Autonomous Vehicle Fleet Coordination With Deep Reinforcement Learning

Cane Punma
PwC AI Accelerator
canepunma1@gmail.com

Abstract

Autonomous vehicles are becoming more common in city transportation. Eventually, autonomous vehicles must learn to coordinate and efficiently navigate a city. We believe that complex intelligent behavior can be learned by these agents through Reinforcement Learning. In this paper, we discuss our work for identifying the optimal operational rules by adapting the Deep Q-Learning (DQN) model to the multi-agent setting. Our approach applies deep reinforcement learning by combining convolutional neural networks with DQN to teach agents to fulfill customer demand in an environment that is partially observable to them. We also demonstrate how to utilize transfer learning to teach agents to balance multiple objectives such as navigating to a charging station when their energy level is low. The evaluations presented show that our solution has shown that we are successfully able to teach agents cooperation policies while balancing multiple objectives.

1 Introduction

Many business problems that exist in today's environment consist of multiple decisions makers either collaborating or competing towards a particular goal. In this work, the challenge is applying multi-agent systems for autonomous fleet control. As Autonomous Vehicles (AVs) become more prevalent, companies controlling these fleets such as Uber/Lyft will need to teach these agents to make optimal operational decisions. The goal of this work is to train these agents/cars to learn optimal relocation strategies that will maximize the efficiency of the fleet, while satisfying customer trip demand. Recent industry solutions in the use case of fleet control may use dynamic agent-based simulation modeling to optimize over a chosen objective function and generate heuristics that agents abide by. This approach requires various hand coded rules as well as assumptions to help the model converge on a solution. This becomes an extremely difficult problem when there are many outside environment dynamics that can influence an agent's/car's decision making (e.g. charging, parking). Furthermore, a solution to a particular environment may become outdated with new incoming information (e.g. new demand distribution).

An algorithm that can adapt and learn decision making organically is needed for these types of problems; recent research in Reinforcement Learning and particularly Deep Reinforcement Learning has shown to be effective in this space. Deep Mind's success with Deep Q Learning (DQN) has proven to be very successful in learning human level performance for many Atari 2600 games, a previously difficult task due to highly dimensional unstructured data. In this work, we will pull from prior work in Multi-Agent Deep Reinforcement Learning (MA-DRL) and extend one of the temporal-difference learning algorithms, Deep Q Networks, to our multi-agent system of autonomous vehicle fleet coordination. Our city environment that holds the cars and customers will be represented as an image-like state representation where each layer holds specific information about the environment. The environment is only partially observable to agents in the simulation within a given distance. We will introduce our agents to this environment and will show how this approach facilities scaling of the system. Following, we will show how we took advantage of Transfer Learning to teach agents multiple objects in particular charging an import aspect of AVs. Our results show that we are successfully able to teach coordination strategies with other cars so that they can optimize the utility of the system all while balancing multiple objectives. Finally, we show that our approach is able to scale to a real life scenario where we pulled demand, geographical, and infrastructure data to form a San Francisco testing experiment.

2 Related Works

Multi-agent cooperation problems have been well studied, [Panait and Luke, 2005] provides a survey of literature in this field. Previous work like [Tan, 1993] have shown the successful use of Reinforcement Learning in social environments that require cooperation. The application of RL for the purpose of taxi revenue maximization has also been explored [Verma *et al.*, 2017] although this example assumes one single learning agent in the environment, and does not take into account multiple learning agents. Deep learning has pushed the possibilities of single agent reinforcement learning to be able to solve previously complicated and challenging tasks. This domain of Deep Reinforcement Learning has garnered

