Coordinated Multi-Agent Learning

Mohit Sharma
mohits1@andrew.cmu.edu

Arjun Sharma
arjuns2@andrew.cmu.edu

Abstract

Many real world tasks involve multiple agents acting synchronously in a cooperative setting. Communication policy between multiple agents in most environments is often fixed. In [3], the authors come up with a new DQN [6] architecture to learn a communication policy in addition to the environment policy. In this work we extend the above architecture to a novel grid world environment. We use the above environmental setup to find correlation between the agents' communication and environment policies. We show through extensive experimentation that architecture is able to learn effective communication policies while acting independently. Further, we analyze these communication policies to see if some patterns emerge.

Introduction

A Multi-Agent system is usually defined as a group of autonomous entities interacting with each other in some shared environment. Multi-agent systems find use in a wide array of tasks ranging from robotic teams, collaborative learning, competitive learning etc. In traditional Reinforcement Learning literature the agent learns by trial and error. At each step, the agent observes the environment and takes an action which in turn affects the environment. However, traditional RL algorithms cannot be directly applied to multiple agents. The foremost problem in their application, is the problem of non-stationarity which result from the dynamics of multiple agents acting in the same environment simultaneously. This breaks down the usual assumptions of traditional RL algorithms and hence convergence is no longer guaranteed [1]

Multiagent Reinforcement Learning (MARL) algorithms focus on three different kinds of agent strategies i.e. Fully Cooperative, Competitive and Mixed. In this work we focus on the Fully Cooperative strategy. For fully cooperative settings multiple agents usually share the same reward signal and use message passing to communicate between themselves for coordinated learning. These messages are usually fixed in traditional algorithms e.g. agents might share their locations with other agents. With the recent success of single agent RL there has been a renewed interest in MARL. In [3] the authors come up with different architectures to solve the problem of Cooperative MARL while implicitly learning optimal communication policies. The network architecture allows agents to send messages to each other in a fully differentiable setting allowing the whole network to be trained using gradient descent in an end to end manner. The authors experiment their method for the prisoner's dilemma and an MNIST game.

In this project, we try to see whether learning communication policies in MARL settings is tractable in more realistic environments. Taking inspiration from traditional MARL algorithms we use a Predator-Prey as our target environment. Within this target environment we design a predator prey game which intrinsically motivates coordination between multiple agents. Through extensive experimentation we show that our agent is able to learn effective communication policies, which allows it to complete the task, while acting independently, but sharing information with other agents. Further, we analyze the messages sent by the agents to infer if unhindered communication leads to the evolution of a grounded communication policy.

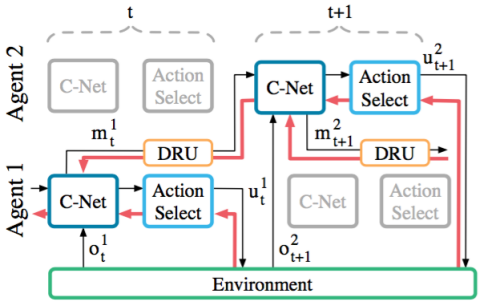


Figure 1: The *Differntial Inter-Agent Learning (DIAL)* architecture as shown in [3]. The red arrow shows the gradients as they are backpropagated in the network. Image taken from [3].

Related Work

Multi-agent systems have been used to solve problems in a variety of domains, including robotics, distributed control, economics, etc. Within this broad stream of work a lot of focus has been dedicated to Multi-Agent Reinforcement Learning (MARL) algorithms.

Cooperative Multi-Agent tasks involve agents acting in a shared environment. Tan [9] were among the first ones to show that sharing information between cooperative agents leads to much better performance than using independent Q-learning for each agent. The authors use a variety of ways to show the efficacy of communication between multiple interacting agents. For instance, agents can inform others of their current state by sharing sensory information. Another approach is to share a history of past (state, action, reward) tuples, assuming other agents have not yet encountered these states. However in all of these cases a communication policy exists and is not being explicitly learnt by the agents.

In other more traditional work agents are provided with a communication channel but its purpose is not hard coded. For instance, Wagner [10] use a single signal agent vocabulary which can be detected by other agents. Also, in [11] the authors use a fixed but undefined communication library, which allows the agents to associate meanings with words in various trials. The authors show that as environments change the agents learn to associate different meanings with the same words. Both of these are similar to our setting, however in our method we learn these communication policies together with the environment action in a DQN [6] setting.

Recently, there has been a resurgence of multi-agent Reinforcement Learning algorithms being applied to different domains, based on Deep Q-Network [6]. We take inspiration from Foerster et al. [3], where the authors come up with an end to end Differentiable Learning architecture (DIAL) to learn communication policies. We extend this architecture to a more common predator-prey domain. In [3] the authors use a continuous communication channel during training while a discrete channel during evaluation. We remove this dichotomy and instead use a continuous channel for communication policies.

Methods

Independent Q-Learning is an approach to solve MARL tasks in which each agent independently learns its own Q-function $Q^a(s, u^a, \theta_i^a)$ where s refers to the state, u to the action and a is the index of the agent for this Q-function. Although, theoretical convergence is not guaranteed in this setting empirically it has been shown to work well [8].

We extend Independent Q-Learning to a MARL setting taking inspiration from [3]. At each timestep t of the gameplay each agent outputs a message action along with the usual environment action. In [3] the authors send discrete messages but we extend the architecture to real valued messages. This message action is then sent to all the other agents as input at time $t + 1$. This makes the whole network differentiable with respect to time and we can use backpropagatation through time (BPTT) to

train the network end to end. In [3] the authors refer to this architecture as *Differentiable Inter-Agent Learning (DIAL)*. In [7] the authors use a Bidirectional RNN for a similar architecture and refer to it as *Bidirectionally-Coordinated Net (BiCNet)*.

Figure 1 shows the *DIAL* architecture for two agents as described in [3]. *DRU* in the image above refers to *Discretize/Regularize Unit*, which is used to discretize the generated message during execution phase. In our project we use continuous valued messages and hence we remove DRU from our network.

For our baseline we experiment with various settings. Firstly, we try independent Q-learning where each of the agent receives its own state and performs actions independently of all other agents. Thus each agent in this setting formulates a Q function as $Q^a(s^a, r, \gamma, u^a)$. This reduces the problem to a traditional RL setup but with none of the RL convergence guarantees holding. Secondly, we try sharing state between the independent Q-agents however the actions of each agent are still independent i.e. each agent only knows about its own action. This reduces the problem to find $Q^a(s, r, \gamma, u^a)$. Joint action learners in MARL architecture refer to case where each agent knows about the actions of all other agents. This reduces the Q-function formulation to $Q^i(s, r, \gamma, u^i_t, u^{i'}_{t-1}), i' = [a_0, a_1, ..a_{i-1}, a_{i+1}, ..]$. In our implementation, the Q-function is formulated as $Q^a(s^a, r, m^{a'}_{t-1}, u^a)$ where $m^{a'}_{t-1}$ is a list of messages from all other agents at $t-1$. Thus each agent get its own state observation from the environment along with the messages sent by other agents. We hypothesize that even without explicitly sharing state, the agents would be able to learn appropriate messages to send and infer their meaning for optimal joint actions.

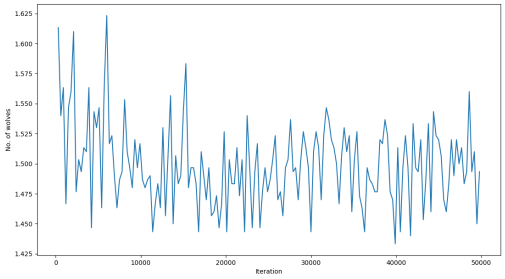
We experiment with our settings in the *Wolfpack* environment. *Wolfpack* is an implementation of the standard predator-prey environment often used in MARL. In [9] the authors show that using cooperative strategies in such environments leads to better rewards. However we modify the usual predator-prey setting by explicitly rewarding cooperative behavior. We thus implement the *Wolfpack* settings as described in [5]. Specifically, in this setting the cumulative reward for agents is much higher if both of the predators catch the prey simultaneously.

Results

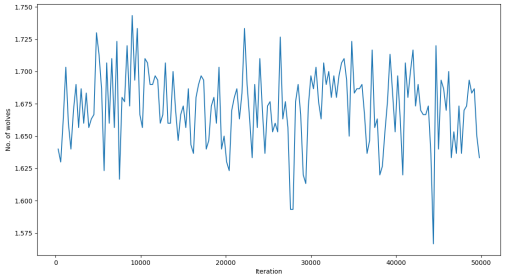
Environment: As discussed in the previous section, the environment is implemented based on the description in [5]. The game consists of two wolves trying to catch a prey. While a single wolf is capable of bringing down the prey, it is unable to save the carcass from the scavengers unless the second wolf is nearby. Therefore, the wolves receive a smaller reward r_{lone} when the second wolf is not within a capture radius of the prey when it is caught. A larger reward r_{team} is obtained if the two wolves are close by. We use a 11×11 grid with a capture radius of 3. We set r_{lone} to 1 and r_{team} to 100. Additionally, the wolves receive a negative reward of $-1/11$ at each time step. We have successfully implemented the environment and and have verified its correctness using the experiments described below.

Baseline: We run two sets of experiments to see whether our hypothesis that knowledge of the other wolf's state would be beneficial for the wolves holds. In one set of experiments, each wolf only has access to its own location and the location of the prey. This corresponds to the independent Q-learning setting discussed in the previous section. In the second set of experiments, each wolf also has access to the other wolf's location. This is the shared state setting. Intuitively, sharing of states should lead to a better performance since it would allow the wolf to capture the prey only when the second wolf is within the capture radius and hence collect more reward.

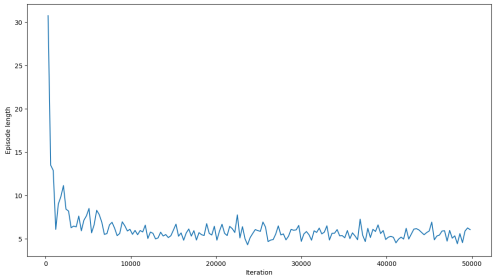
Our experiments show that this is indeed the case. Figure 2 shows the results for the independent Q-learning setting. The column on the left shows plots for an experiment where the prey was not allowed to move from its initial position (chosen randomly in each episode). The column on the right shows plots for the experiment when the prey moves around the grid by choosing random actions. The top row shows the number of wolves on average withing the capture radius when the prey was caught. The bottom row shows the episode length as training proceeds. For clarity, we plot these values after every 300 episodes and show the average value over the preceding 300 episodes. Figure 3 shows plots for the same settings of the experiments when the state is shared between the wolves. As can be seen from the plots, the average number of wolves per capture does not improve over



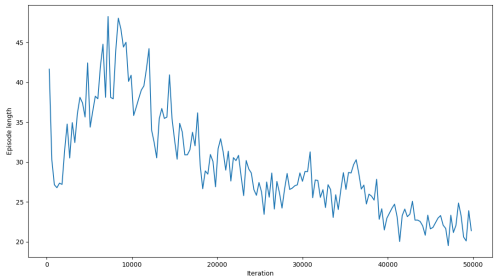
(a) No. of wolves per capture, stationary prey



(b) No. of wolves per capture, random prey

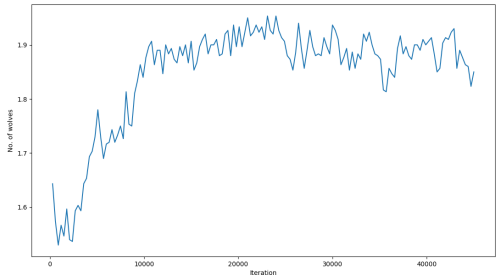


(c) Episode length, stationary prey

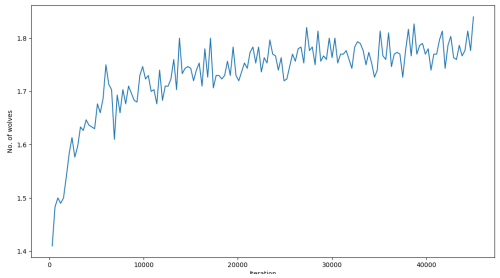


(d) Episode length, random prey

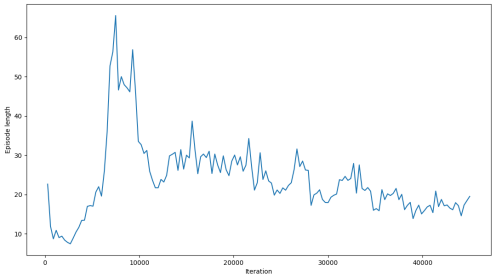
Figure 2: The average no. of wolves that capture the prey and episode length for stationary (left) and random (right) prey for the independent Q-learning scenario. The average is computed over the preceding 300 episodes.



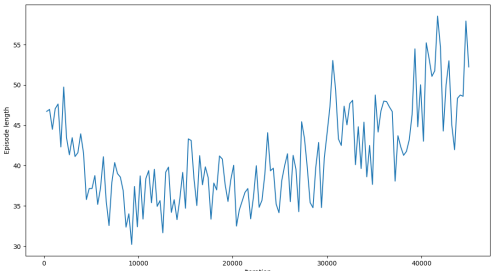
(a) No. of wolves per capture, stationary prey



(b) No. of wolves per capture, random prey

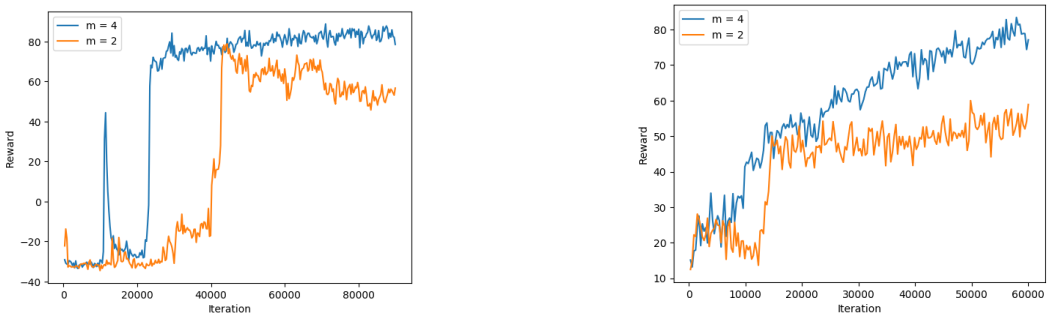


(c) Episode length, stationary prey



(d) Episode length, random prey

Figure 3: The average no. of wolves that capture the prey and episode length for stationary (left) and random (right) prey for the state sharing scenario. The average is computed over the preceding 300 episodes.



(a) Static prey

(b) Random prey

Figure 4: Comparison of performance using message length m = 2 and m = 4

episodes in case of independent Q-learning. On the other hand, when the state is shared, the wolves gradually learn to catch the prey only when the other wolf is within the capture radius. This can be seen from the increasing trend of the number of wolves plots. Also, the episode length plots show that the wolves take longer time to capture the prey in a shared setting, as they wait for the other wolf to come close to the prey. The performance plateaus after about 20000 episodes while it failed to improve even after 50000 episodes in the independent Q-learning setting.

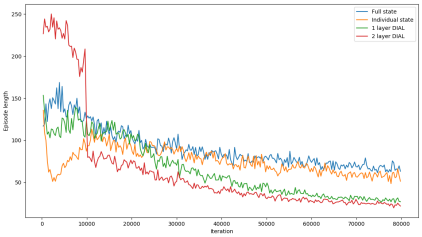
These results clearly agree with our hypothesis and indicate the correctness of our implementation of the environment.

DIAL: Having confirmed our hypothesis through the baseline experiments, we now compare the baseline results to those obtained using the DIAL architecture. DIAL was implemented as a recurrent network. The architecture consists of n layers of Gated Recurrent Units (GRU) [2] which are shared for the generation of the action and the message. The hidden layer activations of the topmost GRU layer are then used as an input to two separate linear layers. One linear layer has $|A|$ units where $|A|$ denotes the number of possible actions. This layer learns to predict the Q-value for each action given the current state. The second linear layer has m units to output a message $\in \mathbb{R}^m$. Unlike [3], we do not discretize the message as test time. The input to this network consists of the state of the prey and wolf at time t and the message generated at time $t-1$ by the other wolf. The input size is thus $4+m$. We set the hidden layer size to 16. The action space in *Wolfpack* consists of 5 possible actions - up, down, left, right and no movement. The network was trained by minimizing the one step TD-error. We used Adam [4] for optimization with an initial learning rate of 0.0001. The network was trained for 50000 episodes. However, early in the training process, the wolves often fail to capture the prey for a long time when using the DIAL policy and hence, progress is slow. To overcome this, we set a maximum episode length of 400 for the 11×11 grid version of *Wolfpack* for which we discuss the results below. We found it to be necessary to increase this length in accordance with the grid size to ensure that the agent gets sufficient time to capture the prey.

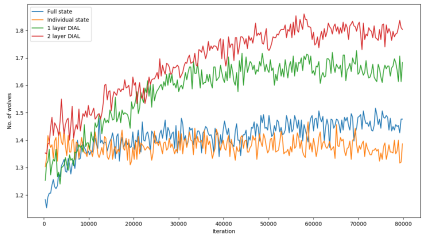
Figure 5 compares the performance of DIAL with the baselines. We tried two settings of the number of hidden layers $n = 1, 2$. As can be seen from the figures, DIAL achieves the maximum reward and at the same time has the lowest episode length and the maximum average wolves in the capture radius. Amongst the two DIAL models, the 2-layer architecture performs better than the single layer architecture. DIAL converged after about 60000 iterations which is approximately the same as the tabular Q-learning baselines.

Figure 6 shows the comparison between the baseline methods and DIAL on the static prey setting. This setting offers an interesting insight. As can be seen from the figures, while the full state tabular Q-learning method achieves a higher average number of wolves within the capture radius, it takes a much longer to do so. This can be seen in the figure comparing the episode lengths of the various methods. On the other hand, DIAL achieves an almost comparable average number of wolves per capture but does so by taking considerably fewer steps. This leads to the 2-layer model achieving a higher average reward. This shows that the policy learnt using DIAL leads to the prey being captured by a single wolf if it believes that the second wolf is too far away.

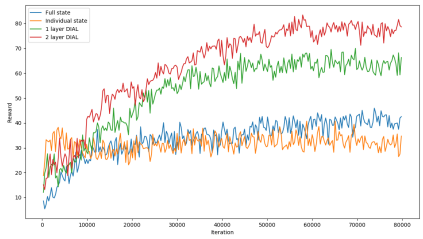
Effect of message length: Next, we analyze the effect of message length on the performance. We compared the performance of a DIAL model with message length $m = 2$ with the results shown previously (where $m = 4$). Figure 4 show the results of this comparison. The performance when



