Technical University of Crete

School of Electrical & Computer Engineering

Diploma Thesis

Deep Reinforcement Learning for Multi-Agent Search and Rescue Operations

Author:
Chanialakis Theofilos

Thesis Committee:
Associate Prof. Chalkiadakis Georgios

Associate Prof. Samoladas Vasilios

Associate Prof. Partsinevelos Panagiotis

A thesis submitted in partial fulfillment of the requirements for the degree of Diploma in Electrical and Computer Engineering

September 2020

Περίληψη

Οι περιπτώσεις έκτακτης ανάγκης, όπως οι φυσικές καταστροφές, αποτελούν ένα από τα πιο σημαντικά προβλήματα της σύγχρονης κοινωνίας καθώς απαιτούν προετοιμασία ώστε να προστατευθεί το σύνολο του πληθυσμού, όσο καλύτερα γίνεται.

Η προετοιμασία και οι προπαρασκευαστικές ενέργειες, στις περισσότερες περιπτώσεις δεν επαρκούν, καθιστώντας αναγκαία την άμεση δράση υπηρεσιών που ειδικεύονται στην αντιμετώπιση καταστάσεων έκτακτης ανάγκης, όπως Πυροσβεστική, κινούμενες νοσοκομειακές μονάδες κ.α.. Η ομαδική δράση και η συνεργασία μεταξύ των υπηρεσιών αυτών είναι απαραίτητα στοιχεία για αποστολές Έρευνας και Διάσωσης. Η καθολική γνώση των γεγονόντων και η δυνατότητα αξιολόγησης της κατάστασης, είναι πολύ σημαντικά κομμάτια για τη βέλτιστη διαχείρισης της κρίσης. Μια σωστή και γρήγορη απόφαση μπορεί να σώσει ζωές.

Στα πλαίσια αυτής της διπλωματικής εργασίας δημιουργήθηκε ένα σύστημα διαχείρισης δυναμικού για αποστολές Έρευνας και Διάσωσης σε καταστάσεις έκτακτης ανάγκης. Το σύστημα αποτελείται από δύο κομμάτια, τα οποία είναι εξίσου σημαντικά και άρρηκτα συνδεδεμένα μεταξύ τους. Το πρώτο κομμάτι περιλαμβάνει τη συλλογή δεδομένων και τη ζωντανή ενημέρωση μεταβαλλόμενων παραμέτρων της κατάστασης. Το δεύτερο κομμάτι αφορά τη λήψη αποφάσεων και την ανάθεση εργασιών στο διαθέσιμο προσωπικό ώστε να ελαχιστοποιηθεί ο κίνδυνος. Στο κείμενο της διπλωματικής μας εργασίας, αναλύουμε λεπτομερώς τη λειτουργικότητα του συστήματος και τις τεχνολογίες που χρησιμοποιούνται για να λειτουργεί το σύστημα με συνέπεια και αξιοπιστία.

Το σύστημα δέχεται έναν ή περισσότερους διαχειριστές που μπορούν να σημαδεύσουν περιοχές που χρήζουν προσοχής. Η διεπαφή των διαχειριστών με το σύστημα γίνεται μέσω διαδικτυακής σελίδας, με τη χρήση χάρτη και πρόσθετων γραφικών για τη διευκόλυνση της διαχείρισης. Στα χωρικά δεδομένα που εμφανίζονται στο χάρτη, προστίθενται και οι θέσεις του διαθέσιμου δυναμικού, οι οποίες γνωστοποιούνται μέσω εφαρμογής που αναπτύχθηκε για κινητά τηλέφωνα.

Λαμβάνοντας υπόψιν του τις παραπάνω παραμέτρους, το σύστημα παίρνει αποφάσεις για τις ενέργειες που πρέπει να κάνει κάθε ομάδα. Η λήψη αποφάσεων γίνεται μέσω Μηχανικής Μάθησης σε Πολυπρακτορικά Συστήματα. Γίνεται χρήση αλγορίθμων Ενισχυτικής Μάθησης και αρχιτεκτονικής Βαθιών Νευρονικών Δικτύων ώστε η ενέργειες που θα επιλεχθούν να αποτελούν τις βέλτιστες και οι αναθέσεις εργασιών να είναι όσον δυνατόν πιο αποδοτικές. Η Βαθιά Ενισχυτική Μάθηση θεωρείται υπερσύγχρονη τεχνολογία και είναι ενδιαφέρον να εξετάσουμε τη χρήση της σε Πολυπρακτορικά περιβάλλοντα με μεγάλη πολυπλοκότητα. Στην εργασίας μας, προτείνουμε μια καινοτόμα αρχιτεκτονική Βαθιάς Ενισχυτικής Μάθησης για Πολυπρακτορικά περιβάλλοντα, δίνοντας λύσεις σε πολλά προβλήματα που παρουσιάζονται στο τομέα της Μηχανικής Μάθησης. Τέλος, τα βασισμένα σε προσομοιώσεις πειραματικά μας αποτελέσματα αποδεικνύουν ότι το σύστημα διαθέτει όντως την ικανότητα μάθησης του σε ρεαλιστικά σενάρια, παράγοντας πολυπρακτορικά πλάνα δράσης με προοδευτικά όλο και μεγαλύτερη αξία.

Abstract

Emergency situations, like natural disasters, can cause significant problems to our society so they require preparatory actions and immediate response to protect the population to the best of our abilities. Many groups and organizations have been established to aid in Search and Rescue and Emergency Response (ER) operations.

Preparation and preparatory actions, in most cases, are not enough, so it is vital that many agencies and groups, which are specialized in ER situations, like firemen and medics, take immediate action. Collective actions and collaboration, among those groups, are essential components for Search and Rescue operations. Global knowledge of the events and the ability to evaluate the situation are major pieces in ER management. A good and quick decision can save many lives.

In this thesis, we develop an administration system for Search and Rescue operations in ER situations. The system consists of two equally important and inextricable connected parts. The first part consists of the data collection and the live parameter updates. The second part pertain to decision making and task allocation to the work force in order to minimize the danger. Moreover, we provide a detailed analysis of the system's functionality and of the technologies that are responsible for the system's consistency.

The system can be used by two or more administrators, simultaneously, who can markup regions which need attention. The interface is a web-page with the use of augmented map and additional graphics to help with the system handling. The positions of the work-forces groups have been added and are updated frequently to the spatial data of the map. These live updates are possible due to an app which we developed for smartphones.

Decision making procedure makes use of the above information and allocate tasks to every group. Machine Learning algorithms in Multi-Agent Systems/Environments are added in the system in order to make better decisions. In particular, Reinforcement Learning and Deep Neural Network architectures are combined to make sure that the actions are near optimal and the task allocation is the most efficient. Deep Reinforcement Learning is a state-of-the-art technique and it is very interesting to explore how it could be used in Multi-Agent environments with high complexity. In this thesis, we propose a novel Deep Reinforcement Learning architecture in Multi-Agent Settings, giving solutions to many problems which Machine Learning has difficulty to handle. We also provide experimental results, which indicate that the system gradually learns in realistic situations, generating meaningful action plans for all the agents.

Acknowledgments

I would like to express my gratitude to the people who supported me during the whole period of my studies at the Technical University of Crete.

First of all, I would like to thank my advisors, *Prof. Georgios Chalkiadakis* and *Prof. Vassilis Samoladas*, for their advising and continuous guidance, not only on this thesis, but on their courses which I attended as well. I am also grateful to the third member of the committee *Prof. Panagiotis Partsinevelos*, for his kindness, his interest in me when I joined *SenseLab* at 2016, where I had my first contact with academic research and global competitions, and of course his support at any endeavor of mine.

I want also to thank every classmate who I collaborated with, for various class projects over the years in Technical University of Crete. It was a great pleasure of mine to work alongside with such special people.

Last but not least, I am deeply thankful to the members of my family and my friends, who have always been there for me and supported me at all times.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In every emergent disaster, either natural or man-made, the human society faces, the attempt is to contain its societal impact to the minimum degree possible. Emergency response (ER) agencies are responsible for managing these situations and minimizing the losses. Preparatory actions for the population safety are very important. At the same time, quick and precise discussions about the course of action to take in an ER situation, can also be very beneficial. Gaining situational awareness is vital for the disaster management and the task allocation for both emergency responder and search and rescue teams. The people who are in charge of decision making on situations like that must be highly trained and fully prepared for ER situations in order to rise to the challenge, evaluate and handle the situation perfectly. Many challenges can emerge regarding the task allocation of the available forces, on account of this, the field of Artificial Intelligence can provide techniques to manage the situation quick and efficient. Many lives could have been saved if there we could eliminate the miscommunication and misunderstanding between agencies and emergency reponders teams, or even provide a way to minimize the environmental uncertainty.

Search and rescue operations dictate perfect timing, great cooperation and flawless management in order to handle the ER situation. Many systems have been developed to aid these endeavors in real-time. *Artificial Intelligence*, mainly via *Machine Learning* (ML), can provide near optimal solutions and generate several suggestions regarding the possible action plans for every team. ML can help the operation's directors with important metrics about the danger of the situation, can aid their assessment of the potential benefits of agents' cooperation, or even allocate the tasks among all available agent.

Deep Reinforcement Learning (Deep RL), in particular, is a powerful tool which can be very helpful in systems that manage such complicated, challenging and demanding situations. It combines two of the most important techniques in ML, Deep Learning and Reinforcement Learning, and can handle high dimension data. There are many algorithms that have been developed which can be modified to offer enormous aid on search and rescue operations. Modern frameworks provide the ability to build custom environments, based on an abstract template; construct Deep Q-Network architecture model with high-level commands, temper with its parameters and manipulate the data in various ways.

1.1 Motivation

Many disasters that occur every year around the globe could be rendered catastrophic, if we are ill-prepared to confront the challenges these situations pose. As human species, we developed through cooperation. Science and technology both play important roles regarding the quality of our lives. Thus, it is conceivably our duty to utilize our technological growth on ER situation and search and rescue operations for collective benefit. Given this, our motivation in this thesis is to develop a disaster response aid system which could be used by everyone and can help everyone. Our goal was that it would be easy to use, providing intuitive features, such as an augmented map, and support quick functionality usage.

There are many requirements to this end. The system must be stable, robust, consistent and concurrent. Several techniques and algorithms, from the distributed systems field, must be combined for the system's development. A variety of advanced features has to be used in order to develop a fully functional computing server to host the system. Web and mobile application are essential for the system's functionality, making it more user friendly.

A decision making component which will analyse, evaluate and produce action plans, is also vital for this ER and search and rescue system. Machine learning is a key component to guarantee the effectiveness and even optimality of the plans. Due to the large quantity and high dimensionality of the data, deep reinforcement learning is the best and most advanced technique to build upon.

1.2 Contributions

In this thesis we provide a novel ER management system which addresses the challenges that we face on major disasters, such as earthquakes, fires or floods. This system utilizes many state-of-the-art techniques in order to be robust and consistent on every functionality it provides. It is built online, on a remote server and accessible via a web-page and a Android app that we developed. The combination of a web application and a mobile application, provides a user-friendly infrastructure for quick and easy data gathering, which is vital for the machine learning system in order to produce better and near optimal decisions about the action plan. The system uses a ReST API, which is the building block, and combined with Geographic Information Systems, it can fully support the needs regarding the spatial data manipulation.

In our implementation, we populate it with a state-of-the-art Deep RL algorithm, specifically Deep QN [32],which we modified to be able to cope with combinatorial explosion in MAS environments such as ours. The environments that we use are constructed in real-time, and are fully compatible with the Deep RL algorithm that we developed.

In particular, we use a Deep Q-Network (DQN) to cope with the decision making needs of this ER system. A typical DQN is used on single-agent settings. There are many differences regarding a single-agent environment and a multi-agent one. Due to the large state vectors that describe each state of a multi-agent environment, instead of using a typical sequential model for the neural network on the DQN, we leverage the functional model which provides the capability of multi-input layer on the neural network. This is essential for the input size and dimensionality reduction that we need for effectively managing ER situations. On top of that, we use a word embedding layer on the input vectors in order to reproject the data to fewer dimensions and reduce the input data size even more.

Finally, the data that we use for real-time ER situation management is still too large for a low-end system to handle. We would need a very strong infrastructure to process these quantities of data at real-time speed. Since we did not have this computational power at our disposal, we propose a pre-training procedure for some DQN layers in order to learn a variety of ER situations and speed up the real-time weight training of our network.

Chapter 2

Theoretical Background

In this chapter we will discuss web applications, web services and RESTful API management. We will provide a detailed introduction to machine learning, reinforcement learning (RL), deep neural networks (DNN) and learning in multi-agent settings.

Firstly, we will analyse web application environments and the utility of each component on a typical web-app. The analysis includes the services that are provided, the intercommunication architectures and protocols, and the management of application's information, such as spatial data.

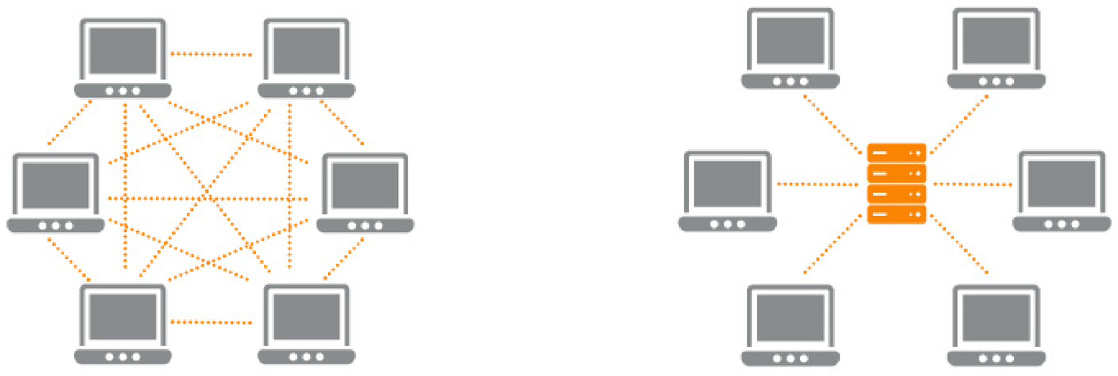
Finally, we will provide background on RL and DNN, and how the combination of those braches of machine learning can be very prospering. We will introduce deep reinforcement learning in multi-agent environments and analyse some of the primary concerns on this field. Following that, we will discuss the state of the art techniques on multi-agent system learning.

2.1 Web Applications

A web application is a computer software program class. A **web-app** runs on a web server instead of the traditional method, in which desktop applications are executed by client's operating system. It is a collection of html pages, servlets, classes and other resources that can accessed through a web browser. There are several advantages on this type of programming, such as platform independence and computation decoupling. The modern development of these software types built Cloud Computing and Software-as-a-Service(SaaS).

A web app includes many components in order to provide a service. Every technology can be shared via the Internet and grant access to many users at the same time. There are many modern techniques to achieve this. The main and most important parts are the following:

- Services that it provides, see 2.1.1
- API to communicate between components, see 2.1.4
- Data servers and clusters to store and handle information



(a) P2P networking (b) Client-Server networking

Figure 2.1: Web services architecture types

2.1.1 Web Services

The term web services refers to a service that a device provides to another one through communication via the WWW (World Wide Web). Web apps are usually composed of only one service type, but they can use a combination of service types, depending on the development needs. There are 2 web apps service types:

- **Peer-to-Peer/P2P**

  P2P networking is an architecture style for distributed application development. Tasks and workloads are partitioned among peers. Peers have the same privileges and responsibilities [1]. Sharing files via torrents is one application of this service type. Blockchain technologies [23] and digital cryptocurrencies are typically operating over peer-to-peer networks [9]. An ordinary P2P network is showed at 2.1a.

- **Client-Server**

  This centralized networking is the architecture style for application development in the majority of web apps. Tasks and workloads are processed exclusively by the server(s). Clients do not interact directly with each other, since the server handles all the information exchange. An ordinary Client-Server network is showed at 2.1b.

2.1.2 Multitier architecture

In software engineering, multilayered architecture, multitier architecture or n-tier architecture is a Client-Server architecture. A typical and widespread multitier architecture is the three-tier architecture in which user interface, business logic and data management are separated. The disjunction of these tiers or layers making capable to develop, maintain and upgrade each tier in isolation to the others. The major challenge in this architecture is the interconnection protocols deployment. Many protocols are available to deploy appropriately to the requirements, e.g. CORBA, SNMP, UDP communication.

