A Bayesian Approach to Learning Command Hierarchies for Zero-Shot Multi-Agent Coordination

Timothy Flavin
University of Tulsa
Tulsa, United States
Timmy-Flavin@utulsa.edu

Sandip Sen
The University of Tulsa
Tulsa, United States
Sandip-Sen@utulsa.edu

ABSTRACT

An ongoing challenge in Multi-Agent reinforcement learning (MARL) is to develop algorithms which can leverage very limited amounts of experience to coordinate with new teammates. Recent work have focused on generating diverse training partners and generalizing to those partners. In this paper, we propose a modular solution called Multi-Armed Two-way Command Heuristic (MATCH) that can be added on to existing agents to learn a command hierarchy within a single episode so that a group of agents may approach the competency of the best agent in the group. We view learning to communicate as a set of non-stationary multi-armed bandit (MAB) problems where each agent has dedicated incoming and outgoing command MAB samplers that adjusts their policies. When giving commands, each agent's goal is to choose the subject who is most likely to follow their command. When receiving commands, each agent uses it's own estimate of the expected benefit of following a particular commander to decide who to follow. We show that competent agents are able to quickly adapt to incompetent teammates by commanding and ignoring them whereas incompetent agents learn to follow commands from more skilled teammates. If pre-trained agents are capable of sending or receiving commands before adding our communication structure, the agent's desired actions are used as a prior distribution which will influence the MAB samplers to mitigate exploration regret.

KEYWORDS

MARL, Trust, Communication, Multi-Armed Bandit

1 INTRODUCTION

One of the main challenges of MARL as opposed to a traditional reinforcement learning (RL) is the problem of nonstationary environment dynamics [21]. MARL algorithms often use centralized training / parameter sharing [13, 21], low learning rates, or learned gradient-based communication [25] to address the problem of non-stationary environment dynamics. These techniques stabilize the environment by ensuring that agents have a good estimate of the behavior of other agents either because they change slowly with low learning rates, or because they have centralized information through duplicated parameters or communication. The problem we attempt to solve in this research is a special case of non-stationarity where agents have to work with never before seen teammates [18]. One approach is to treat other policies as a part of the environment to be generalized away by training with a diverse set of agents. Recent work with this approach has shown success in zero and few-shot scenarios through opponent modeling [2], other play [8], and an evolutionary strategy using two populations to keep diverse pairings between agents [24]. Other studies use Social Learning to allow agents to learn by observing their successful teammates [6, 9] or a hand picked command structure where one agent Coaches it's teammates from a global perspective [11]. Alternatively, one may utilize recent advances in RL sample efficiency such as Transfer learning [19, 26], Meta learning [5, 17, 23], or semantic embedding [15] to allow an agent to simply learn it's policy within a few episodes where adapting to each set of partners is an RL task and the meta-level task is to learn how to adapt to all partners.

This paper takes a different approach. Rather than updating the parameters of a trained policy, we use a communication framework composed of incoming and outgoing commands to enable better group performance in a zero shot scenario. Both incoming and outgoing command selections are treated as non-stationary multi-armed bandit, (MAB), problems [10] where the outgoing MAB learns which teammates are likely to listen and the incoming MAB learns which teammates' commands are worth following. A command in this context is an action provided by one agent, the speaker, to another agent, the listener. The listener may then choose to take either the action contained in the command or it can choose an action based on it's own policy.

Our approach serves as an augmentation to existing agents such as ones trained with the MARL algorithms mentioned above. If giving and receiving commands is already a part of an agent's capabilities, that agent's choices, with regard to commands, can be used as a prior distribution to our method based on the agent's view of the current state of the environment. This approach combines prior state-based information with limited recent experience with particular names teammates to create an enhanced model. Our algorithm allows agents to quickly adapt to new teammates without risking policy collapse by quickly retraining large models.

2 MULTI-ARMED TWO-WAY COMMAND HEURISTIC (MATCH)

We now present details of our approach to a decentralized learning of a command hierarchy by agents.