increased attention recently because of its effectiveness in solving highly dimensional problem spaces. The sub-field of multi-agent systems presents a difficult challenge in how we represents other agents in the environment. Since these agents are non-stationary, how do we train agents to intelligently cooperate and find an optimal policy when each agent's rewards depend on one another? [Tampuu *et al.*, 2015] builds on DeepMind success with the Atari game pong and tests cooperation and competitive policies with each player. This worked laid our motivation for training cooperative agents, although we aimed at proposing a centralized solution rather than decentralized Deep Q-Networks for each agent. This allows us to scale to the training and implementation of the practical example of hundreds of autonomous vehicles operating together. In a more recent work, [Foerster *et al.*, 2016] proposes two novel approaches (RIAL,DIAL) that is an end-to-end learning protocol which allows error to back propagate through multiple agents which is a similar approach taken by [Sukhbaatar *et al.*, 2016]. Although along the same lines of the previous work, this approach has only been tested on riddles where there is a small number of agents and does not seem to scale well to larger problems. [Egorov, 2016] introduces a novel approach to solve multi-agent systems through convolutional neural networks, stochastic policies, residual networks and is what we will build our incremental developed solution for the real world application of large scale autonomous fleet control on. This application will extend [Egorov, 2016] work to address the issues of partially observability and multi-objectives that accompany autonomous vehicle fleet control.

3 Problem Formulation

3.1 Environment Objects

The first step in training a Reinforcement Learning model to to create an environment that is representative of the real world characteristics of your problem. In our case, we needed to create an environment that represents the city dynamics that are associated with the Uber/Lyft ride sharing platform. The general goal of each agent is to travel to customers on the map and fulfill that demand. The different objects and locations that are present in this environment are as follows and can be referenced in the example Figure 1.

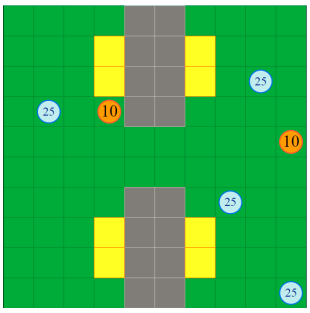


Figure 1: Example environment with agents and customers

Car (Blue Circle): This is the agent or decision maker that will be acting in the environment. Its actions are right, left, up, down, and stay. The car also has an associated energy level that decrements every timestep until it runs out of batter. Agents can occupy the same space as other agents without collision. Agents can not travel outside of the defined map or into obstacles on the map.

Customer (Red Circle): The goal location of where the Agents need to travel to. Once an agent's location matched a customers location, then the customer will be considered 'fulfilled'. Customers also have a 'drop-off location' and 'travel time' associated with them that determines how long an agent/car is removed from the system while in transit. Each customer begins with a random wait time of 5-15 timesteps which decrements every step. Once the wait time has expired, the customer is no longer available and removed from the map. Customers are generated from a demand distribution that we created based on real trip data for the test city location. Each timestep, new customers are generated by pulling from this distribution.

Obstacles (Gray Square): These are the locations on the map that the agents cannot travel to. Customer also will not appear at these locations.

Charging Stations (Yellow Square): These are the locations that agents can travel to refill their energy levels. Agents must choose the *Stay* action at these locations for *1/2* of their battery to be refilled. Customers can appear at these locations.

Open Road (Green Square): These are the standard spaces that agents and customers can exist on.

3.2 Environment Reward Structure

The order of activity are as follows: advance each agent sequentially, resolve their pickups or collisions, decremented car energy levels and customer wait times, remove customers that have waited too long or have been picked up, and generate new customers from the distribution. Each of these transitions will have a reward associated with each agent. The events and associated reward structure are shown in Table 1.

Event	Reward
Agent picks up customer	+1
Agent Collision (Obstacle, Outside Map)	-1
A customer's wait time runs out	-1
Standard Movement (E.g. left, right)	-0.1
Agent chooses 'Stay' in charging location (Charging)	+0.01
Agent chooses 'Stay' in open road (Parking)	-0.05
Agent's energy reaches 0	-10

Table 1: Event Reward Structure

After each timestep, the rewards are aggregated and attributed to that specific agent for that initial state. A small negative reward is attributed for standard movement. This was essential to incentive agents to find the quickest path to the customers. Another important reward structure decision was the small positive reward for charging. We found that there was not enough signal to just have a large negative penalty for losing all of its energy. We had to incentivize charging without giving too strong a reward to detract from the agents picking up the customers.

