1 Multi-Agent Deep RL

The field of Deep RL ultimately advances as researchers try to tackle increasingly complicated and more realistic problems. It is well understood that most real-life tasks involve multiple actors, especially when cooperation is involved. Hence, Multi-Agent Deep RL (MADRL) is the extension of single-agent Deep RL to multi-agent problem settings. The following notes will describe Multi-Agent Systems, and the MADRL problem setup. We will then cover two key challenges with MADRL, namely non-stationarity and partial observability. Next, we step back into the single-agent setting and examine how DQN’s were extended to handle partial observability via RNN’s, hence the term Deep Recurrent Q Network. We will also see how DQN’s were extended to multi-agent settings through Independent DQN’s. Lastly, we will see how researchers combined elements of DRQN’s and Independent DQN, and with some slight modifications, to successfully tackle multi-agent problems via an architecture called Distributed Deep Recurrent Q Networks.

1.1 Multi-Agent Systems

MADRL is applied to Multi-Agent Systems (MAS). Intuitively, a MAS is a computerized system comprised of multiple interacting agents. These agents must either compete or cooperate to obtain the best overall results, however that may be defined. Examples of MAS’s include:

- Multi-player online games
- Cooperative robots in production factories
- Traffic control systems
- Unmanned Aerial vehicles, surveillance, spacecrafts

Note that in a MAS, not all agents are required to be of the same type. While this increases complexity, it adds flexibility when modeling real-world problems.

1.2 MADRL Problem Setup

- MDP generalized to a stochastic game, or a Markov game.
- Let \( n \) denote the number of agents, \( S \) as a discrete set of states,
- \( A_i, i = 1, 2, \ldots, n \) as a set of actions for each agent.
- The joint action set for all agents is \( A = A_1 \times A_2 \times \cdots \times A_n \)
- The state transition probability function is \( p : S \times A \times S \rightarrow [0, 1] \)
- The reward function is \( r : S \times A \times S \rightarrow \mathbb{R}^n \)
- The value function for each agent depends on the joint action and joint policy: \( V_\pi : S \times A \rightarrow \mathbb{R}^n \)
1.3 MADRL Challenges

There are two key challenges that arise when applying single-agent Deep RL to MAS’s: **Non-stationarity**, and **partial observability**. We will investigate these two in more detail.

1.4 MADRL Challenge: Non-stationarity

In a single agent environment, an agent is concerned only with the outcome of its own actions. In a multi-agent setting, an agent observes the behavior of other agents in addition to its own outcomes. Learning among multiple agents is complex - agents interact and learn concurrently. These interactions constantly reshape the environment and lead to non-stationarity. Estimated potential rewards of an action would be inaccurate, and so good policies at a given point may not remain so in the future. Q-learning is not guaranteed to converge to most multi-agent problems since the Markov property does not hold (because of non-stationarity). To account for this, collecting and processing information must happen recurrently, while ensuring that it does not affect the agents stability. The exploration-exploitation dilemma is more involved under multi-agent settings.

1.5 MADRL Challenge: Partial Observability

In real-world applications, often times agents only have partial observability of the environment - this type of problem can be modeled using a partially observable Markov decision process (POMDP).

1.5.1 POMDP Definition

A discrete-time POMDP models the relationship between an agent and its environment. Formally, a POMDP is a 7-tuple \((S, A, T, R, \Omega, O, \gamma)\), where

- \(S\) is a set of states
- \(A\) is a set of actions
- \(T\) is a set of conditional transition probabilities
- \(R : S \times A \rightarrow \mathbb{R}\) is the reward function.
- \(\Omega\) is a set of observations
- \(O\) is a set of conditional observation probabilities
- \(\gamma \in [0, 1]\) is the discount factor

1.6 Deep Recurrent Q Network

To better understand how partial observability is tackled in the multi-agent setting, it is informative to understand how partial observability is handled in single-agent settings. This leads us to Deep Recurrent Q Networks (DRQN). Deep Q Network’s (DQN) are limited in that the learn a mapping from a limited number of past states. For example, in the case of Atari, a DQN is trained on the last four game screens the agent encountered. This means that by passing in the last four game screens, the problem can be framed as a MDP. However, we know that POMDP’s better capture the dynamics of real world environments. This lead researchers to add **recurrency** to DQN’s, hence DRQN.

