# FULLY COOPERATIVE MULTI-AGENT DEEP REINFORCEMENT LEARNING

**Nikunj Gupta**

**Master of Technology Thesis**

June 2019


International Institute of Information Technology, Bangalore

# FULLY COOPERATIVE MULTI-AGENT DEEP REINFORCEMENT LEARNING

Submitted to International Institute of Information Technology, Bangalore
in Partial Fulfillment of
the Requirements for the Award of
Master of Technology

by

**Nikunj Gupta**
**IMT2014037**

International Institute of Information Technology, Bangalore
June 2019

*Dedicated to*

*The spirit of learning!*

# Thesis Certificate

This is to certify that the thesis titled **Fully Cooperative Multi-Agent Deep Reinforcement Learning** submitted to the International Institute of Information Technology, Bangalore, for the award of the degree of **Master of Technology** is a bona fide record of the research work done by **Nikunj Gupta**, **IMT2014037**, under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

---

Prof. G. Srinivasaraghavan

Bengaluru,
The 15$^{th}$ of June, 2019.

# FULLY COOPERATIVE MULTI-AGENT DEEP REINFORCEMENT LEARNING

## Abstract

Coordination of autonomous vehicles, automating warehouse management system or another real world complex problem like large-scale fleet management can be easily fashioned as cooperative multi-agent systems. Presently, algorithms in Reinforcement Learning (RL), which are designed for single agents, work poorly in multi-agent settings and hence there is a need for RL frameworks in Multi-Agent Systems (MAS). But, Multi-Agent Reinforcement Learning (MARL) poses its own challenges, some of the major ones being the problem of non-stationarity and the exponential growth of the joint action space with increasing number of agents. A possible solution to these complexities is to use Centralised learning and Decentralised execution of policies, however the question of using the notion of centralised learning to the fullest still remains open. As a part of this thesis, we developed an architecture, adopting the framework of centralised learning with decentralised execution, where all the actions of the individual agents are given as input to a central agent and it outputs information for them to utilize. Thus, the system of individual agents are given the opportunity of using some extra information (about other agents affecting the environment directly) from a central agent which also helps in easing their training. Results for the same are showcased on the Multi-Agent Particle Environment (MPE) by OpenAI. An extension of the architecture for the case of warehouse logistics is also shown in the thesis.

# Acknowledgements

First of all, I would like to sincerely thank my advisor Prof. G. Srinivasaraghavan for the constant guidance in my Master's Thesis and for his motivation and immense knowledge. His support enthused me throughout the time my of research.

My sincere thanks also goes to Dr. Swarup Kumar Mohalik for offering me the summer internship at Ericsson Research, Bangalore, finally extending to a thesis, leading me to work on a project in a domain which I enjoy very much (Reinforcement Learning) and so close to real world problems. I am grateful for his continuous steering me in the 'practical' direction of my research.

I must thank Prof. Manisha Kulkarni, Prof. Dinesh Babu, Prof. Sujit Kumar Chakrabarti, Prof. Shrisha Rao, Prof. V Ramasubramanian and all my professors at IIIT-B for extending the boundaries of my thinking.

Last but not the least, I would like to thank my family and friends for supporting me throughout my life.

# Contents

**Abstract** **iv**

**Acknowledgements** **v**

**List of Figures** **ix**

**List of Tables** **x**

**List of Abbreviations** **xi**

**1 Introduction** **1**

1.1 Reinforcement Learning . . . . . 1

1.1.1 Agent interaction with environment . . . . . 1

1.1.2 Markov decision processes . . . . . 2

1.2 Deep Reinforcement Learning . . . . . 2

1.2.1 Deep Q-Network (DQN) . . . . . 3

1.3 Multi-Agent Reinforcement Learning . . . . . 3

1.3.1 Markov Game . . . . . 4

1.4 Motivation . . . . . 5

1.5 Problem Statement . . . . . 5

**2 Background** **6**

2.1 Challenges in MARL . . . . . 6

2.1.1 Non-Stationarity . . . . . 6

2.1.2 Complexity in Training . . . . . 7

2.1.3 Continuous Action Spaces . . . . . 7

2.1.4 The ”Lazy” Agent Problem . . . . . 7

2.2 Centralised Learning and Decentralised Execution . . . . . 8

**3 Related Work** **9**

**4 Proposed Architecture** **13**

4.1 Environment / Simulation . . . . . 13

4.2 Architecture . . . . . 14

4.2.1 Central Agent . . . . . 14

4.2.2 Individual Agents . . . . . 15

4.3 RL Algorithms . . . . . 16

4.4 Workflow . . . . . 17

**5 Experiments and Results** **19**

5.1 Hyper-Parameters . . . . . 19

5.2 Experiments . . . . . 19

**6 Application in Warehouse Logistics** **22**

6.1 Case Study: Amazon Warehouses . . . . . 23

6.2 Extension of our framework . . . . . 24

6.3 Workflow . . . . . 27

**7 Conclusions** **28**

**Bibliography** **29**

# List of Figures

FC1.1 The agent-environment interaction in a MDP . . . . . 2

FC1.2 Multiple RL agents in the same environment. . . . . 4

FC3.1 Algorithm for Q Actor-Critic approach . . . . . 10

FC3.2 COMA's Overall Architecture . . . . . 11

FC3.3 Overview of multi-agent centralized critic, decentralized actor approach 12