To encourage agents to act efficiently with one another, a penalty was applied for a missing customer in vicinity. Each agent feels the negative reward of missing a customer even if it is far from them. This incentivises the agents to learn a divide and conquer technique as to minimize the probability of missing a future customer. In the Results/Tests section we will show how effective this has been in a 7x7 and 10x10 simple bridge example.

3.3 Environment State Representation

Now that we defined our various objects and reward structure, we need to represent the environment as a vector that the Deep Reinforcement Learning model can learn from. Along with the suggestions of previous work, we found that the best way to do this while keeping spatial representations was with an image-like structure. Just like how images provide three matrices stacked on top of each other to represent *rgb* values, the same can be done for our environment. Our state representation has 5 layers that encode a different piece of information and result in a tensor of *5 X W x H*. The following are what each channels represents.

Self Layer: Encodes the location of the agent of interest. A value of 1 is given to the (x,y) location where the agent exists. Each self layer is unique to each agent.

Other Agents Layer: This encodes the locations of all the other agents excluding yourself. A value of 1 is given to locations where other agents exists and if there are two agents at a location the value 1 is still used.

Customer Layer: This encodes the locations of all the customers on the map or within each agent's vision. Here an integer value that represents that customers wait time remaining is placed at the location for that specific customer. When customers are removed from the map than the value will return to 0.

Obstacles Layer: This encodes the locations of obstacles and charging locations. The value 1 is used to represent locations where the agent can't go to and the value 2 is used to encode the location of the charging station.

Extra Agent Information: This encodes the energy and priority of the agent of interest. For energy we placed the integer value of how much energy is left on the (0,0) location of the matrix and the priority on the (0,1) location

4 Methods

In the following sections we will build on the simple Deep Reinforcement Learning technique Deep Q Learning (DQN) and walk through the adaptations that make it the final partially observable multi-agent system (PO-MADRL). We will describe how these methods relate to our environment and how they affected our implementation decisions for this environment.

4.1 DQN W/ Convolutional Neural Networks

The goal of reinforcement learning is to learn an optimal decision making policy that will maximize the future expected reward. One of the most common Reinforcement Learning algorithms is Q-Learning which represents the maximum discounted reward when we perform action a in state s, and continue optimally from that point on. Q-Learning is a form of model-free learning, meaning that it does not have to learn a model of the environment to optimally settle on a policy. The following formula is the Q-Function which is recursively calculated with transitions of *(s,a,r,s')*. In our case, s represents the image like array of the state representation holding all the agents, customers, and obstacles. The values *a* and *r* are the straightforward action taken by the agent and the associate reward. *s'* represents the image-like state after the agent has moved to its next location and collision detection or customer pick up has been resolved.

$$Q(s,a) = r + y * maxQ(s',a') \tag{1}$$

This formula is based on the Bellman equation which is simply the current reward plus the discounted future reward of the next time step if you were to take the best action. This can be implemented as a large table of states by actions and can be recursively solved if the environment is explored enough. If iterated enough times, the Q-Function should approach the true value of taking a certain action in a state and an optimal decision policy can be looked up by taking the action that will return that maximum value.

When environments become complex with very large state representations, like Atari games, then it becomes computationally difficult to converge on a reasonable Q-Function because the look-up tables can become very large. This is where the recent success in Deep Q Learning has come into play. Neural networks do a great job in generalizing highly dimensional data in a lower space. We can represent that large Q-Function table with a neural network. The Q(a,s) value can be estimated with the net and the loss of this network simply is as follows.

$$L = 1/2 * [r + maxQ(s',a') - Q(s,a)]^2 \tag{2}$$

Along the lines of [Egorov, 2016], we also decided to use Convolutional Neural Networks (CNNs) to interpret these image like state representations. CNNs have shown to be successful for image classification tasks because of its ability to understand spatial relationships within the image. The same can be said with the importance of the geospatial location of agents in relation to other agents and customers. In addition, CNNs allow us to scale our environment to large sizes without the exponential increase in processing time that may have occurred with fully connected networks.

