MARL with Communication

Ashish Mittal, Nilay Pande, Adwait Godbole, Sumit Chaturvedi

Abstract. We explore various methods of facilitating communication between learning agents. Our aim is to see interesting emergent behaviour when multiple agents cooperate to fulfill some goal. Many animals, from spiders to lions, hunt in packs. In our predator-prey environment, we hope to recreate this behaviour.

Contents

1 Introduction

Multi-Agent Reinforcement Learning models reality pretty accurately. Most of us aren't lone wolves. We live in communities. Our basic needs of food and water are a result of symbiotic interactions between millions of people. In order to make AI mimic this phenomena, they should be taught to communicate.

Leaving needless hyperbole aside, practically, learning agents do benefit from communication. In [1], it is shown that n agents, who share everything can reduce the training time by a factor of $\Omega(n)$.

All experiments in this report are adapted from [2].

1.1 Setting

We chose the predator-prey environment for all our experiments. We consider a Grid World environment for our tasks. In Figure 2, the blue squares represent the predators and green squares represent the prey. The red square represents that the predator has finally tasted blood. The light blue are surrounding the predator is the *perceptual window*. The predator can sense the position of the the prey only if it ventures into this area. In the example given, we have 5 predators and 2 prey. A predator has just managed to attack the prey (marked by the red square).

The predators are the learning agents. In the initial variants the prey simply does a random walk. Later we treat a variant with 'intelligent' prey. There is variation in the allowed actions and allowed states across experiments. We'll explain them when we look at each experiment in turn.

Figure 1: Expectation

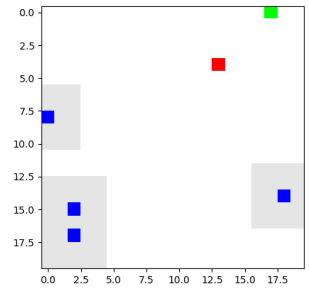

Figure 2: Reality

1.2 Algorithms

We briefly detail the algorithms used. There are only minor modifications made from what we did in class.

1.2.1 Q-Learning

In the Q-Learning algorithm, the only change we make is how to select action from the given action-value function and state.

```
# Initialize action-value function
for all non_terminal(s) :
    for all actions :
        Q(s, a) = random.rand()
Q(terminal_s, _) = 0
for each episode :
    s = random.state()
    for each step of episode :
        # We only change this.
        a = selectAction(Q, s)
        r, s_ = env.next(s, a)
        Q(s, a) += alpha * [r + gamma * argmax(Q(s_)) -
            Q(s, a)]
        s = s_
```

The action is selected from the probability distribution obtained by doing softmax over $Q(s)$.

$$p(a_i|s) = \frac{e^{Q(s,a_i}/T}{\sum_a e^{Q(s,a)/T}} \tag{1}$$

1.2.2 REINFORCE

Again, the algorithm is taken from Sutton and Barto.

```
episodeCount = 0
while not stop(episodeCount) :
    st = []
    at = []
    rt = []
    s = initialise()
    while not episode.terminate() :
        # pi is represented by a two-layer neural network
        # theta are the weights of the network.
        a = selectAction(pi(s, theta))
        s_, r = env.next(s, a)
        st.append(s)
        at.append(a)
        rt.append(r)
        s = s_

    for each step of episode :
        G_t = return(t, st, at, rt)
        # We take the gradient of ln(pi(s, a))
        # while updating theta.
        grad = ln(pi).gradient(a)
        theta = theta + alpha * G * grad
```

In our case, the policy is parametrized by a two-layer neural network. The input is a state vector and the output is a probability distribution over the available actions.

2 Experiments

In Table 1, you can see the action-value function for stationary prey and predator. This is done just as a sanity check to ensure that our implementation is correct.

←	←	↑	→	→
←	←	↑	→	→
←	←	X	→	→
←	←	↓	→	→
←	←	↓	→	→

Table 1: *Certificate:* $argmaxQ(s, a)$ for each s in perceptual window when prey is not allowed to move. 1 predator, 10 prey.