DRQN’s are applied in single-agent partially observable settings. DRQN’s replace the first fully connected layer of a DQN with a recurrent LSTM layer of the same size. The RNN can maintain an internal state and aggregate observations over time. DRQN’s was tested on Atari games, and to make them more like POMDPs, the authors introduced flickering, where they would randomly omit past frames. [Hausknecht and Stone 2017]
Figure 1: DQN Architecture (https://kam.al/blog/drqn/)

Figure 2: DRQN Architecture (https://kam.al/blog/drqn/)

<table>
<thead>
<tr>
<th>Game</th>
<th>DRQN ± std</th>
<th>Ours</th>
<th>QN ± std</th>
<th>Minh et al.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asteroids</td>
<td>1029 ± 1312</td>
<td>1070 ± 1345</td>
<td>1639 ± 1542</td>
<td></td>
</tr>
<tr>
<td>Beam Rider</td>
<td>3269 ± 1167</td>
<td>6923 ± 1207</td>
<td>6846 ± 1619</td>
<td></td>
</tr>
<tr>
<td>Bowling</td>
<td>62 ± 5.9</td>
<td>72 ± 11</td>
<td>42 ± 6.8</td>
<td></td>
</tr>
<tr>
<td>Centipede</td>
<td>3333 ± 1101</td>
<td>3653 ± 1184</td>
<td>3209 ± 5237</td>
<td></td>
</tr>
<tr>
<td>Chopper Cnl</td>
<td>2070 ± 785</td>
<td>1460 ± 597</td>
<td>6687 ± 2916</td>
<td></td>
</tr>
<tr>
<td>Double Dunk</td>
<td>-2 ± 7.8</td>
<td>-10 ± 3.5</td>
<td>-18 ± 2.5</td>
<td></td>
</tr>
<tr>
<td>Frostbite</td>
<td>2878 ± 350</td>
<td>519 ± 303</td>
<td>328 ± 250.5</td>
<td></td>
</tr>
<tr>
<td>Ice Hockey</td>
<td>-4.4 ± 1.6</td>
<td>-3.5 ± 3.5</td>
<td>-1.6 ± 2.5</td>
<td></td>
</tr>
<tr>
<td>Ms. Pacman</td>
<td>2048 ± 653</td>
<td>2365 ± 735</td>
<td>2311 ± 525</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3: DRQN vs. DQN on Atari Games: On standard Atari games, DRQN performance parallels DQN, excelling in the games of Frostbite and Double Dunk, but struggling on Beam Rider. (Hausknecht and Stone, 2017)

Figure 4: DRQN Performance Degradation: When trained on normal games (MDPs) and then evaluated on flickering games (POMDPs), DRQNs performance degrades more gracefully than DQNs. Each data point shows the average percentage of the original game score over all 9 games in Figure 3. (Hausknecht and Stone, 2017)
1.7 Independent DQN

Independent DQN is a simplistic way of extending DQN to cooperative multi-agent settings. Each agent observes the global state, selects an individual action, and receives a team reward shares among all agents. Each agent independently and simultaneously learn their own Q functions. However, this could lead to non-stationarity issues because each agent is treating other agents as part of the environment. (Jakob N. Foerster, 2016)

1.8 Distributed DRQN (DDRQN)

We have seen how DRQN can be applied in single-agent settings to deal with partial observability. We have also seen how DQN can be extended to multi-agent setting via Independent DQN. Therefore, a natural next step would be to combine DRQN with Independent DQN. However, researchers found that simply combining the two leads to poor performance. To improve the performance, the researchers made three key modifications:

- **Last Action Input**: Provide each agent with its previous action as input into the next time step. Since the agents employ stochastic policies for the sake of exploration, they should in general condition their actions on their action-observation histories, not just their observation histories. Feeding the last action as input allows the RNN to approximate action-observation histories.

- **Inter-Agent Weight Sharing**: Tying the weights of all agents networks. Essentially only 1 network is learned, but the agents can still behave differently because they receive different observations, and thus evolve different hidden states. In addition, each agent receives its own index as an input, so this makes it easier for agents to specialize. This significantly reduces the number of parameters that needs to be learned, and so this speeds up learning.

- **Disabling experience replay**: While this is effective in single agent settings, this leads to non-stationarity in multi-agent settings as multiple agents learn simultaneously.

With these three modifications, the authors called this new architecture Distributed DRQN (DDRQN). (Jakob N. Foerster, 2016)
2 Emergent Communication

The ability to cooperate through language is a defining feature of humans. As the capabilities of deep artificial networks increase, researchers are interested in knowing whether they also can develop a shared language to interact. This development of a language protocol is defined in the literature as emergent communication (Lazaridou and Baroni, 2020).