What makes the convergence of this deep Q-Learning algorithm possible is the addition of Experience Replay, e-greedy exploration, and target networks. Experience Replay allows us to store all of our *(s,a,r,s')* transitions in a bucket that we can access. During training, mini batches of transitions are pulled from the bucket and used to fit on the network. E-greedy exploration allows us to initially take random actions but over time the agent will take actions it know result in the maximum expected reward. Lastly, target networks is a separate network that is a mere copy of the previous network, but frozen in time. This network is updated at slower intervals than the actual network as to stabilize the convergence of the Q-values during training.

4.2 Multi-Agent Deep RL (MADRL)

The biggest challenge with multi-agent systems is how to optimize each agent's control policy while taking into account the intentions of the other agents. There are a few approaches we can consider to address this challenge. One approach is to fully centralize the training and execution of agents. This would mean that all the agents perform their actions simultaneously so output of the centralized network would be the action space of the size proportional to the number of agents. Another approach is fully decentralized training, where each agent has its own Deep Q-Network policy and Replay Memory that it is optimizing on. The first approach has the advantage of the ability to converge easier on an optimal cooperation policy due to the nature of simultaneous action. The second approach allows for the training of unique policies and personalities for agents which can translate to unique roles when applying to the AV fleet control application. However, both approaches fail when scaling up to larger environments and more agents due to the exponential growth in training time. In the first approach, this exponential growth is a result of the proportional size of the observation and action space. In the second approach, the exponential growth is a result of the need to train multiple networksFrom our review of related works, we decided to implement an approach that took advantage of both. With this methodology, we chose to have all the agents train and learn on their policies on the same network and share parameters with the other agents. Execution of the agents occur individually so it is a decentralized control and execution of the policy. Each agents experiences are added to a common replay memory bucket so the accumulation of all the agents experiences are optimized over when fitting a batch. In addition, by having multiple agents add experiences to the same bucket, this allows for a faster convergence of the parameter shared network.

To achieve this we must have an execution protocol for performing each agents actions. Referring back to our image representation of the state, we have one layer that represents the agent of interest and another layer that represents the all the other agents. Each agent will have their respective view defined by their index and the partial observability which we describe in the next section will allow for independent behavior to be learned. When we defined *s'* as the next state or the updated image-like state after the agent has moved earlier, we must now modify it to account for the other dynamic agents in the system. The next state, *s'*, represents the current time in the environment after all the agents have consecutively moved and all the conflict resolutions of agent-agent and agent-customer events have been resolved. This allows each agent to not look at the other agents as a stationary objects, rather as intentions that need to be accounted for when making its own optimal decision.

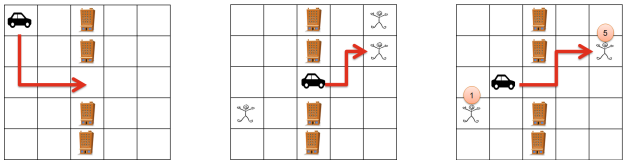
4.3 Partially Observable Multi-Agent Deep RL (PO-MADRL) + Transfer Learning

In this application, we chose to make the environment partially observable to each agent. This means that each agent would receive a private window of the environment from their perspective. Each agents recorded transition becomes *(o,a,r,o')* where *o* at timestep t represents the slice of the full environment state representation outlined above. This private window is the surrounding vision that the agent can see around it. From the current location of the agent, we take *v* spaces in all directions to form *(2v + 1)* sized matrix for each agent which is padded for locations off the map. We made an assumption that in the field, agents are offline and that there is no communication with the central station or controller. In other words, the dynamic locations of other agents and customers are partially observable to the agents in vision. This implementation is advantageous because as we scale the size of our map up, the time to train remains consistent because the size of the input observations will always be relative to the vision. We fully utilized this in our San Francisco test case which modeled geospatial and temporal demand distribution and found that its performance was comparable to the fully observable case.