FC4.1 Cooperative Navigation . . . . . 14

FC4.2 The Proposed Framework . . . . . 17

FC5.1 Sample Episode 1 . . . . . 20

FC5.2 Sample Episode 2 . . . . . 20

FC5.3 Sample Episode . . . . . 21

FC6.1 A picture of an Amazon Robot . . . . . 24

FC6.2 The Framework for the Warehouse Logistics Application . . . . . 26

# List of Tables

TC5.1 Hyper-parameters . . . . . 19

TC5.2 Experiment 1 . . . . . 20

TC5.3 Experiment 2 . . . . . 21

TC5.4 Experimental Results . . . . . 21

TC5.5 Average number of episodes per episode . . . . . 21

TC6.1 Typical MADRL Applications . . . . . 22

# List of Abbreviations

**COMA** . . . . . Counterfactual Multi-Agent Policy Gradients

**DDPG** . . . . . Deep Deterministic Policy Gradients

**DRL** . . . . . Deep Reinforcement Learning

**DQN** . . . . . Deep Q-Network

**MADDPG** . . . . . Multi-Agent Deep Deterministic Policy Gradients

**MADRL** . . . . . Multi-Agent Deep Reinforcement Learning

**MARL** . . . . . Multi-Agent Reinforcement Learning

**MAS** . . . . . Multi-Agent Systems

**MDP** . . . . . Markov Decision Process

**MPE** . . . . . Mult-Agent Particle Environment

**PPO** . . . . . Proximal Policy Optimization

**RL** . . . . . Reinforcement Learning

**TRPO** . . . . . Trust Region Policy Optimization

# CHAPTER 1

# INTRODUCTION

## 1.1 Reinforcement Learning

*"**Reinforcement learning**, like many topics whose names end with "ing," such as machine learning and mountaineering, is simultaneously a problem, a class of solution methods that work well on the problem, and the field that studies this problems and its solution methods."*

*-Richard Sutton*

### 1.1.1 Agent interaction with environment

Reinforcement Learning (RL) starts with an agent that has explicit goals, can interact with an environment and choose to perform actions in the environment thus having an effect on the them. For instance, the charge level of a robot's battery being monitored by an agent. The actions could be sent to the control architecture of the robot by the agent. Framing problems in Reinforcement Learning can be done using Markov Decision Processes (MDPs). The agents perform actions in the environment and the latter in return sends rewards and new states to the agent. This happens continuously. Rewards are numerical values representing the immediate feedback of an action per-

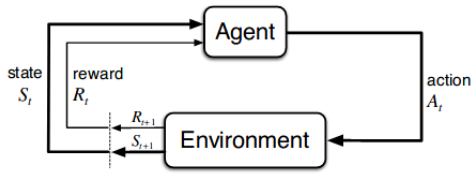


Figure FC1.1: The agent-environment interaction in a MDP

formed and the agent attempts to maximize them over time by choosing better actions. See Figure FC1.1 [1]

### 1.1.2 Markov decision processes

A Markov decision process (MDP) is defined by: [1]

- A set of states $S$
- A set of actions $A$
- Transition probabilities $P$ (define the probability distribution over next states given the current state and action) $P\ (S_{t+1} \mid S_t, A_t\ )$
- Reward signal $R : S \to \mathbb{R}$, mapping states to reward values (real numbers) (rewards can also be defined over state-action pairs; $R : S \times A \to \mathbb{R}$)

$$\boldsymbol{M = (\ S, A, P, R\ )}$$

## 1.2 Deep Reinforcement Learning

Using neural networks as function approximators in RL makes it Deep RL. Recently, this became a very big break-through when Mnih et al., [2] presented the first deep learning model which learnt to play a collection of Atari games by approximating control policies directly from high-dimensional sensory input using reinforcement

learning. Deep RL in a broad sense indicates combining deep learning with RL in order to deal with high-dimensional environments. Similar to the distinct difference between ”shallow” machine learning and deep learning, here also it is the function approximator used. Generally, stochastic gradient descent is utilized to update weight parameters in deep RL. Recent work has achieved stable learning and outstanding results, like deep Q-network (Mnih et al.) [2], and AlphaGo (Silver et al.) [3]

### 1.2.1 Deep Q-Network (DQN)

A note-worthy work is on Deep-Q Network. This paper was published in Nature on 26th February 2015, where the authors combined Deep Neural Networks with RL for the first time at a big scale. Good results were demonstrated on a wide range of Atari 2600 games using just images and score as inputs. This work was a major step forward in the quest for general AI as it represented the first general-purpose agent to be able to continually adapt its behaviour without any human intervention. [4]

## 1.3 Multi-Agent Reinforcement Learning

Artificial Intelligence’s (AI) grand vision since its inception has been to build autonomous agents which can interact with the environment and also amongst each other. A number of complex problems in todays society can be modeled as a Multi-Agent Learning problems, for instance, Multi-Agent Systems (MASs) are of great use in situations involving finding solutions for large tasks because they work through cooperation of individual agents. Agents communicate with each other accompanying interaction with the environment as shown in figure FC1.2. [5]

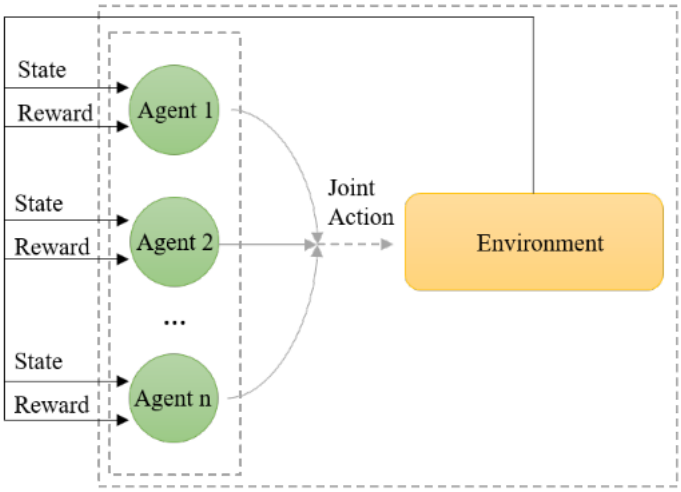


Figure FC1.2: Multiple RL agents in the same environment.

### 1.3.1 Markov Game

In a multi-agent setting, formulating an RL problem can be done using a stochastic game- Markov Game (generalisation of the single agent's MDP). If $n$ denotes the number of agents, $S$ denotes the discrete set of environment states and $A_i$, $i = 1, 2, 3, ..., n$ is the set of actions for each agent, then the following can be defined:

- the joint action set for all agents $A = A_1 \times A_2 \times ... \times A_n$.
- The transition function $p : S \times A \times S \rightarrow [0, 1]$
- the reward function $r : S \times A \times S \rightarrow \mathbb{R}^n$ .
- The value function of each agent (dependent on the joint action and joint policy) $V^{\pi} : S \times A \rightarrow \mathbb{R}^n$ . [5]

## 1.4 Motivation

There are particular challenges in MARL which we wanted to target. They include the problem of non-stationary environments when dealing with an MAS, working with continuous action spaces, complexity in training a large network and the ”lazy” agent problem. All of these have been described in Chapter 2 in brief.