2.1 Learned Value of Commands

From the perspective of an agent $a_i \in \mathcal{A}$ where $\mathcal{A}$ is the set of agents in the environment, an outgoing command is a suggested action given by a_i to some other agent $a_j \in \mathcal{A}$. An incoming command for agent a_i is a suggested action from agent a_j to agent a_i. The selection of outgoing commands is modeled as a multi-armed bandit problem where the reward for instructing $a_j \in \mathcal{A}$ is 1 if a_j follows the command, and -1 if it does not. Note that an agent may command itself when $i = j$, which is desirable in the case that other

Proc. of the Adaptive and Learning Agents Workshop (ALA 2024), Avalos, Milec, Müller, Wang, Yates (eds.), May 6-7, 2024, Auckland, https://ala2024.github.io/. 2024.

agents stop listening due to a_i sending poor commands. The outgoing MAB problem is necessarily non-stationary because teammates may change their probability of listening over time. The reward for incoming commands is based on the recipient agent's estimation of advantage after following a command. While listening, agent's must estimate either a value function $V(s)$ or a Q function $Q(s, a)$ to evaluate if the command helped. Advantage, A, is calculated with either of the following equations.

$$A = r_t + \gamma V(s_{t+1}) - V(s_t),$$

$$A = r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t).$$

Here, s_t refers to the state at time t and a_t refers to the action taken at time t. The discount factor γ accounts for the discount for future rewards, and r_t is the reward received at time t.

If an agent chooses to follow a command, the advantage will serve as an estimate of whether the command was better or worse than the agent expected to do on it's own. Agents will learn to listen to instructions coming from agents resulting in a positive advantage and they will learn to instruct agent's that follow them most often. A natural, and effective, "command" hierarchy can emerge where agents listen to more competent teammates and instruct less competent ones.

2.2 Incorporating Prior information

In addition to the learned value of giving and receiving commands based on particular pairs of agents, it may be useful to base decisions on context. One option is to model communication as a contextual multi-armed bandit problem, but we believe that learning contextual interactions on top of pair-wise relations for arbitrary environments is not feasible given the data limitations of the zero-shot scenario. If the agent's are capable of giving commands before our system is introduced, then their commands are used as prior information to the MAB. We used three families of MAB samplers to solve the incoming and outgoing communication problem where the hyper-parameters prior strength, p, and experience strength, e, are used by a sampler to determine how quickly an agent will adjust from it's own policy to new information about a teammate.

$$\theta_t = \theta_0 + p\hat{a}_t + er_t \tag{1}$$

The sampler estimate, θ_t, in equation 2.2 is a linear combination of an initial value, θ_0, a desired action generated by the pre-trained agent $\hat{a}_t$ multiplied by the prior strength, p, and recent rewards for communication, r_t, multiplied by the experience strength, e. The first sampler used is a Thompson Sampler [22] based on the Dirichlet distribution [16]. For ϵ-greedy and UCB sampling [10], we used a constant learning rate to serve the non-stationary nature of the problem where the reward for pulling a given arm is the observed advantage. θ_t represents the set of αs for the Dirichlet distribution in Thompson sampling, and the estimated mean rewards for each arm in the ϵ-greedy and UCB approaches.

2.3 Algorithm

We now describe the basic flow of the learning algorithm in MATCH. Given a set of pre-trained agents $\mathcal{A}$, initialize two MAB samplers for each agent where the number of arms in each sampler is equal to the number of agents $|\mathcal{A}|$. One sampler will serve as the incoming or listening sampler that will decide which commands to follow, and the other will serve as the outgoing or speaker sampler that will choose which agents to command. At each time-step, allow agents to communicate in addition to their actions to update their samplers with recent experience.

Algorithm 1: MATCH algorithm

Input: Set of agents A, environment E, Prior weight: p, Evidence weight: e, Discount Factor: γ;
for $a,i \in A$ **do**
  $a.inMAB \leftarrow MAB\_Sampler(|A|, p, e)$;
  $a.outMAB \leftarrow MAB\_Sampler(|A|, p, e)$;
$obs, done = E.start$;
while *not done* **do**
  $c\_dirs \leftarrow O_{|A|x|A|}, commands \leftarrow O_{|A|x|A|}$;
  for $a,i \in A$ **do**
    $t \leftarrow a.outMAB.sample(prior = a.target(obs_i))$;
    $c\_dirs_{t,i} \leftarrow 1$;
    $commands_{t,i} \leftarrow a.command(obs_i, t)$;
    $commands_{i,i} \leftarrow a.policy(obs_i)$;
  $actions \leftarrow list(size = |A|), l \leftarrow list(size = |A|)$;
  $c\_rewards \leftarrow O_{|A|x|A|}$;
  for $a,i \in A$ **do**
    if $sum(c\_dirs_{i,*}) > 0$ **then**
      $recieved \leftarrow c\_dirs_{i,*}$;
      $recieved_i \leftarrow 1$;
      $l_i \leftarrow a.inMAB.sample(choices = recieved)$;
      for $c, j \in c\_dirs_{i,*}$ **do**
        if $c > 0$ *and* $l_i != j$ **then**
          $A_j.outMAB.update(r = -1, arm = i)$
        else if $c > 0$ *and* $l_i == j$ **then**
          $A_j.outMAB.update(r = 1, arm = i)$
      $actions_i \leftarrow commands_{i,l_i}$;
    else
      $actions_i \leftarrow commands_{i,i}$;
      $l_i \leftarrow -1$;
  $n\_obs, r, done \leftarrow E.step(actions)$;
  for $a, i \in A$ **do**
    if $l_i > -1$ **then**
      $Adv \leftarrow (r_i + \gamma * a.V(n\_obs)) - a.V(obs)$;
      $a.inMAB.update(r = Adv, arm = l_i)$;