Finally, in this work we take advantage of Transfer Learning to teach agents how to achieve multiple objectives. Autonomous vehicles run on a limited supply of battery and need to learn to recharge when necessary. In our AV fleet control problem, agents have the main objective of fulfilling customer demand but have the secondary objective is to keep itself alive on battery. As stated earlier in the reward structure, agents received a positive reward when picking up customers but a large negative reward when it loses all of its energy. We found that when training using this reward structure, the agents did not seem to converge on any successful strategy. In some iterations, the agents only figured out how to go to the charging locations and stay there. In other cases, the agents just continued picking up customers until all their batteries were depleted. To solve this, we took advantage of transfer learning to consecutively teach these objectives. We began with training the standard PO-MADRL model where the agents were given an infinite amount of energy. On the next iteration, the map and reward structure stayed consistent but this time, each agent was initialized with a random energy level from (50 minutes to 150 minutes). Now with a finite amount of energy, the agents began to experience the large negative reward associated with running out of battery. Over time, they were able to balance the two objectives and travel to the charging locations when needed. In our third test case with multiple objectives, we show that the agents learned to strike a balance between picking up customers and charging to where it outperforms a conservative baseline in customers fulfilled and the aggressive baseline on minimizing the number of times the battery depletes or the vehicle is left stranded.

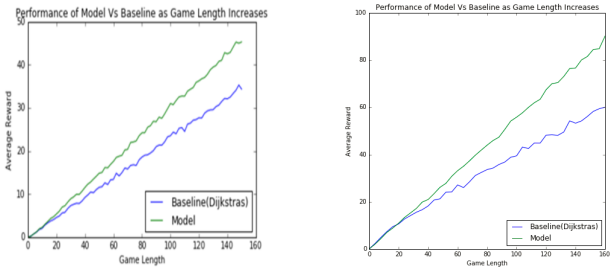
5 Experiments

In this section we provide experimental results that demonstrate the performance of our approach on the environment outlined above. This section will take you from our experiments on single agents to multiple agents with multiple objects. Finally, we will apply this approach and framework to a more larger scale and more realistic scenario which contains real data. In these tests, we compared the model agents to a baseline policy that employs Dijkstra's algorithm to fulfill customer demand. This baseline creates a graph of the map and calculates the shortest distance (including obstacles)



(a) Relocation (b) Future Reward (c) Timer

Figure 2: Single agent learns intelligent behavior



(a) 7x7 - 2 Agents (b) 10x10 - 4 Agents

Figure 3: Model vs Baseline on Average Customers Fulfilled

to all the customers. The heuristic built then ranks the customer distances by length and moves towards the customer that it can possibly make it to based on wait time remaining. In the multi-agent case, this baseline performs decentralized execution for all the agents from the first indexed one. The goal of these experiments are to show the progression of our approach from the simple DQN to the multi-agent partially observable method with the last test case.

5.1 Single Agent

To begin with, we wanted to test the effectiveness of the single-agent deep reinforcement learning algorithm in this environment before building on top of it. Would it be able to catch the small nuances of the environment and make optimal decisions to maximize future reward? Our experiment has shown that it has indeed been able to learn intelligent behavior on top of the ability to fulfill outstanding customer demand. Figure 4 shows simple illustrations, not representative of the actual environment, for three behaviors that were learned by the agent that allowed it to outperform the baseline heuristic.

1. When there is no outstanding customers, the agent relocates to the middle of the map because it is the optimal location that would result in the shortest distance when a new customer would appear.
2. In a situation where two customers are equal distance, the agent chooses the one that will result in a higher future reward. In this case it is another customer nearby. In other cases it may be that the chosen customer was in a more optimal location, like the middle.
3. The agent is able to recognize that the customer's wait time is running out and correctly makes a decision to not pursue that customer.

These examples provided a sanity check that the single-agent DQN was able to teach intelligent behavior that can be adapted to different spatial environments. Next we will test the ability of agents to mold these policies based on the surrounding agents.

5.2 Multi-Agent

The goal of this evaluation is to test the ability of the agents to learning coordination policies that maximize the reward of all the agents in the system. Our test environment consisted of a 7x7 map with 2 agents and a 10x10 map with 4 agents. Our experiment randomly placed these agents on the board and randomized their priority. Customers appear on the board uniformly during each timestep. Similar to Figure 1, the map has a bridge in the middle that separates two opposite sides of the map. The optimal cooperation policy here would be for the agents to split up and pick up customers on opposite sides of the map. We tested if our agents can perform this divide and conquer coordination technique versus our baseline heuristic, which would not be able to. Figure 2 shows the average customers fulfilled against the length of the game for the model versus the baseline on both experimental environments.