## 1.5 Problem Statement

We developed a Multi-Agent Deep Reinforcement Learning (MADRL) framework that exploits the standard MAS paradigm of *”Centralised Learning and Decentralised Execution”* in Reinforcement Learning to accomplish complex **strategic task planning**, cooperatively.

# CHAPTER 2

# BACKGROUND

This chapter will give some background covering some challenges in MARL and the concept of Centralised Learning and Decentralised Execution.

## 2.1 Challenges in MARL

### 2.1.1 Non-Stationarity

As opposed to the single agent scenario, where the agent is concerned with the outcomes of its own actions only, in a multi-agent scene, the environment is affected by the other agents too, and so the observations made by an agent include the behaviour of other agents also. Learning, in the latter case, is more complex because all agents interact and learn concurrently, and the constant reshaping of the environment due to interactions from multiple agents leads to the non-stationarity problem. Optimal policy of an agent can get adversely affected; the estimated potential rewards of actions can become inaccurate and a good policy at a particular time may not remain so in the future. Non-stationarity in MAS has been studied from various angles along with a variety of assumptions made, making it difficult to maintain an overview of the state-of-the-art. [5] [6]

### 2.1.2 Complexity in Training

Although, policies for complex visual tasks have been successfully learned with DRL, (using approaches like DQN), achieving good performance needs relatively large, task-specific networks and extensive training. This problem is particularly severe for a system of multiple agents. [7] Intuitively, one might consider extending a single agent deep RL to multi-agent environment (like in independent deep q-learning [8]) but that method over-fits and is computationally expensive. [9]

### 2.1.3 Continuous Action Spaces

Most common deep RL models work on discrete spaces only. For instance, Deep Q-Network (DQN) works only on problems with discrete, low-dimensional action spaces (although DQN is fine with high-dimensional observation spaces). Many tasks of interest, especially physical control tasks, have high dimensional continuous action spaces. Since DQN finds the action with maximum action-value, it cannot be applied in continuous domains where an iterative optimization process at every step is desired. [10] One intuitive approach to attack the concerned problem could be discretizing the action space however, this would mean dealing with additional problems like the curse of dimensionality: the exponential increase of action numbers against the number of degrees of freedom.

### 2.1.4 The ”Lazy” Agent Problem

The phenomenon of the ”Lazy” Agent problem occurs particularly in Cooperative settings with a single joint reward signal. The authors of [11] present a simple experiment where they prove that the centralised approach fails by learning inefficient policies. An agent could be discouraged from learning if another agent has already

learned a good policy because then its (i.e., 'lazy' agent's) exploration could lead to a worse team reward.

## 2.2 Centralised Learning and Decentralised Execution

Shifting to a multi-agent scenario from a single agent results in an exponential growth in the joint action space the agents and that is why the reinforcement learning methods designed for single agents do not scale properly on cooperative multi-agent tasks, for instance, the coordination of autonomous vehicles. In such complexities, it is sometimes wise to use decentralised policies where each agent selects its own action conditioned only on its own local observations in the environment. Using decentralised policies will also help in problems involving constraints like partial observability or communication issues. This puts a pressure for development of RL methods that can learn decentralised policies efficiently. Using decentralised policies that are trained centrally is a popular multi-agent deep reinforcement learning paradigm. [12] [5]

# CHAPTER 3

# RELATED WORK

As mentioned earlier, MADRL has a lot of diverse research going on in the direction of solving the aforementioned challenges with each having some advantages and disadvantages. This chapter talks about some of these related works.

**"Deep repeated update Q-network (DRUQN)"** by Castaneda, et al., [13] a DQN variant, was proposed to handle non-stationarity in MAS systems. Based on the repeated update Q-learning (RUQL) [14], DRUQN updates the action value inversely proportionally to the likelihood of an action, thus, not suffering from the policy-bias problem of Q-Learning. **"Deep loosely coupled Q-network (DLCQN)"**, another variant of DQN, propsed by Castaneda et al., [13] relies on LCQN [15] involving specification and adjusting of an independence degree for each of the agents in the MAS using its negative rewards and observations. Another method named **"Lenient DQN (LDQN)"** was proposed by Palmer et al., [16] where they apply leniency to MADRL. Lenient agents map state-action pairs to decaying temparature values which control the amount of leniency applied towards the policy updates that are negative (sampled from the experience replay memories). Thus, a notion of 'optimism' is introduced in the value-function update facilitating cooperation in fully cooperative multi-agent reinforcement learning problems.

**Algorithm 1** Q Actor Critic

Initialize parameters $s, \theta, w$ and learning rates $\alpha_\theta, \alpha_w$; sample $a \sim \pi_\theta(a|s)$.
**for** $t = 1 \dots T$: **do**
  Sample reward $r_t \sim R(s, a)$ and next state $s' \sim P(s'|s, a)$
  Then sample the next action $a' \sim \pi_\theta(a'|s')$
  Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \log \pi_\theta(a|s)$; Compute the correction (TD error) for action-value at time t:
    $\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$
  and use it to update the parameters of Q function:
    $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)$
  Move to a $\leftarrow a'$ and s $\leftarrow s'$
**end for**

Figure FC3.1: Algorithm for Q Actor-Critic approach

Lillicrap, et al., [10] introduced an architecture (**"deep deterministic policy gradient-DDPG"**) which uses an actor-critic approach (see algorithm in figure FC3.1) to handle continuous action spaces. The deterministic policy gradient [17] (which can be defined as the expected gradient of the action-value function) can be estimated much more efficiently than the usual stochastic policy gradient. [10] States are deterministically mapped to specific actions using a parameterized actor function while the critic side learns using DQN. A large number of training episodes are required to find solutions. [5] **"Trust Region Policy Optimization (TRPO)"**, by Schulman, et al., [18] can also be extended to continuous action spaces. Complex non-linear policies implemented as neural networks can be effectively optimized using TRPO. The proposed algorithm is scalable and has strong theoretical foundations. These features assure a great scope for training large and rich function approximators for many challenging problems. Schulman et al., proposed a new family of policy gradient methods (after TRPO) for RL, named **"Proximal Policy Optimization (PPO)"**, [19] which reduced the complexity of implementation of TRPO by converting the KL Divergence constraint to a simple penalty in the loss function. PPO has some of the benefits over TRPO, namely, they have better sample complexity (empirically), are less complex to implement, and are more general.