2.1 Research Perspectives

Emergent communication is generally studied through two perspectives:

- First, from a scientific perspective, understanding the conditions under which language evolves in communities of deep agents and its emergent features can shed light on human language evolution.

- Second, from an applied perspective, endowing deep networks with the ability to solve problems interactively by communicating with each other and with us should make them more flexible and useful in everyday life.

2.2 Applied Perspective

This lecture will be focusing on the later. That is, how do we endow multi-agent systems with the ability to communicate under different requirements. This can be further categorized into:

- Communication that facilitates inter-agent coordination.

- Communication between self-interested and competing agents.

- How do we enable communication in agents such that they are able to interact with humans?

2.3 Communication Facilitating Inter-Agent Coordination

As the complexity of tasks and the number of agents grow, the coordination abilities of agents become of fundamental importance. Humans excel at large-group coordination, and language clearly plays a central role in their problem solving ability (David-Barrett and Dunbar, 2016; Lupyan and Bergen, 2016; Tomasello, 2010). Therefore, this insight is inspiring algorithmic innovations in multi-agent learning, where communication is used to facilitate coordination among multiple agents interacting in complex environments. We will look at one particular work where communication is used to facilitate coordination among multiple agents.

2.3.1 Learning to Communicate with Deep Multi-Agent Reinforcement Learning

We consider the work by Foerster et al. (2016). This paper considers RL problems with both multiple agents and partial observability. All the agents share the goal of maximising the same discounted sum of rewards $R_t$. While no agent can observe the underlying Markov state $s_t$, each agent receives a private observation $o'_a$ correlated to $s_t$. At each time-step, the agents select an environment action $u \in U$ that affects the environment, and a communication action $m \in M$ that is observed by other agents but has no direct impact on the environment or reward. We are interested in such settings because it is only when multiple agents and partial observability coexist that agents have the incentive to communicate. As no communication protocol is given a priori, the agents must develop and agree upon such a protocol to solve the task. They focus on settings with centralised learning but decentralised execution. In other words, communication between agents is not restricted during learning, which is performed by a centralised algorithm; however, during execution of the learned policies, the agents can communicate only via the limited-bandwidth channel.

While not all real-world problems can be solved in this way, a great many can, e.g., when training a group of robots on a simulator. Centralised planning and decentralised execution is also a standard paradigm for multi-agent planning (Kraemer and Banerjee, 2016; Oliehoek et al., 2008).
**RIAL - Reinforced Inter-Agent Learning**  The most straightforward approach, which they call reinforced inter-agent learning (RIAL), is to combine DRQN with independent Q-learning for action and communication selection (See figure 5). Each agents Q-network conditions on that agents individual hidden state and observation. Couple things to note, first, they disable experience replay to account for the non-stationarity that occurs when multiple agents learn concurrently, as it can render experience obsolete and misleading. Second, to account for partial observability, they feed in the actions $u$ and $m$ taken by each agent as inputs on the next time-step.

**Figure 5:** At each step for each agent, the agent receives a partial observation from the environment and discrete messages from other agents. Then, it produces an action and messages using epsilon-greedy over q-values. Here, the red-lines describe the flow of gradients [Foerster et al., 2016].

In this model, they treat each agents as independent networks and so the learning phase is not centralised, even though the problem setting allows it to be. Consequently, the agents are treated exactly the same way during decentralised execution as during learning. It is also possible to share parameters among the agents which speeds up training.

**DIAL - Differentiable Inter-Agent Learning**  While RIAL can share parameters among agents, it still does not take full advantage of centralised learning. In particular, the agents do not give each other feedback about their communication actions. Human communication is rich with tight feedback loops. RIAL lacks this feedback mechanism, which is intuitively important for learning communication protocols. To address this limitation, they propose differentiable inter-agent learning (DIAL) (See Figure 6). The main insight behind DIAL is that the combination of centralised learning and Q-networks makes it possible, not only to share parameters but to push gradients from one agent to another through the communication channel. Thus, while RIAL is end-to-end trainable within each agent, DIAL is end-to-end trainable across agents. Letting gradients flow from one agent to another gives them richer feedback, reducing the required amount of learning by trial and error, and easing the discovery of effective protocols. DIAL works as follows: during centralised learning, communication actions are replaced with direct connections between the output of one agents network and the input of anothers. Thus, while the task restricts communication to discrete messages, during learning the agents are free to send real-valued messages to each other. Since these messages function as any other network activation, gradients can be passed back along the channel, allowing end-to-end backpropagation across agents. But during decentralized execution, the messages are discretized with a regularizing unit (DRU).