(a) Episode length, random prey

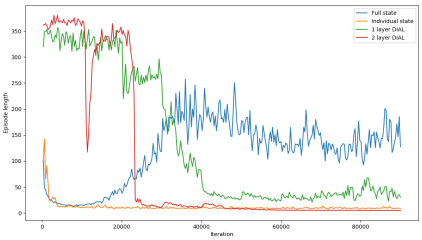


(b) No. of wolves per capture, random prey

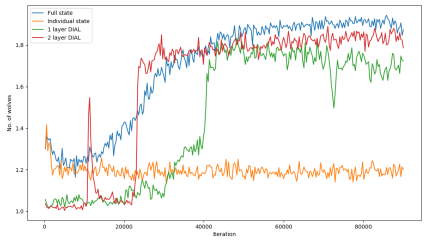


(c) Reward, random prey

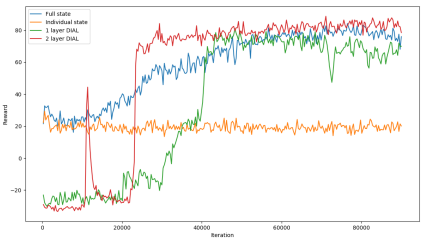
Figure 5: The average episode length, no. of wolves and reward for random prey using DIAL. The average is computed over the preceding 300 episodes.



(a) Episode length, stationary prey



(b) No. of wolves per capture, stationary prey



(c) Reward, stationary prey

Figure 6: The average episode length, no. of wolves and reward for stationary prey using DIAL. The average is computed over the preceding 300 episodes.

Table 1: Correlation between generated messages and agent state and action

(a) Correlation for Random prey

		Message	
		m_0	m_1
State	s_0	0.40	-0.11
	s_1	0.36	-0.05
Action		0.14	0.51

(b) Correlation for Static prey

		Message	
		m_0	m_1
State	s_0	0.28	0.12
	s_1	0.14	0.16
Action		0.14	0.46

using $m = 2$ was lower compared to $m = 4$ in both random and static prey settings. In the static prey setting, the network performance degraded after some time, indicating divergence. This indicates that choosing an appropriate message length is crucial for performance.

Analyzing the message: Finally, we try to analyze the content of the message. Intuitively, one would expect the message to contain information about the current location and possibly the action at the current time step that the wolf intends to take help the other wolf decide its trajectory towards the prey. We try to see if the messages communicated between the agents have any correlation with their current state and chosen action by computing the Pearson correlation coefficient between the message and the agent state and action. We used the model with $m = 2$ for this experiment. Table 1 shows the results for this experiment. While we saw a moderate correlation between the message and the action the wolf takes at that time step, we did not find a strong correlation between the state and the message. Figures 7 and 8 show how the messages are distributed in 2-d. We color code the message according to the state and action of the agent at the time step the message was generated. The figure clearly shows how messages are well correlated with the actions but barely so with the state of the agent.

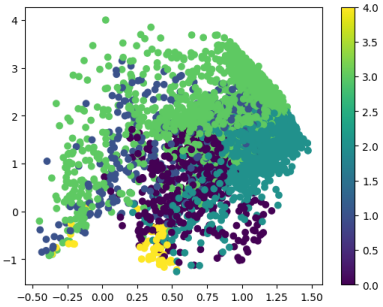
Discussion and Conclusion

In this project, we looked at Multi-agent Reinforcement Learning. We performed a literature review of the existing methods in this field and focused on one particular type of method where agents learn to communicate with each other.

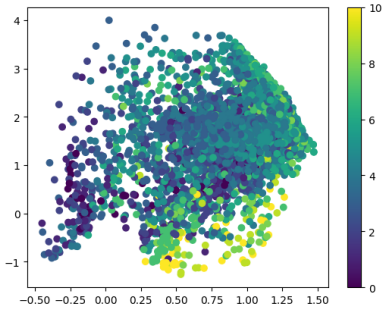
In particular, we looked at an existing method called *Differentiable Inter-Agent Learning (DIAL)* and applied it to a different environment which we believe allows for more interpretable analysis of the technique and the messages it generates. The environment is the classic predator-prey game of *Wolfpack* where two wolves collaborate with each other to catch a prey. To this end, we implemented the environment from scratch. As a baseline and to confirm our hypothesis that sharing of information between agents should help them gain higher rewards, we implemented tabular Q-learning in two scenarios - full state sharing and individual state. As expected, having access to the state of the other agent allows learning of much better policies. Next we implemented recurrent architecture in [3] where rather than allowing the agents to share states, the agents are allowed to share messages with each other. These messages however are not hand-coded like much of the prior work in this field and must be learnt. We compared various design choices in this architecture such as the number of hidden layers of recurrent units and message size. We showed that choosing an appropriate message length is crucial for performance and that deeper architectures outperform shallower ones. Moreover, we analysed the messages generated by the agents and studied their correlation with information such as the agents current state and action which intuitively should help the agents achieve higher reward.

Future work

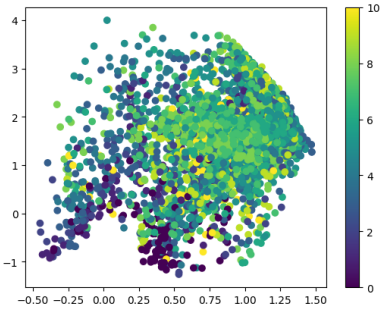
In future, we aim to work on learning to communicate in an inverse reinforcement learning setting. We plan to look into the problem of learning a communication strategy between collaborating agents when given state-action demonstrations of agents working together to solve some task. We aim to use the *Wolfpack* setting we created in this project and learn the expert policy by using value iteration in the full state setting. Given trajectories from this expert trajectory we aim to simultaneously learn the reward function and the communication mechanism the agents must employ when the policy they learn no longer has access to the state of the other agent.



(a) Messages colored by wolf action

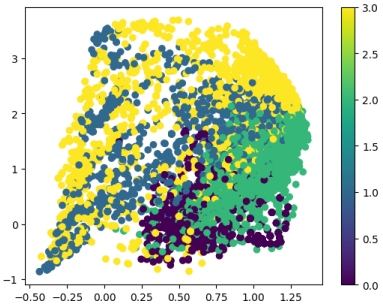


(b) Messages colored by x-coordinate of wolf state

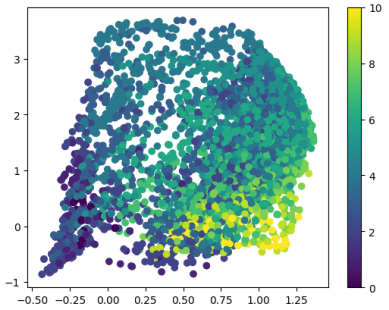


(c) Messages colored by y-coordinate of wolf state

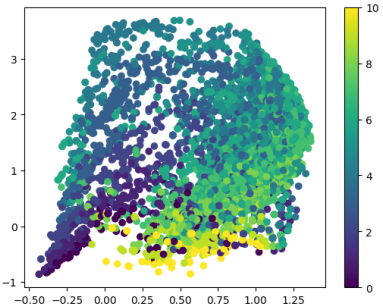
Figure 7: 2-d plot of messages for the random prey setting. Colors represent the action or state of the agent at the same time step.



(a) Messages colored by wolf action



(b) Messages colored by x-coordinate of wolf state



(c) Messages colored by y-coordinate of wolf state

Figure 8: 2-d plot of messages for the static prey setting. Colors represent the action or state of the agent at the same time step.

References

[1] 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):156, 2008.

[2] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

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

[4] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

[6] 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.

[7] Peng Peng, Quan Yuan, Ying Wen, Yaodong Yang, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent bidirectionally-coordinated nets for learning to play starcraft combat games. *arXiv preprint arXiv:1703.10069*, 2017.

[8] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017.

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

[10] Kyle Wagner. Cooperative strategies and the evolution of communication. *Artificial Life*, 6(2): 149–179, 2000.

[11] Holly Yanco and Lynn Andrea Stein. An adaptive communication protocol for cooperating mobile robots. In *Meyer, JA, HL Roitblat, and S. Wilson (1993) From Animals to Animats 2. Proceedings of the Second International Conference on Simulation of Adaptive Behavior. The MIT Press, Cambridge Ma*, pages 478–485, 1993.