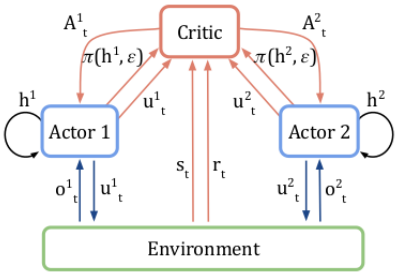


Figure FC3.2: COMA's Overall Architecture

**"Independent reinforcement learning (InRL)"**, one of the simplest form of MARL, is where each agent does not explicitly worry about what other agents are doing to the environment, that is, they treat all of the experience as part of their own environment (non-stationary). But in InRL, policies have been observed to over-fit other agents' policies, thus failing to sufficiently generalize. [9]

A popular approach is ***"Centralised learning and Decentralised execution"***. In this approach, multiple agents are trained simultaneously by applying a centralized learning function and decentralised policies are learned by each agent. Basically, each agent takes actions based on its local observation histories thus having an advantage under conditions of partial observability or in case of compromised communications during actions. **"Counterfactual Multi-Agent Policy Gradients (COMA)"**, by Foerster et al., [12] uses a centralised critic, which is used only during learning. The actor, on the other hand, is needed during execution. The centralised critic conditions on all the state information and the joint action, while each agents policy conditions only on its own action-observation history (refer Figure FC3.2). Lowe et al., proposed **"Multi-Agent Deep Deterministic Policy Gradient (MADDPG)"** [20] based on actor-critic policy gradient algorithms. MADDPG, where they introduced centralised learning with decentralised execution where the critic uses extra information to ease the training process while the actors take actions based on their local observations. Figure FC3.3 shows an overview of multi-agent centralized critic, decentralized actor approach. [20]

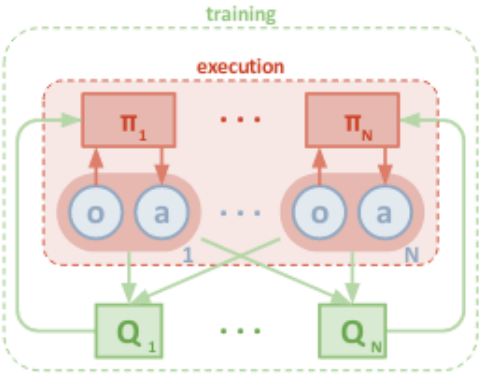


Figure FC3.3: Overview of multi-agent centralized critic, decentralized actor approach

Sunehag et al., [11] tackled the problem of the ”lazy” agent by introducing a **value decomposition network architecture**, in which the team value function is decomposed into agent-wise value functions. The authors empirically showed that learning such value-decompositions gives better results, especially when combined with weight sharing, role information and information channels. (They performed experiments on a wide range of partially observable multi-agent domains).

# CHAPTER 4

# PROPOSED ARCHITECTURE

## 4.1 Environment / Simulation

We used the following environment- **Multi-Agent Particle Environment** [21] which can be best described as by the creators themselves:

*"A simple multi-agent particle world with a continuous observation and discrete action space, along with some basic simulated physics."*

Out of these set of environments, our interests lie in the following one:

**Cooperative navigation:** In this environment, there are **N** agents and **L** landmarks. The task of the agents is to cooperatively reach and cover all the landmarks. The local observations involve factors like their positions, velocities, etc. in the environment. They are rewarded based on their distance from the landmarks. These agents occupy a physical space and are penalized for collisions. [21] Figure FC4.1 shows the looks of the concerned environment. These environments were used in the following cited paper [20].

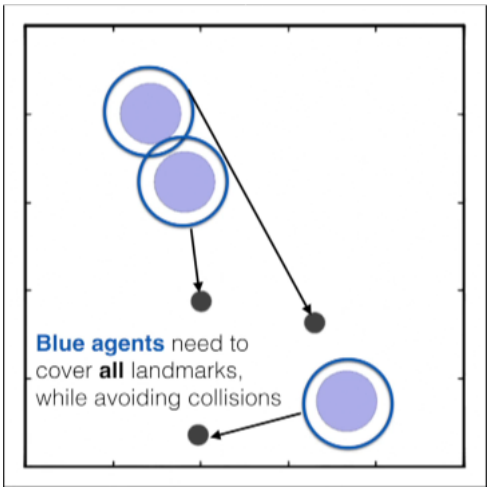


Figure FC4.1: Cooperative Navigation

## 4.2 Architecture

There are two major components of our proposed architecture. Refer Figure FC4.2.

- Central Agent
- MAS of RL agents (A set of individual RL agents)

### 4.2.1 Central Agent

The task of the central agent is to enable centralised learning and therefore make sure to handle the problem of non-stationary environment.

**State:** The input state of the central agent will be the actions of all the individual agents of the environment.

$$S = [\, a_1, a_2, ..., a_n \,]$$

where $a_1$, $a_2$ ... are actions of respective agents and n is the number of agents.

**Action:** This agent would output a vector, which would basically act as the relevant information required by a particular individual agent at a given time-step. The size of this vector is a hyper-parameter.

$$A = [\ v_1,\ v_2,\ ...\ ]$$

**Reward:** The reward for the central agent is the sum of rewards of the individual agents when the take their respective actions in the environment.

$$R = sum\ (\ r_1,\ r_2,\ ...\ r_n\ )$$

### 4.2.2 Individual Agents

The individual agents are part of the MAS which actually exists in the environment and can execute their actions to have an effect on the environment. Their task is to use information from central agent and make a decision (action) considering their local observations too. Although there are multiple agents here, we are defining a single network for the MAS because the agents here are interchangeable, that is, any agent can replace any other agent and do the latter's tasks.