In Algorithm 1 below l refers to a matrix which keeps track of which agents listened to the commands of other agents during a given step, O_{mxn} refers to the zero matrix of dimensions m and n.

3 BENCHMARK ENVIRONMENTS

We now introduce the environments we used to evaluate our MATCH framework.

3.1 Generic Communication Environment

The first environment we experimented with was designed to show a minimal representation of the problem of learning a command hierarchy. The environment contains three agents and an advantage matrix, A, which specifies the expected advantage, $A_{ij} = \mathbb{E}[Adv(a_i, a_j)]$, that an agent a_j obtains when listening to a_i's commands. Additionally, $\mathbb{E}[Adv(a_i, a_j)] = 0$ when $i = j$ because an agent should do as well it expects when receiving no commands from other agents. At each time step, one of the three agents is chosen at random to send a command to a listener which it will select based on it's outgoing MAB model. The targeted listener of that command will then either follow or ignore the command based on it's incoming MAB model for that commander. The commanders outgoing MAB model will receive a 1 or 0 reward depending on if the listener followed the command or not. If the command is followed, a reward for the incoming MAB model is generated as follows. The advantage recorded in the advantage matrix at A_{ij} is used as the mean to a normal distribution with a variance of 0.25 to sample a value for the advantage of that command. The normal distribution represents a stochastic environment with an approximate value function which would be used to calculate the advantage for having listened to a command.

3.2 Cart Pole Listener

The second environment we use is OpenAI Gym's Cart Pole [4] environment where one agent, the listener, is playing the game and another agent, the speaker, is giving instructions. For each experimental run, one policy is chosen to be the listener and another to be the speaker. At each time step, the speaker gives the listener a command and the listener must decide to follow that command, or to follow it's own policy instead. The reward is the number of frames that the cart is able to balance the pole before either the pole falls or the cart drifts too far from the origin.

3.3 MARL Grid world

The next environment used in this paper is an 8x8 grid world with four agents and consisting of multiple paths and pits. Each agent receives a small negative reward for moving along a path and a large positive reward for reaching the exit. Agents are given a large negative reward and are forced to move randomly if they enter a pit. The game ends when all agents exit. Due to individual rewards, the value function for an optimal agent in this environment was solved using policy iteration [7].

3.4 Modified Reference Environment

The final environment we experiment with allows for speaking, listening, and the incorporation of prior information from communication capable agents to improve team performance. This environment is a modified version of the MPE simple reference environment [12, 14] found in The Farama Foundation's Petting Zoo API [20]. Our modified version of this environment is made up of three agents and three landmarks. Each agent is randomly assigned a landmark as their goal. Agents do not know their own goal, but they do know the assigned goals of their teammates. An agent's individual reward is calculated as the negative square of its distance from its own goal. Agents are also rewarded globally based on the average squared distance of all three agents to their goals. We modified the environment to allow for three agents instead of two, and to allow for agents to give each other commands rather than an arbitrarily learned vocabulary of nine words as in the original environment. Each agent observes its velocity, its relative position with respect to each landmark as well as to each other agent, and the commands received from the other agents. We also used only the continuous action version of the environment. Each agent is able to send a message in this environment at every time step and that message is acted on or ignored in the next time step. We wanted to use this environment as a more realistic scenario where agents are mostly competent, rewards are stochastic and dependant on group performance, and any agent can communicate with any other agent on a given step.

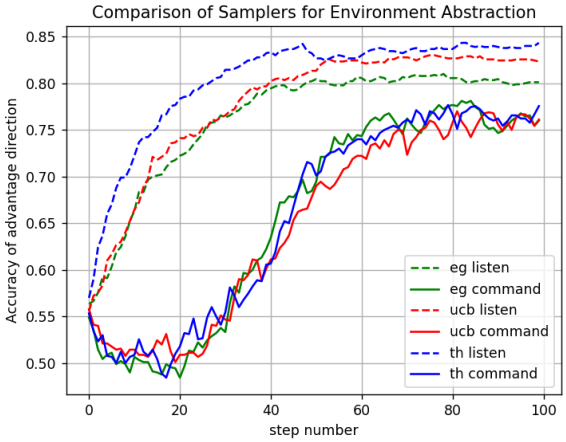

Figure 1: The accuracy of each sampler over time about whether each possible command relationship is advantageous or not in the Generic Communication Environment.

