DSDF: COORDINATED LOOK-AHEAD STRATEGY IN STOCHASTIC MULTI-AGENT REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-Agent reinforcement learning has received lot of attention in recent years and have applications in many different areas. Existing methods involving Centralized Training and Decentralized execution, attempts to train the agents towards learning a pattern of coordinated actions to arrive at optimal joint policy. However if some agents are stochastic (noisy) in their actions to varying degrees, the above methods provides poor coordination among agents. In this paper we show how the stochasticity of agents, which could be a result of malfunction or aging of robots, can add to the uncertainty in coordination and thereby contribute to unsatisfactory global rewards. In such a scenario, the deterministic agents have to understand the behavior and limitations of the stochastic agents while the stochastic agents have to plan taking in cognizance their own limitations. Our proposed method, Deep Stochastic Discounted Factor (DSDF), tunes the discounted factor for the agents by using a learning representation of uncertainty to update the utility networks of individual agents. DSDF also helps in imparting an extent of reliability in coordination thereby granting stochastic agents tasks which are immediate and of shorter trajectory with deterministic ones taking the tasks which involve longer planning. Results on benchmark environments shows the efficacy of the proposed approach when compared with existing deterministic approaches.

1 INTRODUCTION

Multi-agent reinforcement learning (MARL) has been applied to wide variety of applications which involve collaborative behavior such as in Traffic management (Chu et al., 2019), power distribution (Nasir & Guo, 2019), fleet management (Lin et al., 2018) etc. There are different methods to encourage this collaboration among agents. While a set of algorithms focus on learning centralized policies (Jaques et al., 2019; Moradi, 2016), some others learn decentralized policies (Zhang et al., 2018). To improve the performance of the decentralized policies, some works leveraged centralized training while learning these policies (Tan, 1993; Rashid et al., 2018; Mahajan et al., 2019). In literature these methods are known as centralized training and decentralized execution (CTDE) methods.

Subsequently, there are many CTDE methods which are proposed for obtaining collaborative multi-agent policy. These include preliminary methods like IQL (Tan, 1993; Xu et al., 2021) which has challenges dealing with non-stationarity and then extends to more recent methods like COMA (Foerster et al., 2018), VDN (Sunehag et al., 2017), QMiX (Rashid et al., 2018), MAVEN (Mahajan et al., 2019), QTRaN (Son et al., 2019) etc. In some other works we can also find some of the variants of these methods like (Xu et al., 2021; Rashid et al., 2020).

All these methods generally assume the agents behave exactly the way as the policy instructed it. However in many cases, the agents can behave inconsistently/stochastically i.e. sometimes they execute the actions different from the actions given by the policy. The degree of inconsistency can be different for different agents i.e. the probability of executing an action different from that of policy can vary. In the rest of the paper, we call the stochastic agents with the term noisy/degraded agents. This is a common phenomena in industries where agents (machines etc.) may undergo wear and tear and subsequently are degraded which can result in noisy actions. To explain it better let us consider the following example.

Consider the case of soccer match wherein one or more players are injured and hence their movement and precision(for taking long shots) are impacted. Let us assume that these players cannot be replaced and all the players have similar skills. So the intuitive team strategy would be to let the injured players operate within a

small radius and just perform small distance passes while leaving the dribbling/running with the ball to other fit players. Effectively, injured players take simpler short term objectives (passing the ball) while other fit players take relatively longer and complex goals (like dodging the opponent and running with the ball for longer distances). This in turn means all the players would need to refactor their look-ahead strategy and re-tune the respective policies such that reward of overall joint policy is maximized. Such refactoring would also be essential in a robotic environment, like robots coordinating in a warehouse or in an Industry 4.0 use case for assembly line manufacturing, where some of the robots may get degraded due to wear-n-tear over time. Maintenance or replacement for robots is a costly process and needs to be done even when one of the many robots is slightly degraded in its abilities. The concept described here could enable continuation of the manufacturing process as one robot adjusts to other's short commings. This in turn saves cost, increases efficiency and partially contribute to reducing the carbon footprint by increasing lifetime of a robot.

In the context of RL, the refactoring of look-ahead strategy for each robot can be achieved by adjusting the discounted factor. For example, noisy (inconsistent) agents should use lower discounted factor i.e. they should plan for short-term rewards as they are uncertain on the future whereas the good and accordant agents should use higher discounted factor i.e. they should plan for long-term rewards. However the choice of tuning of discounted factor for each agent is non-trivial. Since the discounted factor of one agent will have an effect on another agent's strategy, such a tuning can only be done based on a unified representation of state of the environment and differential capabilities of other agents.

Hence in this work we propose an efficient representation which could help all the agents to automatically understand an effective discounted factor. We call this method as Deep Stochastic Discounted Factor (DSDF) method, which predicts the discounted factor based on a hypernetwork based Learning representation. The method leverages observation space of the agents and state of the environment. The proposed method computes the discounted factor for all agents simultaneously and thereby promotes effective cooperation in context of the degraded scenario. More details on the method are proposed in Section 3.

In addition, we also came with an iterative penalization method which will penalize the discounted factor by a factor for each agent based on the noisy action. It should be noted that this method does not consider the dependency of other agents in tuning the discounted factor .