**State:** The state input of this network is the action output from the central agent, the current agent's local observation and the current agent (id).

$$S = [\ v_1,\ v_2,\ ...\ ] + [\ o_1,\ o_2,\ ...\ o_k] + [\ a_i\ ]$$

where k is the size of the local observation vector, $o_k$ is an observation value (local) and $a_i$ is the current agent i.

**Action:** The individual agents' action is a discrete value ranging from 0 to 4.

- 0 → Don't move
- 1 → Move Right
- 2 → Move Left
- 3 → Move Up
- 4 → Move Down

**Reward:** The individual rewards are defined as a function of the landmarks' minimum distances from all the agents. Also, a penalty of 1.0 is given to the total reward in case of two agents colliding.

## 4.3 RL Algorithms

The first component, the central agent, as discussed above, is required to output an action vector of real values which means we are dealing with a continuous action space here. Deep Deterministic Policy Gradients (**"DDPG"**), as discussed in Chapter 3, is a state-of-the-art approach for it. It is an off-policy algorithm and can be understood as being deep Q-learning for continuous action spaces. The second component, on the other hand, is a network outputting an action from a discrete set of actions for each of the individual agents and so we are using Deep Q-Networks (**"DQN"**) for our experiments. DQNs learn beautifully over continuous state spaces/discrete action spaces as already discussed in Chapter 1. The algorithm uses a combination of two important techniques-experience replay and target networks, which give excellent results using just an input of raw real values. A new class of policy gradient algorithms introduced by Schulman et al., namely, "TRPO" [18] and "PPO" [19] can also be used for any of the above components. 3

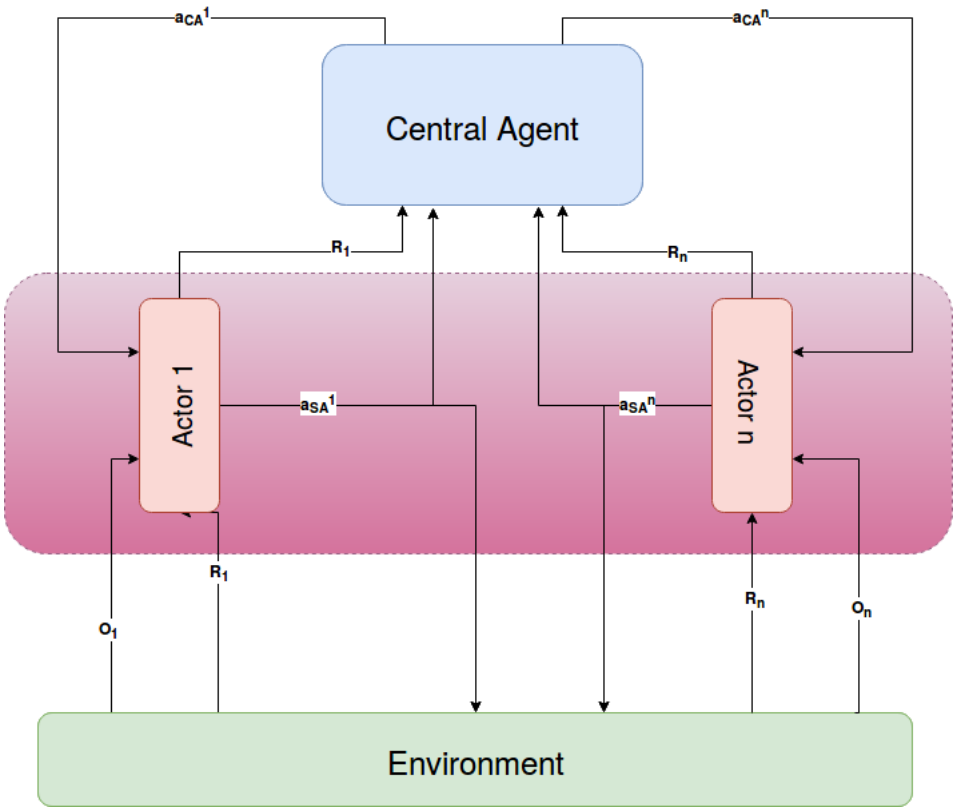


Figure FC4.2: The Proposed Framework

## 4.4 Workflow

The workflow of the framework is as follows:

First the Central Agent looks at the actions of all the individual agents and then makes a decision of what action vector it must output for the first individual agent. Then the first individual agent takes in this vector input along with its local observation and makes a decision of from its discrete action space. After performing its action on the environment, it receives an immediate reward which it utilises to take better actions in future. The same reward is forwarded to the Central Agent also, for it to analyse its action given the individual agents's actions in the previous time step. Now its time for the second individual agent to take an action in the environment and so it

asks the CA to take an action and send an action vector as a help. Similarly, the whole queue of individual agents keeps taking actions and the models keep updating (or rather improving).

# CHAPTER 5

# EXPERIMENTS AND RESULTS

## 5.1 Hyper-Parameters

Some of the hyper-parameters used in the architecture are tabulated in table TC5.1 :

Table TC5.1: Hyper-parameters

| Variable | Value |
|---|---|
| Number of Episodes | 30000 |
| Maximum Episode Length | 200 |
| Learning Rate (Adam's Optimizer) | 0.001 |
| Replay Memory Buffer size | $10^6$ |
| Batch Size | 20 |
| Exploration Decay Rate | 0.995 |
| Gamma | 0.95 |
| Central Agent Action Space | A=5 |
| Central Agent Action Bound | 5.0 |
| Individual Agent State Space | A + O(=18) + 1 |
| Individual Agent Action Space | 1 {0-4} |

## 5.2 Experiments

We used Adam's Optimizer in all the experiments. The learning rate was set to 0.001 and $\gamma$ was set to 0.95. The replay memory buffer size was $10^6$ and before making an

update, a batch of 20 episodes was formed. The agent learned the task in 1284 episodes and the screenshots of the output for the case in Table TC5.2 are as in Figures FC5.1 and FC5.2.