**Switch Riddle**  Now, let us consider a communication game adapted from a riddle. The riddle is as follows: "One hundred prisoners have been newly ushered into prison. The warden tells them that starting tomorrow, each of them will be placed in an isolated cell, unable to communicate amongst each other. Each day, the warden will choose one of the prisoners uniformly at random with replacement, and place him in a central interrogation room containing only a light bulb with a toggle switch. The prisoner will be able to observe the current state of the light bulb. If he wishes, he can toggle the light bulb. He also has the option of announcing
that he believes all prisoners have visited the interrogation room at some point in time. If this announcement is true, then all prisoners are set free, but if it is false, all prisoners are executed. The warden leaves and the prisoners huddle together to discuss their fate. Can they agree on a protocol that will guarantee their freedom?” [Wu (2002)]

This riddle can be framed as a multiagent game with a discrete communication channel.

**Question:** What do you think the state, action, reward and partial observations are?

- Multi-Agent: N agents
- Communication Channel: 1-bit (on/off)
- State: N-bit array: has i-th prisoner been to interrogation room
- Action: *Tell / None*
- Reward: +1 (freedom) / 0 / -1 (all die)
- Observation: *None* Or *Switch*

In their formalisation, at time-step \( t \), agents observe that is either 0, 1, which indicates if the agent is in the interrogation room. Since the switch has two positions, it can be modelled as a 1-bit message, \( m_a^t \). If agent \( a \) is in the interrogation room, then its actions are \( u_a^t \in \{None, Tell\}; otherwise the only action is None. The episode ends when an agent chooses Tell or when the maximum time-step, \( T \), is reached. The reward \( r_t \) is 0 unless an agent chooses Tell, in which case it is 1 if all agents have been to the interrogation room and 1 otherwise. Following the riddle definition, in this experiment the previous message is only available to the agent \( a \) in the interrogation room.

**Switch Riddle experimental results:** All four communication based methods (DIAL, DIAL-NS, RIAL, RIAL-NS) learn an optimal policy substantially outperforming the NoComm baseline. DIAL with parameter sharing reaches optimal performance substantially faster than RIAL. Furthermore, parameter sharing speeds both methods. In this setting, RIAL without parameter sharing (RIAL-NS) was unable to beat the NoComm baseline. These results illustrate how difficult it is for agents to learn the same protocol independently. Hence, parameter sharing can be crucial for learning to communicate. They also show the policy that is discovered for \( n=3 \). See figure 7. Therefore, we can see that cooperative agents are able to develop a communication protocol in order to succeed at the task.

---

**Figure 6:** DIAL architecture. Note the additional DRU unit which discretizes messages during decentralized execution. ([Foerster et al.] 2016).
2.4 Beyond Cooperation: Self-interested and Competing Agents

While most deep agent emergent communication work considers interactions between co-operative agents, there is increasing interest in cases where agents interests diverge. Communication between self-interested and competing agents has been extensively studied in game theory and behavioral economics (Crawford and Sobel, 1982), since in human interaction and decision making tensions between collective and individual rewards constantly arise. From a practical point of view, a better understanding of emergent communication in non-fully-cooperative situations can positively impact applications such as self-driving cars.

Theoretical results suggest that, when the agents incentives are not aligned, meaningful communication is not guaranteed (Farrell and Rabin, 1996). This is also seen in practice. When agent interests diverge, senders could choose to communicate information increasing their personal reward only (and potentially decreasing that of others), and consequently disincentivize receivers from paying attention to what is communicated.

### 2.4.1 Emergent Communication through Negotiation

We will look at a work in this setting by Cao et al. (2018). The authors of this work study language emergence in a semi-cooperative model of agent interaction. In particular, they consider a negotiation environment (see figure 8) consisting in a multi-turn version of the ultimatum game. Agents are presented with three types of items: peppers, cherries and strawberries. At each round:

- An item pool is sampled uniformly, instantiating a quantity (between 0 and 5) for each of the types.
- Each agent receives a utility function for which specifies how rewarding one unit of each item is. Each agent only has access to its own utilities.
- The agents negotiate for N timesteps by exchanging messages and proposals.