We focus on the analysis of the three-tier architecture as it is the fundamental building block of our Web system. The three-tier architecture contains:

1. **Client Tier**: This tier includes the presentation layer, as commonly known, the User Interface(UI). It is built mainly on markup and web languages like JavaScript, CSS, HTML. On this layer, they are developed the functionalities of AJAX [14] and asynchronous request handling. The UI should be developed with respect to the following characteristics: clarity, concision, familiarity, responsiveness, consistency, aesthetics and efficiency.

2. **Middle tier**: This tier encapsulate the business logic of the application. It is considered as a web server which coordinates the application, processes commands, makes logical decisions and evaluations. Is also handles all the clients' requests and data exchange between layers. This layer includes any intercommunication API on the application. In this thesis we developed an RESTful API, so our implementation on this tier conforms with REST constrains as well, see 2.1.4.

3. **Data Tier**: This tier is the data access layer. It contains the data persistence mechanisms of the application. Databases and data servers/clusters are parts of this tier, so the biggest concerns are scalability, availability, redundancy, consistency, backup and database administration.

2.1.3 The Golang programming language

The Golang or Go programming language was conceived in late 2007 to aid the software development infrastructure at Google. Go developed at Google and is a compiled, garbage-collected, concurrent, statically typed language. It is an open source project. Golang is efficient, productive, and scalable. Its qualities compose a language that is not a breakthrough research language but is nonetheless an excellent tool for engineering large software projects.

Golang was a building block on the development of the RESTful API that webapp is based on. The selection of the Go language over other popular languages for RESTful API development, such as Python, was made in order to gain experience on this language too.

2.1.4 Representational State Transfer

Representational state transfer (**REST**) is a software architectural style that defines a set of constraints to be used for creating Web services. Web services that were developed with the REST architectural style, called **RESTful** Web services, provide interoperability between computer systems on the Internet. RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of stateless operations.

Web resources were first defined on the World Wide Web as files or documents identified by their URLs.In a RESTful Web service, requests made to a resource's URI will get a response with a payload formatted in JSON, XML or some other format. By using a stateless protocol and standard operations, RESTful systems aim for fast performance, reliability, and the ability to grow by reusing components that can be managed and updated without affecting the system as a whole, even while it is running.

The representational state transfer term was introduced and defined by Roy Fielding in 2000 in his doctoral dissertation[11]. Fielding's dissertation explained the REST principles that were known as the "HTTP object model" beginning in 1994, and were used in designing the HTTP 1.1[10] and Uniform Resource Identifiers (URI) standards[3]. The term evokes an image of how a well-designed Web application behaves: it is a network of Web resources where the user progresses through the application by selecting resource identifiers and resource operations such as GET or POST, resulting in the next resource's representation being transferred to the end user for their use.

In RESTful API, a web service must adhere to the following six REST architectural constraints:

1. **Use of a uniform interface**. Resources should be uniquely identifiable by a single URL, and a resource can be manipulated only by using the underlying CRUD methods of the network protocol[21].

2. **Client-server based**. There should be a clear delineation between the client and server. Data access, workload management and security are the server's domain. Uniform Interface and request-gathering concerns are the client's domain.

3. **Stateless operations**. All client-server operations should be stateless. Server does not store state information for the clients. Any state management that is required should take place on the client, not the server.

4. **RESTful resource caching**. Unless explicitly indicated that caching is not possible, all resources should allow caching.

5. **Layered system**. REST allows for an architecture composed of multiple layers of servers. So the client cannot know if it is connected directly to an end server or an intermediate one. Multiple layers of servers are used to improve systems scalability and manage load balancing.

6. **Code on demand**. Every time a server will send back static representations of resources in the form of XML or JSON. However, when it is necessary, servers can send executable code to the client.

2.2 Geographic Information System

A Geographic Information System(**GIS**) is "a computer based information system for the storage and manipulation of map-based land" as the father of GIS, Roger Tomlinson noted in the year 1968, in his paper[34] that posited the Geo-Information Systems.

Contemporary GIS technologies are far more advanced than those back in 1968, nevertheless the concept of labeling data points on maps and creating augmented maps will always be very intriguing. Spatial analysis is even older than the GIS concept and many people through the last decades have used it for topography or even epidemiology.

Modern geoprocessing methods have been developed to manipulate spatial data. Many processing operations can handle datasets, analyse them and discover hidden patterns, apply supervised and unsupervised learning algorithms, such as feature selection, use topologies and raster data in order to create new layers. Many GIS frameworks and spatial databases have been developed, with a vast variety of features. In this thesis we use a *PostGIS database* to handle and optimize storing and querying spatial data which our system utilizes.

2.2.1 PostGIS

PostGIS is a spatial database build on PostgreSQL object-relational database. It provides support for geographic objects and allows location queries to be run in SQL. PostGIS adds various data types to the PostgreSQL DBMS, such as geometry or raster, in order to function as a spatial database. It also adds operations, index enhancements and functions. Some of the main features that PostGIS provides are:

- Typical spatial functions, like Intersection, ConvexHull etc.
- Spatial reprojection for vector and raster data types.
- Handling ESRI shapefiles from commandline, GUI and 3rd-party tools
- Support for GeoTiff, GeoJson, WKT, KML and other formats

2.3 Machine Learning

Machine learning is an branch of artificial intelligence (AI) that provides systems the ability to automatically learn, behave and improve from experience without being explicitly programmed to do so. Machine learning focuses on computer programs development that can access data and use it to learn for themselves.

The learning process begins with datasets or observations, such as examples, direct experience, or instruction, in order to look for hidden patterns in data and make better decisions in the future based on the provided information. The main goal is to allow the computers learn automatically without human help or intervention and adjust actions accordingly.

The term machine learning was introduced by Arthur Samuel in 1959[31]. Tom M. Mitchell provided a formal definition of the machine learning algorithms, "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."[24]. Alan Turing in his paper "Computing Machinery and Intelligence"[35], asked the question "Can machines think?" but now it is replaced with the question ”Can machines do what we (as thinking entities) can do?”.

Machine learning have become a main segment of AI. Many algorithms have been developed and several types of learning have been introduced through the last decades, as a result of the technological evolution that is taking place simultaneously.

The machine learning algorithmic types differ in their approach, type of task or problem that they are intended to solve and data type they input and output.

There are 2 learning categories:

- **Supervised Learning**

  The typical goal is to learn a function that maps the input values to the output values. The training process contains both input and output data, as labeled training data. Every input-output pair includes an expert output value, an supervisory signal, which the model has to use in order to learn and generalize from the training dataset to new different inputs. Some widely used supervised learning algorithms are:

  – K-nearest neighbor
  – Linear regression
  – Logistic regression
  – Support Vector Machines
  – Decision trees
  – Neural Networks

- **Unsupervised Learning**

  This approach to the ML problems handles data that are not labeled. The unsupervised learning algorithm has to work by itself to discover hidden patterns and information. Unlike supervised learning, the unsupervised algorithms are draw inferences from input data only. Some common unsupervised learning algorithms are:

  – Clustering
  – Anomaly detection
  – Expectation-maximization
  – Principal component analysis
  – Singular value decomposition
  – Hidden Markov models

2.3.1 Markov Decision Processes

The term Markov Decision Process (**MDP**) was introduced at 1957 by Richard Bellman[2]. It is a discrete time stochastic control process which forms a mathematical framework for decision making procedures. MDPs are used to solve optimization problems via dynamic programming and reinforcement learning algorithms. They

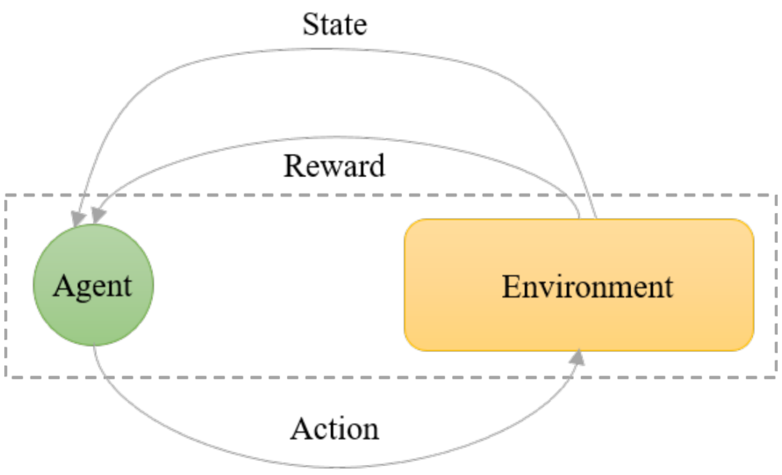


Figure 2.2: A single agent interacting with its environment.

are based on the validity of the Markov property, which states that the next state depends only on the previous state and not any other state that came before it.

A typical MDP is a tuple $\langle S, A, P, R, \gamma \rangle$ where S is a finite set of states, A is a finite set of actions and also A_s is a finite set of actions available in state s. Probability distribution $P(s'|s,a) = Pr(s_{t+1} = s'|s_t = s, a_t = a)$ refers to the transition model from $s \rightarrow s'$, where s is the current state, s' is the next state, s_t and a_t is the state being and the action performed at the time t. Reward function $R(s, a, s')$ produces the reward received on the transition from $s \rightarrow s'$ due to action a. Discount factor $\gamma \in (0, 1)$ is used in order to lower the reward obtained accordingly to the number of steps taken by that time. The actual obtained reward at time t with the proper discount will be $R = R_t \times \gamma^t$.

Given these parameters we are one step before we compose this thesis' MDP. This thesis addresses a problem on multi-agent environment, so we need the multi-agent extension of MDPs. We will discuss Multi-agent MDPs at 2.10.

2.3.2 Reinforcement Learning

The techniques of Reinforcement Learning (**RL**) were introduced into literature by M. Waltz and K. Fu, at 1965 [36]. RL is trial and error learning. Its goal is learning to map situations to actions in order to maximize a numerical reward. A decision maker, or learner, typical refereed to as agent, has no instructions on which action to take, instead must discover which actions grant bigger reward by trying them. On a typical RL applications there are 2 entities, the agent and the environment. Environment is considered everything that is not agent-centered.

In RL settings, which usually are stated as MDP ones, agent and environment interact with each other via three elements, state s, action a and reward r, as illustrated in Fig. 2.2. Specifically, the environment state at time-step $t = 0, 1, 2, 3, ...,$ is denoted as $s_t \in S$, the agent based on s_t performs the corresponding action $a_t \in A$. The environment state changes from s_t to s_{t+1} and the agent obtains its reward $r_{t+1} \in R$, as illustrated at 2.3.

The interactions between environment and agent compose a series of states,

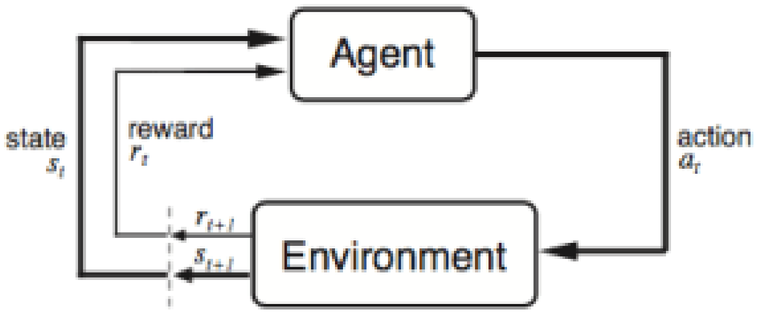


Figure 2.3: An agent interacting with its environment at time t [17].

actions, and rewards: $s_0, a_0, r_1, s_1, a_1, ..., r_n, s_n$. Although n could approach infinity, in practice, we limit n by defining goal or terminal states $s_n = s_T$. The series of states, actions, rewards from time-step $t = 0$ to terminal state $t = n$, is considered an *episode*.

On the trail and error procedure of RL, the agent learns to estimate how good, for itself, can be a state of the environment. Richard Bellman contributed the fundamental methodology to tackle this problem, the well known "Bellman equation" and "principle of optimality". Both rely on dynamic programming problem solving and breaks into simpler subproblems which leads to optimality. According to Bellman, we can use a value function to help the agent keep track of the reward expectations that each state has to offer. With the use of the value function, the agent can compose a policy over the state-space. A policy expresses the action selection on every state in order to maximize the corresponding value function's results. Policy maps each state $s \in S$ to an action $a = a_s \in A$ with probability $\pi(a|s)$ of picking the action a on state s. The obvious goal is to choose the best policy, therefore the best action on each state.

The value function calculation emerges from the value estimation on state s under policy $\pi, V_\pi(s)$. This is the expected reward when starting on state s and following policy π. We define $V_\pi(s)$ as:

$$V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s] \tag{2.1}$$

where G_t is the discounted reward, R is the reward model, γ is the discount factor, t is the time-step and $\mathbb{E}_\pi$ is the expected value given that the agent follows policy π. The value function links state and value for a given policy π.

Correspondingly, we define $Q_\pi(s, a)$ as the expected reward when starting from s, perform action a and then following the policy π:

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a] \tag{2.2}$$

The Q_π links state,action for a given policy π.

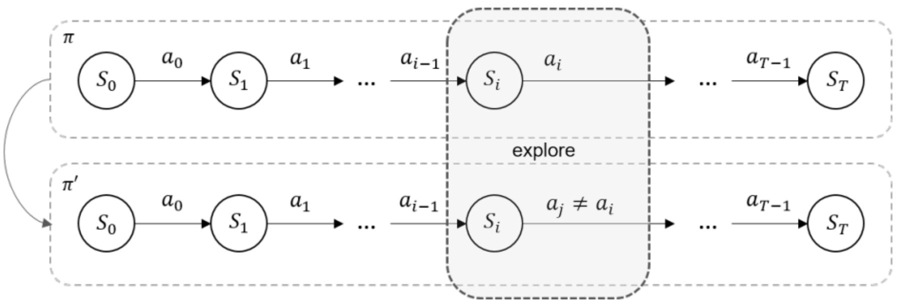


Figure 2.4: A naive approach to finding a better policy π' from π [27].

Based on 2.1, we expand V_π to represent two consecutive states s, s' [33] where $s = s_t$ and $s' = s_{t+1}$:

$$
\begin{aligned}
V_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\
&= \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2}|S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma\,\mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2}|S_{t+1} = s']] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V_\pi(s')]
\end{aligned}
\tag{2.3}
$$

Similarly, from 2.2 expand Q_π to:

$$Q_\pi(s, a) = \sum_{s'} p(s'|s, a)[R(s, a, s') + \gamma \sum_{a'} \pi(a'|s')Q_\pi(s', a')] \tag{2.4}$$

From V_π to Q_π:

$$V_\pi(s) = \sum_a \pi(a|s)Q_\pi(s, a) \tag{2.5}$$

and vise versa:

$$Q_\pi(s, a) = \sum_{s'} p(s'|s, a)[R(s, a, s') + \gamma V_\pi(s')] \tag{2.6}$$

Equations 2.3 and 2.4 are called the Bellman equations for V_π and Q_π respectively, and are used in policy improvement. On a trial and error approach, the agent can explore different actions $a_j \neq a_i$ on state s_i in order to achieve $Q_\pi(s_i, a_j) > Q_\pi(s_i, a_i)$. If the previous inequality is true, we replace a_i with a_j on the policy π and derive a new and improved policy π'. A policy comparison can be considered as:

$$\pi_1 \geq \pi_2 \iff V_{\pi_1} \geq V_{\pi_2}, \quad \forall s \in S \tag{2.7}$$

We define an optimal policy π^*, an optimal value function V^* and an optimal q-function Q^* as following:

$$\pi^* \geq \pi, \quad \forall \pi \tag{2.8}$$

$$V_{\pi^*}(s) = V^*(s), \quad \forall s \in S \tag{2.9}$$

$$Q_{\pi^*}(s,a) = Q^*(s,a), \quad \forall s \in S, \forall a \in A \tag{2.10}$$

We can expand 2.9 and 2.10 to form the Bellman optimality equations:

$$\begin{aligned} V^*(s) &= \max_a Q^*(s,a) \\ &= \max_a \sum_{s'} p(s'|s,a)[R(s,a,s') + \gamma V^*(s')] \end{aligned} \tag{2.11}$$

and

$$\begin{aligned} Q^*(s,a) &= \sum_{s'} p(s'|s,a)[R(s,a,s') + \gamma V^*(s')] \\ &= \sum_{s'} p(s'|s,a)[[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')] \end{aligned} \tag{2.12}$$

We can use these equations to solve a large variety of RL problems, usually in form of MDPs. This technique cannot handle problems with large state-space due to lack of memory or computational power of computers, even the modern ones.