Table TC5.2: Experiment 1

| **Agents** | **Landmarks** |
|---|---|
| N = 2 | L = 2 |

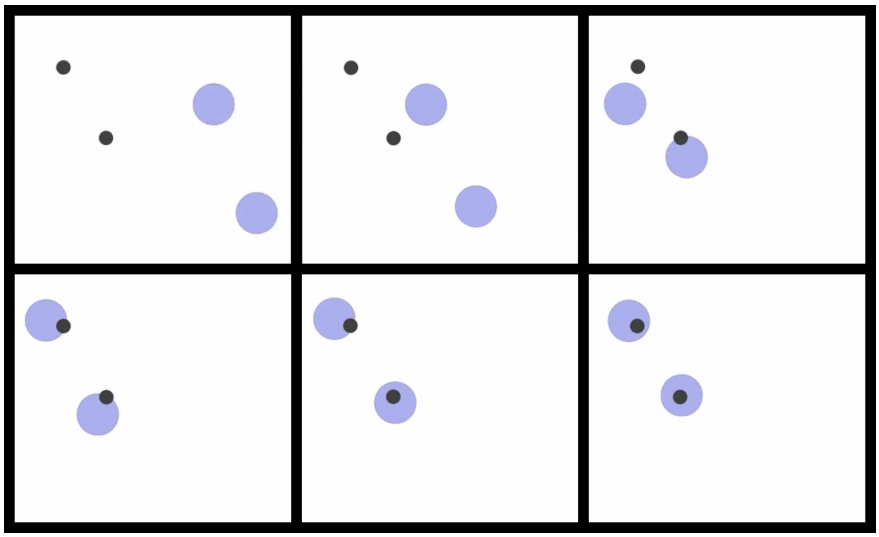

Figure FC5.1: Sample Episode 1

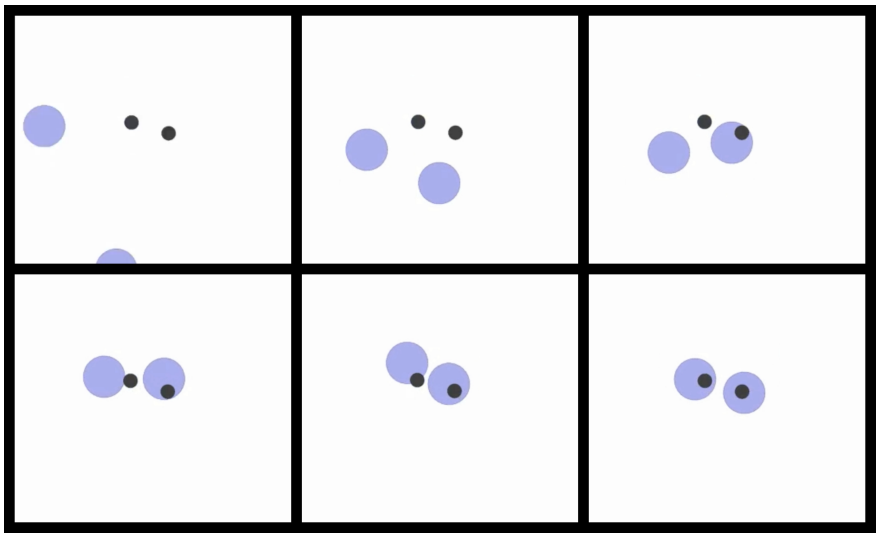

Figure FC5.2: Sample Episode 2

The screenshots of the output for the case in Table TC5.3 are as in Figures FC5.3.

Table TC5.3: Experiment 2

| **Agents** | **Landmarks** |
|---|---|
| N = 5 | L = 5 |

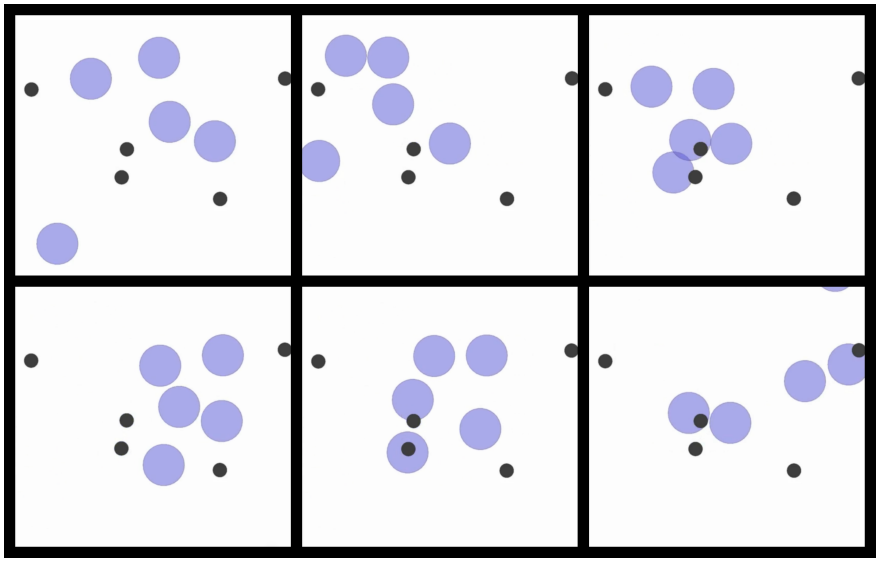

Figure FC5.3: Sample Episode

Table TC5.4 and Table TC5.5 show some observed and calculated measures of the experiments.

Table TC5.4: Experimental Results

| **N** | **L** | **Solved in (Runs)** | **Average distance** |
|---|---|---|---|
| N=2 | L=2 | 1284 | 0.273 |
| N=5 | L=5 | 1569 (stopped) | 4.437 |

Table TC5.5: Average number of episodes per episode

| **N** | **L** | **# collisions** |
|---|---|---|
| N=2 | L=2 | 0.11 |
| N=5 | L=5 | 1.767 |

# CHAPTER 6

# APPLICATION IN WAREHOUSE LOGISTICS

Deep Learning has been used in MARL too, since the success of DQN. [4] A few illustrative examples are given in Table TC6.1 . [5]

Table TC6.1: Typical MADRL Applications

| Application | Algorithm | References |
|---|---|---|
| Traffic lights control | DDDQN and IDQN | [22] |
| Swarm systems | DQN and DDPG | [23] |
| Large-scale fleet management | Actor-Critic and DQN | [24] |
| Federated control | Hierarchical-DQN (h-DQN) | [25] [26] |
| Task and resources allocation | CommNet | [27] [28] |

In this chapter, we describe an extension of our framework in Warehouse Logistics.