In particular, to achieve negotiation, the agents need to communicate. One obvious communication protocol is to directly transmit the proposed division of items. This is referred to as the proposal channel. They also give the agents the ability of transmitting strings of arbitrary symbols with no a priori grounding. They refer to this as the linguistic channel, a specific instantiation of the more general concept of cheap talk.

**Question:** What are the key differences in properties between the two channels?

- **Non-bindingness:** Messages sent via the linguistic channel do not commit the sender to any course of action, unlike directly transmitting a proposal which binds the sender to the proposal.
- **Unverifiability:** There is no inherent link between the linguistic utterance and the private proposal made, meaning that the agents could potentially lie.
This work therefore looks into whether and under what circumstances the agents can make use of this channel to facilitate negotiation and establish a common ground for the symbols.

**Learning:** In this work, the agents are actually trained via REINFORCE with an entropy regularization term. Each agent learns 3 policies:

- A Binary policy for the termination action.
- A Policy for linguistic utterances.
- A Policy for the proposal the agent generates

Each agent receives a reward \((R_A, R_B)\) based on the utilities for each of the items. They then also introduce a **prosocial reward** \(R\) that is shared across A and B, which is the sum of the agents individual (selfish) rewards, i.e., \(R = R_A + R_B\). Prosocial agents are therefore incentivised to communicate, as finding the optimal joint allocation requires communicating the hidden values between agents. In their experiments, they consider 4 separate communication configurations: only the proposal channel, only the linguistic channel, both channels open, and no communication at all.

**Experiment 1: Can Self-interested Agents Learn to Negotiate?:** The first experiment tries to answer whether self-interested agents can learn to negotiate in this task. The answer is yes! They observe that the proportion of total utility each agent receives is roughly equal, and above 50%, suggesting that the agents have learnt to keep items with higher utility, while giving away items with low utility in the top two plots. However, if one only uses the linguistic channel, agents do not negotiate optimally. They claim that this is because the agents find it difficult to ground cheap talk in order to exchange meaningful information about how to divide up the items. (See figure 9)

**Experiment 2: Can Pro-social Agents Learn to Coordinate?:** In the next experiment, they try to observe whether pro-social agent can learn to coordinate: For prosocial agents with aligned interests, theoretical results suggest that communication using cheap talk is a Nash equilibrium. That is indeed what they observe, as the linguistic channel results in much better task success than any other communication scheme (See figure 10). The information bandwidth of the proposal channel is limited by the type and number of items, whereas the linguistic channels bandwidth is unconstrained and can (in theory) be arbitrarily large, helping the effective exchange of task-specific information. One thing you can note is that, in the prosocial setting, having both the proposal channel and linguistic channel, results in worse performance.
Figure 9: Results: Can self-interested agents learn to negotiate? Self-interested agents can learn to negotiate fairly. However, self-interested agents do not appear to ground cheap talk. By using only the linguistic channel, agents do not negotiate optimally. (Cao et al., 2018)

<table>
<thead>
<tr>
<th>Agent sociality</th>
<th>Proposal</th>
<th>Linguistic</th>
<th>Both</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-interested</td>
<td>0.87 ± 0.11</td>
<td>0.75 ± 0.22</td>
<td>0.87 ± 0.12</td>
<td>0.77 ± 0.22</td>
</tr>
<tr>
<td>25th &amp; 75th percentiles</td>
<td>[0.81, 0.95]</td>
<td>[0.61, 0.94]</td>
<td>[0.81, 0.95]</td>
<td>[0.62, 0.97]</td>
</tr>
<tr>
<td>Turns taken</td>
<td>3.55 ± 1.12</td>
<td>5.36 ± 1.20</td>
<td>3.43 ± 1.10</td>
<td>3.00 ± 1.13</td>
</tr>
<tr>
<td>Prosocial</td>
<td>0.93 ± 0.10</td>
<td>0.99 ± 0.02</td>
<td>0.92 ± 0.11</td>
<td>0.95 ± 0.11</td>
</tr>
<tr>
<td>25th &amp; 75th percentiles</td>
<td>[0.89, 1.0]</td>
<td>[1.0, 1.0]</td>
<td>[0.88, 1.0]</td>
<td>[0.93, 1.0]</td>
</tr>
<tr>
<td>Turns taken</td>
<td>3.10 ± 0.99</td>
<td>3.71 ± 0.58</td>
<td>2.98 ± 0.97</td>
<td>2.27 ± 0.69</td>
</tr>
</tbody>
</table>

Figure 10: Results: Can pro-social agents learn to coordinate? Pro-social agents have higher total utility as compared to self-interested agents. Pro-social agents perform best when using just the linguistic channel. (Cao et al., 2018)

Question: Why do you think that including the proposal channel in addition to the linguistic channel worsened the performance?