Two well-known learning schemes in RL for model-free use are TD and MC learning, so they do not require knowledge of transition probabilities $p(a_i|s)$ in order to approximate the value and q functions.

Many techniques and algorithms have been developed to tackle RL problems. Some of them are:

- Actor-Critic (AC) architecture[19].
- State-Action-Reward-State-Action (SARSA) algorithm[33].
- Least-Squares Policy Iteration (LSPI) algorithm[28].
- Q-Learning algorithm[37].

In this thesis we will use the Q-learning algorithm as a fundamental building block for our Deep Q-Network, which we will analyze at 2.3.4.

Q-learning is an algorithm which does not require a environment model to solve RL problems, it is model-free. For finite MPDs, Francisco Melo showed that Q-learning always converges to an optimal action-selection policy given infinite exploration time[22]. Q-learning uses a table to store and update the values learned from previous interactions with the environment. The following equation is considered as the update rule for Q-learning:

$$Q^{new}(s_t,a_t) = Q(s_t,a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)) \tag{2.13}$$

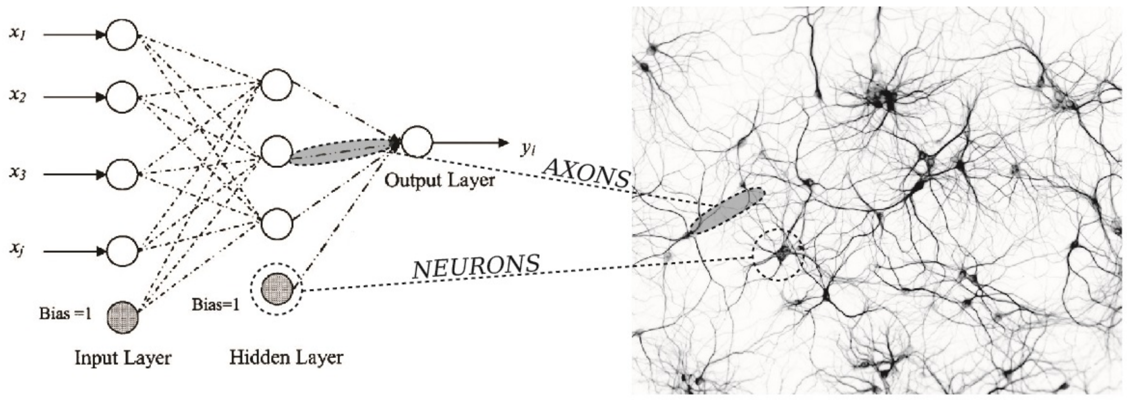


Figure 2.5: Analogy between brain NN and ANN.

where r_t is the reward received after moving from s_t to s_{t+1}, Q^{new} is the updated version of the old Q function, the term $\alpha \in (0, 1]$ is the learning rate, which refers to evolution of the learning procedure over time. Altering the learning rate can force learning to shift focus on the latest Q values and ignore those which came some steps before; or vise versa, it can be used to attach importance to the initial values and learn with a very small ratio.

2.3.3 Artificial & Deep Neural Networks

Walter Pitts[29] and Warren McCulloch[17] introduced the computational model for Artificial Neural Networks (**ANN**) in 1943. The core idea was that ANNs can mimic the human brain with neurons and axons, in a way that is illustrated at Fig. 2.5. As humans, we receive stimulus from our input detectors or organs, our NN process these signals and produce different ones in order to send them to other parts of our body. Just like that, an ANN is composed of an input layer, which feeds the data into the ANN, connected with that is a layer of neurons where every neuron processes the input signals and produces another signal to send to the next layer with the use of a synapse. The final layer is the output layer where the neuron produces an output signal, meaning that the ANN has fully process the given inputs. Fig. 2.6 illustrates how a ANN typical neuron functions, by collecting inputs with different importance, so every input value has a weighted effect on the neuron, and after processing the inputs produce an signal which is filtered by the activation function in order to eliminate or adjust the signals.

A typical ANN is composed of three layers: an input layer, an output layer and an intermediate one. The intermediate layer is usually called hidden layer because an external observer can not know if this layer exists, see Fig. 2.7a. If the ANN contains more than one hidden layer, we call it a Multi-Layer ANN or a Deep NN (**DNN**), see Fig. 2.7b.

Since the ANN's conception, many techniques and architecture have been developed which became popular for their ability to tackle major problems in ML. The major differences refer to:

- Layer interconnection types

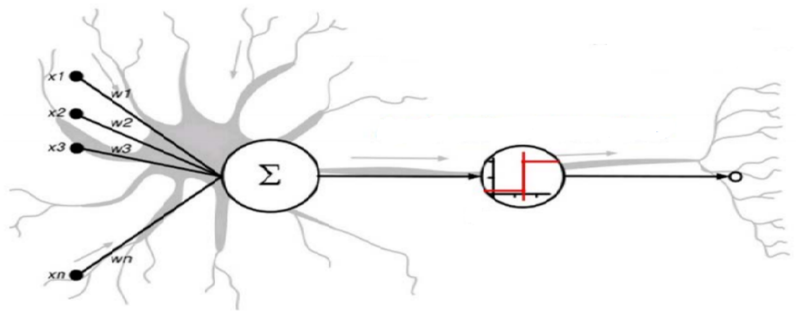
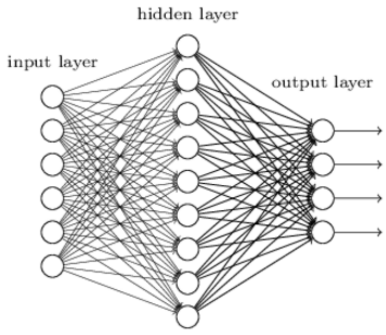
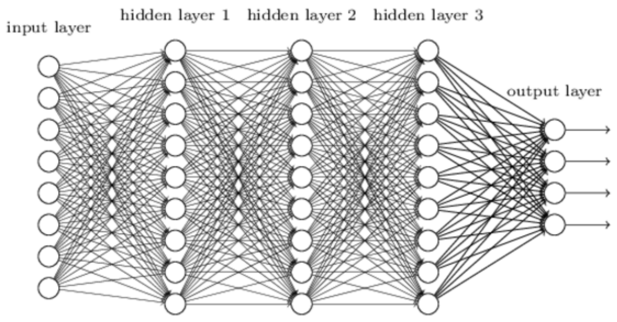


Figure 2.6: A neuron-oriented input and output procedure.



(a) Typical ANN



(b) Typical DNN

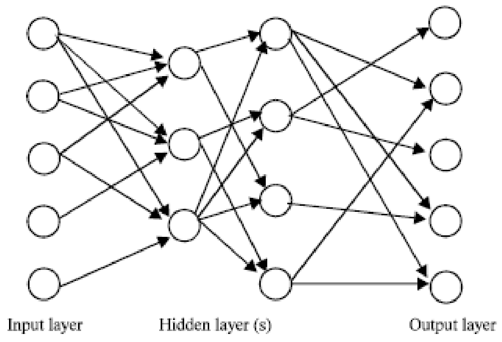
Figure 2.7: Difference between ANN and DNN

  - Fully connected networks, Fig. 2.8b
  - Partially connected networks, Fig. 2.8a

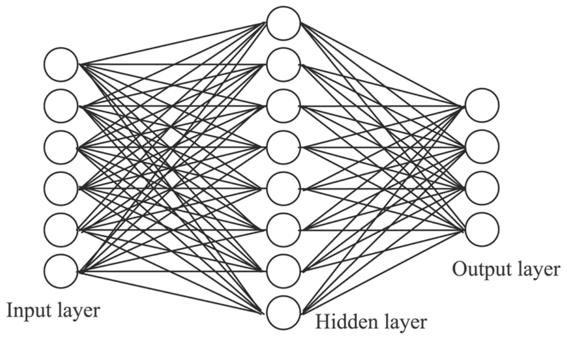
- Information flow direction types
  - Feed-forward NN, Fig. 2.8c
  - Recurrent NN (RNN), Fig. 2.8d
    - Fully recurrent
    - Simple recurrent
    - Long short-term memory (LSTM)

- Layer functionality types
  - Convolutional NN (CNN), Fig. 2.8e
  - Associative NN (ASNN)

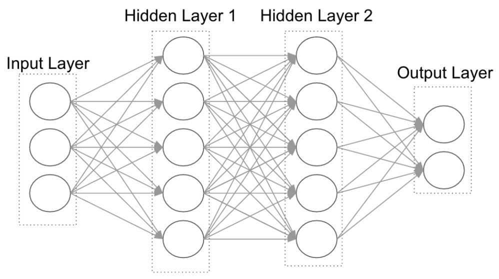
- Input-output flexibility capability types
  - Sequential model, linear stack of layers
  - Functional model, multi-input-output layers



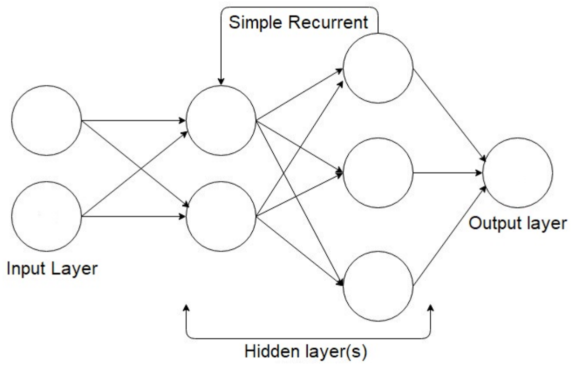
(a) A partially connected ANN



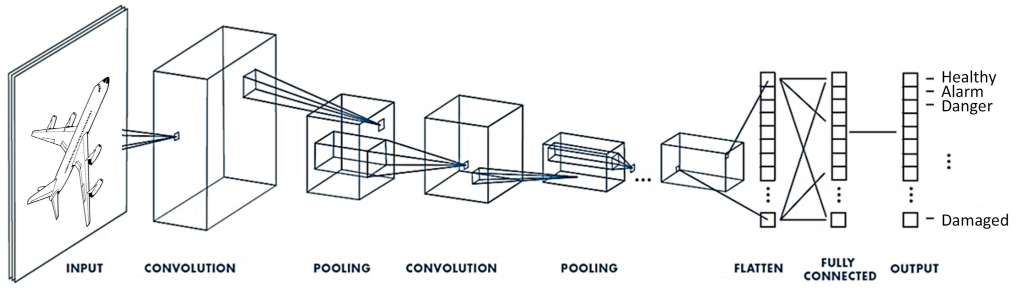
(b) A fully connected ANN



(c) A feed-forward NN



(d) A Simple Recurrent NN



(e) A Convolutional NN

Figure 2.8: Different types of ANNs

The goal on a typical ANN is to minimize the error on its predictions as fast as possible. A ANN makes a guess based on the input and the existing weights. The prediction error derive from the difference between prediction and ground truth. The ANN has to adjust its weights properly in order to minimize the previous error.

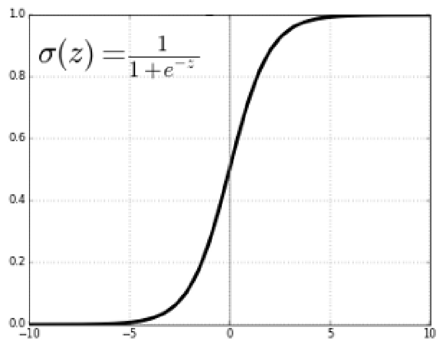
$$\begin{aligned} prediction &= input \times weight \\ error &= ground\ truth - prediction \\ adjustment &= error \times weight's\ contribution\ to\ error \end{aligned} \tag{2.14}$$

Those are the three main functions of DNNs, and of ANNs in general:

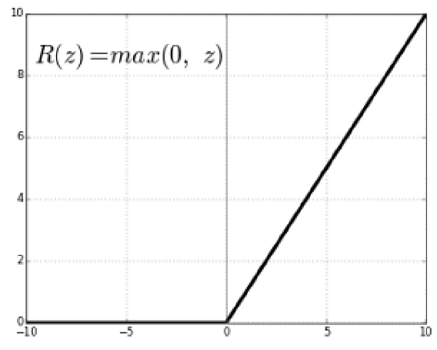
1. Scoring input
2. Calculating loss
3. Update the model

The most important step of this process is the update. The most commonly used optimization functions, typically refer to as optimizers, for weight adjustment are: Gradient Descent, Stochastic Gradient Descent, ADAM[16], Conjugate Gradient, ADADELTA and Line Gradient Descent.

Except the utility of optimizers, big role in the learning process have the activation functions as well. Some of the most used are: Rectified Linear Unit (ReLU), Sigmoid, Logistic Sigmoid, Softmax, Softplus, Linear and Tanh.



(a) Sigmoid



(b) ReLU

Figure 2.9: Activation functions examples

2.3.4 Deep Reinforcement Learning

Deep Reinforcement Learning (**Deep RL** or **DRL**) is a combination of Deep Learning and Reinforcement Learning. It can deal with high-dimensional environments and tackle the curse of dimensionality problem which is a constant concern in computer science and specifically on analysis and learning procedures. Perhaps the most well known application of Deep RL was the 2013 paper by DeepMind Technologies team, who used a CNN in order to get a graphical representation of the input state

s and produce Q-values for all possible actions at state s, they named it Deep Q-network (DQN).

The learning procedure consists of a RL problem based on a MDP where the Q-table has been replaced with a Deep NN. We discussed how we can get the optimal Q^* at subsection 2.3.2, so we will explicate the corespondent method on DeepRL, for a typical Deep Q-network. DQN leverages a function approximator to estimate the optimal action-value function,

$$Q(s,a;\theta) \approx Q^*(s,a). \tag{2.15}$$

A neural network approximator with weight θ is called a Q-network. It can be trained by minimising the loss of its predictions, which is calculated as,

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\Big[(y_i - Q(s,a;\theta_i))^2\Big], \tag{2.16}$$

where $\rho(s,a)$ is the probability distribution over states s and actions a, $Q(s,a;\theta_i)$ is the prediction of the Q-network at iteration i and $y_i = \mathbb{E}_{s'\sim\mathcal{E}}[r+\gamma \max_{a'} Q(s',a';\theta_{i-1})|s,a]$ is the target for iteration i. Differentiating the loss function with respect to the weights we arrive at the following gradient,

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\Big[\Big(\underbrace{r+\gamma\max_{a'} Q(s',a';\theta_{i-1})}_{\text{target}} - \underbrace{Q(s,a;\theta_i)}_{\text{prediction}}\Big)\nabla_{\theta_i} Q(s,a;\theta_i)\Big]. \tag{2.17}$$

Algorithm 1: Deep Q-learning with Experience Replay [26]

```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights
for episode = 1, M do
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
    for t=1, T do
        With probability $\epsilon$ select random action $a_t$
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action $a_t$ in emulator and observe reward $r_t$ and state $x_{t+1}$
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient decent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
        according to equation 2.17
    end
end
```

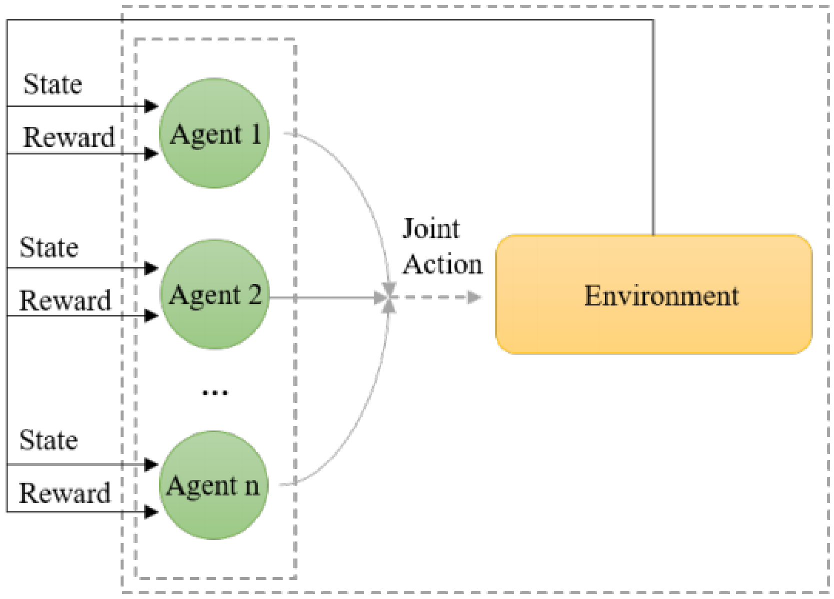


Figure 2.10: Multiple agents interacting with the same environment[27].