Briefly, in a self-sustained large scale warehouse, the daily routine would be something like as follows:

- There would be a queue of customer orders getting lined up.
- Products are kept in various racks (shelves) spread across the warehouse.
- Manual labour is procuring the products as and when required.
- There is a supply buffer room in the warehouse where all the supply for future has been dumped, ready to be placed on appropriate racks.

- If the warehouse is automated, then there would be robots roaming around inside, picking up or dropping off products.
- The overall objective of the warehouse would be to fulfill all the customer orders and possibly utilizing minimum resources in terms of energy and time.
- The warehouse must forecast the demands of the products appropriately and to save time and energy during the procurement of products from their racks could be achieved by an optimal placement of them in racks.

## 6.1 Case Study: Amazon Warehouses

Amazon is arguably the most prominent e-commerce provider and online retailer in the world today. They follow the following techniques in their warehouses as of today:

**Chaotic Storage:** Chaotic storage is a modified form of random storage in which incoming items are assigned randomly to available space, i.e., in chaotic storage different products can be stored within the same location or bin. This provides greater flexibility and higher space utilization than regular random storage assignment. However it adds complexity in that the pick worker must not only find the correct location or bin but then must search within the bin for the correct item. Amazon is using the concept of chaotic storage currently at its fulfilment centers. The video of the process of storing in amazon warehouses can be viewed in the following link: **Chaotic Storage Video**.

**Amazon Bots:** Kiva Systems, were acquired by Amazon to bring in robots into their warehouse facilities (in 2012). The result was an increase in robotic services, robotic-human shared workspaces, and spearheading the collaborative environment for other companies every following year. These robots move the whole rack and bring it near an


Figure FC6.1: A picture of an Amazon Robot

output gate where the required product is manually picked up. So, Amazon proposed a system involving cooperation of humans and robots. The following video shows how Amazon Robots work in their warehouse: **Amazon Robots Video**

## 6.2 Extension of our framework

We define three components of the Automated Warehouse to manage the whole logistics scene (Refer Figure FC6.2):

- The **Central** Agent (CA)
- The **Staff** Agents (the MAS of individual agents inside the warehouse) (SAs)
- The **Warehouse Manager** Agent (WMA)

The CA will be incharge of handling the customer's orders along with managing the group of SAs in the warehouse. Basically, it will take the list of orders as an input in addition to the SAs' actions at a given time. The CA will output a relevant action vector as an additional information given to the SAs for them to make a decision of what

product they need to worry about, given the condition of the environment (as given to it by the CA). The WMA will take care of what action should an SA practically do, i.e., fetch a product from the supply buffer and STORE it in a particular rack or GET a product from a rack and try to fulfil an order. The MDPs of these agents can be formed as follows:

***CA:***

- **State**: Current Customer's orders + Actions of all SAs
- **Action**: A Vector (of info for an SA)
- **Reward**: Based on order fulfilled or not

***SA:***

- **State**: CA's vector + Local Observation + Agent (SA) (id)
- **Action**: Product Number
- **Reward**: Based on order fulfilled or not

***WMA:***

- **State**: Rack-Product Configuration + Agent (SA)
- **Action**: GET / STORE FROM (rack number)
- **Reward**: Based on order fulfilled or not