We make an assumption here that the change in the agent's noisy behavior can be a represented by a step function rather than being a continuous one. That means the noisy agents would retain their performance until certain threshold of further degradation. For each such threshold breach, the agents will require a bit of retraining, however the discounted factor learning representation does not need any retraining. The quantification of the threshold is not part of this paper and left for future work.

2 BACKGROUND

In this work, we assume a fully cooperative multi-agent task which is described with a decentralized partially observable Markov decision process (Dec-POMDP) which is defined with a tuple $G =< S, \mathbf{U}, P, r, Z, O, N, \gamma >$ where $s \in S$ describes the state of the environment Bernstein et al. (2002). At each step in time, each agent i out of these N agents will take an action a_i and for all the agents the join action is represented by U. Due to the applied actions, the system will transit to $P(s'|s, \mathbf{U}) : \mathbf{S} \times \mathbf{U} \times \mathbf{S} \longrightarrow [0, 1]$. All the agents share common reward function $r(s, \mathbf{U} : \mathbf{S} \times \mathbf{U} \in \mathcal{R}$ and $\gamma : N \times 1 \longrightarrow [0, 1]$ is the discounted factor chosen for N agents.

Here we consider the environment is partially observable in which agent draws individual observation $z \in \mathbf{Z}$ according to a observation space $O(s, a) : \mathbf{S} \times \mathbf{A} \longrightarrow \mathbf{Z}$. Each agent can have their own observation history $\tau_i \in \tau : (\mathbf{Z} \times \mathbf{U}$ which influences the underlying stochastic policy $\pi^i(u^i|\tau^i)$. The joint policy π has a join action-value function $Q^\pi(s_t, u_t) = E_{s_{t+1:\infty}, u_{t:\infty}}[R_t|S_t, u_t]$, where $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$.

3 PROPOSED METHOD

In this section, we describe the proposed approach which leads to a significantly improved joint policy even when some agents are noisy and degraded. An important assumption here is **"All the agents are conformal and accordant with policy while starting the experiment. However, after a while, some of the agents degrade and we need to retrain the policy from scratch so that we address these noisy agents. During**

the execution phase also we assume the same agents are noisy with same degree of degradation. This is required to circumvent the usage of continuous learning during execution which is outside the scope of this paper."

The proposed approach is explained in context of QMIX (Rashid et al., 2018) as base collaborative mechanism between agents since QMIX is considered as one of state of art collaborative mechanism. But, DSDF can be extended to any other collaborative mechanisms which satisfies centralized training and decentralized execution.

As discussed, we propose to compute the discounted factor for each agent based on the current observation and also the global state of the environment. We propose two methods to compute the optimal γ_i value for agent i. (i) Iterative penalization method where we penalize the discounted factor of each agent for every noisy action taken and (ii) DSDF- A hypernetwork based representation of system to compute the appropriate discounted factors for each agent. For this we train a fully connected neural network which will output the discounted factor values for each agent i. In both the above cases, the computed discounted factor is fed to the individual utility networks and used to compute the y, where $y = r + Q_{tot}(s', \mathbf{u}, \theta^-, \gamma)$, where θ^- is the target network. The previous equation can be rewritten as $y = r + Q_{tot}(s', \mathbf{u}, \theta^-, \gamma_i)$, where γ_i is the discounted factor chosen for the i^{th} agent. Now, we update both the utility networks, mixing network (hypernetwork to be exact) using the different discounted factors for each agent.

3.1 Proposed methods to calculate appropriate γ for all the agents

In this work, we propose two methods to compute the appropriate γ for all the agents.

3.1.1 Iterative penalization Method

In this method, we assume the discounted factor is 1 for all the agents during the starting of the experiment i.e. during retraining from scratch. If the action executed by the agent i is different from that of action given by the policy we will penalize the discounted factor for the agent i by a factor P. At every one time step, with every mismatch we will decrease the discounted factor with the factor P. Now, we will use the latest discounted factor at the time step to compute the utility function. Finally, we use the regular QMIX approach to update the utility networks and mixing networks.

The penalization factor P should decrease with time steps just like we do in exploration factor in ϵ-greedy method. The choice of optimal value for P can be itself posed as optimization problem which is out of scope of the paper. However, the method proposed in Section 3.1.2 does not require any hyperparameter selection.

3.1.2 Deep Stochastic Discounted Factor (DSDF) Method

In this case, we propose a method to compute the appropriate $\gamma_i, \quad i = 1, \cdots, N$ using a trained network. The proposed method is shown the Figure 1.

The idea is to utilize the agent local observations $o_{t,i}$ and global state s_t at time instant t of the environment to compute the discounted factor for individual agents. The reason behind is explained below:

Since we assume the underlying collaborative mechanism works for accordant agents, the only issue behind poor global return is the degradation of some agents and the noise in action selection. Each agent have their perspective of the global state in form of local observations. Hence local observations will have an idea on the noise in the degraded agents and we need to estimate the discounted factor depending on the local observations and also the global state of the environment.

Returning to the main discussion, we use the local observations of agents to estimate the discounted factors using a fully connected network θ_γ. However, we require additional information about the global state to the process. Since the local observations and global state are in different scale, we cannot combine together in same network. Also, we have non-negativity constraint on the γ values which is enforced by training the θ_γ using the concept of hypernetwork as described in (Ha et al., 2016).

The local observations of all N agents are sent to the network θ_γ which will compute the discounted factor values $\gamma_i, \quad i = 1, \cdots, N$. We utilize the hypernetwork θ_h to estimate the network weights θ_γ. The training process to update the θ_h is explained below.

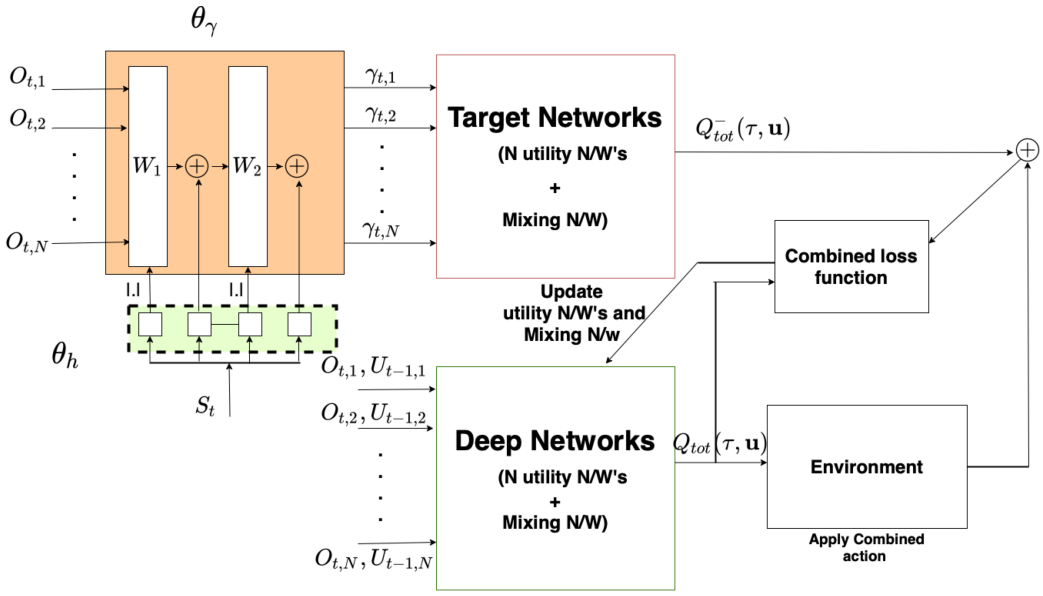


Figure 1: Proposed DSDF approach. The individual target Q-values are multiplied with predicted discounted values and combined with reward obtained from environment. Finally we use this value to update the N utility networks and mixing network (In case of QMIX approach)

The θ_h, θ_u (utility networks) and θ_m (mixing hypernetwork) are interlinked with each other. The general loss function of QMIX is

$$\mathcal{L}(\theta) = \sum_{t=1}^{B} \left(y^t - Q_{tot}(\tau, \mathbf{u}, s : \theta)\right) \tag{1}$$

where θ is the set of parameters of N utility agents $(\theta_{u,i}, i = 1, \cdots, N)$ and mixing hypernetwork θ_m, computed over B episodes. Now, if we expand y^t

$$y^t = r + \gamma \max_{u'} Q_{tot}(\tau', \mathbf{u}', s' : \theta^-) \tag{2}$$

Now, instead of using single γ value we will take the γ inside the equation and we use the mixing network function to calculate the value of Q_{tot}

$$y^t = r + \max_{u'} g(\mathbf{u}', s', \gamma_i Q_{u,i}, \theta_{tot}) \tag{3}$$

Here $g(.)$ is the mixing network architecture which is parametrized by θ_{tot}, $Q_{u,i}$ is the individual utility function of agent i. Now, we will replace the γ_i with output of the network θ_γ. The replaced equation is

$$\begin{aligned} y^t &= r + \max_{u'} g(\mathbf{u}', s', f_\gamma(o_1^t, \cdots, o_N^t, \theta_\gamma) Q_{u,i}, \theta_{tot}) \\ &= r + \max_{u'} g(\mathbf{u}', s', f_\gamma(o_1^t, \cdots, o_N^t, f_h(\theta_h, s')) Q_{u,i}, \theta_{tot}) \end{aligned} \tag{4}$$

where f_γ is the discounted factor hyper network which is parametrized by θ_γ and f_h is the hypernetwork function which is parametrized by θ_h.

Replacing the value of y_i^{tot} from equation 11 with that of equation 6 we obtain the loss function as

$$\mathcal{L}(\theta, \theta_h) = \sum_{t=1}^{B} \left(r^t + \max_{u'} g(\mathbf{u}', s', f_\gamma(o_1^t, \cdots, o_N^t, f_h(\theta_h, s')) Q_{u,i}, \theta) - Q_{tot}(\tau, \mathbf{u}, s : \theta\right) \tag{5}$$

There are two unknown parameters in the above equation (i) θ, (ii) θ_h. Since the parameters are interdependent on each other i.e. θ_h on θ and vice-versa, we need to solve them iteratively. For every B samples, first we will update the hyper network θ_h for fixed θ and then update θ for the computed θ_h. So at each step we update the both θ_h and θ iteratively.