2.3.5 Multi-Agent Deep Reinforcement Learning

A Multi-Agent System (MAS) is a loosely coupled network of software agents who interact to solve problems which are beyond the individual knowledge or capacities of each software agent. In a MAS agents collaborate in order explore the environment or to solve tasks. Due to their ability to make autonomous decisions and learn from the interactions between the MAS group and the environment, they compose a flexible unity of individual entities. It is this flexibility which makes MAS suited to tackle problems in a variety of disciplines including civil engineering, computer science, and electrical engineering. A typical example of a MAS is illustrated in Fig. 2.10.

We have discussed MDPs and their usage in order to find optimal policies for single-agent tasks at subsection 2.3.1. However, in a MAS where agents collaborate in an environment, which is modeled by each agent with a different MDP, optimal behavior is not defined in the same way [8]. The goal of the MAS is to select actions that maximise the expected discounted accumulative reward of the the multi-agent group over time. In the domain of multi-agent learning, the MDP is generalized to a stochastic game, or a Markov game [8].

A multiagent MDP is a 5-tuple $\langle a, S, A, P, R \rangle$, in which a is a finite collection of N agents, $S = S_1 \times S_2 \times ... \times S_N$ is the joint state set and $A = A_1 \times A_2 \times ... \times A_N$ is the joint action set. A joint action $(a_1, a_2, ..., a_N)$ represents the concurrent execution of the actions a_i by agent i. The state transition probability function is represented by $P : S \times A \times S \to [0, 1]$ and the reward function is specified as $R : S \times A \times S \to \mathbb{R}^n$. The value function of each single agent is dependent on the joint state, the joint action and joint policy, which is defined by $V^\pi : S \times A \to \mathbb{R}^n$.

In Multiagent Reinforcement Learning (**MARL**) there are many challenges to overcome in order to solve real-world problems effectively. The main are:

- Non-stationarity
- Partial observability
- MAS training schemes
- Continuous action spaces
- Transfer learning

We will not discuss more the solutions that have been proposed to overcome these problems. However, in this thesis we use some techniques from Multiagent Deep Reinforcement Learning (**MADRL**) in order to tackle our MAS settings problems.

Chapter 3

Related Work

In this chapter we brief review realated work. We first present other systems which have been developed to cope with emergency response situations. In particular, we discuss in some detail the HAC-ER system which was an inspiration for this thesis. Finally, we present essential techniques on Multi-agent RL and Multi-agent Deep RL which have been developed in the past few years.

3.1 Emergency Response Systems

Emergency response agencies face a number of challenges when a disaster occurs. First, given that infrastructure may have been damaged, gaining situational awareness is vital in order to determine where aid is required and the way it can be delivered [30]. Information about the damaged area is very important and many sources can contribute on these updates, including satellite imagery, people on the ground or even unmanned aerial vehicles (UAVs) can provide key information regarding the situation. Second, the proper allocation of the available forces is grievous. Based on their situational awareness, groups like first responders (FRs) can dig people out of the rubble, extinguish fires, or provide medical aid to those who need it. A major parameter is the time required to travel to the corresponding task, as it is non-productive time and can be crucial. Finally, due to the uncertainty of the environment, every team should inform the others about the changes on the current situation in order to detect previous assumptions that are no longer valid and change the course of plan for the future.

Ramchurn at el. [30] developed a system which can handle and coordinate large groups of responders in major disasters. It was an inspiration for this thesis, and we thus discuss more about it in the following subsection.

3.1.1 Human-Agent Collectives for Emergency Response

The prototype disaster management system, Human-Agent Collectives for Emergency Response (**HAC-ER**), was introduced in the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015). It proposed a novel system which can address many of the challenges that concern the research

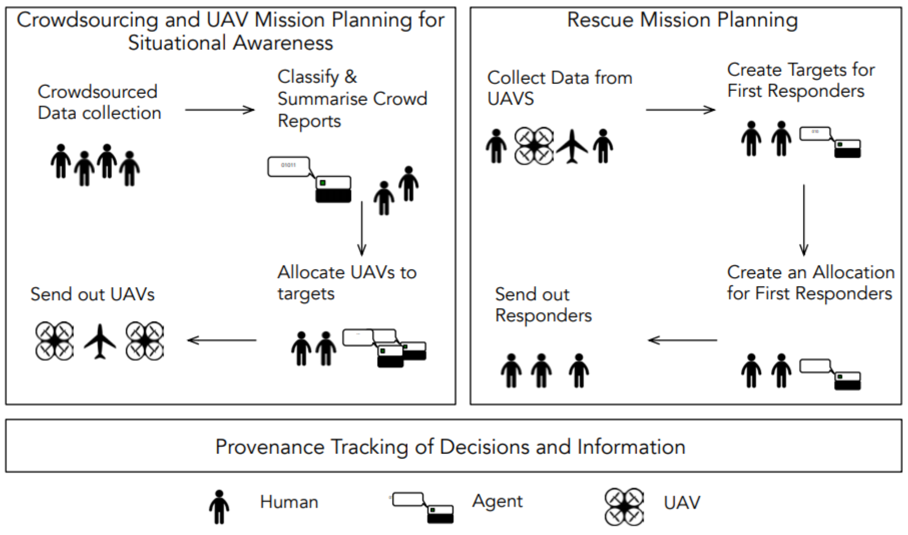

Crowdsourcing and UAV Mission Planning for Situational Awareness
Crowdsourced Data collection
Classify & Summarise Crowd Reports
Allocate UAVs to targets
Send out UAVs
Rescue Mission Planning
Collect Data from UAVS
Create Targets for First Responders
Create an Allocation for First Responders
Send out Responders
Provenance Tracking of Decisions and Information
Human
Agent
UAV

Figure 3.1: The left box describes the information gathering process while the right box describes the task allocation process[30].

community about emergency response (ER) operations, and won the best Demo award in that major AI conference. A overview of the system is provided in Fig.3.3.

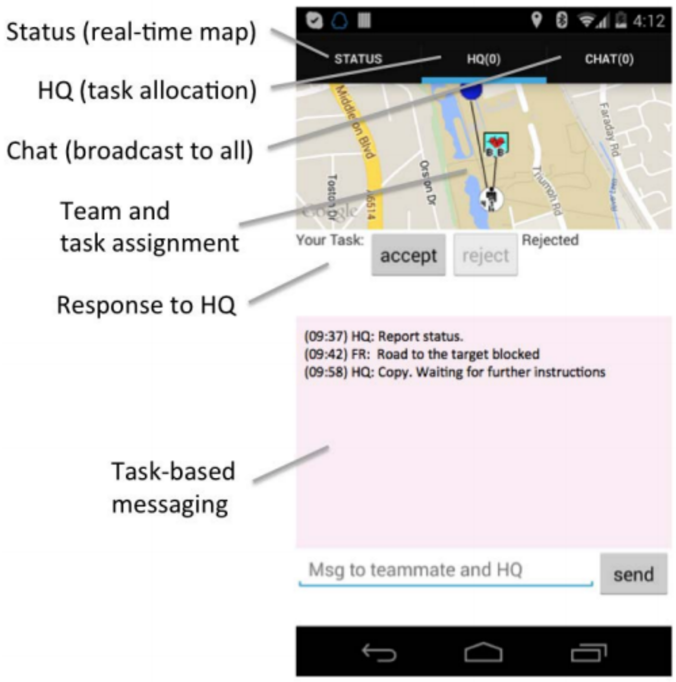
HAC-ER makes use of state-of-the-art algorithms in order to achieve collaboration between humans and agents with the goal to carry out tasks as a team. It utilises crowdsourcing and machine learning techniques to get situational awareness from large streams of data on social media platforms such as Facebook and Twitter. The system also incorporates a provenance manager in order to track and analyse the information which the system uses.

There are three aspects on decision making, the strategic one, the tactical one and the operational one. HAC-ER distinguishes three levels based on these aspects, "Gold", "Silver" and "Bronze" respectively. Each level with its own responsibilities and purpose.

With the use of the CrowdScanner component, HAC-ER obtains information from Twitter and construct heatmap layers in spatial maps in order to better geo-tag the disaster zone and evaluate the emergency intensity. The construction of the heatmaps combines two ML techniques, independent Bayesian classifier combination and the Gaussian Process.

The UAV allocation is a mixed-initiative control with both coordination algorithms and human input decisions. The system employs the max-sum algorithm to cope with coordination. Nevertheless, the manual allocation of the UAVs is always very important, so the HAC-ER team designed a fully functional user interface (UI) to allow human-agent teams to coordinate the UAVs. The max-plus algorithm provides suggestions to the correspondent Silver commander via this UI. Bronze operators can tele-operate the UAVs via this UI as well. The decision making and the information gathering in HAC-ER are illustrated at Fig. 3.1

Another allocation task includes the first responders on the ground. The system leverages the Planner Agent to collect GPS locations and with the use of a Multi-Agent Markov Decision Process (see Section 2.3.5) finds a policy that maximizes the



(09:37) HQ: Report status.
(09:42) FR: Road to the target blocked
(09:58) HQ: Copy. Waiting for further instructions

Figure 3.2: The Mobile Responder Tool[30].

reward obtained. The developed a UI for task allocation which contains FR feedback, hover-over alignment, drag-and-drop editing and task-based communication channels. In order to communicate with the responders more efficiently, the HAC-ER team developed a Mobile Responder Tool as well, which is updated real-time and every responder can get the information that he needs on its task, as showning Fig. 3.2.

The Tracking Provenance is the component which analyzes and provides detailed informing about every part of the system. It checks information validity and can revoke every decision made based on a invalid piece of information. It is the most important tool on the situation monitoring, therefore its employment can change the planned course of action completely.

3.2 Multi-Agent Reinforcement Learning Techniques

Many techniques and algorithms have been developed to control multiple agents for RL applications. All these endeavors tackled several challenges as compared to single agent RL. Agents' heterogeneity is a major problem in multiagent RL (MARL). The need to define proper collective rewards is vital to every proposed solution [27]. Additionally, every design requires compact representations in order to provide scalability for large number of agents. One of the most pressing problems in MARL is the non-stationarity of the environment. We discuss how the research community confront these barriers and analyse briefly some of the algorithms that are primarily used to cope with MARL and MADRL.

To begin, Jelle R. Kok and Nikos Vlassis proposed scalable techniques for learn-

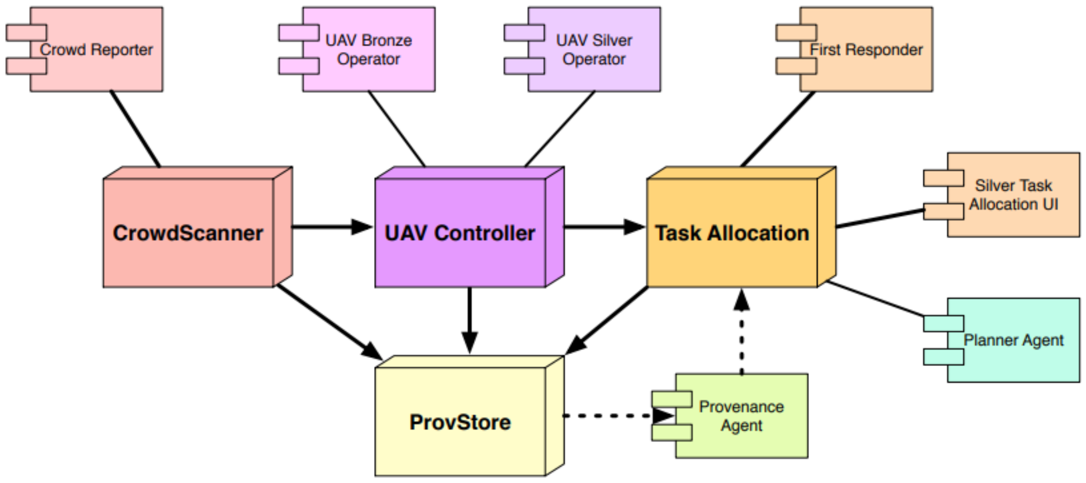


Figure 3.3: Information flows between HAC-ER's components[30].

ing in collaborative multiagent settings, at [18]. They introduced Sparse Cooperative Q-learning which consist of model-free RL techniques. It utilizes the topology of a coordination graph in order to approximate the global action-value function, and the updates for each agent are based on its contribution to the global action value. They manage to provide a linear scalability via the combination of edge-based (or agent-based) decomposition on a global payoff propagation algorithm for action selection.

3.2.1 Multi-Agent Deep Reinforcement Learning

Castaneda proposed the *Deep Repeated Update Q-network (DRUQN)* and the *Deep Loosely Coupled Q-network (DLCQN)*, to deal with the non-stationarity in MAS[7]. DRUQN algorithm updates the action-value inversely proportional to the likelihood of action selection, aiming to avoid policy bias. DLCQN adjusts an "independence degree" for any individual agent using its observations and negative rewards. Through this degree, every agent learns to evaluate, based on the circumstances, if it is better to cooperate with another agent or act independently.

In real-world applications, partial observability is a common feature of the environment. Hausknecht and Stone introduced the *Deep Recurrent Q-network (DRQN)* [15] which was the building block for the work of Foerster at el. [13] who proposed the extension *deep distributed recurrent Q-network (DDRQN)* extension to handle partial observability via the use of a multi-agent partially observable Markov decision process (POMDP). DDRQN relied on three features: inter-agent weight sharing, disabling experience replay, and last-action inputs. The inter-agent weight sharing requires from all agents to use weights of one network, which it learns during the training process. Weight sharing can decrease learning time, as a result of the reduction on the parameters to be learned. Each agent has different hidden state and observations, DDRQN however assumes that the agents have an identical set of actions. This is not true on complicated or real-world problems. Often autonomous

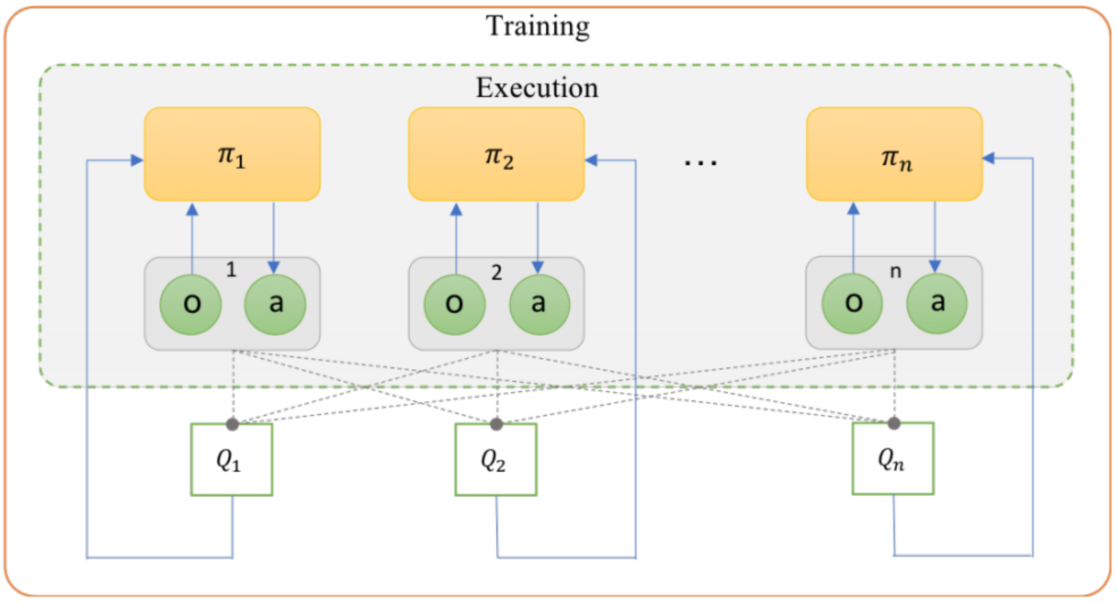


Figure 3.4: Centralized learning and decentralized execution based on MADDPG where policies of agents are learned by the centralized critic with augmented information from other agents' observations and actions[20].

agents have different set of actions [27].