For both cases, we can see that as time (game length) increases, the model's reward begins to diverge from the baseline. This shows that the agents are performing are more efficiently than the baseline heuristic. After inspecting the individual games, we found that the agents did indeed perform a divide and conquer technique policy. When randomly placed on the same side of the map, one agent would take the initiative and travel to the other side of the map. When both agents started on different sides, they would stay at their sides and pick up their respective customers. This was also true of the 4 agent example, where each side of the map had an even distribution of agents. This example successfully demonstrates that these agents can learn a policy that maximizes the overall expected future system reward based on the other agent in the environment and the map structure.

5.3 Multi-Agent + Multiple Objectives

The goal of this experiment was to demonstrate the ability to teach multiple objectives to agents through transfer learning and show its strength over a rules based approach. The two objectives are to fulfill outstanding customer demand and keeping one's car battery alive. Our environment was a 10x10 bridge map with 4 agents. As described in our approach of transfer learning, we trained our agents using a two staged approach. In the first stage, we trained a model to pick up customers with infinite energy. In the second stage, we took the weights from the model trained during the prior stage and fine-tuned it. In the stage 2 environment, we gave each agent a random amount of energy and activated the penalty for losing all of your energy. During the fine-tuning stage we set the epsilon for the e-greedy exploration to 0.5. We linearly decayed this value to 0.01 over 3000 iterations. We found this approach to be successful in teaching the agent to keep itself alive and still fulfill customer demand versus training a model altogether to learn to manage both objectives from scratch.

In our experiment we demonstrated that the RL model was successfully able to learn how to balance these two objectives effectively. Would the model stray too closely to keeping itself alive and only pick up a few customers? Or, would it pick up many customers and sparingly recharge itself? We adapted our current Dijkstra's baseline heuristic to include an additional heuristic for energy management. This additional heuristic would force the agent to perform Dijkstra's to the closest charging location after its energy level is below a certain threshold. The first baseline (Conservative Baseline) would would travel back to the charging location if it's energy was below 30 percent. The second (Aggressive Baseline) would only go charge its energy when it was below 10 percent. And finally, the Standard Baseline was the middle of the prior two. Based on previous research with simulation modeling, rule based simulation modeling would have to set these thresholds according to business requirements and goals. Setting a hard threshold is not optimal because in some scenarios it may be more beneficial to be more aggressive and get the customers near you. In others, it helps to be more conservative as customers may be far away. Table 2 shows the performance of our model against the baseline agents.

Metric	Conservative	Aggressive	Model
Customers Fulfilled	37.15	40.32	**52.91**
Energy Depletions	0.63	11.76	**2.31**

Table 2: Performance of RL Model Vs Different Baseline Heuristics

We can see that the conservative baseline model successfully charges its battery when low but ends up picking up fewer customers than the aggressive baseline. Unfortunately, the aggressive baseline loses its energy more frequently and has experience around 11 battery depletions over the course of the trials. Our RL model shows that it is able to balance the average amount of times that agents lose all their energy while still fulfilling more customer demand than all three baselines. While the agents may still run out of battery from time to time, the overall performance boost of balancing these two objectives far exceeds the general baseline models.

5.4 Scaling to San Francisco Data-set

Finally, to test the ability of our methodology to scale, we implemented our approach on a proprietary San Francisco data set. Our environment adapted a geo-fenced representation of the San Francisco metropolitan area. The map was a 30x30 grid where each space represented a one square mile. This data-set contained real life demand distributions on a 24 hour clock for two weeks that we scaled down for the purpose of this experiment. Various charging locations were chosen as possible candidates for infrastructure in the city. We chose to incorporate 7 charging locations in areas of high demand in our experiment. Lastly, we created a drop off period after each customer was fulfilled. Each customer generated would have an associate drop off location and trip time to complete. The vehicles would be taken out of commission for that amount of time where it cannot pick up other customers and decremented an associated energy level to complete the trip.