Algorithm 1 DSDF method to estimate discounted factor with QMIX

Require: Initialize parameter vector θ_h, hypernetwork parameters and θ (agents utility networks, maxing network, hyper network), Learning rate $\leftarrow \alpha_\gamma$ and α_θ, $\mathcal{B} \leftarrow \{\}$
Require: step = 0, $\theta^- = \theta$
  while step < step_{max} **do**
    $t = 0$, s_0 = Initial state
    while $t \neq$ terminal and $t <$ episode limit **do**
      for each agent i **do**
        $\tau_t^i = \tau_{t-1}^i \cup \{(o_t, u_{t-1}\}$
        $\epsilon =$ epsilon-schedule (step)
        $$u_a^t = \begin{cases} \underset{u_t^i}{\text{argmax}} \; Q(\tau_t^i, u_t^i) & \text{with probability } 1-\epsilon \\ \text{randint}(1, |U|) & \text{with probability } \epsilon \end{cases}$$
      end for
      $s_{t+1} = p(s_{t+1}|s_t, \mathbf{u}_t)$
      $\mathcal{B} = \mathcal{B} \cup \{(s_t, \mathbf{u}_t, r_t, s_{t+1}\}$
      $t = t+1, step = step+1$
    end while
    if $|\mathcal{B}| >$ batch-size **then**
      b $\leftarrow$ random batch of episodes from $\mathcal{B}$
      if θ_h not converged **then**
        Update $\theta_h = \theta_h - \alpha_\gamma \nabla_{\theta_h} (\Delta Q_{tot})^2$
      end if
      Update $\theta_\gamma = f_\gamma(O, \theta_h)$, where O is the set of observations for all agents in the sampled batch.
      Update Q_{tot} using the latest updated θ_γ.
      Update $\theta = \theta - \alpha_\theta \nabla_\theta (\Delta Q_{tot})^2$
    end if
    if update-interval steps have passed **then**
      $\theta^- = \theta$
    end if
  end while

An important point to note here is that θ needs to be updated for every batch of samples as each batch will have new information on the environment. However in real world the degree of degradation might become stationary after some time (an injured soccer player or a robot degrades till a point but holds on to the level of degradation). Consequently the noise in the degraded agents may remain constant for a while. Hence the θ_γ network will converge after some iterations once it has figured out the right values of discounted factor. Hence at that point there is no need to update the θ_γ and θ_h. However it should be noted that the discounted factor value will change depending on the local observations and state of the environment during the execution. Please refer to Figures 2a, 2b and 2c, where training of θ_γ is done on one experiment (one scenario of degradation) and the other experiments used the same θ_γ.

We can also apply the proposed DSDF approach to the case where all the agents are deterministic. In this case, we expect it will give improved performance when compared with existing method (like QMIX) as it will dynamically tune the discounted factor γ for each agent, based on its local observation also, rather than using constant γ for all the agents.

The proposed DSDF method to estimate the discounted factor with existing QMIX method is explained in algorithm 1. It has the following two advantages when compared with iterative penalization method.

1. DSDF method can dynamically change the discounted factor whereas the simpler method of penalizing it will only decrease it. Also, for the accordant agents, the discounted factor is always 1 and hence it may degrade those agents performance for accidental deviation.

2. DSDF method calculates the discounted factor of an agent relative to other agents. Hence it can consider complex interactions between agents while deciding discounted factor for the agents.

4 RESULTS AND DISCUSSION

The proposed approach is validated on two different benchmark environments (i) SMAC (Samvelyan et al., 2019b) and (ii) Modified lbForaging (Papoudakis et al., 2020). To induce the degradation/noise in the agents, a noisy component is added which will decide whether to execute the action given by the policy or to take a random action from set of valid actions.

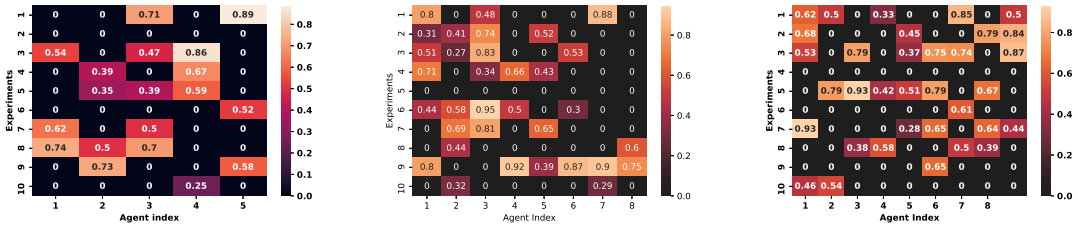
4.1 DESCRIPTION OF THE ENVIRONMENTS

4.1.1 STARCRAFT ENVIRONMENT

SMAC (Samvelyan et al., 2019b) simulates the battle scenarios of a popular real-time strategy game StarCraft II. In this challenge, a team of units, each controlled by an agent observes other units within a fixed radius and takes actions based on their local observations. These agents are trained to solve challenging combat scenarios known as maps. Similar state space, observation space, action space and reward are used as with the (Rashid et al., 2018). The results were compared on three different set of maps (i) 2s_3z, (ii) 8m and (iii) 1c_3s_5z.

For each map, 10 **different sets of experiments were performed with different number of noisy agents in each experiment and with varying degree of degradation values.** For each experiment, the probabilistic degradation values β were assigned to the agents, as shown in the Figures 2a, 2b and 2c.