3.2.2 Centralized Learning and Decentralized Execution

A popular approach regarding the training scheme in a MAS is the centralized learning and decentralized execution (**CLDE**) one. Using this technique, a group of agents is trained simultaneously via a centralized method through a communication channel. Decentralized policies are then emerging, where every agent take actions relying on its (local) observations. CLDE has become a standard archetype in MAS settings due to the simplicity on the learning process, which can be performed at a simulator in a laboratory having no constrains regarding the communication and obtaining extra information.

Lowe et al. [20] introduced the *Multi-agent Deep Deterministic Policy Gradient (MADDPG)* which features actor-critic policy gradient algorithms. MADDPG utilizes CLDE in which each actor picks actions based only on its local observations, and the critic evaluates these actions using extra information, such as global observations. The segregation of training and execution regarding the information locality is illustrated in Fig. 3.4

Foerster et al. [12] proposed another multi-agent actor-critic method, called *Counterfactual Multi-agent (COMA)*, which is also based on a CLDE scheme. COMA focuses on handling the problem that the agents cannot quantify their contribution in cooperative settings, regarding the global rewards which obtained by joint actions. However, this method can only handle discrete action spaces, contrary to MADDPG which can learn continuous policies as well.

Chapter 4

Our Approach

In the previous chapter, we analysed some systems which have been developed to deal with emergency response situations, how to manage search and rescue operations and how ML can contribute to coordinate a set of agents in order to achieve better joint actions and obtain larger rewards. In this chapter we will analyse our system, which is developed to overcome some often difficulties that we detected in other emergency response systems, and to simplify some operations to manage a disaster situation better.

We begin by providing details regarding the system architecture and the techniques used in order to optimize the interconnectivity of its components, as well as the system's GUI developed for web browsers and smartphones. We then describe in detail all aspects of the decision making component. This is composed of the environment creating and manipulation, the Deep Reinforcement Learning algorithms, we used in this work and the novel deep Q-network architecture we developed for use in realistic search and rescue operations.

4.1 System Architecture

In this thesis we developed a system in order to operate in an emergency response situation. We ensure the proper functionality, using a web-page environment based on a map, with live-updates from multiple sources, users or administrators. We placed emphasis on the building blocks of the system, like the API that was developed for the system's intercommunication between its components; the User Interface (UI) on the web-page and the functionality that it provides, and the mobile application which every acting entity can use to obtain information about the on going situation and the task that he has to fulfill.

We developed a distributed system as a server, on the Okeanos platform which provides free virtual machines to university students. We use a PostGIS database to store and update spatial data which are used by the decision making component to perform task allocation for each agent. The dataset which we use regarding the spatial data of Chania, has been generated by the TUC SenseLab's research team and provided to us for this thesis purposes. We developed a ReST API alongside with the server in order to connect every component of the system properly and leverage the simplicity of the ReST usage. We developed a web-page which is online at

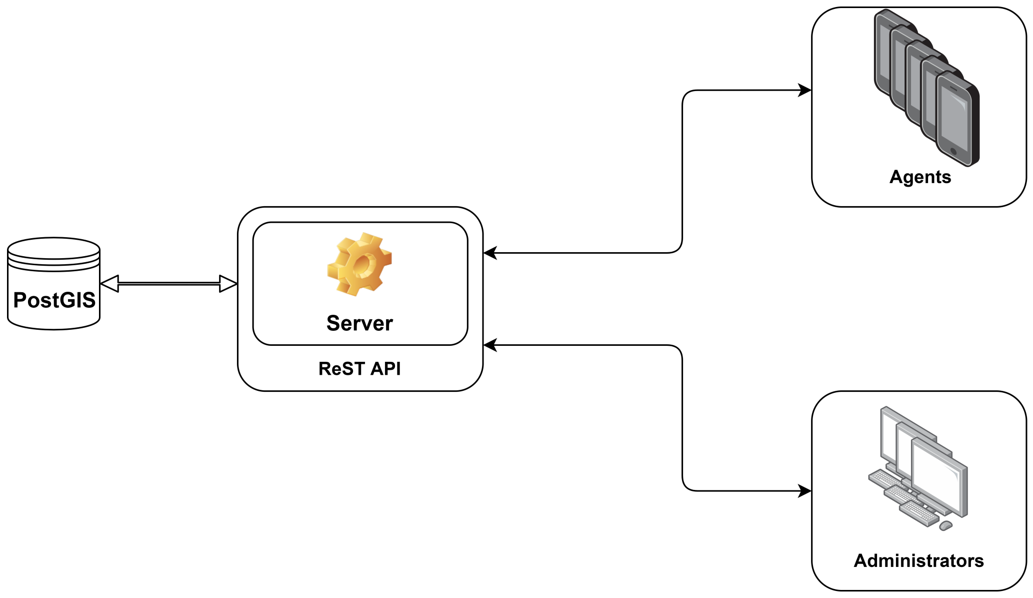


Figure 4.1: System Architecture.

the url http://snf-871151.vm.okeanos.grnet.gr:8080/ and an application for Android smartphones to provide better UI for both administrators and users groups. Fig. 4.1 illustrates the overview of the system's architecture.

4.1.1 Web-Page

The graphical user interface (GUI) is essential for the simplicity that a page like this needs. The web-page is developed over Google Maps Javascript API. Techniques like AJAX are used to update the attributes on the current action plan periodically. The layout and design of the page were constructed with HTML and CSS. The typical overview of the web-page with a simple plan is illustrated at Fig. 4.2. The web-page's GUI provides the following functionalities regarding an ER situation:

- Plan
  - Load previous plan from DB
  - Create new plan
  - Reset current plan
- Area
  - Add new circle-like area
  - Add new polygon area
  - Modify area's attributes, like center, radius and boundaries
- Agent
  - See agent's location
  - Search for specific agent

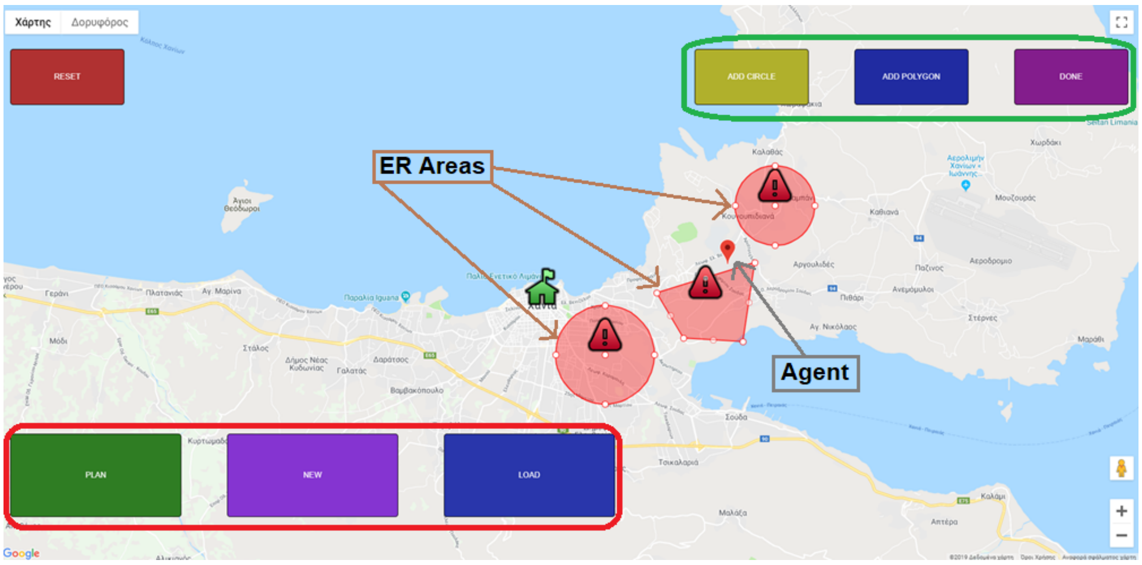


Figure 4.2: Web-page overview.

The page can be used for a variety of areas and agents, see Fig. 4.3. The search agent functionality provides fast and efficient search over the agents' names. After clicking a name from the list, the map shifts its focus on this agent and its icon begin to bounce to indicate its position more clear, see Fig. 4.4.

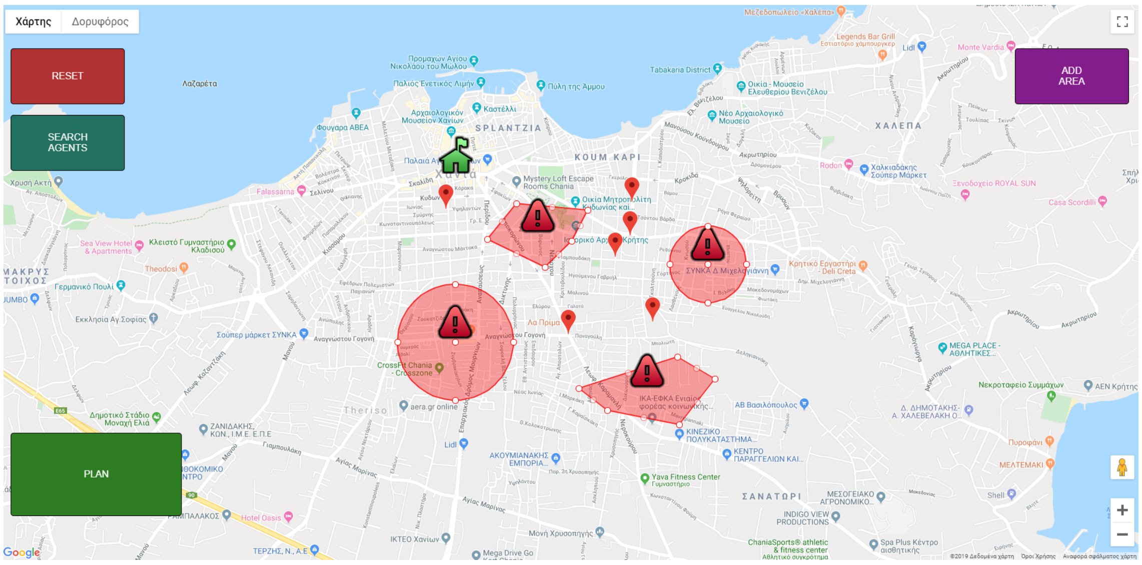


Figure 4.3: A more realistic plan.

4.1.2 Mobile application

We developed the mobile application for Android using the Android Studio platform. Alongside with that, we used once again the Google Maps API for Android applications by the Google Inc. which is provided for free. The utility of the Android application includes the location updates of the corresponding agent, with the

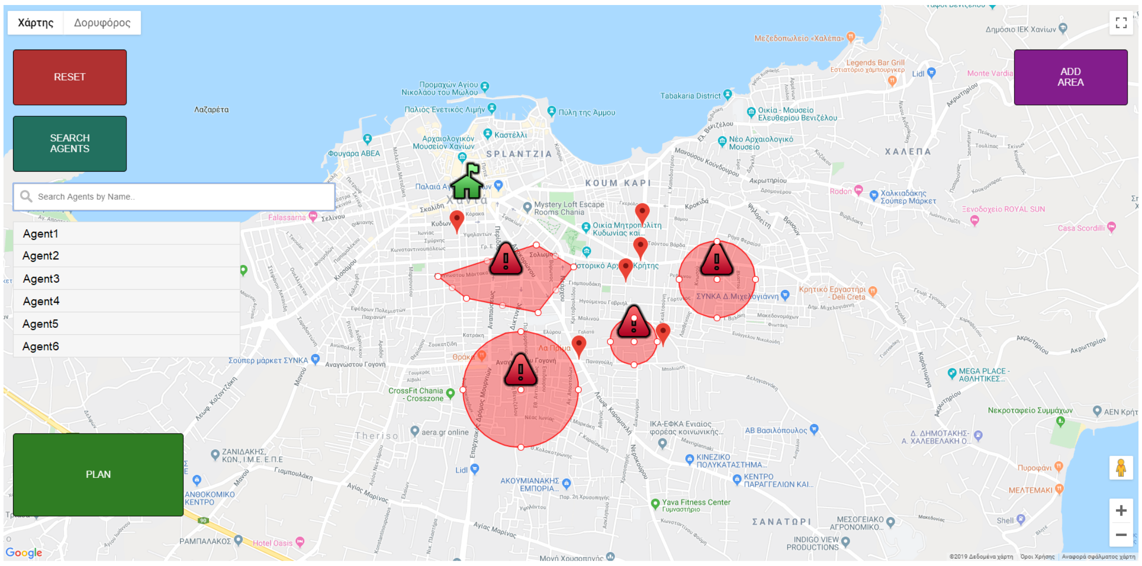


Figure 4.4: Searching for specific agent.

use of the device's GPS sensor, and the update on the task allocation for the agent.

4.1.3 ReST API

We use Golang as the programming language to develop the ReST API. Every information exchange is formed as JSON file, so we define the following structures to handle the communication between the system's components:

- Plan
  - ID (unique for every plan)
  - Start Date (of the plan)
  - Status (of the plan, e.g. started, stopped)
- Agent
  - ID (unique for every agent)
  - Plan ID (at which the agent pertains to)
  - Latitude (position of the agent)
  - Longitude (position of the agent)
  - Type (of the agent, e.g. firefighter, medic)
  - Name (of the agent as additional data)
- Area
  - ID (unique for every area)
  - Plan ID (at which the area pertains to)
  - Latitude (position of the area's center)

  - Longitude (position of the area's center)
  - Info (of the area as additional data regarding its radius or its polygon points)

- Action
  - Agent ID (at who the action is referred to)
  - Plan ID (at which the action pertains to)
  - Area ID (at which the action pertains to)
  - Status (of the action, e.g. accepted, done)

The API handles every request from and to the server as soon as it is formed with the proper and predefined specifications. For example, every update on an agent's position has to go to home page ("http://snf-871151.vm.okeanos.grnet.gr:8080/") following by the "api/agentupdate", http://snf-871151.vm.okeanos.grnet.gr:8080/api/agentupdate and only with the POST method, everything else but that, will get excluded from the API as an invalid request. Some of the valid API functions are:

- home page + "api/action"
- home page + "api/area"
- home page + "api/agent"
- home page + "api/plan"
- home page + "api/mobile/areas"

The API is a highly important component of the system because it connects every part of the system with each other. It makes sure that the web-page is up and serves the necessary files to the internet. The API handles the information exchange between the web-app and the PostGIS DB. It also provides and updates the data on the mobile application. Thus, the development of this part was a very crucial step on this thesis.

4.2 Decision Making

In the previous section, we explain how we are able to collect large amount of information regarding ER situations with the usage of the web-page and mobile application. In this section we discuss how this information can be transformed to develop a MAS environment which can be used to perform state-of-the-art ML techniques to coordinate the agents efficiently. We will analyze the environment construction from the data stored in our PostGIS DB, how this environment updates its state and how the rewards are given to the agents, how we developed our Deep Q-network (DQN) for MAS settings, the methods used to train the model, and the challenges that we faced during all these procedures.

4.2.1 Data Transformation and Pre-processing

We make a pre-processing and transformation of the available GIS data, which is described below, in order to produce a graph for DQN algorithms to operate on. The dataset that we used contains information about Chania, Crete, Greece. It is a dataset which constructed by SenseLab research team, at 2018. The data refers to roads in and around the Chania area. Their coordinate system is EPSG:2100, which is one of the coordinate systems for the Greek territory and the Greek road network. We reproject every EPSG:2100 point to the global standard EPSG:4326 which was established by the National Geospatial-Intelligence Agency in 1984. The Chania dataset, the agents' possitions and the ER locations, are stored at our PostGIS DB.