4 RESULTS

We now present experimental results with MATCH on each of the environments introduced in the previous section. When not specified, the value functions used for each experiment's advantage calculations were approximated using a two layer Multi-Layer Perception (MLP) Neural Network with ReLU [1] activation functions and layer sizes [64,64]. The networks were trained using Bellman squared error [3] between the value network and the recorded discounted rewards for each episode after 100 episodes of play for each policy.

4.1 Generic Communication Environment

We ran 100 experiments with MATCH in this domain and recorded the number of times each agent was correct about the sign of the advantage for it's given communication options (results are presented in Figure 1 where the vertical axis is the percent of the time that the sign of the advantage for listening to each other agent was correct at each time step). The three dashed lines represent the listening MAB samplers while the solid lines represent the commanding MAB samplers. During the early time steps, the listening

L \S	P1	P2	P3	P4
P1	0.97, **4%**	79, **66%**	201, **90%**	262, **97%**
P2	3.9, **5%**	0.26, **0.1%**	-18, **-5%**	6.4, **1%**
P3	106, **64%**	12, **3%**	7.1, **1%**	1.2, **0.2%**
P4	115, **60%**	-16, **-3%**	-1.6, **-0.3%**	-4.6, **-1%**

Table 1: Cart Pole Listener: Each cell represents both the absolute and percent difference when the agents used our algorithm to listen to another agent compared to listening randomly.

samplers have very few examples and so they listen close to randomly. During this time, the outgoing MAB samplers are unable to differentiate between the different agents based on how often they listen. This graph highlights a common behavior with our model in that without prior information the outgoing MAB samplers will lag behind the incoming MAB samplers before the latter start to stabilize. E-Greedy, UCB, and Thompson sampling perform comparably in our environments but Thompson sampling performs the best in most cases. Therefore, we report the results from using MATCH with Thompson sampling for the remaining environments.

4.2 Cart Pole Listener

We ran the cart pole problem for 100 episodes where the listener's experience with it's speaker is reset before each episode to measure zero-shot performance. We compared our algorithms performance with a listener that chooses to follow or ignore commands with a probability of 0.5. For this experiment, we used four policies with mean scores of 22, 203, 484, and 498 respectively when playing with no communication. We ran every combination of policies for a total of 16 experiments where no learning is done when a policy is paired with itself as both speaker and listener. The average score when running all four policies without no communication is 301. With random mixing, the score across all combinations of policies increases 355, and with our learning algorithm it increases to 447. We present, in Table 1, the actual and percent differences in average score between our algorithm and that with random policy mixing. For cases where policies choose the same actions often, such as with policies three and four, our algorithm is not able to differentiate policy quality well within a single episode, and hence performs comparable to randomly listening. When agent strategy's are varied, e.g., with player 1 and 3, our algorithm significantly improves performance.

4.3 MARL Grid World

For the MARL Grid World environment, we used 4 policies varying in quality from random to optimal: choosing optimal actions 0%, 30%, 80%, and 100% of the time and choosing randomly otherwise. Each agent gives a command to an agent of it's choice. If an agent is given multiple commands, it must choose only one to follow. For this experiment, we ran 5000 trials with no communication, random communication, and communication learned using MATCH. Results from these experiments are presented in Figure 2. The mean reward for the team as a whole was -9.3 for no communication, -7.7 for random communication, and -4.3 for learned communication.

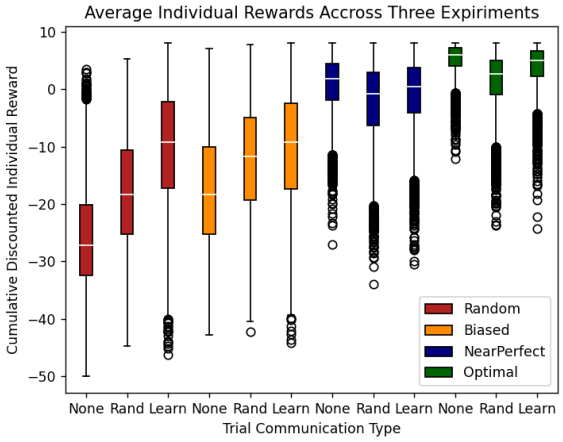

Figure 2: Each color represents a single policy's scores over the course of the three scenarios: no communication, random communication, and MATCH in the MARL Grid World.