Here β means the agent will perform actions given by the policy with $1 - \beta$ probability and will perform random actions with β probability. The degradation value of 0 implies the agent is good or accordant with policy. It should be noted that the experiments also included the case where all the agents are accordant to demonstrate the applicability of the proposed approach to general case where there are no noisy agents.



(a) 2s_3z (b) 8m (c) 1c_3s_5z

Figure 2: Degradation values of agents considered in each experiment for SMAC

4.1.2 LBFORAGING ENVIRONMENT

This environment contains agents and food resources randomly placed in a grid world. The agents navigate in the grid world and collect food resources by cooperating with other agents. The environment was modified by adding a couple of additions to the environment as mentioned in the appendix A.

100 food resources were placed in the 30×30 grid and 6 agents were chose to consume those resources. The episode is terminated either when there are no food resources available in the grid or number of time steps reached 500. Here out of 6 agents, three agents 3 are deterministic (1,3,4) and 3 are stochastic (2,5,6) with degree of stochasticity of $0.2, 0.4$ and 0.6 respectively. The targets were chosen for individual agents such that there are exact amount of food resources available to achieve targets for all the agents.

4.2 RESULTS ON ENVIRONMENT

In this section, experiments were performed on the above two environments with proposed DSDF, iterative penalization, QMIX and IQL. The pymarl library was utilized for the same (Samvelyan et al., 2019a).

4.2.1 SMAC

To evaluate the performance of the agents here, the training was paused after every 100 episodes and testing for 20 episodes. The plot of the % test winning episodes for the experiment index 1 (as per Figure 2a) is shown in the Figure 3a. The discounted factor plot is not given for this example as the environment is complex and will take many time instants to converge which makes the graph difficult to read.

Notably, the learning representation of discounted factor which is the key part of proposed DSDF approach is trained only on the experiment 1 for each of the three map scenarios. For remaining 9 experiments, the respective trained learning representation was utilized to provide discounted factor γ_i value for each agent i.

From the plot it can be seen that the IQL approach performed poorly since the two of the agents are noisy and thus it resulted in poor collaboration between the agents. Although QMIX gave good performance when compared with IQL, it also settled at around 40% winning rate. Both these methods performed below expectation as they look too much into future with the choice of highest discounted factor ($\gamma = 0.92$) which leads to poor planning.

The iterative penalization method improves the collaboration further to 60%, whereas the proposed DSDF method improved collaboration by achieving the top value of 95%. The reason being the discounted factor is being predicted by non-linear network which utilizes the complex interactions between agents to come with an appropriate discounted factor value when compared to iterative penalization method.

For each experiment in all the three map scenarios, the proposed DSDF was evaluated along with the existing methods. The plot of the % test wins (obtained at the time instant given next to the map names) obtained for each experiment in respective map scenarios (i) 2s_3z (220×10^4), (ii) 8m (200×10^4) and (iii) 1c_3s_5z (300×10^4) is shown in Figures 3b, 3c and 3d respectively. From the plots it is evident that the proposed DSDF approach performs better than all the existing methods, which shows the efficacy of the proposed DSDF approach. It is also evident that the proposed DSDF approach performs better even when composition of noisy agents comprises about 75% of total number. In addition the iterative penalization method also performs better than the existing methods. The proposed approach also performs well when compared with QMIX for the case where all the agents are deterministic (experiment 2 for 2s_3z, experiment 5 for 8m and experiment 4 for 1c_3s_5z) which shows the generalization of the approach. The reason is that the proposed DSDF approach dynamically chooses the best discounted factor based on the agent's degradation (current capability of the agent) whereas the existing methods use constant discounted factor.

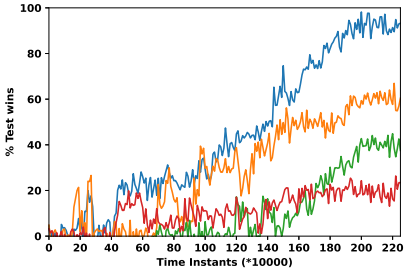
Since credit assignment of individual agents is a challenge in SMAC environment, the performance of individual agents is not depicted. It is for this very reason, the results are then analyzed on the lbForaging environment.

4.2.2 LBFORAGING

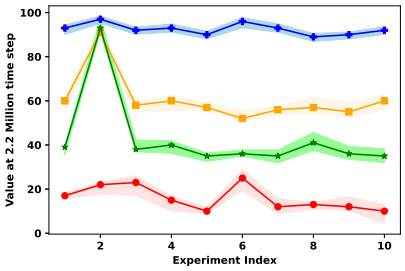
The individual targets for respective agents are $20, 20, 30, 60, 30$ and 60. The sum of all food levels available in the grid were restricted to a number T such that total sum of targets for individual agents is equal to T. **An important point to be noted that the value of T is not known to any agent during training or in execution.** Here the sum of targets is chosen to be $T = 220$. The setting necessitated the agents to collaborate within themselves to reach their respective targets.

The predictions of the discounted factor values using DSDF method for all the time steps is shown in the Figure 4a. As can be seen, the agents discounted factor changes whenever we update the discounted factor network. From the plot, it can be observed that the values got almost saturated after some updates which shows the hypernetwork is converging and hence after this time the discounted factor hyper network need not be updated. It should be noted the discounted factor values will change (depending on the local observations and state of the environment) with every batch of samples.