Using the Chania dataset, we can extract the points where a road intersects with another one. These points are the intersections of the area. We use these intersections to build a graph. This graph will be used by our DQN algorithms to operate on. Fig. 4.5 illustrates an example with the pre-processed intersections on the map near an agent.

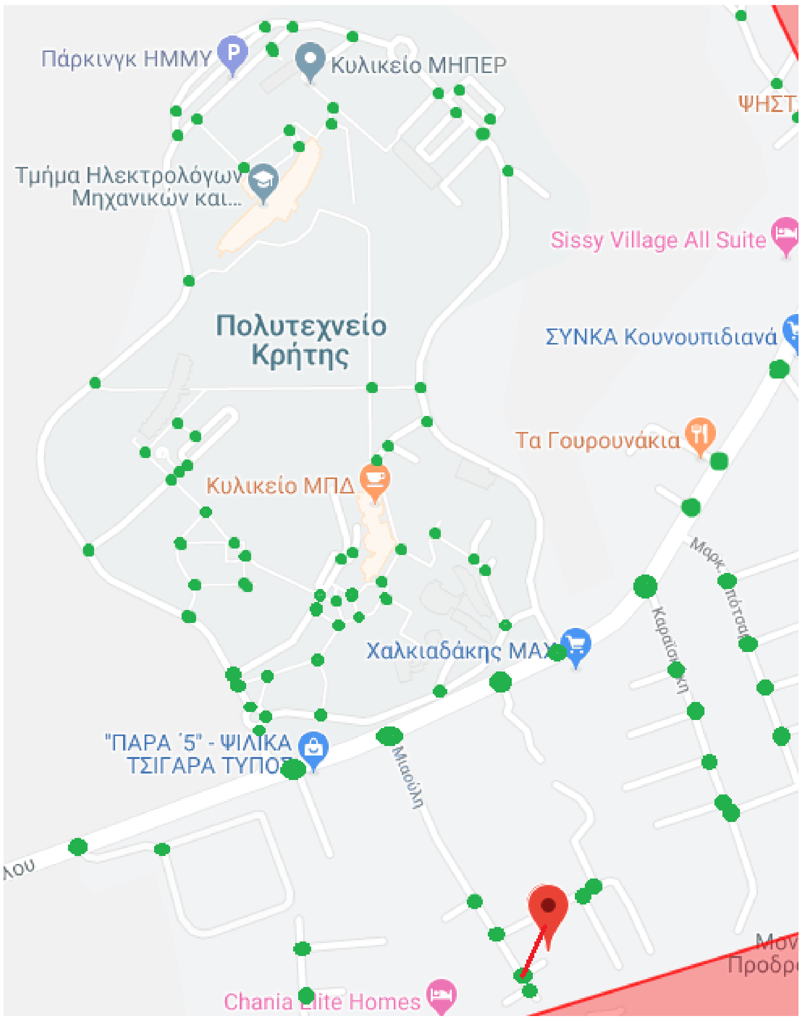


Figure 4.5: Intersections on the map are colored with green dots, and the closest intersection to the agent is connected with it via a red line.

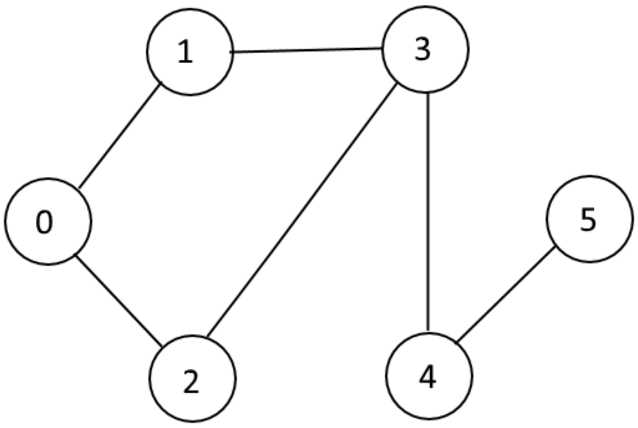


Figure 4.6: An example of a cyclic graph.

The graph built by the intersections is a cyclic graph like the simple one in Fig. 4.6. Every node represents an intersection and every edge refers to the road which connects the two intersections. This means that an agent can be placed on the graph based on the road intersection which is nearest to it. It can move via edges to another intersection.

4.2.2 Constructing the Learning Environment

The most commonly used tool for ML environment construction, as well as developing and comparing RL algorithms is Gym, from OpenAI. Gym toolkit provides essential libraries for an easy-to-use suite for RL environments. It provides a straightforward interface which can fully support the building of an environment for any RL applications. Gym has developed many ready-to-use environments in order to test and evaluate ML algorithms with ease.

In this thesis we had to develop a new environment based on our needs. We leverage the Gym default interface which contains the unimplemented functions: *Init, Step, Reset, Close.*

Those subroutines provide the essential functionality for the environment manipulation that we need in order to perform complicated DRL techniques during the training process. We also implemented a reward function which produces a global reward at every single time-step based on the global state.

We developed two types of environments. The most important type is the one that handles real world data, which developed for real-time emergency usage for the Chania area, as we described at section 4.2.1. The other type was developed for training and testing purposes. It uses synthetic data with specific and user defined attributes. The development of this type was crucial for this thesis due to the limitations that the real world environment type has. Using the synthetic environments, we could adjust the data parameters as we desire, in order to train and test the decision making system in various situations quickly and easy. We will analyze in the following subsections why the synthetic environments play huge role in the training step and how we used them to scale-up our system performance.

4.2.3 A Novel Deep Q-Network for Search and Rescue Operations

The heart of every decision making system is the learning methods and techniques that the developers used in order to tackle the problem efficiently. In this thesis we developed a Deep Q-Network (DQN) for multi-agent environment settings. This choice was made due to the large state and action space of the environment, it could most probably not have been feasible to create a Q-table for massive environments like those that we build for this thesis. Based on the environment types that we developed, we constructed a multi-input deep NN for Q-function approximation. Basically, we replaced the Q-table (or the Q-fucntion) with a deep NN, as the techniques for DQN dictate.

For an ER situation and the corresponding environment, the DQN inputs include four attributes:

1. Agents' positions
2. Agents' types
3. Emergencies' positions
4. Emergencies' types

Those are the attributes which describe the state of the environment. In Fig. 4.7 the DQN architecture used in this work is illustrated. There is a hidden layer for each input vector followed by a concatenation layer and through two more hidden layers, it produces the output values. The output values are Q-values, for each agent and for each available agent's action. The system will pick the best action for each agent based on the Q-value that the DQN is outputting at every time-step. We want our DQN to gradually learn the Q-function in order to take advantage of the optimality principals that RL methods guarantee. We use the Q-learning updates of Eq. 2.13. Our goal for the DQN, is to eliminate any difference between the target and the prediction in the NN loss function (Eq. 2.17), for every possible state of the environment.

Additionally, we leverage the techniques of the experience replay in order to decouple the episodes from consecutive time-steps and produce uncorrelated results eliminating any time oriented bias. We use an epsilon-greedy (ϵ-greedy) strategy for action selection through the learning process in order to address the exploration vs exploitation dilemma [33]. In the next subsection, we describe our novel DQN architecture of Fig. 4.7 in detail.

For the development of the DQN architecture and algorithms we leverage the functionality of the Keras API. Keras is a high-level NN API, which is compatible with many open source platforms for ML applications, such as TensorFlow and Theano. It provides fast experimentation, easy prototyping and both CPU and GPU deployment. The Keras API provides a large collection of activation functions, loss functions, optimizers and metrics to choose from. It is fully compatible with

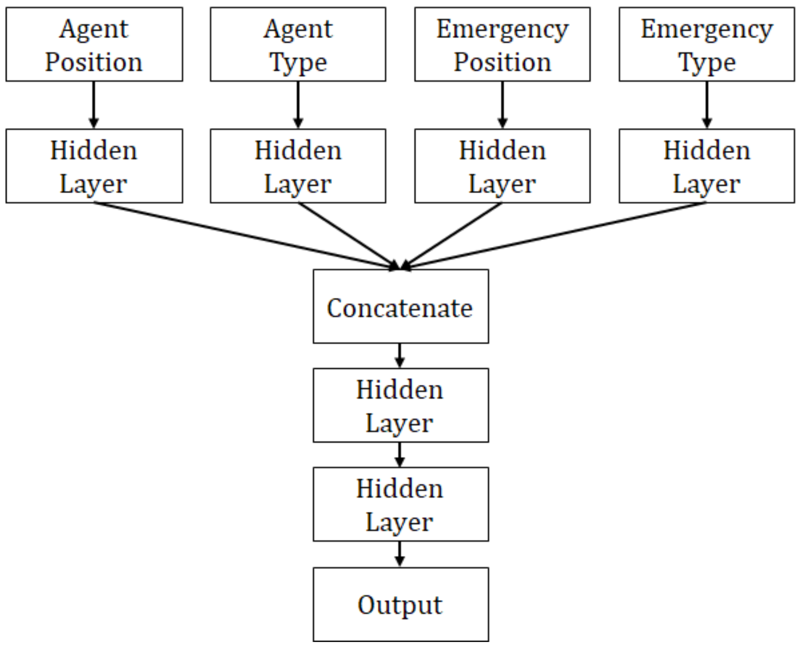


Figure 4.7: DQN Architecture with Functional model

Gym environments. The Gym and Keras combination is used by the majority of the state-of-the-art ML algorithms for training and testing. It was a challenge for us to develop our decision making component using these top-class platforms.

4.2.3.1 A Novel DQN Architecture

In the previous sections we discussed every aspect of the ML component construction. In this section we analyze the peculiarities of our ML system and how we overcome many challenges that Deep RL applications face when they get combined with MAS.

The main problem is that a typical DQN is developed for sinle-agent applications with a simple Deep NN construction. Therefore, we had to adjust the network to work for multi-agent settings. Additionaly, the DQN takes as input a one-hot state vector. A one-hot state vector is a binary vector that represents the state of the environment, the agent's possition. On this vector every value is zero, expect the one that represents the agent's possition. Fig. 4.8 illustrates the architecture of a typical DQN. If we try a similar approach for our environments that are multi-agent and contain many emergencies, it would most probably fail to be effective.

A one-hot state vector for this thesis' settings would be a one hot state 6D matrix. Each dimension will refer to an attribute of the state space. A 6D matrix built that way, would consist of the following dimensions, with the corresponding value range:

1. Number of agents
2. Number of possible agent's position (environment states)
3. Number of agent types

4. Number of emergencies
5. Number of possible emergency's position (environment states)
6. Number of emergency types

This means that on a typical ER situation at the Chania area, with 20 available agents of 5 different types, 7 emergencies of 2 different types, considering that the dataset that we use for the environment contains 35000 data points; if we use the one-hot state matrix as input for the DQN we would need:

$$\underbrace{20}_{\text{agents}} \times \overbrace{35000}^{\text{env states}} \times \underbrace{5}_{\text{agent types}} \times \overbrace{7}^{\text{emergencies}} \times \underbrace{35000}_{\text{env states}} \times \overbrace{2}^{\text{emergency types}} = 1.715 \times 10^{12} \quad \text{bits} \tag{4.1}$$

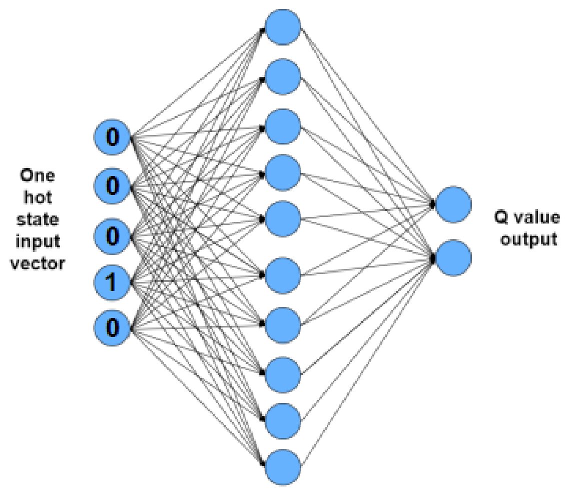


Figure 4.8: A typical DQN with an input example.

This means that the DQN would require 1.715×10^{12}bits $\approx$ 200GB of RAM just as input. This is not possible with the technology that was available to us for this thesis, so we altered the DQN model to overcome this major problem. Instead of using the typical sequential model of Fig. 4.9, we leverage on a "functional" model architecture. The functional model is a multi-input, multi-output NN model which Keras provides. For this thesis needs, we do not use the multi-output feature, nevertheless the multi-input feature gives us the opportunity to break down the input to separate matrices. Fig. 4.9 and Fig. 4.7 illustrate the main architecture differences of sequential and functional models.

The use of the functional model brings down the input size. However, this is not enough, the input vectors are still very large as soon as we use the one hot state represenation. The "Agent Position" input is: $20 \times 35000 = 700000$ bits $\approx$ 85kB.

We can utilize word embeddings techniques which are commonly used in Natural Language Processing (NLP), in order to manipulate the data differently. With this method, our goal is to reproject the data from the initial dimensions to fewer and

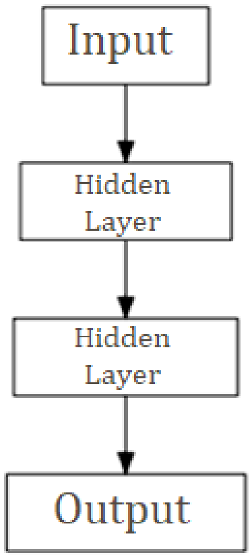


Figure 4.9: Sequential model

better distinguishable ones. Word embeddings reproject words to vectors based on a given vocabulary. We adapt this feature by considering an agent's position as a word and the possible environment state which the agent could be, as the vocabulary. We use this data manipulation for every functional model's input that is showed at Fig. 4.7. In this way we decrease the model's input size by a lot more, ensuring that we would not face any memory problem regarding our application. Fig. 4.10 shows the input attributes followed by the word embedding layer. This is the final form of our DQN architecture, and the one we use in our experiments.

With the novel combination of the techniques that we discuss above, we can address the memory complexity of our application. Another important parameter that defines a real-time system is its speed. Especially for a system which manages ER situations, the time taken to produce an action plan is essential. As such, the system should be able to handle large quantities of data in real-time. Unfortunately, we did not have the resources to make sure that our system will always have the desired computational power; and it is expected that many ER organizations' system in the real world lack sufficient computational power also. It is much preferable to build a system that can run on ”everyday” devices instead of relying on access to exceptionally strong servers. Thus, we developed two solutions for this problem.

The first one requires a preprocessing operation for the system's DQN. We could use a synthetic environment like those we discuss in Section 4.2.2. We train the DQN with various combinations regarding the number of the agents, the agents' types, the number of emergencies and their types as well. After every training session we save a large number of the DQN weights/parameters, which are uncorrelated with the input/output vectors' size, and load them back at the next training session. In this way we can combine the training results for the combinations we feed the DQN to learn. We can pre-train and save almost 70% of all the parameters and load them back again for a real-time scenario usage, without any need for training. Therefore, we have to train only the 30% of the DQN parameters at real-time. This procedure

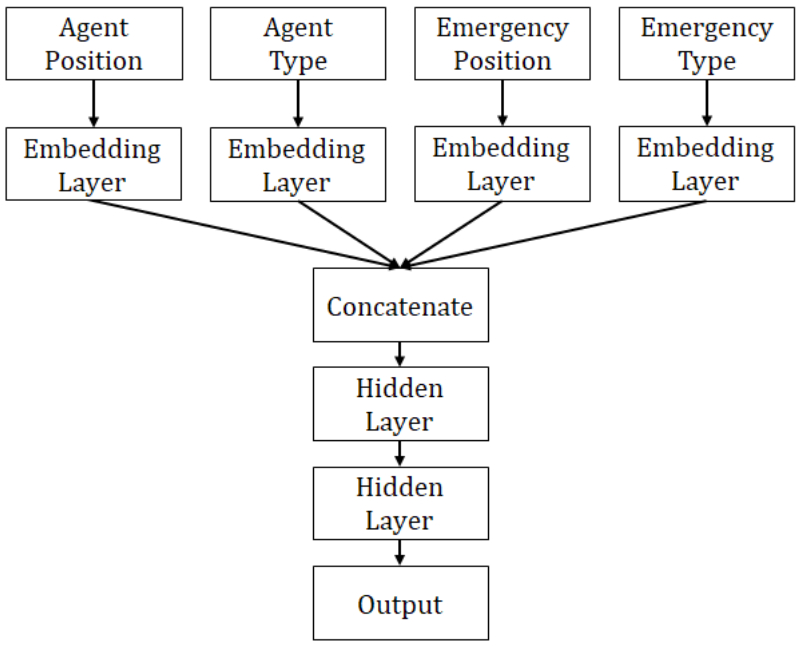


Figure 4.10: Functional model with word embedding layer on the DQN.

could benefit our system's performance very much. The layers which are pre-trained are circled in Fig. 4.11.