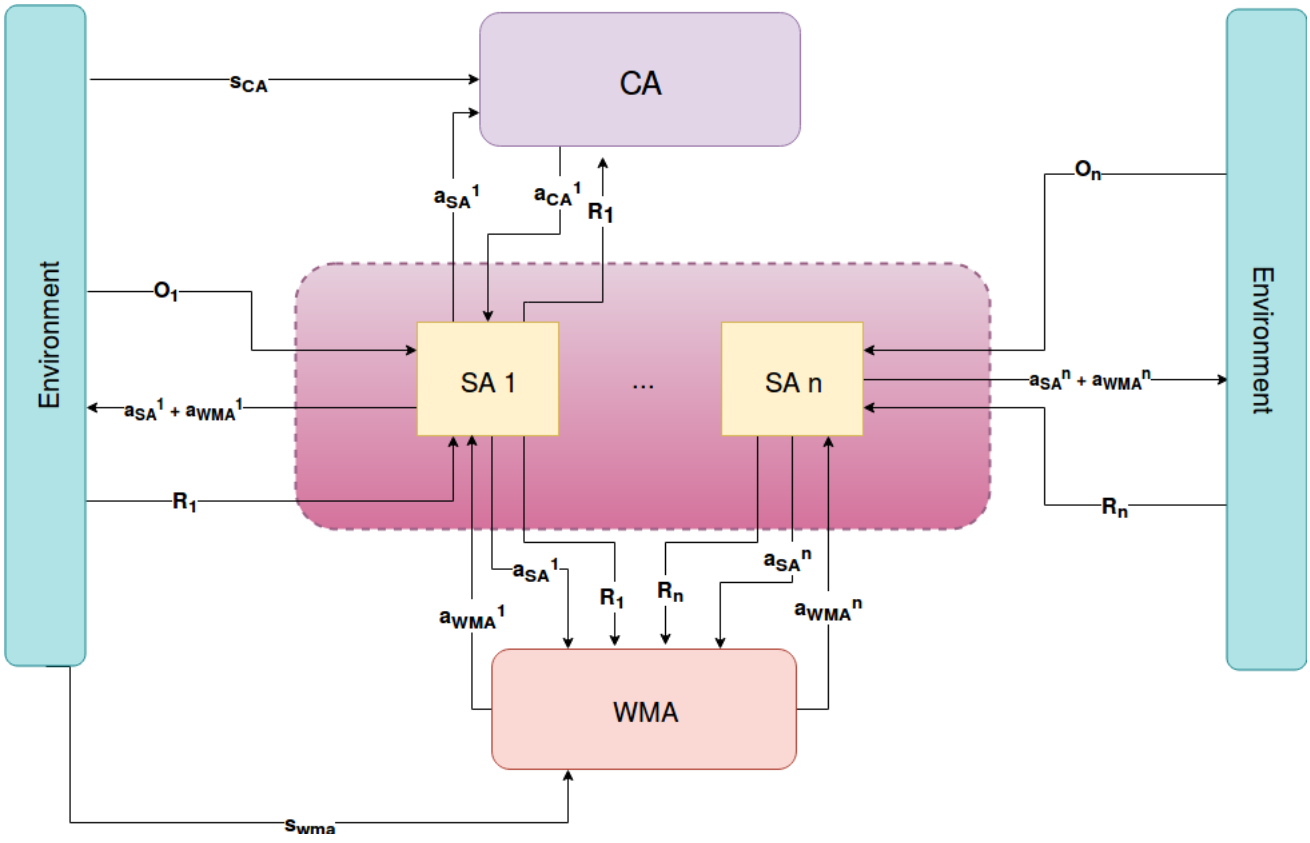


Figure FC6.2: The Framework for the Warehouse Logistics Application

## 6.3 Workflow

The flow is as follows - the central agent will gather information about the current orders and input the previous actions of all the SAs, and then output an action vector for current SA (seeking information for taking an action in current time step) to use. This SA network will take in CA's vector along with the local information of the current SA and the identification information of this current SA, to take an action of what product this SA must consider to fetch. WMA takes this action of the SA as its state input along with the information about the availability of the product in racks or in the supply buffer and accordingly outputs an action of GET or STORE and FROM. The GET FROM action means that the SA is commanded to fetch the particular product from the outputted rack number and give it at the output counter. And, the STORE FROM action maps to the command of getting the required product from the supply buffer and storing it in the outputted rack number for future orders. Here, intuitively, the WMA takes care of demand forecasting and optimal warehouse product placements. Note that, the reward for each of these components will be received only after one full iteration of taking actions.

# CHAPTER 7

# CONCLUSIONS

A multi-agent reinforcement learning framework has been proposed in this thesis where an agent learns what all the agents in the environment are doing and accordingly alerts the individual agents affecting their decisions in a given state situation, thus avoiding the problem of non-stationarity in MAS, easing the training to some extent, dealing with large scale continuous action spaces and also avoids the 'lazy' agent problem empirically. Further analysis and improvement of results can be done by conducting a few more experiments and maybe by figuring out the perfect set of hyper-parameters too. (There are many hyper-parameters involved, especially in the first component- the central agent.) The architecture seems to work nicely for a low number of agents and landmarks, but may not scale well for a very large number. We leave this investigation to future work along with the aim to develop variants that require fewer samples and are scalable for practical applications like autonomous navigation.

# Bibliography

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2017.

[2] Volodymyr Mnih and Koray Kavukcuoglu. Playing atari with deep reinforcement learning. In *arxiv*, 2013.

[3] David Silver, Aja Huang, Chris J. Maddison, and Arthur Guez et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489.

[4] Volodymyr Mnih, Koray Kavukcuoglu, and David Silver et al. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[5] Nguyen, Thanh Thi, Ngoc Duy Nguyen, and Saeid Nahavandi. Deep reinforcement learning for multi-agent systems: A review of challenges, solutions and applications. In *arxiv*, 2018.

[6] Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag, and Enrique Munoz de Cote. A survey of learning in multiagent environments: Dealing with non-stationarity. In *arxiv*, 2017.

[7] Andrei A. Rusu, Sergio Gomez Colmenarejo, and Caglar Gulcehre et al. Policy distillation. In *arxiv*, 2015.

[8] Ardi Tampuu and Tambet Matiisen et al. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4), 2017.

[9] Lanctot and Marc et al. A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4190–4203, 2017.

[10] Timothy P. Lillicrap, Jonathan J. Hunt, and Alexander Pritzel et al. Continuous control with deep reinforcement learning. In *arxiv*, 2015.

[11] Sunehag and Peter et al. Value-decomposition networks for cooperative multi-agent learning. In *arxiv*, 2017.

[12] Foerster and Jakob N. et al. Counterfactual multi-agent policy gradients. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[13] Castaneda and Alvaro Ovalle. Deep reinforcement learning variants of multi-agent learning algorithms. In *Master's thesis, School of Informatics, University of Edinburgh*, 2016.

[14] Abdallah, Sherief, and Michael Kaisers. Addressing the policy-bias of q-learning by repeating updates. In *International conference on Autonomous agents and multi-agent systems*, pages 1045–1052, 2013.

[15] Yu and Chao et al. Emotional multiagent reinforcement learning in spatial social dilemmas. In *IEEE transactions on neural networks and learning systems*, pages 3083–3096, 2015.

[16] Palmer and Gregory et al. Lenient multi-agent deep reinforcement learning. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 443–451, 2018.

[17] Silver and David et al. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, 2014.

[18] Schulman and John et al. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[19] Schulman and John et al. Proximal policy optimization. In *arxiv*, 2017.

[20] Lowe, R., and Wu et al. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017.

[21] Lowe, Ryan, and Wu et al. Multi-agent actor-critic for mixed cooperative-competitive environments. *Neural Information Processing Systems (NIPS)*, pages 6379–6390, 2017.

[22] Calvo et al. Heterogeneous multi-agent deep reinforcement learning for traffic lights control. In *The 26th Irish Conference on Artificial Intelligence and Cognitive Science*, pages 2–13, 2018.

[23] Huttenrauch, Sosic, and Neumann. Guided deep reinforcement learning for swarm systems. In *arXiv*, 2017.

[24] Lin, Zhao, Xu, and Zhou. Efficient large-scale fleet management via multi-agent deep reinforcement learning. In *arXiv*, 2018.

[25] Kulkarni, Narasimhan, Saeedi, and Tenenbaum. Hierarchical deep reinforcement learning: integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 3675–3683, 2016.

[26] Shah P. Hakkani-Tur D. Kumar, S. and L. Heck. Federated control with hierarchical multi-agent deep reinforcement learning. In *arXiv*, 2017.

[27] Gharbi A. Noureddine, D. and S. Ahmed. Multi-agent deep reinforcement learning for task allocation in dynamic environment. In *Proceedings of the 12th International Conference on Software Technologies (ICSOFT)*, pages 17–26, 2017.

[28] Szlam A. Sukhbaatar, S. and R. Fergus. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, pages 2244–2252, 2016.