In this experiment, we wanted to test the ability of the PO-MADRL approach to perform just as effectively while each agent had a partial view of the environment. Figure 3 shows that, our model (M) does indeed outperform the baseline (B) in number of customers fulfilled (Reward) and average number of cars fulfilled that have their energy deleted (Depletions). While the percentage difference between model and baseline is not as strong as the previous test cases, we have shown that we are able to implement a scalable methodology to this multi-agent problem without much loss of performance.

Agents	B-Reward	M-Reward	B-Depletions	M-Depletions
25	178.3	**195.2**	7.6	**6.3**
50	256.5	**278.5**	13.1	**8.05**
100	399.6	**438.8**	23.5	**11.5**

Table 3: Performance of RL Model Vs Baseline on San Francisco Dataset

6 Conclusion and Future Work

Deep Reinforcement Learning provides a great approach to teach agents how to solve complex problems, even better than humans. For instance, Deep Mind successfully taught agent to defeat the world champion in Go. More specifically, multi-Agent Reinforcement Learning problems provide an interesting avenue to investigate agent-to-agent communication and decision protocols. Since agents must rationalize the intentions of other agents, the dimensionality of the problem space becomes difficult to solve. In our use case, we wanted to see if we could scale a DRL solution up to an actual ride sharing environment that maintains the same dynamics as it would in real life. For this to be possible, we were tasked with the problem of teaching these agents effective cooperation strategies that would optimize the reward of the system along with the problem of teaching these same agents multiple objectives. This work demonstrated how we successfully applied a partially observable multi-agent deep reinforcement solution to this ride sharing problem. Along with that, we showed how we can effectively take advantage of transfer learning to adapt decision policies to account for multiple objectives. This approach can be effectively adapted to other multi-agent problems that require scalability and multiple objectives.

For future work, we plan to explore a decentralized version of this where each agent has its own Deep Q-Network. It would be interesting to see the different personalities that groups of agents develop respective of the other group. This would allow us to test competition strategies and employ a zero-sum game theory problem. Just as how there are competing interest in the ride-sharing community between Uber and Lyft, we feel that this can be an interesting competition environment to learn from. Another area of future work can be to test and compare the effectiveness of the other classes of deep reinforcement learning algorithms, namely policy gradients, and actor-critic.

References

[Egorov, 2016] Maxim Egorov. *Multi-Agent Deep Reinforcement Learning*. Stanford University, 2016.

[Foerster *et al.*, 2016] Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. *Learning to Communicate with Deep Multi-Agent Reinforcement Learning*. University of Oxford, United Kingdom, Canadian Institute for Advanced Research, CIFAR NCAP Program, Google DeepMind, 2016.

[Palmer *et al.*, 2017] Gregory. Palmer, Arthur Szlam, and Rob Fergus. *Lenient Multi-Agent Deep Reinforcement Learning*. Courant Institute, New York University and Facebook AI Research, 2017.

[Panait and Luke, 2005] Liviu Panait and Sean Luke. *Cooperative Multi-Agent Learning: The State of the Art*. George Mason University, 2005.

[Sukhbaatar *et al.*, 2016] Sainbayar. Sukhbaatar, Arthur Szlam, and Rob Fergus. *Learning Multiagent Communication with Backpropagation*. Courant Institute, New York University and Facebook AI Research, 2016.

[Tampuu *et al.*, 2015] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, LIlya Kuzovkin, and Kristjan Korjus. *Multiagent Cooperation and Competition with Deep Reinforcement Learning*. Computational Neuroscience Lab, Institute of Computer Science, University of Tartu Department of Mathematics, ETH Zurich, 2015.

[Tan, 1993] Ming Tan. *Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents*. GTE Laboratories Incorporated 40 Sylvan Road Waltham, MA 02254, 1993.

[Verma *et al.*, 2017] Tanvi Verma, Pradeep Varakantham, Sarit Kraus, and Hoong Chuin Lau. *Augmenting Decisions of Taxi Drivers through Reinforcement Learning for Improving Revenues*. School of Information Systems, Singapore Management University, Singapore, 2017.