The second solution we developed in order to minimize the required computational power is using fully real-time training but eliminates a part of the environment; as such, it probably not constitute a part of the optimal action plan. However, there is always a tradeoff between optimality and the need to perform only real-time training and decision making with the resources we have at hand.

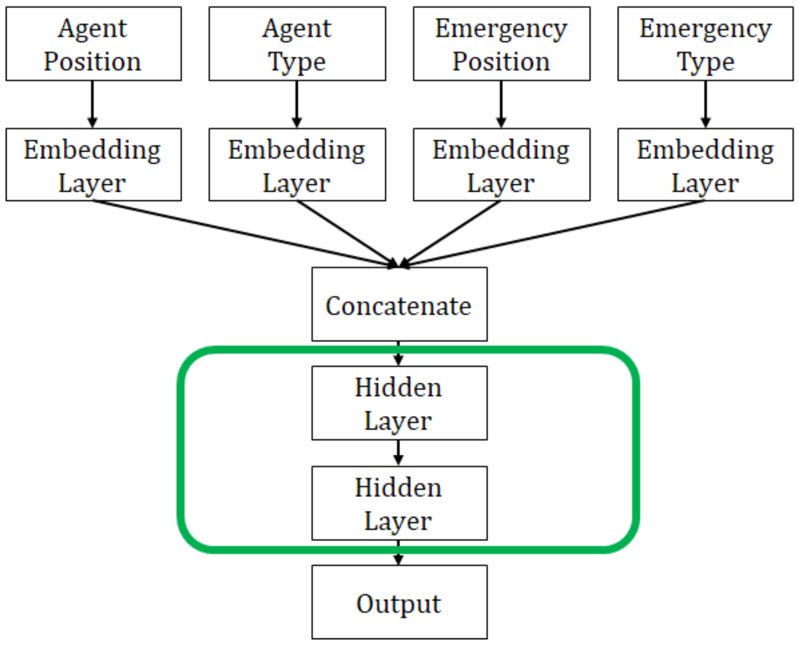


Figure 4.11: The saved parameters of the functional model on the DQN.

Chapter 5

Experimental Evaluation

In this chapter, we begin with a description of the hardware and software setup that we used to evaluate the functionality and the performance of the system, and the optimality regarding the ML decision making component. We discuss the reward function, the ways and the attempts of building one which can help the system utilize fully every agent's abilities. The system's infrastructure is of great significance because it affects the performance of every other feature. However, the most important aspect of the evaluation refers to the decision making optimality of the DQN that we developed.

5.1 Experimental Setup

The best and most precise way to evaluate our system is by using real disasters data. This is not possible due to the complete absence of datasets describing any disaster with the details our system dictates to be provided. In order to test and evaluate our system's performance we have to synthesize several ER scenarios.

The synthetic scenarios, that we developed, refer to two different environment types. In Section 4.2.2, we detailed the differences of these two types. This way we can categorize the scenarios to those which use the Chania dataset as the environment, and those which use only synthetic data (randomly generated) as the environment.

For the fully synthetic environments, we are able to alter the following parameters:

1. N_e: Number of environment's positions, $N_e \in \mathbb{N}$
2. N_a: Number of agents, $N_a \in \mathbb{N}$
3. T_a: Type of every agent, $T_a \in \{Firemen, Transporters, Medics, Soldiers, UAVs\}$
4. N_d: Number of disasters, $N_d \in \mathbb{N}$
5. T_d: Type of every disaster, $T_d \in \{Fire, Earthquake, Flood\}$

We use the first parameter, we construct a randomly generated graph, which represents the environment. On this graph, we place every agent in a node randomly and it gets assigned its corresponding type. Disaster points are placed and assigned with a type, randomly as well. These fully synthetic environments can be parametrized easily, so we can construct, test and evaluate many environments, using either a computationally weak infrastructure for small environments or a powerful computer system to handle large environments. Typically, we run all scenarios with small environments exclusively on the hosting server. In Table. 5.1 we provide the hardware specifications of the online hosting server, among two other testing computer units. The personal computer which used to conduct some experiments as well and powerful computer in SenseLab to run every large scale scenario.

Regarding the real world environment and the Chania dataset, we had to add a lot of computational power in order to conduct experiments with scenarios using the whole Chania dataset or a part of it. We used a personal computer to handle scenarios which include only a part of the dataset and the server located in TUC SenseLab for scenarios which make use of the whole dataset. Hardware specification on both the PC and SenseLab's Server are in Table 5.1.

Hardware Specifications				
Components		PC	Online Server	SenseLab Server
CPU	(Model)	Intel i7-5500U	QEMU v2.1.2	Intel i7-9700K
Speed	(GHz)	2.40	2.10	3.6
Cores	(logical)	4	2	8
RAM	(Manuf.)	Samsung	QEMU	HyperX
Capacity	(GB)	8	4	32
Speed	(MHz)	1600	Unknown	3200
GPU	(Model)	GTX-950M	None	RTX-2080
Speed	(MHz)	914	-	1800
Memory	(GB)	2	-	8
Cuda Cores	(#)	640	-	2944

Table 5.1: Hardware details of three machines used for experimental evaluation.

5.2 Building the Reward Function

Civil Protection Agencies around the world have proposed many meaningful action plans in order to deal with ER situations. The coordination of peoples' actions alongside with the police, medics, firemen and military are in the spotlight of every action plan. We tried to build a reward function which conforms with the protocols that Civil Protection Agencies created and follow. Many attempts were made in order to build a stable reward function which could leverage the important features that our DQN can provide.

The main goal for a reward function on a MAS is to compel the agents to take actions which can benefit not only them, but the team they belong to as well. Thus, we experimented with building several reward functions, attempting to incorporate the collective rewards according to the needs of the situation and the collaboration of the agents. The proper reward function for our application is based on three key aspects. The first aspect that we evaluated was the realistic point of view for the reward function. We build the reward functions with reference to the Civil Protection Agencies's protocols, this way we know that the DQN's action plan could fit into the protocols' specification and take advantage of the protocols' peculiarities. For example, a commonly proposed tactic in ER is first deal with the extrication of the people and prioritize everything around it. Inspired by this, we provide the firemen greater reward than anyone else if they show up first at a disaster point.

Secondly, we needed to address the collaboration between the agents. We mapped every interaction within the agent groups with a value regarding to their ability of collaborating with each other. For example, a UAV can be very useful for monitoring and reporting the status of an area while a major disaster is taking place; on the other hand it can be almost useless when at this area there are firemen or military troops near by which can report as well and evaluate the status better by being there. Therefore, the reward function has to deter the UAV to go on an area which is populated by other agent groups and impel it to go monitor another area of interest where no other agent is there. This is possible by giving the UAV a large and small reward respectively.

The third aspect of the reward function regards the type of disasters. We had to build separate parts of the reward function in order to integrate all the possible disasters that our application can cover. We mapped the agents' interactions regarding the disaster that is taking place in an area of interest. This way we can differentiate the collaboration patterns between the agent groups and handle more than one disaster at the same time.

The building of the reward function is very important as it can change the whole logic behind the DQN's action plans and affects the performance and efficiency of the system. It is also very difficult to fully adapt in the protocols due to their generality on many issues and their absolute trust on the commanders of every team force they include in their plans. We needed exact values and weights for every agent and all its possible actions as well as its collaboration with other agent groups, that none protocol could provide. The reward function is not continuous, we used tables to represent the reward function and in the next sections we utilize them to test and evaluate the ML component. In Appendix Listing A.1 we provide the reward table regarding three disaster types in Python programming language.

5.3 Experiments and Results

The system's infrastructure and the ML component are the two pillars of this thesis. The most important parameters in order to evaluate the infrastructure's quality are the system's stability and proper functionality. In this thesis, as long as we make sure that the system is stable and functional, we focus the experimental results on the ML optimality using the novel DQN.

We discussed, at section 4.2.3.1, how we could manage the lack of computational power that the combination of DQN and MAS demands, in order to produce near optimal action plans for ER situations.

The DQN evaluation includes three points of interest:

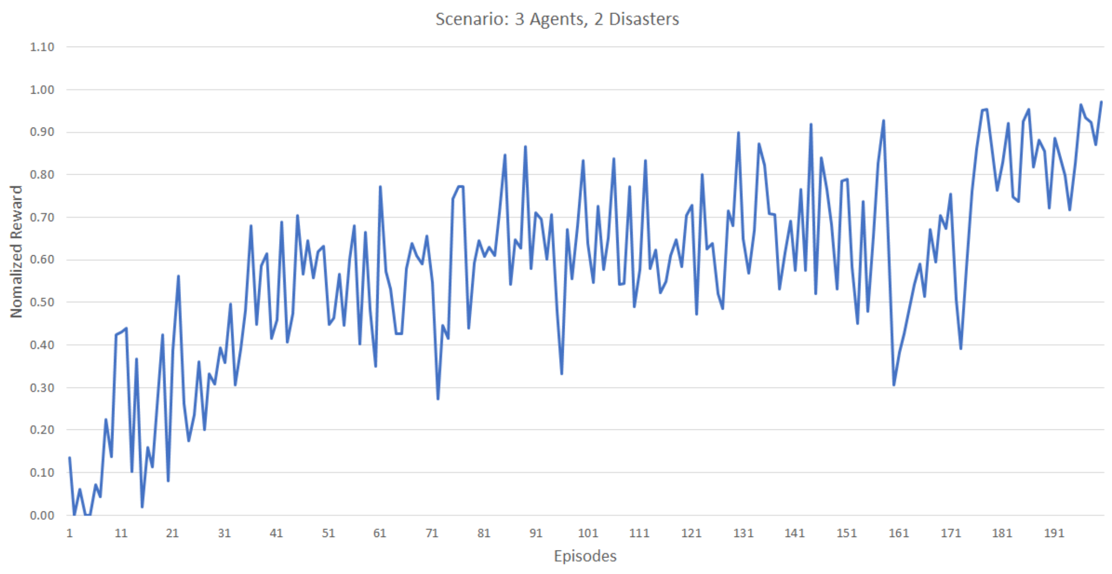


Figure 5.1: Rewards obtained by 3 agents with 2 disasters on synthetic environment, normalized w.r.t. the optimal.

1. Collective Reward, during the action plan that the DQN produced
2. Normalized Reward, regarding the optimal reward and the Collective Reward
3. Step Count, that the agents take following the action plan

The most significant part on the DQN evaluation is the optimality level of the action plan it produces. Due to the data and environment specifications, we are able to calculate the optimal values for some simple scenarios. We use a brute-force algorithm to calculate the optimal reward by exploring all the possible states of the environment. In this way we can find optimal for lightweight scenarios only, because in large scale scenarios the number of states in environment is massively high because of its exponential growth over the state's attributes. Thus, we are able to find the optimal values on scenarios that include a small number of agents and disasters. A future addition could be an implementation of different algorithms to find the optimal in every scenario, or even approximate its value.

The first type of scenarios that we tested is about the optimality over the action plan that the DQN produced for a scenario on a synthetic environment. We use the optimal and the collective reward to calculate the normalized reward:

$$normalized = \frac{collective}{optimal}. \tag{5.1}$$

Using a synthetic environment in Fig. 5.1 we provide a graph based on 200 training episodes for a scenario which includes 3 agents (1 fireman, 1 medic, 1 transporter) and 2 disasters (1 fire, 1 earthquake). We see that by the end of the training, the reward from the action plan that the DQN produced almost reached the optimal one, picking at a value of 0.97.

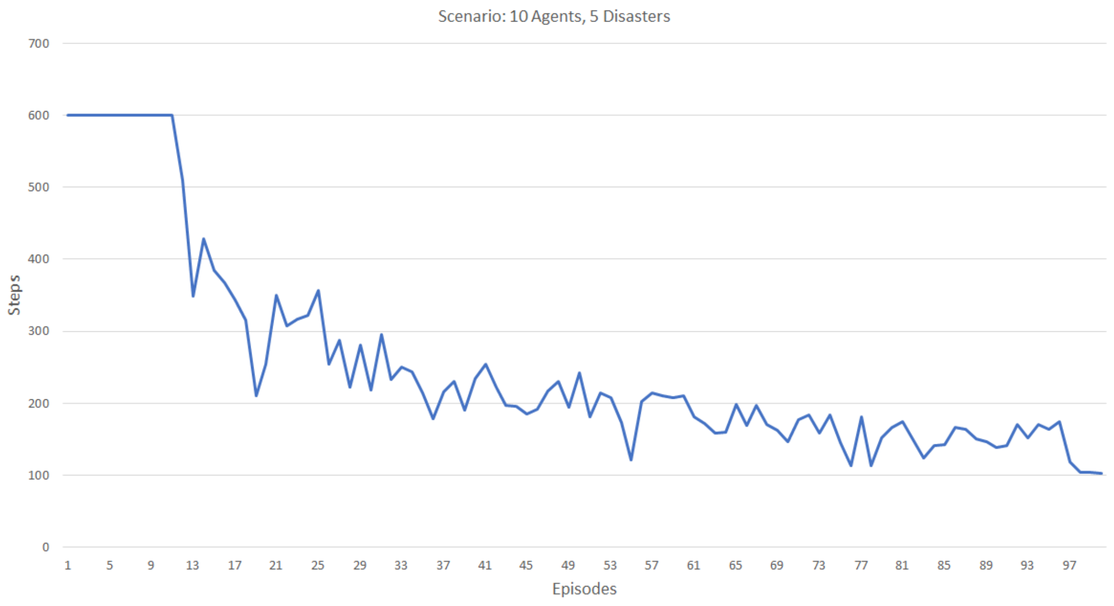


Figure 5.2: Steps needed to reach a goal state for every agent, with 10 agents and 5 disasters, on synthetic environment data.

The second type of scenarios that we tested is about the steps (number of actions) that the agents took in order to reach a goal state on the environment. In Fig. 5.2 we provide a graph based on 100 training episodes for a scenario on a synthetic environment with 10 agents (random types) and 5 disasters (random types). This graph shows the maximum steps that were needed so that every agent has reached a goal state (a disaster position). Concerning about the execution time, we capped the steps at 600 for every agent, meaning that every agent could take up to 600 steps/actions before reaching a goal state of the environment. For the first 10 episodes many agents did not manage to reach a goal state. After that we can see that the step count goes down till the last 2 episodes that it dropped and stabilized at 107. The consecutive decrease of the step count through the training indicates the learning of the DQN. Unfortunately, we cannot know if this is the optimal number of steps due to the large state space that this scenario creates.

The third type of scenarios that we tested is on the Chania dataset and refers to the cumulative rewards that the agents obtain by following the action plan which the DQN produced. In Fig. 5.3 we provide a graph based on 500 training episodes for a real world scenario with 10 agents and 5 disasters. This graph shows the collective reward of the agents through the training session. There is a large fluctuation on the reward values due to the large negative rewards that we implement in the reward function. Despite the variance between episodes, there is a overall increase on the reward indicating once again that the DQN can adapt in a scenario like that and learn over time.