They say that the difficulties in optimisation is caused by the results for both channels open to be similar to those for only the proposal channel open, as the proposal channel is pre-grounded, which poses a strong local maxima. With better optimisation, they believe that the performance of both channels open will more closely match those of just the linguistic channel open.

2.5 Machines Cooperating with Humans

One of the most ambitious goals of AI is to develop intelligent agents able to interact with humans. Endowing agents with communication is an important milestone towards reaching this goal. So far, we have seen that we can build and train agents to use a communication channel to collaborate. However, for such communication to be useful for humans, we would require that the communication occur in natural language. One way to go about building such agents is by creating an instructor-follower model, sometimes also known as latent language policies (Andreas et al., 2017). It consists of two agents: an instructor and a follower. An instructor observes the state and produces an instruction. The follower observes this instruction and performs action.
2.5.1 Hierarchical Decision Making by Generating and Following Natural Language Instructions

This work by [Hu et al., 2019] builds an hierarchical instructor-follower model in a challenging RTS game. This game has rock-paper-scissors dynamics. They also collect instruction-execution pairs from human collaboration. The architecture is as follows: the instructor observes the current game state and produces high-level natural language instructions. The executor is then conditioned on this natural language instruction as well as on the game state and makes low-level actions in the environment. See figure 12.

**Experimental results:** The hierarchical behaviour cloned model wins 57.9% of the games against a non-hierarchical model. The non-hierarchical model produces the low-level actions directly. This result shows that exploiting the compositional structure of natural language improves generalization for both the instructor and executor model, significantly outperforming agents without an hierarchical structure.
2.5.2 Key Challenge: Language Drift

A key challenge in this regime is language drift. We just saw that we can train instructor-follower models through behaviour cloning. How can we learn better language policies? We can use RL! However, that’s when we encounter a problem. The transcript in figure 13 was produced in an environment similar to the negotiation environment we saw earlier. The agents were initialized using human transcripts. However, when finetuning towards a reward for high utility, they encounter language drift. Language drift is the problem when the natural language syntax and semantics diverge. Recent work (Jacob et al., 2021; Lu et al., 2020; Lee et al., 2019) have tried to tackle this problem.

Bob: i can i i everything else . . . . . . . . . . . . . .
Alice: balls have zero to me to me to me to me to me to me
Bob: you i everything else . . . . . . . . . . . . . .
Alice: balls have a ball to me to me to me to me to me to me

Figure 13: Example of language drift from Yarats and Lewis (2018).

Figure 14: Example of semantic drift in the MiniRTS environment (Hu et al., 2019) from Jacob et al. (2021). In the RTS environment, fine-tuning results in semantic drift. For example, the instructor produces “build a spearman”, however, the executor builds a dragon instead.

3 Summary

- Emergent communication: Language protocols produced by multiagent systems to communicate.
- We can train agents to develop and use language protocols in co-operative and competitive settings.
- In co-operative settings, a communication channel becomes important for good performance.
- Collaboration between humans and trained agents require the communication channel to be grounded in natural language.
- Language drift: Syntax and semantics of the language diverge when fine-tuning towards a specific reward.

4 Questions:

- Training Instructor-follower agents with reinforcement learning is reminiscent of hierarchical RL. What are some of similarities or differences?
• We looked briefly at language drift. What are some ideas to tackle this problem?

5 Future work:

With success in games like Go, Chess and Poker, researchers are becoming increasingly interested in games that require communication for collaboration between players. Hanabi (Bard et al., 2020) is a multiplayer game which requires players to collaborate by signalling to each other. Diplomacy (Anthony et al., 2020; Gray et al., 2020; Paquette et al., 2019) is a 7-player game that requires players to collaborate and compete using language. These games could act as a good test-bed to test human-machine collaboration.

6 Contributions

Raghav Kedia presented and scribed the notes for part 1 of this lecture on Multi-Agent Deep RL. Athul Paul Jacob presented and scribed the notes for part 2 of the lecture on Emergent Communication. TA Sirui Li reviewed the draft and helped with the preparation for the presentation.

References