2.1 Independent Agents

This is our baseline experiment. Each agent has its own separate action-value function. No information is shared between any agent. Note that to catch the prey, the predator only needs to know the relative position to the prey. It doesn't need to know it's own or the prey's absolute position on the board. To reflect this, each agent has a perceptual window. Think of it as the predator's line of sight. In Figure 2, the grey boxes represent these windows. The predator can be in two types of states.

1. *Prey in perceptual window*: A tuple (Δ_x, Δ_y) representing the relative position of the prey with respect to the predator. If there are more than one prey in the window, the closest one is chosen.

2. *Otherwise* : A state unique to this predator. Each predator has a unique id which is used to represent this state. The idea is that if a predator cannot see the prey, due to absence of any other information, it is best to do a random walk.

The allowed actions are move left, move right, move up, move down and stay. You might wonder why the predator will ever stay (i.e. prowl) in its own position and reap negative rewards? Disallowing the agent to stay results in situation where even if the prey is right next to the predator, there is no way that they'll move to the same spot in one step. In this situation, the Manhattan-distance is 1. Finally, the rewards given to the agent are as follows:

1. *Prey Capture:* A reward of $+1$ is given and the episode is terminated.

2. *Otherwise:* A negative reward of -0.1 is given.

In Figure 3 notice that in the x-range $[175, 200]$ the plot of cumulative time steps has become linear. This means that the agents have learnt an optimal policy. As a result, the number of time-steps per episode has become a constant.

In Table 2, the action-value function behaves somewhat strangely. For example, the action-value function advises the agent to move upwards when the agent is just beneath it. Also, if the prey is in the top-left corner, the advise is to move down. Other than these few oddities the suggested actions seem right.

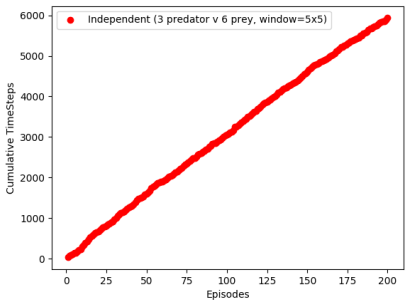

Figure 3: Time-steps averaged over 10 runs with different seeds

↓	↑	↑	→	↑
←	↑	→	↑	→
←	←	X	↑	·
·	↓	↑	→	→
↓	←	↓	↓	↓

Table 2: Since the prey is moving randomly, this table isn't as consistent as the one above. The dot means stay.

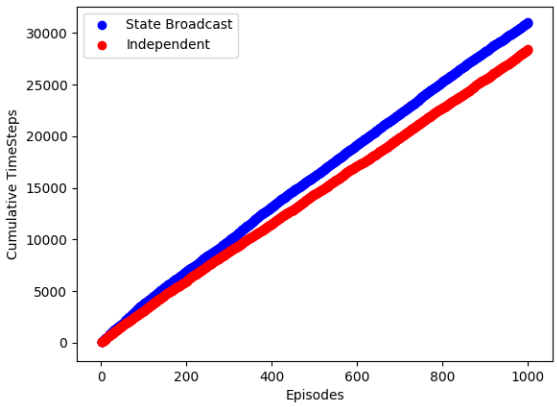

Figure 5: Comparison of Expt. 1 and Expt. 2

2.2 State Broadcast

In this experiment, at the start of each step, each predator shares its state with the rest. Each predator knows where the others are. Consider two predators, Mufasa and Simba, and a prey, Zazu. Mufasa can see Zazu (i.e. Zazu is in his perceptual window) but Simba can't. But because Simba knows where Mufasa is and Mufasa knows where Zazu is, Simba can calculate where Zazu is using the information Mufasa gives him. Figure 4 offers you a visual picture.