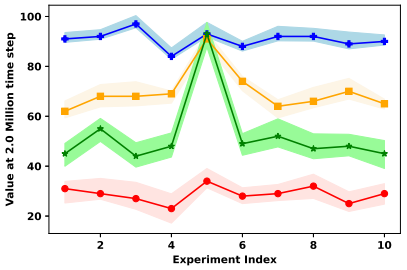
Also, it can be observed the discounted factor values for accordant agents are higher ($>= 0.9$) which suggests the algorithm makes the respective utility functions depend more on the future values. This is quite evident as these agents need to look more into future and decide current actions. On the other hand noisy agents should choose lower discounted factor so that they won't take future rewards into consideration. From Figure 4a, it can be observed that the agents with high degree of degradation i.e. they have less probability of executing actions given by the policy, uses less discounted factor. This is in accordance with the assumption that the more the noise in the agents, the less the discounted factor value should be and vice-versa.



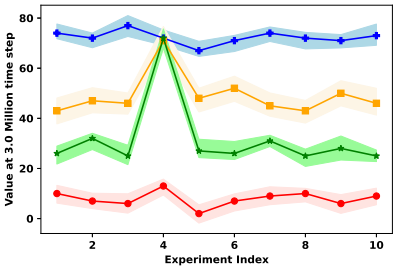
(a) Time instant wise rewards for Experiment 1 for 2s_3z map



(b) % battles won for the case 2s_3z

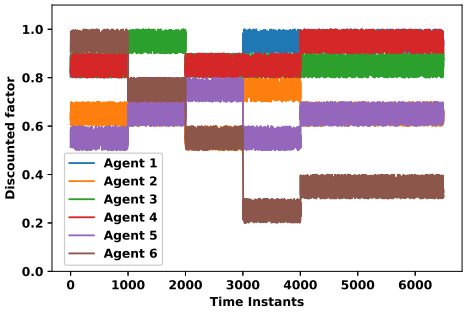


(c) % battles won for the case 8m

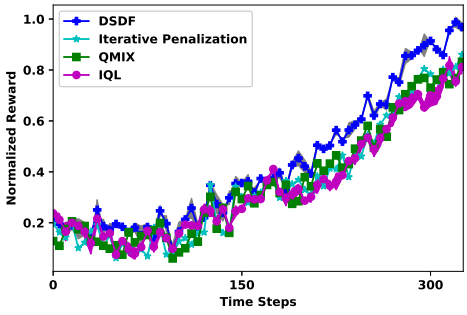


(d) % battles won for the case 1c_3s_5z

Figure 3: Returns obtained for StarCraft environment for all the experiments for different map scenarios along with confidence intervals. The dots in the figures 3b, 3c and 3d are connected for intuitive comparison. The proposed DSDF approach outperforms the existing methods and iterative penalization technique.



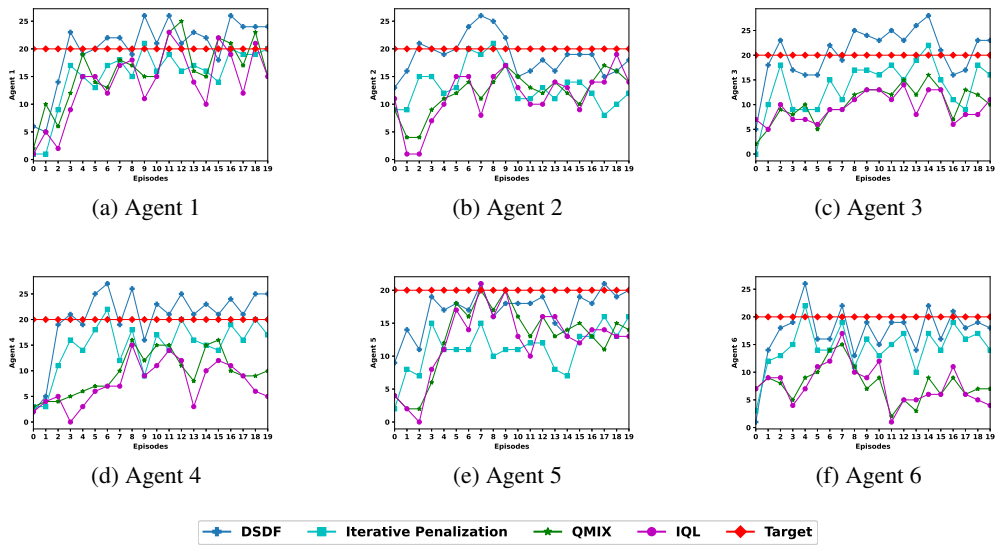
(a) Discounted factor prediction



(b) Normalized global returns for correct resources

Figure 4: Returns obtained for lbForaging environment along with discounted factor prediction plot. The advantage obtained using DSDF approach when compared with existing methods is not significant here since the environment requires lesser collaboration between agents when compared with complex SMAC environments. However, the results for this environment is depicted as credit assignment for individual agents is possible here.

The performance of the individual agents are shown in Figure 5. In all the plots, first 10 indices correspond to latest 10 training episodes and next indices correspond to 10 execution episodes. From the plots, it can be concluded that accordant agents, except agent 4, trained using DSDF method reached their targets in



(a) Agent 1 (b) Agent 2 (c) Agent 3

(d) Agent 4 (e) Agent 5 (f) Agent 6

Figure 5: Comparison of individual agents performance using proposed DSDF approach vs existing approaches for lbForaging environment. One can observe the deterministic agents achieved targets in almost all the test episodes whereas the stochastic agents consumed more resources when compared with existing methods. This confirmed our look-ahead strategy works better and results in good collaboration

all the execution episodes. On the other hand, the accordant agents trained with vanilla QMIX and IQL have also reached their respective targets. The performance of the noisy agents trained using the proposed DSDF method as well as iterative penalization method outperforms the vanilla QMIX and IQL. All the noisy agents with smaller targets reached their respective targets while the agent with higher target settled closer to the target value. For the case of accordant agents, the agents trained using proposed DSDF method either performed better (Agent 3 and 4) or shown comparable performance (Agent 1) when compared with QMIX and IQL. This could be attributed to the DSDF learning representation which helps accordant agents realize the limitation of noisy agents and thereby assume additional work-load to compensate for underperformance of noisy agents. On the other hand, the agents trained with iterative penalization method demonstrates at-par performance with QMIX and IQl.

The mean reward obtained for every time step with 95% confidence interval during evaluation is shown in Figure 4b. From the plot, it is evident that the proposed DSDF method resulted in higher average reward when compared to iterative penalization method and other state-of-art methods. The agents trained with iterative penalization method also fared comparably well when compared with QMIX and IQL methods.

5 Conclusion

In this paper, we propose a novel method, DSDF, to handle a mix of accordant and noisy agents which together learns a collaborative joint policy. Along with the above, an Iterative Penalization method is proposed, which though has lesser gains than DSDF. Our proposed methods can be combined with any state of art MARL algorithms without much impact to existing computational complexity.

Our proposed method is tested on the two different environments and demonstrated clear improvement in results when compared with QMIX and IQL methods both for individual and global returns. Future directions include extending the technique to situations where environment might also be noisy in regions (like oil patches on floor). To the best of our knowledge, this is the first time, such an interaction involving degraded/noisy and accordant agents have been explored in context of multi-agent reinforcement learning.

The method could have significant usage in reducing maintenance cost for future Industry 4.0 scenario aided by low-latency 5G communication where multiple robots might need to co-ordinate in a factory floor.

REFERENCES

Daniel S Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of operations research*, 27(4):819–840, 2002.

Tianshu Chu, Jie Wang, Lara Codecà, and Zhaojian Li. Multi-agent deep reinforcement learning for large-scale traffic signal control. *IEEE Transactions on Intelligent Transportation Systems*, 21(3):1086–1095, 2019.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 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. PMLR, 2019.

Kaixiang Lin, Renyu Zhao, Zhe Xu, and Jiayu Zhou. Efficient large-scale fleet management via multi-agent deep reinforcement learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1774–1783, 2018.

Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. Maven: Multi-agent variational exploration. *arXiv preprint arXiv:1910.07483*, 2019.

Milad Moradi. A centralized reinforcement learning method for multi-agent job scheduling in grid. In *2016 6th International Conference on Computer and Knowledge Engineering (ICCKE)*, pp. 171–176. IEEE, 2016.

Yasar Sinan Nasir and Dongning Guo. Multi-agent deep reinforcement learning for dynamic power allocation in wireless networks. *IEEE Journal on Selected Areas in Communications*, 37(10):2239–2250, 2019.

Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V Albrecht. Comparative evaluation of multi-agent deep reinforcement learning algorithms. *arXiv preprint arXiv:2006.07869*, 2020.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 4295–4304. PMLR, 2018.

Tabish Rashid, Gregory Farquhar, Bei Peng, and Shimon Whiteson. Weighted qmix: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning. *arXiv preprint arXiv:2006.10800*, 2020.

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philiph H. S. Torr, Jakob Foerster, and Shimon Whiteson. The StarCraft Multi-Agent Challenge. *CoRR*, abs/1902.04043, 2019a.

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philiph H. S. Torr, Jakob Foerster, and Shimon Whiteson. The StarCraft Multi-Agent Challenge. *CoRR*, abs/1902.04043, 2019b.

Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 5887–5896. PMLR, 2019.

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.

Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *In Proceedings of the Tenth International Conference on Machine Learning*, pp. 330–337. Morgan Kaufmann, 1993.

Xing Xu, Rongpeng Li, Zhifeng Zhao, and Honggang Zhang. Stigmergic independent reinforcement learning for multiagent collaboration. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

Kaiqing Zhang, Zhuoran Yang, Han Liu, Tong Zhang, and Tamer Basar. Fully decentralized multi-agent reinforcement learning with networked agents. In *International Conference on Machine Learning*, pp. 5872–5881. PMLR, 2018.

Appendix

A MODIFICATIONS TO LBFORGAING ENVIRONMENT

The modifications to the environment are as follows.

1. We added two additional actions '**Carry on**' and '**Leave**' to the agents. The '**Carry on**' action enables the agent to store the food resources which they can subsequently leave in another time step and/or state for consumption of another agent. The '**Leave**' action enables to agent to drop the resources which they consumed.
2. Each agent is given a target of the resources they need to consume. For this, we modify the reward function by adding targets to the them. Our eventual goal is to ensure all agents reach their targets. This means if some of the agents consumed in excess they must realize and give up the extra resources for benefit of other agents.