Scenario \Noise σ	0.1	0.4	0.8
Random Communication	-15.47	15.51	-15.91
r_σ	5.79	5.73	5.71
On Policy (no MATCH)	-14.28	-14.35	-14.31
r_σ	5.28	5.30	5.00
Few Shot Avg (5 episodes)	-13.86	-14.00	-14.07
r_σ	5.28	5.30	5.00
Few Shot Final episode	-13.12	-13.24	-12.89
r_σ	5.28	5.30	5.00

Table 2: Reference Env: Average and standard deviation (r_σ) of policy returns in the Modified Reference environment with varying noise levels.

4.4 Modified Reference Environment

We present two experiments on the modified reference environment. For each experiment, we ran 500 trials for each of the four groups of policies. The first group communicates randomly, and the second group communicates with a hand programmed algorithm with a hand chosen probability to communicate. The third group is the average performance over five episodes where experience gathered for MATCH is maintained between episodes to measure few shot performance. The final group shows average performance on the last episode of each few shot learning trial. In the first experiment Gaussian noise is added to agent relative location vectors to simulate sensor inaccuracy. Typical relative location vector magnitudes are below five. We varied noise levels from a standard deviation of 0.1 to 0.8. We also tested a case where one of the agents is malicious and gives commands to guide itself and it's teammates away from the landmarks to evaluate how MATCH can add robustness to MARL algorithms against malicious attackers.

In Tables 2 and 3, each row refers to a group of policies run for 500 trials. In the first row, communication is not learned and decisions are drawn from the uniform distribution. The second row uses

Scenario	Good	Good	Malicious
Random Communication	-21.20	-21.26	-32.43
r_σ	7.07	7.00	11.84
On Policy (no MATCH)	-20.46	-21.01	-27.18
r_σ	7.63	7.78	10.67
Few Shot Avg (5 episodes)	-21.47	-21.12	-27.28
r_σ	8.05	7.94	11.61
Few Shot Final episode	-21.44	-20.63	-24.44
r_σ	8.92	7.85	10.36

Table 3: Reference Env: Average and standard deviation (r_σ) of policy returns in the Modified Reference environment where one agent is a bad actor that moves and commands in the opposite direction of the others.

our hand programmed policy which is designed to communicate efficiently with other competent agents while the third row is the average over five episodes running MATCH in a few-shot scenario and the fourth row is the average score of the last episode of the few show learning scenario.

5 DISCUSSION

In this section we discuss some key properties of MATCH as well as analyze certain aspects of its performance on the benchmark environments.

5.1 Sample Efficiency and Data Requirements

An important property of MATCH is sample efficiency. In the Generic Communication environment, at each time step, one command is sent. This results in the update of a single outgoing MAB parameter and a single incoming MAB parameter. There are a total of $|\mathcal{A}|^2$ outgoing parameters and a total of $|\mathcal{A}|^2$ incoming parameters as each agent needs to learn two parameters for each other agent. This means that in the single command scenario, a given arm of a given bandit sampler is expected to be pulled $\frac{1}{|\mathcal{A}|^2}$ times per time step, or for the parameters used in this particular environment for our experiments, $11.\bar{1}$ times over the course of an experiment comprising of 100 time steps.

We may expect to improve on this by allowing each agent to give a command at every time step such as in the MPE environment. Listening agents then must use a masked MAB sampler where they may choose only from the arms on which they have received commands and the arm that represents their own policy. While it might seem like this would increase the efficiency up to $\frac{2|\mathcal{A}|}{2|\mathcal{A}|^2} = \frac{1}{|\mathcal{A}|}$ expected samples per arm per time step, this is not the case. The actual sample efficiency gained is less than that, and it depends on a number of factors. Consider the case where all agents sent a command to agent a_0. The agent, a_0, will choose one command to listen to and ignore the rest. This will update one outgoing parameter for each commander, but only a single arm on the listening agent will be pulled so the expected number of pulls per arm in this case is $\frac{1|\mathcal{A}|+1}{2|\mathcal{A}|^2}$. Efficiency is lost when a single agent receives multiple commands.