Another real world scenario that we tested includes 8 agents and 3 disasters. We used the Chania dataset to simulate a real time and the two aspects that we tested are the steps which took for all the agents to reach a goal state, and the collective reward for the agents. The overall decrease of the steps count as the number of

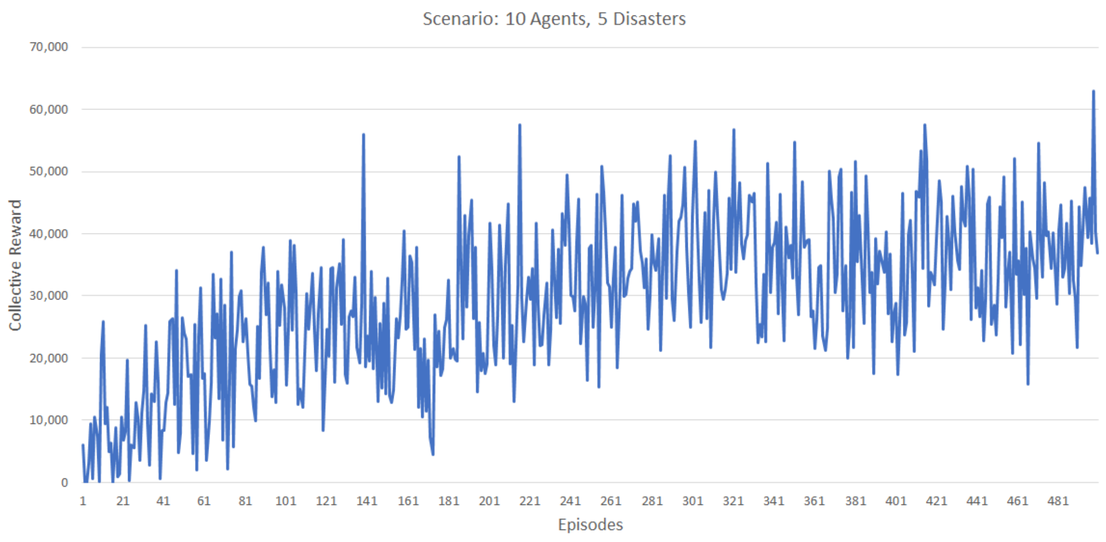


Figure 5.3: The collective reward obtained by the agents in real time scenario, with 10 agents and 5 disasters, on Chania Dataset.

training episodes increasing is illustrated in Fig. 5.4. The collective reward of the agents on this scenario is showed in Fig. 5.5. Despite the fluctuations on the step and reward values during these training episodes, both graphs indicate once again that the DQN adapts and learns over time.

We also provide the full parameter table, regarding N_e, N_a, T_a, N_d, T_d as mentioned at Sec. 5.1, for all the figures and the corresponding experiments on the Appendix in Table A.1.

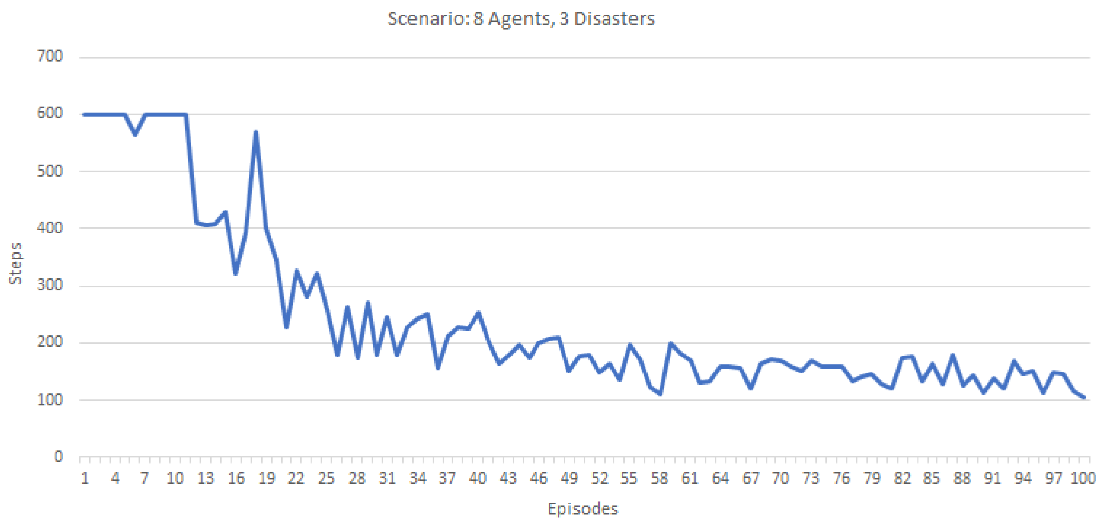


Figure 5.4: Steps needed to reach a goal state for every agent in real time scenario, with 8 agents and 3 disasters, on Chania Dataset.

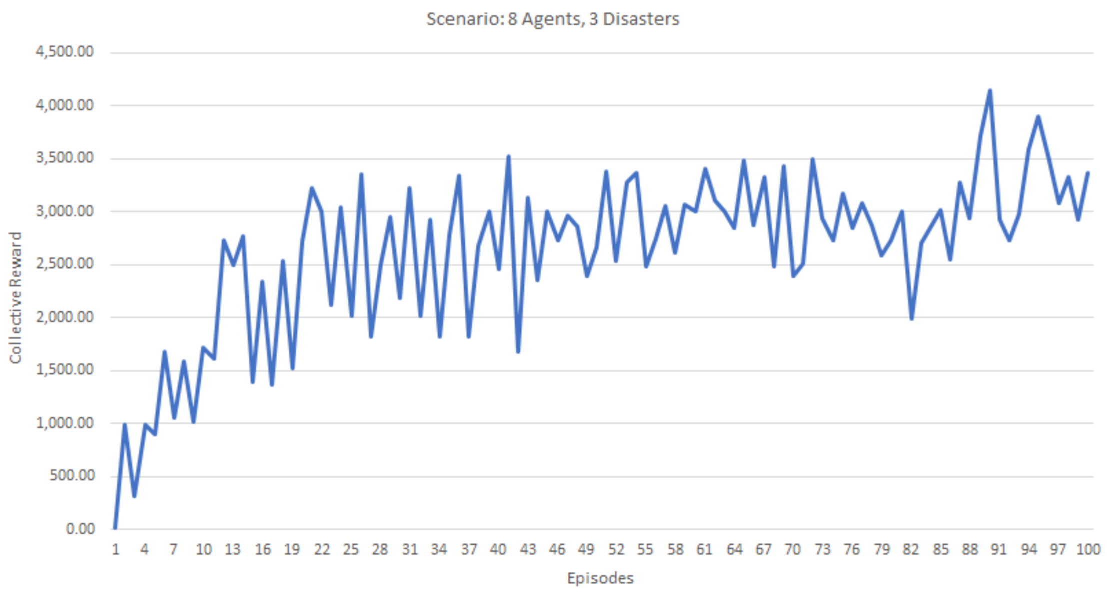


Figure 5.5: The collective reward obtained by the agents in real time scenario, with 8 agents and 3 disasters, on Chania Dataset.

Chapter 6

Conclusions and Future Work

In this thesis we developed a novel ER management system which is a very useful and vital tool on major disasters, such as earthquakes, fires or floods. After the development of the hosting server on a online and remote location, using the web page and the mobile application, that we created utilizing a ReST API, we could build ER scenarios and examine the functionality of the system, as well as the effectiveness of our decision making algorithms.

We managed to build ER scenarios by implementing multi-agent environments on OpenAI Gym. The custom environments, which we could dynamically create, were essential for Keras framework to work on, in order to try several Deep RL algorithm implementations. Using Keras, we were able to develop a large variety of these algorithms with the intention of making a DQN that could serve us, as the decision making component, in several Search and Rescue operations. The DQN that we developed can handle both synthetic and real world data environments. On a scenario with real world data, some preprocessing was necessary in order to reproject the data on the Chania's GIS dataset.

The large state space of the environment's MDP was a serious problem which we manage to overcome using a set of optimizations. We implemented a different input processing model for our DQN, which helped reduce the input data size. With the focus on input data size, we also added a word embeddings layers in the DQN architecture, which helped with the reduction as well. These two additions could not provide the ability to handle ER situations in real-time for large scale scenarios. Thus, we developed a optional pre-train method which can reduce the amount of computational time needed to produce a real-time action plan. This is possible after a number of pre-training sessions on the dataset and can provide a huge reduction of the training parameters.

The experimental evaluation showed that the DQN produced meaningful action plans, with the ability to learn over time and gradually increment the cumulative reward of the agents through the training session. We expect that with more parameter tuning the DQN could provide better action plans for every agent. Accomplishing this requires a lot of testing, due to the large number of the parameters that can be tuned. A better infrastructure and computational environment for testing are essential for the system, and the DQN especially, to reveal its maximum potential. In order to do that, other algorithms could be implemented to adapt in this type of environment and scenarios.

A different approach to ER situations could utilize online algorithms for decision making. Online algorithms have the potential to perform very well and are able to provide live updates generating new and improved plans ”on the fly”. An online extension of our DQN approach is feasible due to the efficiency of the neural networks and could probably produce even better action plans.

Furthermore, there are approaches in the algorithmic game-theoretic and advanced search techniques literature that could be of value to our problem. For instance, the Cooperative Game-Theoretic approach as Bistaffa at el. [5] using Graph-Constrained Coalition Formation could provide a baseline for decision making problems like ER situation management.

Comparing the results of the other approaches and algorithms, which can quarantine optimal or near optimal solutions, with the action plan our DQN provides, would allow us to determine the optimality level that the DQN performs for any given input data range, either large or small scale.

Bibliography

[1] David Barkai. *Peer-to-peer computing : technologies for sharing and collaborating on the net.* Intel Corporation, 2002.

[2] Richard Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 1957.

[3] T. Berners-Lee, R. Fielding, and L. Masinter. *Uniform Resource Identifiers (URI): Generic Syntax.* RFC Editor, 1998.

[4] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, 2000.

[5] Filippo Bistaffa, Alessandro Farinelli, Georgios Chalkiadakis, and Sarvapali D. Ramchurn. A cooperative game-theoretic approach to the social ridesharing problem. *Artificial Intelligence*, 2017.

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *CoRR*, 2016.

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

[8] Georgios Chalkiadakis. Multiagent Reinforcement Learning: Stochastic Games with Multiple Learning Players. 2003.

[9] U. W. Chohan. Cryptocurrencies: A Brief Thematic Review. 2017.

[10] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1.* RFC Editor, 1999.

[11] Roy Thomas Fielding and Richard N. Taylor. *Architectural Styles and the Design of Network-Based Software Architectures.* PhD thesis, University of California, Irvine, 2000.

[12] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual Multi-Agent Policy Gradients, 2017.

[13] Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. Learning to Communicate to Solve Riddles with Deep Distributed Recurrent Q-Networks. *CoRR*, 2016.

[14] Jesse James Garrett. Ajax: A new approach to web applications. 2007.

[15] Matthew J. Hausknecht and Peter Stone. Deep Recurrent Q-Learning for Partially Observable MDPs. *CoRR*, 2015.

[16] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2014.

[17] Stephen Cole Kleene. Representation of events in nerve nets and finite automata. *RAND Corporation*, 1951.

[18] Jelle R. Kok and Nikos Vlassis. Collaborative multiagent reinforcement learning by payoff propagation. *J. Mach. Learn. Res.*, 2006.

[19] Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. In S. A. Solla, T. K. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12*. MIT Press, 2000.

[20] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. *CoRR*, 2017.

[21] J. Martin, S.J. James Martin, and J.T. Martin. *Managing the Data-base Environment*. Prentice-Hall, 1983.

[22] F. S. Melo. Convergence of q-learning: a simple proof. *Technical Report*, 2001.

[23] M. Memon, S. S. Hussain, U. A. Bajwa, and A. Ikhlas. Blockchain beyond bitcoin: Blockchain technology challenges and real-world applications. In *International Conference on Computing, Electronics Communications Engineering (iCCECE)*, 2018.

[24] T.M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[25] V. Mnih, K. Kavukcuoglu, and D. Silver et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, 2013.

[27] Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. *IEEE Transactions on Cybernetics*, 2020.

[28] M.G. Lagoudakis R. Parr. Least-squares policy iteration. *Journal of machine learning research*, 2003.

[29] W.S. McCulloch W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics 5*, 1943.

[30] Sarvapali D. Ramchurn, Trung Dong Huynh, Feng Wu, Yuki Ikuno, Jack Flann, Luc Moreau, Joel Fischer, Wenchao Jiang, Tom Rodden, Edwin Simpson, Steven Reece, Stephen Roberts, and Nicholas R. Jennings. A disaster response system based on human-agent collectives. *Journal of Artificial Intelligence Research*, 2016.

[31] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 1959.

[32] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *CoRR*, 2014.

[33] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, Cambridge, Massachusetes, 1998.

[34] Roger F. Tomlinson. A geographic information system for regional planning. *Journal of Geography*, 1969.

[35] A. M. TURING. Computing machinery and intelligence. *Mind*, 1950.

[36] M. Waltz and K. Fu. A heuristic approach to reinforcement learning control systems. *IEEE Transactions on Automatic Control*, 1965.

[37] Christopher John Cornish Hellaby Watkins. *Learning from Delayed Rewards.* PhD thesis, King's College, 1989.

Appendix A

Experimental Values

Parameters in Sec. 5.3							
Figures	Testing	Data Type	N_e	N_a	T_a	N_d	T_d
Fig. 5.1	Normalized Reward	Synthetic	5000	3	1 f, 1 m, 1 t	2	fire, earthquake
Fig. 5.2	Steps	Synthetic	10000	10	Random	5	3 fires, flood, earthquake
Fig. 5.3	Reward	Chania Dataset	32000	10	5 f, 3 m, 1 t, 1 s	5	3 fires, flood, earthquake
Fig. 5.4, Fig. 5.5	Steps, Reward	Chania Dataset	20000	8	3 f, 2 m, 2 t, 1 u	3	2 fires, earthquake

Table A.1: Detailed parameters regarding the experiments.

```
######################## Disaster: Fire ########################

fire[0] = 500              # Only Firemen
fire[1] = 200              # Only Medics
fire[2] = 50               # Only UAV
fire[3] = 150              # Only Transporters
fire[4] = 150              # Only Soldiers

fire[0, 1]    = 2000          # Firemen - Medics
fire[1, 3]    = 1000          # Medics - Transporters
fire[0, 4]    = 1500          # Firemen - Soldiers
fire[1, 2]    = 0             # Firemen - UAV
fire[2, 3]    = 0             # UAV - Transporters
fire[2, 4]    = 0             # UAV - Soldiers
fire[3, 4]    = 1000          # Transporters - Soldiers

fire[0, 1, 3] = 5000          # Firemen - Medics - Transporters
```

```
fire[0, 1, 4] = 3000          # Firemen - Medics - Soldiers
fire[1, 3, 4] = 1500          # Medics - Transporters - Soldiers

####################### Disaster: Earthquake #######################


earthquake[0] = 500              # Only Firemen
earthquake[1] = 100              # Only Medics
earthquake[2] = 50               # Only UAV
earthquake[3] = 200              # Only Transporters
earthquake[4] = 150              # Only Soldiers

earthquake[0, 1]    = 1500          # Firemen - Medics
earthquake[1, 3]    = 1000          # Medics - Transporters
earthquake[0, 4]    = 2000          # Firemen - Soldiers
earthquake[1, 2]    = 0             # Firemen - UAV
earthquake[2, 3]    = 0             # UAV- Transporters
earthquake[2, 4]    = 0             # UAV- Soldiers
earthquake[3, 4]    = 1500          # Transporters - Soldiers

earthquake[0, 1, 3] = 5000          # Firemen - Medics - Transporters
earthquake[0, 1, 4] = 4000          # Firemen - Medics - Soldiers
earthquake[1, 3, 4] = 1500          # Medics - Transporters - Soldiers

####################### Disaster: Flood #######################

flood[0] = 500              # Only Firemen
flood[1] = 100              # Only Medics
flood[2] = 50               # Only UAV
flood[3] = 200              # Only Transporters
flood[4] = 200              # Only Soldiers

flood[0, 1]    = 2000          # Firemen - Medics
flood[1, 3]    = 1000          # Medics - Transporters
flood[0, 4]    = 2000          # Firemen - Soldiers
flood[1, 2]    = 0             # Firemen - UAV
flood[2, 3]    = 0             # UAV- Transporters
flood[2, 4]    = 0             # UAV- Soldiers
flood[3, 4]    = 1000          # Transporters - Soldiers

flood[0, 1, 3] = 5000          # Firemen - Medics - Transporters
flood[0, 1, 4] = 4000          # Firemen - Medics - Soldiers
flood[1, 3, 4] = 2000          # Medics - Transporters - Soldiers
```

Listing A.1: Reward Table in Python for 3 types of disasters: fire, earthquake, flood