Note, that with sharing of state, a predator, potentially, has a view of the entire board. Since the board size is 20 by 20, the predator has an actual perceptual window of 39 by 39 (it can see 19 grid cells away on either side as well as up and down). The state space, however, has exploded tremendously.

With the same number of prey and predators as before, we launched this experiment. You would hope that with wider visibility the number of time-steps to finish a number of episodes would decrease. However, this doesn't happen (Figure 3).

To debug this, we looked at the action-value function at the end of training. We examined the predicted action when both Δ_x and Δ_y are negative. Ideally, in most such states, the chosen action should be left or down. The reality was much worse. Only around 40% of the actions were one of left or down. It is as if the actions were chosen at random.

We hypothesize that this is because it is very unlikely that a predator will act upon the shared message and be successful in catching the prey. In other words, it is likely that the closer predator will snatch the prey. As a result, the predator who received the message, won't see any value in pursuing the prey.

Although this experiment doesn't seem immediately useful, we'll come back to it when we consider joint termination. Over there, more that one predator will be required to capture the prey.

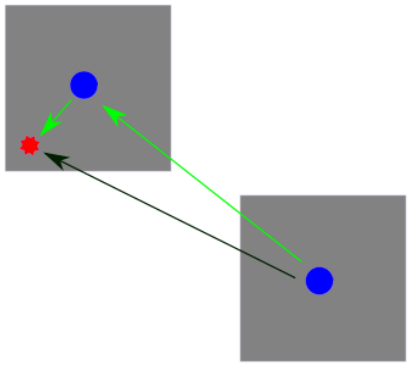

Figure 4: Vector addition

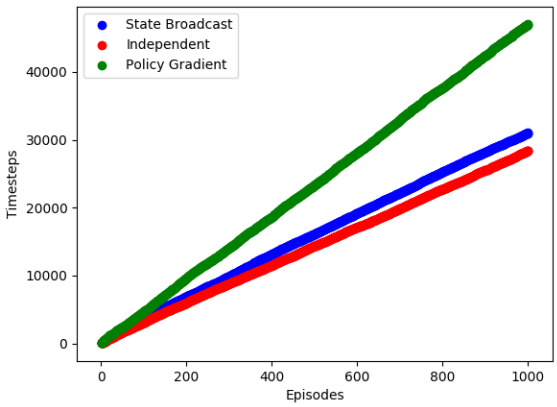

Figure 7: Comparison of the three experiments

2.3 Policy Gradient

We thought that it perhaps the results in [2] could be improved by an application of Reinforcement Learning. More concretely, as a first step, we implemented a policy gradient method ([4], [5]). Each predator had its own policy network.

In Figure 6, the figure represents the policy network architecture. The inputs to the network are Δ_x, Δ_y and an indicator variable Un. In case there is no prey in the perceptual window, this variable is set to 1, else it is set to 0. Hopefully, the network learns to ignore Δ_x and Δ_y when $Un = 1$.

The output of the network is a probability mass function over the 5 available actions.

In Table 3, we show the action which had the highest probability at a given state. If you compare it with Table 2, you'll find that there are a lot more actions which are indefensible. For example, in the 4^{th} row, for both the 3^{rd} and 4^{th} column, the prescribed action is up. This seems pretty nonsensical!

The plot in Figure 7 backs this up. The high slope indicates that more steps are wasted in each episode. Although this is a sad state of affairs, having this architecture in place means that we can tackle much larger state spaces. Later, when we consider Joint Termination, the state will become a four tuple. Having a action-value table is impractical in such a setting.

We feel that policy gradient doesn't work well here for two reasons.

1. *Skewness in stationary state distribution:* The predator spends way more time in its unique state (when there is no prey in the perceptual window) than in any other state. Due to this, the network's parameters are disproportionately biased towards fitting this state. One solution can be that the policy network comes into play only if the prey is in the perceptual window. That is, if there is no prey visible, the predator will take a random move. In the presence of prey, it will move as prescribed by the policy network.