The probability of multiple speakers sending commands to the same listener depends on both the learned outgoing probability distribution for selecting listeners and also potentially a context based prior available with a pre-trained agent. Outgoing samplers get an average of $\frac{1}{|\mathcal{A}|}$ samples per time step, but incoming sample efficiency depends on the probability of conflicts. The expected number of samples per listening arm per time step, $\mathbb{E}[n_s]$, based on the probability of conflicts, is shown in Equation 2, where $c_i \in C$ and C is the set of all possible sets outgoing commands for a given time step with cardinality $|\mathcal{A}|^{|\mathcal{A}|}$. Here, $n_c(c_i)$ is the number of unique agents that receive at least one command given the set of commands c_i and $P(c_i|\theta_t)$ is the probability of this set of commands given our agents outgoing probability parameters at time t, θ_t.

$$\mathbb{E}[n_s] = \frac{\sum_{c_i \in C} n_c(c_i) * P(c_i|\theta_t)}{|\mathcal{A}|^2} \tag{2}$$

When the listening samplers use an uniform distribution, $P(c_i|\theta_t) = \frac{1}{|C|}$, $\forall c_i \in C$, the sample efficiency for $|\mathcal{A}| \in \{3, 4, 5, 6, 7, 8\}$ are respectively $[0.235, 0.170, 0.134, 0.111, 0.094, 0.082]$ samples per listening arm per time step. These estimates give a rough idea of the suitability of our algorithm for tasks with different episode lengths.

There are two more considerations with regards to sample efficiency. First, when a listener is commanded by a speaker and no one else, including itself, it may choose either to follow the speaker's command, or to follow it's own policy, but when a listener commands itself and has no other incoming commands, it will always follow it's own policy, i.e., execute the command it gave itself. If the agent's internal advantage calculation does not have mean zero, this biases agents towards themselves, so we treat this case as if the agent sent no command at all and we do not update the incoming or outgoing sampler because no choices have been made. In other words, if an agent does not receive a command from a teammate, it does not update it's listening MAB sampler. The second consideration is when more than one agent command a single listener. Only one command can be accepted. This makes a single agent more likely to command that listener in the future, and other agents become more likely to seek out other listeners. This behavior causes the number of conflicts to decrease over time as agents avoid targeting commands to the same listener as each other. We have found that in practice these two forces tend to balance out such that the above simplified sample efficiency list is a good rule of thumb.

5.2 MATCH deficiencies for Zero/Few-shot learning

There are a few scenarios in which our algorithm should not be used. First are environments where slight policy changes can cause greatly degraded performance. For example, if the MARL grid-world pits cause the agent to restart, a single poor step may cause a perfect agent to lose ten steps of progress while a single good step for a random agent may increase it's odds of completing the maze by only a fraction of a percent. In this environment, following bad commands is so much worse for a good agent than it is beneficial for a bad agent to follow good commands, that our algorithm may not learn a good command structure fast enough to outweigh the cost of exploration. In this case, no communication has an average

team score of -6.98 with random communication scoring -9.98 and our algorithm scoring -7.2 over 5,000 trials. Our algorithm still performs comparable to no communication, but it cannot overcome the inherent drawback of policy mixing in such an unforgiving environment within a single episode. Additionally, the sample efficiency calculated in Section 5.1 allows for the estimate of how many arm pulls are expected, but with poor enough value function approximation, the number of samples required becomes prohibitive for zero or few shot learning. This means that in highly stochastic environments, or environments in which learning a value function is impossible, our algorithm may not be suitable.

5.3 Modified Reference Environment

This environment is meant to be a more difficult and realistic test for MATCH. First, sample efficiency is paramount because the episode length is only 25 time steps. Using our estimations in Section 5.1, for twenty samples per listening arm, we would need $20/0.235 = 84$ time steps, or about 4 episodes. Knowing this, we will use few shot learning for this environment. Additionally, the reward at each time step is based on both global and local factors, so approximate value function accurately is challenging in this environment. Lastly, the hand-crafted policies for this environment perform well together, but asymmetric information of this environment means that an effective command structure can increase agent performance above that of what the best solo agent can achieve. This combination make for a challenging but hopeful use case for MATCH.

Our hand-crafted policy works in the following way. If no commands are received, agents move towards the center of mass of the three landmarks with a 50% probability, or the closest landmark with a 50% probability. Agents send a command to the furthest ally half the time, the closer ally one third of the time, and to themselves one sixth of the time. If there is one incoming command, it will be listened to with probability 3/4. If two commands are received, a random one of the two is followed with probability 6/7. These agents are mostly trusting with no preference between teammates and their default policy performs very well if teammates are competent/altruistic. The malicious teammate moved and sent commands in the opposite direction of what it would have done as a normal policy.