B COMBINED LOSS FUNCTION USED IN THE WORK

The general loss function of QMIX is

$$\mathcal{L}(\theta) = \sum_{t=1}^{B} \left(y^t - Q_{tot}(\tau, \mathbf{u}, s : \theta\right) \tag{6}$$

where $\theta = [\theta_{u,i} \; \forall \; i = 1, \cdots, N \;\; , \theta_{tot}]$ is the set of parameters of N utility agents and mixing network. Now, if we expand y^t

$$y^t = r + \gamma \;\; \max_{u'} \;\; Q_{tot}(\tau', \mathbf{u}', s' : \theta^-) \tag{7}$$

Now, instead of using single γ value we will take the γ inside the equation to handle the stochastic agents

$$y^t = r + \max_{u'} \;\; g(\mathbf{u}', s', \gamma_i Q_{u,i}, \theta^-_{tot}) \tag{8}$$

with i ranges from $1, \cdots, N$. Here $g(.)$ is the target network of mixing network architecture which is parametrized by θ^-_{tot}, $Q_{u,i}$ is the individual utility function of agent i. The individual utility function can be represented as

$$Q^t_{u,i} = f_{u,i}(o^t_i, u^{t-1}_i, \theta^-_{u,i}) \tag{9}$$

where $f_{u,i}$ is the function of the i^{th} utility network paramterized by $\theta_{u,i}$.

Substituting equation 9 in equation 8 gives

$$y^t = r + \max_{u'} \;\; g(\mathbf{u}', s', \gamma_i f_{u,i}(o^t_i, u^{t-1}_i, \theta^-_{u,i}), \theta^-_{tot}) \tag{10}$$

In the paper, the idea is to predict the $\gamma_i \;\; \forall \;\; i = 1, \cdots, N$ and use them to scale the predicted Q-values of individual agents from the network. Now, we will replace the γ_i with output of the network θ_γ. The replaced equation is

$$\begin{aligned} y^t &= r + \max_{u'} \;\; g(\mathbf{u}', s', f_\gamma(o^t_1, \cdots, o^t_N, \theta_\gamma) f_{u,i}(o^t_i, u^{t-1}_i, \theta^-_{u,i}), \theta^-_{tot}) \\ &= r + \max_{u'} \;\; g(\mathbf{u}', s', f_\gamma(o^t_1, \cdots, o^t_N, f_h(\theta_h, s')) f_{u,i}(o^t_i, u^{t-1}_i, \theta^-_{u,i}), \theta^-_{tot}) \end{aligned} \tag{11}$$

where f_γ is the discounted factor hyper network which is parametrized by θ_γ and f_h is the hypernetwork function which is parametrized by θ_h. The hypernetwork θ_h is

Replacing the value of y_i^{tot} from equation 11 with that of equation 6 we obtain the loss function as

$$\mathcal{L}(\theta, \theta_h) = \sum_{t=1}^{B} \Big(r^t + \max_{u'} g(\mathbf{u}', s', f_\gamma(o_1^t, \cdots, o_N^t, f_h(\theta_h, s')) f_{u,i}(o_i^t, u_i^{t-1}, \theta_{u,i}^-), \theta_{tot}^-) - Q_{tot}(\tau, \mathbf{u}, s : \theta \Big) \tag{12}$$

Replacing the agent utility networks function equation 9 in the equation 12 gives

$$\mathcal{L}(\theta, \theta_h) = \sum_{t=1}^{B} \Big(r^t + \max_{u'} g(\mathbf{u}', s', f_\gamma(o_1^t, \cdots, o_N^t, f_h(\theta_h, s')) f_{u,i}(o_i^t, u_i^{t-1}, \theta_{u,i}), \theta_{tot}^-) - Q_{tot}(\tau, \mathbf{u}, s : \theta \Big) \tag{13}$$

In QMIX, the θ_{tot} is computed by a separate hypernetwork which will update the weights from a hypernetwork.

In this work, we use the combined loss function in equation 13 to estimate the θ and θ_h.

C Details of the networks used in SMAC environment example

We used the exact QMIX network architectures for this environment as used in pymarl library (Rashid et al., 2018).

D Details of the networks used in lbForaging environment example

Below are the details of the networks we used in the implementation example

---

Implementation details used in the paper

Environment Parameters

1. Number of agents - 6
2. Number of food resources - 100
3. Grid size - 30×30
4. Sight - 10
5. Max Episode steps - 1000
6. Cooperation - True
7. Max player level - 20
8. Stochastic level of agents - [0,0.2,0,0,0.3,0.6]

Network Parameters

1. Agents utility networks are 3-layer networks. First layer of the network is MLP with 318 nodes followed by GRU layer with final layer being MLP layer with 8 nodes output.
2. Mixing network is chosen as two layer fully connected network with 6 inputs nodes in the first layer and eight output nodes in the last layer.

3. Mixing hypernetwork is also chosen as two layers with 40-node first layer followed by another layer matching mixing layer weights dimension.
4. Discounted factor network is also chosen as two layer network with 384 nodes in first node followed by 6 output nodes in the last layer.
5. Discounted factor hypernetwork is chosen as two layers which will match the dimension of the discounted factor network.
6. The learning rate for all the networks is chosen to be 0.96 since at this learning rate we achieved good results.
7. Discounted factor values for the QMIX and IQL methods are chosen as 0.92. This value is obtained after a grid search.