2. *Insufficient Cross-validation:* Honestly we didn't spend near enough time tuning the network hyper-parameters. We only experimented with the learning rate. Maybe by having a deeper or wider architecture, we can get better results.

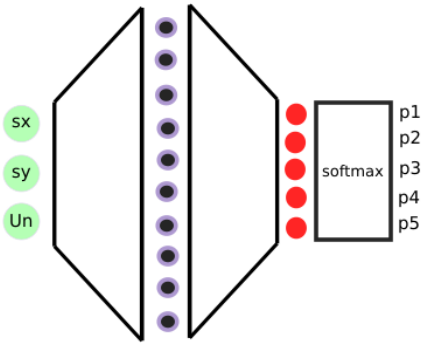

Figure 6: Policy Network Architecture

↑	↑	→	→	↑
←	←	→	↑	←
←	↑	X	↑	→
←	→	↑	↑	←
→	↑	→	→	·

Table 3: $a = \text{argmax } model(s)$ that network assigns for different states.

2.4 Joint Terminate

In this setting, more than one predator is needed to catch the prey. Specifically there have to be two predators such that for some prey:

$$||\text{predator}_1 - \text{prey}||_1 \leq 1 \wedge ||\text{predator}_2 - \text{prey}||_1 \leq 1 \quad (2)$$

Intuitively, communication is necessary to achieve such coordinated action. To keep things simple, we keep 2 predators are 1 prey in our environment. A perceptual window of 4 is employed.

The predators are only given a positive reward $(+1)$ at episode termination. Otherwise, for each wasted step, the predators are penalized by a negative reward of -0.1.

We tried four things in this broad category. *Independent*, *Aware* and *State Sharing + Aware* are reiterations of [2]. The *State Sharing* experiment is our own.

2.4.1 Independent

This is essentially Experiment 1 with a different terminating condition. The states are exactly identical as earlier and each predator learns its own action-value function.

2.4.2 Aware

Here we make modification to the state of the predator. Each predator is aware of the other. Thus, the state of a predator is a 4-tuple :

$$(\Delta_x^{\text{prey}}, \Delta_y^{\text{prey}}, \Delta_x^{\text{partner}}, \Delta_y^{\text{partner}}) \quad (3)$$

The first two entries represent the relative position of the prey (if it is in the perceptual window). The second two represent the relative position of the partner predator.

2.4.3 State Sharing + Aware

In [2], Tan combines the idea presented in Experiment 2 with the idea above. Technically, this is not necessary in our implementation. This is because, implicitly, each predator knows where the other is.

2.4.4 State Sharing

Based on the above observation, we do not consider the *Aware* part of the state. The *Aware* part only causes a state explosion and doesn't increase the predator's knowledge.

2.4.5 Analysis

In Figure 8 we not the following.

Experiment *Shared* is clearly the best. It finishes episodes in the least number of time steps. Moreover, it is able to do so in the least number of episodes. It is the fastest and best learner.

Aware and *Shared + Aware* perform similarly. They are slow learners but are able to eventually beat *Independent*. Some general observations are follows:

- In the case of 1 prey and 2 predators, we observe that only Sharing is good. This is advantageous primarily due to the small state space which leads to fast training. Even then we observed that the performance was at par with and in some cases even better than Aware + State Sharing and Aware.

Expt	Avg. timesteps / episode
Independent	68
Shared	22
Aware	23
Shared + Aware	22