In first experiment where noise was added to agents, MATCH was able to make small but statistically significant improvements over the nearly optimal hand programmed policy in the case of few shot learning. Given the very limited time steps of 25 per episode, it is to be expected that multiple episodes are necessary to augment an already proficient policy. We set the prior weight to 5.0 and the experience weights to 0.1 and 2.0 for speaker and listener samplers respectively.

In the malicious agent case, we expected MATCH to do better initially, but we found that the malicious agent lead to unexpectedly negative rewards at each time step which made our advantage function return large negative values most often. Because the value function was not accurate given the drastic change in expected rewards, the efficiency of MATCH was negatively impacted. MATCH learns to command agents which show the most improvement, so both the hand programmed policy, and the policy augmented with MATCH improved the random agent's performance, but not their own. We were able to train MATCH on 50 episode scenario where each of the agents learned the correct communication relationships, but team performance only increased by about 5 points on average. The large sample size was able to make up for a poor value function, but fifty episodes is too many to be considered few-shot learning and it may be enough to retrain a potential base policy instead.

5.4 MARL Grid World and Cart Pole

In both of these environments, MATCH was able to learn a command hierarchy that preserved the best agent's performance while significantly improving the weaker agents performances. The greater episode lengths of around 300 for Cart Pole and up to 100 for the Gird World allowed our method enough time to learn effective structure in single episodes a majority of the time. The learnability of a reliable value function in both of these environments allowed for MATCH to run very sample efficiently. In the Grid World environment in particular, MATCH was ably to raise the performance of agents one and two greatly while degrading the performance of top agents only marginally as seen when comparing the learned bars for each color to their random counterparts in figure 2.

6 CONCLUSION AND FUTURE WORK

In this work, we proposed a modular algorithm, MATCH, which can be added onto existing agents in order to allow them to communicate with one another through simple commands. The resulting emergent communication network features several desirable properties. The first is that advantageous pairwise relationships tend to be strengthened while undesirable ones are avoided. This structure causes team performance to improve as poor agents align their policies with their best teammates while those skilled teammates learn to stick to their own policy and we have shown this property across four environments of varying levels of complexity. Another desirable property is that our method can utilize pre-existing communication policies that can send commands as a prior distribution and further augment MATCH which combines context with old and recent experience without retraining the original agent and risking policy collapse. In addition, MATCH is completely decentralized. Other agents need not even implement MATCH, for a given MATCH-capable agent to learn incoming and outgoing relationships, so long as agents are capable of sending or receiving commands. Lastly MATCH is a very data efficient algorithm for small numbers of agents with only two tunable hyper-parameters to adapt to a wide variety of environments. MATCH is also computationally cheap. MATCH can be used with an existing agent policy and is designed for zero-shot learning, so it requires no training time besides the burn-in period at the start of an episode and it requires no large computation to sample a multi armed bandit model.

For all it's desirable properties, MATCH has a few weaknesses that we would like to address in future work. When every agent can send a message, MATCH currently ignores the cases where an agent commands only itself in order to avoid a bias towards self-directed commands. It may be the case that there is a way to leverage the instances where an agent desires to command itself rather than its teammates. Another way to improve sample efficiency might involve a delay to the speaker samplers to avoid the initial stagnant

state where speakers are waiting on listeners to stabilize. Perhaps the most impactful change of MATCH would be to weaken it's dependence on a reliable Value function by exploring longer time dependencies or finding another way to judge whether an incoming command has been advantageous to follow. We would also like to develop more concrete advice on tuning the prior experience and recent reward strength for different environments. In the four environments tested in this paper, we found that at the end of an episode, each sampler should have seen around twenty examples to have sufficient experience to improve performance. The prior and recent reward weights need to be tuned so that environments with large rewards do not lead to more greedy samplers than environments with small rewards. Our sample efficiency discussion may provide a ballpark estimate about how many steps in an environment MATCH requires, but a method to provide the proper scale of sampler θs would help MATCH's effective deployment.

Beyond simply mechanical improvements, we would also like to explore similar strategies in other kinds of communication. MATCH focuses on commands, but it would be beneficial if a similar method could augment more general pairwise relationships with decisions involving information sharing or arbitrary learned communication. Less direct communication may require a more complex judging of the value of communication than single step advantage approximations. A more general measure of communication value may open up the possibility of a MATCH-like architecture which takes more complicated priors to influence more than the selection of targets for communication. It also opens the possibility of processing more than one incoming command at a time, which will further increase sample efficiency. As the field of MARL expands, we hope to see a diverse set of methods developed to protect policies from collapsing under novel circumstances or in ad hoc coordination with new teammates. We also hope that methods like MATCH may be used to allow artificial agents to almost-always accept input from humans via simple adjustable prior beliefs for situations without blindly following bad actors.

REFERENCES

[1] Abien Fred Agarap. 2018. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375* (2018).

[2] Stefano V Albrecht and Peter Stone. 2018. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence* 258 (2018), 66–95.

[3] Leemon Baird. 1995. Residual Algorithms: Reinforcement Learning with Function Approximation. In *Machine Learning Proceedings 1995*, Armand Prieditis and Stuart Russell (Eds.). Morgan Kaufmann, San Francisco (CA), 30–37. https://doi.org/10.1016/B978-1-55860-377-6.50013-X

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. arXiv:arXiv:1606.01540

[5] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. 2016. Rl2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779* (2016).

[6] Angelos Filos, Clare Lyle, Yarin Gal, Sergey Levine, Natasha Jaques, and Gregory Farquhar. 2021. Psiphi-learning: Reinforcement learning with demonstrations using successor features and inverse temporal difference learning. In *International Conference on Machine Learning*. PMLR, 3305–3317.

[7] Ronald A Howard. 1960. Dynamic programming and markov processes. (1960).

[8] Hengyuan Hu, Adam Lerer, Alex Peysakhovich, and Jakob Foerster. 2020. "other-play" for zero-shot coordination. In *International Conference on Machine Learning*. PMLR, 4399–4410.

[9] Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Caglar Gulcehre, Pedro Ortega, DJ Strouse, Joel Z Leibo, and Nando De Freitas. 2019. Social influence as intrinsic motivation for multi-agent deep reinforcement learning. In *International conference on machine learning*. PMLR, 3040–3049.

[10] Volodymyr Kuleshov and Doina Precup. 2014. Algorithms for multi-armed bandit problems. *arXiv preprint arXiv:1402.6028* (2014).

[11] Bo Liu, Qiang Liu, Peter Stone, Animesh Garg, Yuke Zhu, and Anima Anandkumar. 2021. Coach-player multi-agent reinforcement learning for dynamic team composition. In *International Conference on Machine Learning*. PMLR, 6860–6870.

[12] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. 2017. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. *Neural Information Processing Systems (NIPS)* (2017).

[13] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems* 30 (2017).

[14] Igor Mordatch and Pieter Abbeel. 2017. Emergence of Grounded Compositional Language in Multi-Agent Populations. *arXiv preprint arXiv:1703.04908* (2017).

[15] Tai-Long Nguyen, Do-Van Nguyen, and Thanh-Ha Le. 2019. Reinforcement learning based navigation with semantic knowledge of indoor environments. In *2019 11th International Conference on Knowledge and Systems Engineering (KSE)*. IEEE, 1–7.

[16] Charles Riou and Junya Honda. 2020. Bandit algorithms based on thompson sampling for bounded reward distributions. In *Algorithmic Learning Theory*. PMLR, 777–826.

[17] Nicolas Schweighofer and Kenji Doya. 2003. Meta-learning in reinforcement learning. *Neural Networks* 16, 1 (2003), 5–9.

[18] Peter Stone, Gal Kaminka, Sarit Kraus, and Jeffrey Rosenschein. 2010. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 24. 1504–1509.

[19] Matthew E Taylor and Peter Stone. 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research* 10, 7 (2009).

[20] J Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S Santos, Clemens Dieffendahl, Caroline Horsch, Rodrigo Perez-Vicente, et al. 2021. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems* 34 (2021), 15032–15043.

[21] Justin K Terry, Nathaniel Grammel, Ananth Hari, Luis Santos, and Benjamin Black. 2020. Revisiting parameter sharing in multi-agent deep reinforcement learning. (2020).

[22] William R Thompson. 1936. On confidence ranges for the median and other expectation distributions for populations of unknown distribution form. *The Annals of Mathematical Statistics* 7, 3 (1936), 122–128.

[23] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. 2016. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763* (2016).

[24] Ke Xue, Yutong Wang, Lei Yuan, Cong Guan, Chao Qian, and Yang Yu. 2022. Heterogeneous multi-agent zero-shot coordination by coevolution. *arXiv preprint arXiv:2208.04957* (2022).

[25] Changxi Zhu, Mehdi Dastani, and Shihan Wang. 2022. A survey of multi-agent reinforcement learning with communication. *arXiv preprint arXiv:2203.08975* (2022).

[26] Zhuangdi Zhu, Kaixiang Lin, Anil K Jain, and Jiayu Zhou. 2023. Transfer learning in deep reinforcement learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2023).