Table 4: 2 predators vs 1 prey

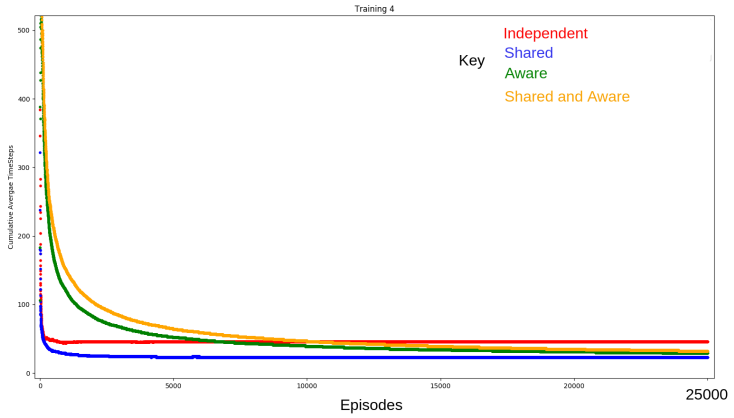

Figure 8: Comparison of the 4 sub-experiments

- In the case of 2 preys and 2 predators however a different trend was observed. Sharing was found to perform worser than Aware, Aware + State Sharing. We hypothesize that this is because the two predators go after two different preys. However in case of Aware + State Sharing, the predators learn to capture state of other predator and use this information. This is possible to the additional components to the state space.

2.5 Communication as an Action

We now consider the setting where an agents state is not broadcast by default, but rather there is a new action dedicated to broadcasting. If the agent takes this action, it doesn't move rather only broadcasts its current state to other agents.

Our experiments agree with our intuition that making turning broadcast into an explicit action severely restricts the ability of agents to efficiently transfer information. In fact we note that after training this our algorithm on this model, broadcast as an action is chosen only about 10% of times.

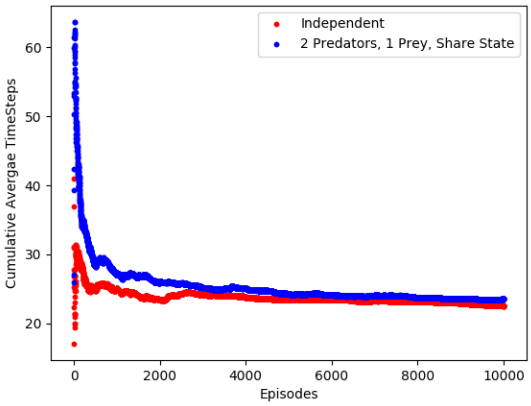

Figure 9: Communicating agents

2.6 Smart Prey

Now we runs a set of experiment where we have a smart prey which tries to actively run away from the nearest predator. We also handle the cases when it is backed up a corner or an edge. In such cases it tries to evade capture by moving along a feasible direction which will take it farthest from the closest predator. We algorithmically specify this policy without any kind of learning (expert policy for prey).

We run this experiment for cases where we have one and two preys. In the case of two preys however the system eventually degenerates to the case for a single prey since once they converge to the same cell, both the preys have identical deterministic strategies for evading the predators.

The following plots exhibit our observations:

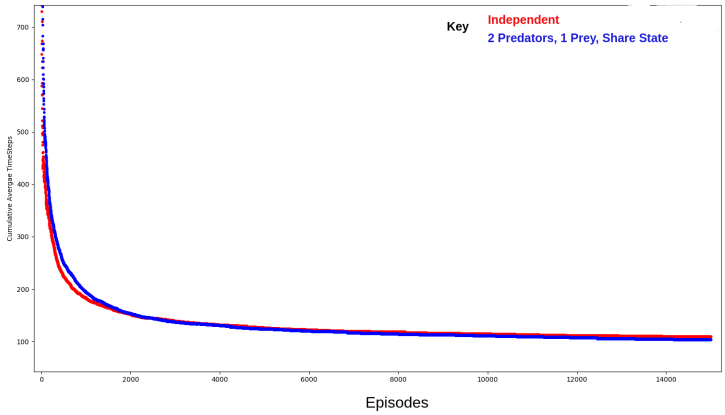

Figure 10: One prey and two predators

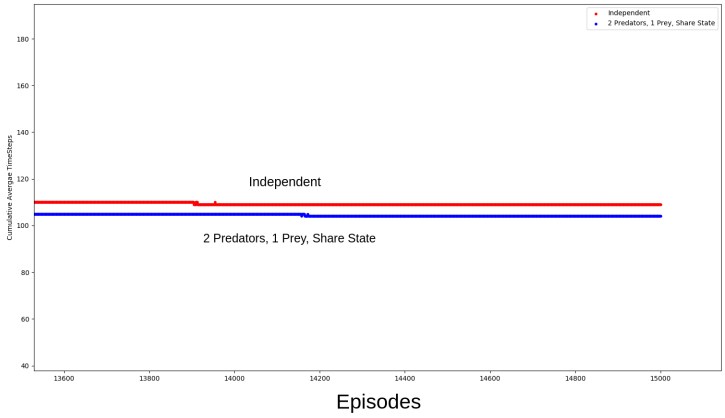

Figure 11: A zoomed in view of the eventual variation

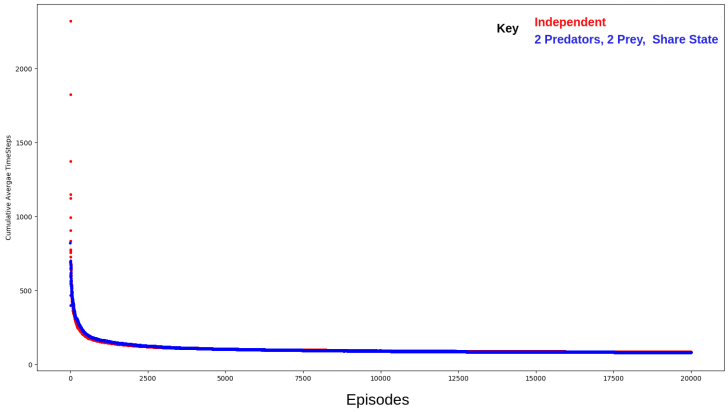

Figure 12: Two prey and two predator

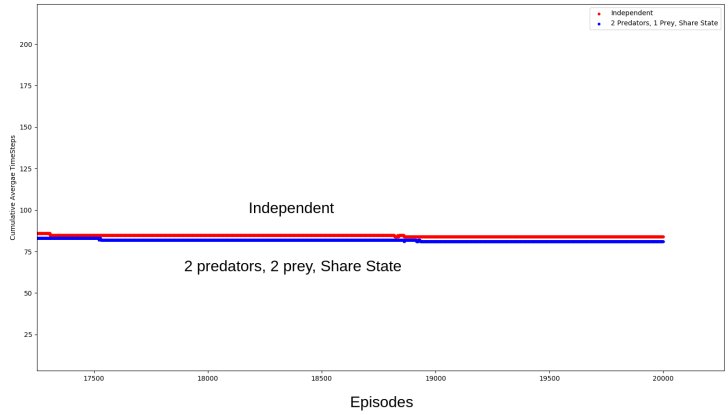

Figure 13: A zoomed in view of the eventual variation

3 Conclusion

We have explored the idea of enabling communication between agents exploring as predators in a predator prey environment. We have tried out two methodologies of enabling communication. The first is direct - the position of partner predators is directly available as a part of the state of any predator. The second is implicit - the at any given time, the predators are only aware of the states of their partners via a broadcasting mechanism. The main advantage of the second is that the training time is lower due to a much smaller state space. The disadvantage however is that the agent needs to learn the correlation between receiving useful broadcasts and catching its prey.

We also tried a related setting where broadcast comes at a cost of not being able to locomote during that time step. We also designed as adversarial variant of the scenario where the prey is smart in that it actively, though algorithmically, tries to evade the predator.

The setup for the experiments considered is available on GitHub.

References

[1] Whitehead, S. D. (1991). A complexity analysis of cooperative mechanisms in reinforcement learning. In Proceedings of AAAI-91.

[2] M Tan (1993). Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents.

[3] Sutton and Barto (2017). Reinforcement Learning: An Introduction.

[4] Sutton et al. (2000). Policy gradient methods for reinforcement learning with function approximation.

[5] Ronald J. Williams (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning.