Multi-agent Energy Sharing in Zero Energy Communities

Amit Umashankar Prasad

A dissertation submitted to the University of Dublin,
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science (Intelligent Systems)

2018

Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work, and has not been submitted as an exercise for a degree at this or any other university.

Signed: _____

Amit Umashankar Prasad

26/08/2018

Permission to lend and/or copy

I agree that the Trinity College Library may lend or copy this dissertation upon request.

Signed: _____

Amit Umashankar Prasad

26/08/2018

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my supervisor, Prof. Dr. Ivana Dusparic for her continuous support, guidance, encouragement and expertise during my Master thesis.

I would also like to thank Dr. Andrei Marinescu for writing such an excellent Ph.D. thesis that I have referred on several occasions.

I would like to thank Anukrity for always encouraging me and being there with me throughout my Masters, and without whom I would have never been able to complete my thesis.

Finally, I would like to thanks my friends and family, for all their support and love.

Amit Prasad

University of Dublin, Trinity College

August 2018

Summary

With advances in self-sustainability, energy users are becoming more self-sufficient due to usage of self-produced energy from renewable sources. Buildings that partially or entirely rely on renewable sources are increasing. If the net energy usage of such buildings over a period of one year is zero, they are called as zero energy buildings (ZEB). A collection of such ZEBs forms a zero energy community (ZEC). Absolute net zero status is difficult to achieve and hence the term *nearly* ZEC (nZEC) is used in practice. We address the problem of energy sharing in such a community. This is different from conventional energy sharing between buildings as the focus is on improvement of community energy status instead of reducing losses due to transmission and/or storage, or economic gains. We model this problem in a multi-agent environment and propose a deep reinforcement learning (DRL) based solution. Each building is represented by an intelligent agent that learns over time the appropriate behaviour to share energy. We evaluate our solution in a self-made multi-agent simulation built using osBrain. Results indicate that, with time agents learn to collaborate and learn a policy comparable to the optimal policy in most cases, which in-turn improves the nZEC's energy status. When compared this with a no-energy-sharing environment an improvement of 40 kWh with 3 houses during winter and over 60 kWh with 4 houses during summer over 3 days in the overall community's energy balance was found. Similarly, with 10 houses an improvement of 97 kWh during Winter and 156 kWh during Summer was found. Buildings with no renewables preferred to request energy from their neighbours rather than the supply grid. Energy distribution amongst peers introduced a greater deficit in batteries of buildings, which indirectly contributed higher renewable energy storage.

Abstract

With advances in self-sustainability, energy users are becoming more self-sufficient due to usage of self-produced energy from renewable sources. Buildings that partially or entirely rely on renewable sources are increasing. If the net energy usage of such buildings over a period of one year is zero they are called Zero Energy Buildings (ZEBs). A zero energy status is difficult to achieve and hence in practice the term *near* ZEB (nZEB) is used. This concept when extended to a group buildings is called as a near Zero Energy Community (nZEC). A typical nZEC is a micro-grid that has distributed energy generation, storage, delivery and consumption with overall energy balance close to zero. There is high uncertainty of energy production in a nZEC due to distributed generation that causes some buildings to produce insufficient energy with respect to their energy demand. Such buildings then request for additional energy from the supply grid at a higher cost. As an alternative to this, buildings can request for additional energy to nearby buildings. This reduces the losses due to transmission and promotes green energy. Moreover, buildings with surplus energy may also benefit economically by sharing with neighbors. Such behavior helps to achieve near zero energy status as a community.

There have been previous works that focus on energy sharing between buildings but very little work has been done on intra-nZEC energy sharing. Previous works focus on energy sharing for economical gains, reducing transmission and storage loses and thereby reducing carbon footprints at individual building level. Although effective, such behavior gives rise to competition which in-turn can lead to grid-destabilization. On the other hand, energy sharing in nZECs considers sustainability at the community level. This means, buildings must leave their selfish goals to help other buildings, sometimes, at the expensive of their own loss to ensure community sustainability. Existing works on intra-nZEC energy sharing consider a community of only nZEBs, whereas in the real-world generally it is a mix of buildings producing renewable energy as well as those that are not. Additionally, these strategies expose consumer consumption and generation patterns to

the entire community using which selfish policies can be adapted by individual buildings to boost their economic gain at the expense of sustainability. nZEC environments are complex and non-stationary due to autonomy of individual buildings.

This thesis addresses the problem of energy sharing between residential buildings to achieve a nZEC status. We model this community as a multi-agent environment where each individual agent represents a building. There are various techniques such as rule based algorithms, evolutionary algorithms and reinforcement learning (RL) which can be used to solve this problem. RL is preferred in decentralized systems and does not require a predefined model of the environment. Agents can learn the optimal behavior using a RL technique called approximate Q learning. Approximate Q Learning works efficiently by avoiding the entire state space exploration with very low inference time. Integration of artificial neural networks as the approximation function with RL is called Deep Reinforcement Learning (DRL). DRL agents have been known to learn the optimal behaviour even in the most difficult problems in the recent years. We therefore propose a DRL based solution to optimize energy sharing between residential buildings to achieve nZEC status.

We have evaluated our approach in a self-made multi-agent simulation built using osBrain. Thorough evaluation revealed interesting results in different scenarios. We show that, with time agents learn to collaborate and learn a policy comparable to the optimal policy in most cases, which in-turn improves the nZEC's energy status. When compared this with a no-energy-sharing environment major differences were found. An improvement of 40 kWh with 3 houses during winter and over 60 kWh with 4 houses during summer over 3 days in the overall community's energy balance was found. Similarly, with 10 houses an improvement of 97 kWh during Winter and 156 kWh during Summer was found. Buildings with no renewables preferred to request energy from their neighbours rather than the supply grid. Energy distribution amongst peers introduced a greater deficit in batteries of buildings, which indirectly contributed higher renewable energy storage.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis addresses the problem of energy sharing between buildings empowered with generation from renewable sources to foster a nearly Zero Energy Community (nZEC). The environment is modeled as a multi-agent system wherein each agent represents a single building. We propose a hybrid online Deep Reinforcement Learning (DRL) based solution that enables agents to learn the optimal policy to achieve a nZEC status. DRL agents exploit the optimal policy to select appropriate actions, in order to improve the community status as well as their own local state. We compare the results our proposed solution against an ideal scenario where all agents act optimally and a non energy sharing environment. This chapter firstly introduces the application domain - nZEC. We then discuss the motivation behind this work, outline the main contributions and lastly present a roadmap for the remainder of the thesis.

1.1 Nearly Zero Energy Communities

Advances in renewable energy generation, lower cost of hardware involved and increased storage capacity has made energy users more and more self-sustainable in the recent years. Users that partially or entirely rely on renewable sources are increasing. A *user* maybe defined as a single home, a building or even a grid. In this thesis we use the term *user*

to refer a building or a independent house. If the net energy usage of a building over time (generally one year) is zero it is called as a Zero Energy Building (ZEB). However, practically achieving a exact net zero status is difficult, and hence the term *nearly* ZEB (nZEB) is used. The definition of nZEB states that –

> The net balance between export and import of energy over a period of time must be zero or even positive [84].

$$|export| - |import| >= 0 \tag{1.1}$$

nZEBs are receiving increased attention in the recent years due to increasing dependency and thus pressure on non-renewable sources of energy. European Union(EU) has therefore stated in its 2010 [77] recast that by year 2020 all new buildings shall have to consume nearly zero energy. Similarly, the Japanese government too has set the same target for 2030 [40]. To achieve this there are two different approaches. One approach considers the renewable generators (PV devices, windmill) either installed on the building or on the ground attached directly to it. The second approach considers the generators placed outside the premises of the building on a land typically owned the building owner. Formally these approaches are called on-site and off-site renewable energy sources [55].

This concept of nZEB when extended to group of buildings is called a near Zero Energy Community (nZEC). The net energy status of such a community is nearly zero over a time period. [51] introduces the concept of cooperative nZEC (CNet-ZEC) and defines nZEC as a collection of *only* nZEBs having its annual energy balance is zero. However, this might not be true. The definition above describes a *nZEB community*, as opposed to a nZEC. We maintain a subtle distinction between a nZEB community and a nZEC. The former is a community in which *all* buildings satisfy the nZEB status whereas in the latter some buildings might not, but as a group they have near net zero energy balance. All nZEB communities are nZEC's but the reverse might not be true. We define nZEC formally as –

A micro-grid that has distributed generation, storage, delivery and consumption having net annual energy balance as nearly zero.

Figure 1.1: (a) A nZEC community consisting of only nZEBs. (b) A nZEC community having buildings with varying generation capacity.

An conceptual illustration of a nZEC is shown in figure 1.1. We consider this definition of nZEC throughout this thesis.

1.2 Motivation

There is high uncertainty of production due to various external factors like solar irradiance, wind speed, sky condition, shade, etc. and internal factors like the efficiency of the hardware used [42, 59]. This uncertainty causes some buildings to produce insufficient energy during different times of the day. Hence, to suffice their energy request buildings demand for additional energy from the *supply grid*. As an alternative, a building may request for additional energy to nearby buildings. This reduces the loses due to transmission as well as the pressure on non-renewable sources of energy. However, this might cause some donor buildings to incur a energy deficit and thus affecting their nZEB status negatively due to uncertainty in production and therefore discourages energy sharing. Nevertheless, as the overall goal is self-sustainability we can up-scale this problem to a group of buildings to

achieve *grid* or *community sustainability*. This can be done by incentivizing energy sharing between buildings with the ultimate goal of achieving community sustainability. Solving the problem on a bigger scale ensures greater sustainability. This community as defined in section 1.1 is called nZEC.

There have been previous works that focus on energy sharing between buildings but very little or no direct work has been done on intra-nZEC energy sharing. Previous works on only energy sharing between buildings focus on economic gains [35, 87, 110], optimizing transactions [92, 112] and thereby reducing transmission and storage loses at the individual building level. These works have further been extended to sustainable microgrids without consideration of nZEC status using traditional programming techniques [35] or modern techniques [43]. There are very few works that particularly focus on nZEC but in a community of only nZEBs [32, 51]. However, in reality very few communities have all buildings exhibiting *energy autonomy*.

nZEC can be modeled as a multi-agent system. Such systems are complex due to the non-stationarity introduced by multiple actors on the environment. To achieve autonomy, often, agents are trained using a learning technique called reinforcement learning. An enhanced variation of reinforcement learning called as deep reinforcement learning (DRL) has proven to be very effective to solve problems with non-linear behaviors in the recent times [57, 58, 102]. Agents can learn the optimal behavior for energy sharing by trail-and-error. This further motivates the research question presented in the next section.

1.3 Thesis Aims and Objectives

The discussion in the previous section motivates the research question addressed by this thesis - *Can deep reinforcement learning (DRL) be used to optimize energy sharing between multiple buildings to enable it to be a zero-energy community?*

This thesis models the environment as a multi-agent system wherein each building is represented by an individual agent which is responsible for handling all transactions on

behalf of that building. As discussed in the previous section energy systems are complex due to stochastic nature of renewable energy generation and non-stationarity in the environment. Additionally, decisions in such systems needs to be taken quickly as they operate in real-time. Based on these factors, a solution for this needs to satisfy the following requirements -

- The solution needs to be autonomous.
- Time taken for decision making should be as small as possible.
- The system should adapt to environmental changes i.e. it should learn and evolve continuously.
- Enable collaboration between agents to achieve a zero energy community status.

This thesis aims to build a Deep Reinforcement Learning (DRL) based energy sharing solution that addresses the requirements outlined above.

1.4 Evaluation

The intelligent energy sharing solution proposed by this thesis is evaluated in a self-made multi-agent simulation using a framework called OSBrain [68]. The focus of evaluation is to see how well does the energy sharing solution address the requirements outlined in the previous section as well as other performance metrics. Evaluation has been performed in 2 seasons, 3 community configurations, and 3 different scales. nZEC status achieved by intelligent agents in these evaluation scenarios has been compared with other strategies such as a no-energy-sharing strategy, always-share-energy strategy and random-actions strategy. This helps us to thoroughly evaluate our solution for adaptability and performance.

1.5 Contribution

This thesis identifies the definition of nZEC and motivates the need for it. The main contribution of this work is a DRL based solution in a multi-agent environment that optimizes energy transactions between buildings to achieve nZEC. The proposed solution is hybrid as it is a composition of independent learners and a monitoring service that provides global state to the learners. Unlike existing solutions that focus on energy sharing between buildings by introducing competition, this work takes the collaborative approach where everyone can benefit. The change shifts the paradigm from a user-centric approach to a community-first approach.

1.6 Assumptions

A series of assumptions have been made for the design and evaluation of this thesis. This thesis assumes that the network is reliable and there are no loss of messages. This means that the agents have uninterrupted communication with each other and the Community Monitoring Service (CMS). We assume that the CMS does not fail to provide relevant information at all times.

Although the environment is non-stationary due to evolving agent policies, it is not completely random as we are using energy data generated using a simulator [72]. Agents are trained on the same data with no uncertainty involved in an episodic manner. The simulated data shows regular usage patterns which can be the case in real-world situations too. Additionally, we use the same solar exposure data for all households considering they are located in the same geographical region. However, this might not be true in the real world due to physical reasons like shade and solar panel angle.

All participating agents are built using the same process - RL, approximate Q learning. This thesis assumes that the environment is not heterogeneous i.e. all agents follow similar a strategy for learning and eventually learn to cooperate. A single agent failing to do so can

disrupt the nZEC status. In real-world situations it is possible that agents have different learning strategies.

1.7 Thesis Roadmap

The remainder of the thesis is organized as follows:

1. Chapter 2 introduces basic concepts of smart grids and provides sufficient background knowledge to understand remainder of the thesis. It reviews existing research on multi-agent systems, energy sharing in microgrids, multi-agent reinforcement learning (MARL) and DRL.

2. Chapter 3 reviews existing literature in intelligent control in smart grids and identifies the research gap that motivates this work.

3. Chapter 4 firstly identifies the requirements for a collaborative intelligent energy sharing solution and then illustrates the overall system architecture.

4. Chapter 5 discusses the implementation and simulation of the proposed architecture.

5. Chapter 6 presents and evaluates the results obtained in a set of different nZEC scenarios.

6. Chapter 7 concludes this thesis with a summary, outlines the issues that remain open and avenues for future work.

Chapter 2

Research Background

This chapter introduces different concepts required for understanding the further work in this thesis. It firstly introduces the concept and challenges faced in smart grids (section 2.1) to provide context for this work. It then presents the basic concepts required to understand this work in a progressive manner (section 2.2, 2.3, 2.5, 2.6).

2.1 Smart Grid

Smart grids are power systems of the future rather than simply a marketing term. While many definitions of smart grids exist, unanimously it is considered as a dynamic interactive real-time infrastructure concept that encompasses the vision of diverse power-system stakeholders [30]. A Smart Grid Strategy Study Group (SG3) setup by International Electrotechnical Commission (IEC) conducted a study towards "IEC Smart Grid Standardization Roadmap Edition 1.0" in 2010. They define smart grids as - "the concept of modernizing electric grid. Smart grids are integrating the electrical and information technologies in between any point of generation and any point of consumption" [2]. The Canadian Electricity Association (CEA) identified key themes of communication, integration, and automation for sustainable, economic, and secure smart grids [3].

Smart grids are motivated by the steady increase in high power over-stressed systems,

increasing distance between generation sites and load centres, distributed generation, reducing carbon footprint, intermittent and non stable energy available from renewable sources, and sustainability amongst other reasons [30]. Additionally, increased customer engagement and demand for transparency in consumption, and pricing, regulator's demand for more competitive energy prices and energy trading are few of the major reasons for driving development towards smart grids. These systems responds to the rate of advances in emerging technologies such as computation power, communication systems, greater generation and storage capacity.

This thesis models the proposed solution for the problem mentioned in section 1.2 in a smart grid environment. It considers all participating buildings are enabled with intelligent power management systems and digital communication technologies, which are capable of monitoring and manipulating their energy loads. We focus on a very particular aspect of smart-grid i.e. zero energy communities as defined in section 1.1.

2.2 Multi-Agent Systems

There has been a increased interest in the recent years to solve complex problems using decentralized approaches. The entities involved, work together to solve complex problems and is called as *distributed artificial intelligence* (DAI). Traditionally, there are two sub areas in DAI. The first is called, *distributed problem solving* which focuses on decomposing a problem and solving it using slave nodes. The micro solutions are then collected and reconstructed to produce the actual solution. The second sub-area is called multi-agent systems (MAS) that emphasizes on joint behavior of agents with some degree of autonomy and complexities arising from their interactions [70]. This thesis models the proposed solution in a multi-agent environment. To understand this, we first define the concept of an agent. An intelligent agent (IA) is a autonomous entity which observes the environment through *sensors* and acts upon it using *actuators* to achieve a particular goal. To achieve autonomy, intelligent agents learn the desired behavior to achieve their goal using suitable

learning techniques (section 2.3.1). Russel and Norvig [81] state goal-directed behavior as the essence of intelligence and use the term 'rational agents' to describe IA. This thesis uses the term *agent* to refer IA or rational agents.

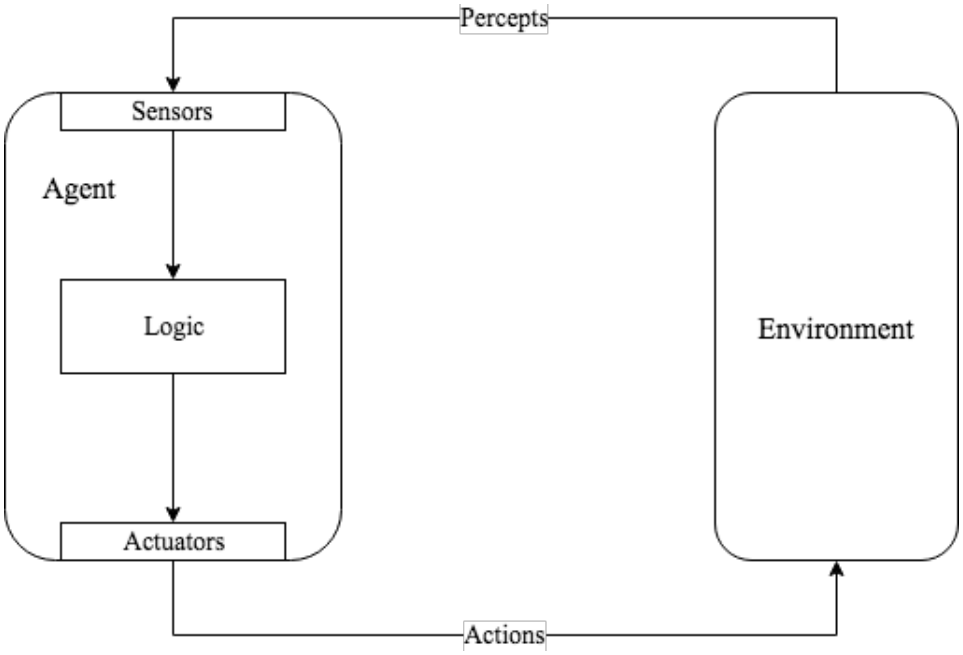

Figure 2.1: Agents interact with the environment using actuators and sensors [81].

Agents act in a environment. These environments are also referred to as task environments. Although many types of task environments can exist, they can be broadly classified into few distinct categories [81] –

- **Fully observable vs Partially observable**

  If an agent can observe the complete state of the environment at each point of time, we then call it as fully observable environment, otherwise the environment is partially observable.

- **Deterministic vs Stochastic**

  If the action taken in the current state fully determines the next state, we say say that the environment is deterministic, otherwise, it is stochastic.

- **Static vs Dynamic**

  If an environment changes without input from the agent to potentially unknown states, it is dynamic. If the environment reacts only to only agent's actions then the environment is static.

- **Episodic vs Sequential**

  In a episodic environment, the agent's experience is divided into atomic episodes. Only the current perception of the environment is relevant. On the other hand, a sequential environment requires memory of past experiences to decide the next best action.

Although, the distinction between MAS and single agent environment may seem simple, MAS are inherently complex. This complexity arises due to multiple agents acting on the environment making it dynamic and stochastic. It is difficult to learn the optimal behaviour in such an environment which is generally the case in most real world scenarios. Small changes in learned behavior can lead to unpredictable changes in the resulting macro-level properties of the multi-agent group [70]. This is discussed in further detail in section 2.3.1. MAS can be further classified into *competitive MAS* and *cooperative MAS*. The term *collaborative MAS* is also used interchangeably for cooperative MAS. Incentives to maximize personal gain can lead to a competitive environment. For example, chess is a competitive environment where two agents are competing against each other to maximize their own performance which inherently reduces the performance of the opponent. Such games are defined as Zero Sum games in Game Theory [63]. In a collaborative multi-agent environment, agents interact with each other to achieve a common goal. For example, playing soccer requires collaboration within the team. Each team member can be considered as an agent, where the common goal is to win. Agents in a collaborative environment might tend to maximize their personal gains but not at the expense of the common goal. [76] discusses the issues of collaboration and trust between agents in MAS in further detail.

2.2.1 Multi-Agent System Architectures

Interaction between agents forms the core of multi-agent systems [76]. MAS have been subject to extensive research to develop models for coordination [18, 38] , collaboration [5, 31, 69] and negotiation [45, 75]. Agents interacting with each other using messages is called as *direct interaction*. However, when agents take actions (no-action is also considered as a action) they affect the environment. This effect on the environment can be perceived by other other agents. Such form of communication is called *indirect interaction* [41]. Majority of the interactions in the real world are indirect. Furthermore, MAS can be categorized on the basis of their organization –

- **Centralized architecture**

  These architectures have a central controlling agent that distributes the tasks amongst other agents. It is similar to master-slave configuration. The interaction between agents generally takes place through the controller agent.

- **Decentralized architecture**

  Agents operate autonomously without the control of central agent. Interactions between agents is direct or indirect.

- **Hybrid architecture**

  These architectures have a central controlling agent that is responsible for management of all other agents and other bookeeping activities. It may store aggregated status of all the agents and the environment. Agents generally communicate with each other directly.

2.2.2 Multi-Agent Frameworks

Research in MAS has lead to the development of languages and tools that are appropriate for the realization of such systems. Since the late 90's many multi-agent frameworks

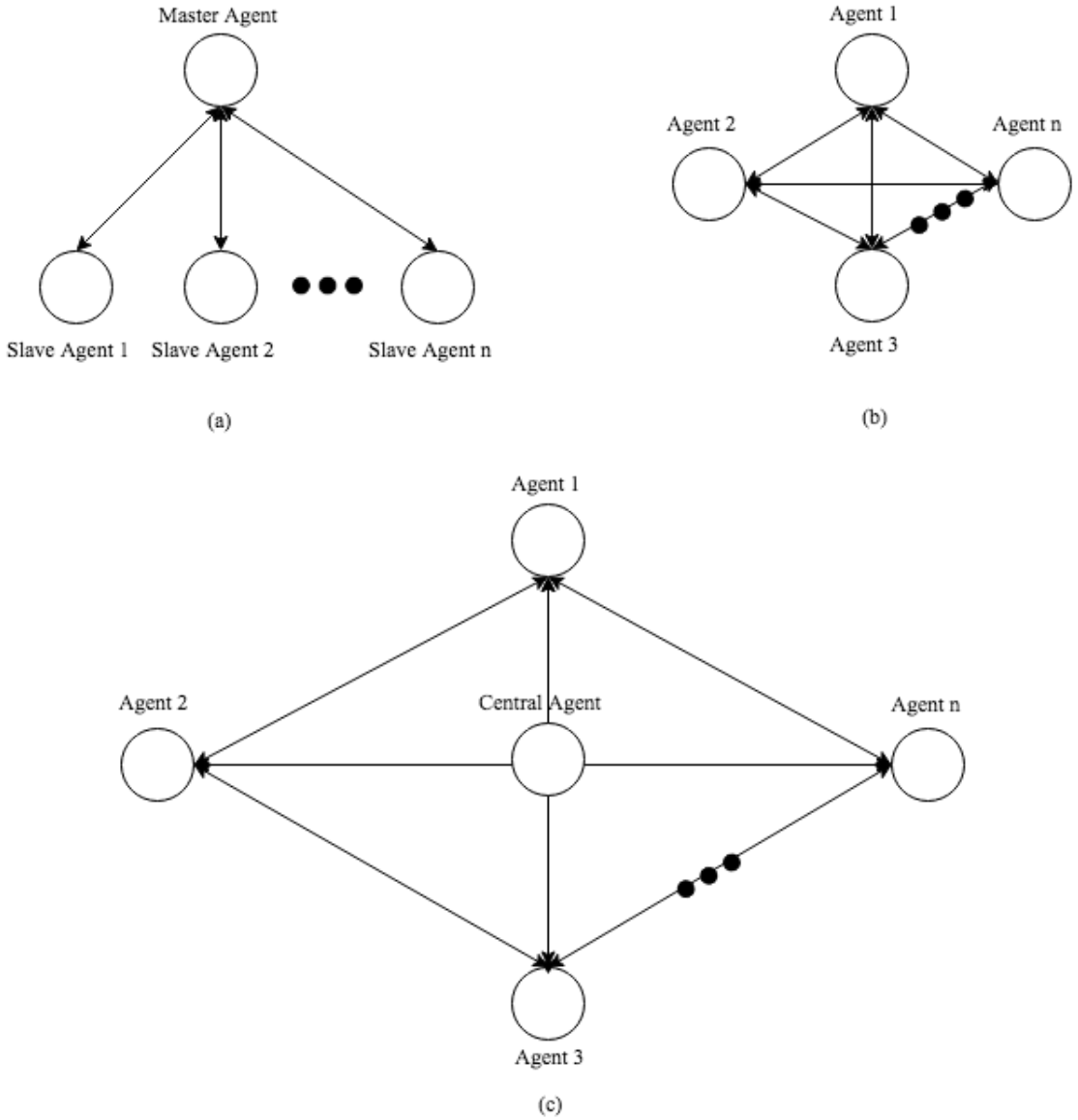

Figure 2.2: MAS architectures on the basis of their organization. (a) Centralized architecture (b) Decentralized architecture (c) Hybrid architecture

(MAF) have been developed. Choosing between frameworks is challenging due to difference in opinions and lack of sufficient survey articles on these frameworks [9, 28, 46]. [46] identifies 5 distinct criteria categories from [12] in MAF - platform properties, usability, operating ability, pragmatics and security management. These can be further subdivided to finer categories for defining the agent platforms. [46] identifies 24 MAFs and provides a in-depth analysis for each of them.

Framework	**Popularity**	**Open Source**
Agent Factory	Low	Yes
AgentBuilder	Medium	No
AgentScape	Low	Yes
AGLOBE	Medium	Yes
AnyLogic	Medium	No
Cormas	Medium	Yes
Cougaar	Low	Yes
CybelePro	Low	No
EMERALD	Low	Yes
GAMA	Low	Yes
INGENIAS Development Kit	Medium	Yes
JACK	High	No
JADE	High	Yes
Jadex: BDI Agent System	High	Yes
JAMES II	Medium	Yes
JAS	Medium	Yes
Jason	High	Yes
JIAC	Low	Yes
MaDKit	Medium	Yes
NetLogo	High	No
MASON	Medium	Yes
The Repast Suite	Medium	Yes
SeSAm	Medium	Yes
Swarm	Medium	Yes
OSBrain	Medium	Yes

Table 2.1: List of multi-agent frameworks.

We analyze and evaluate a few popular MAS frameworks for implementation purposes

–

JADE

JADE stands for Java Agent Development Framework and is fully implemented in *Java* [7, 8]. It is a software framework to build MAS for the management of networked resources in compliance with Foundation of Intelligent Physical Agents (FIPA) [22]. It can be distributed across machines and the agent configuration can be controlled using a remote GUI. JADE is industry-driven and currently is the most popular FIPA-compliant agent platform.

Jadex: BDI Agent System

Jadex is a software framework for developing goal-oriented agents using the belief-desire-intention (BDI) model. It facilitates easy development of intelligent agents with sound software engineering principles [74]. It allows agent development using *Java* and *XML*. It has been used in different domains such as simulation, scheduling and mobile computing [46].

NetLogo

NetLogo is a multi-agent programmable modeling environment. It is used by thousands of researchers and students worldwide [106]. NetLogo is well suited for modeling complex systems with hundred and thousands of agents operating independently. It comes with an extensive library of models of economics, biology, physics, chemistry, psycology and system dynamics.

OSBrain

OSBrain is a general purpose multi-agent module written in *Python* [68]. Agents run as separate processes and communicate with each other using ZeroMQ messaging platform. It uses PyRo for remote object invocation which makes it easy to invoke functions across agents. It was originally developed by OpenSistemas to create a real-time automated

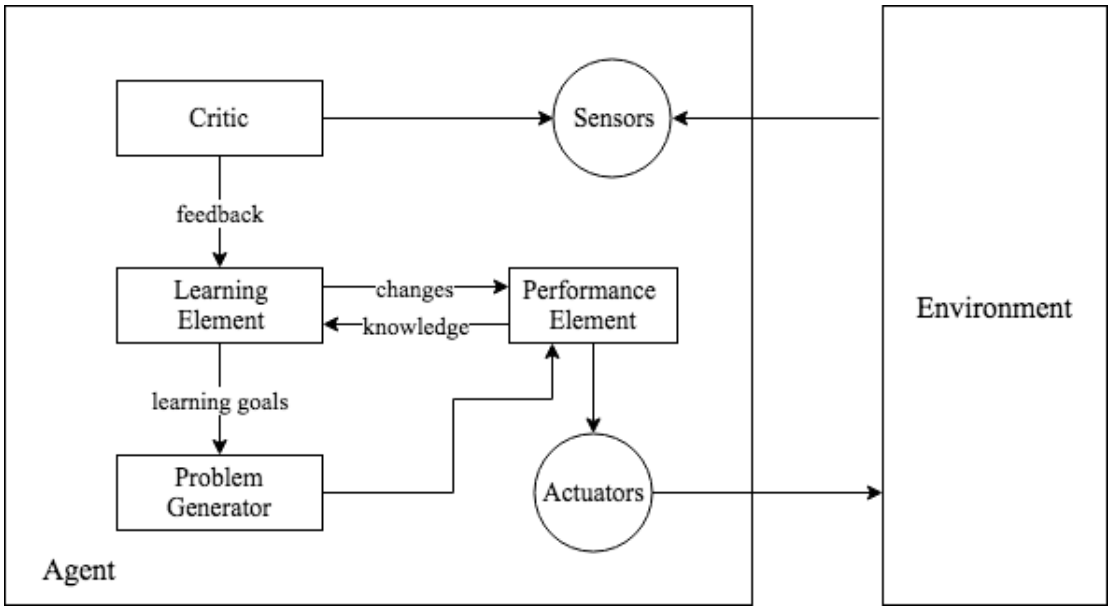

Figure 2.3: General architecture of a learning agent [81].

trading platform. However, the development of this framework was very generic and the company decided to make it public as a multi-agent platform.

2.3 Learning Agents

Russel and Norvig [81] define learning in agents as –

> An agent is learning if it improves its performance on future tasks after making observations about the world.

Intelligent agents have the ability to learn the behavior required to achieve their goals. As part of this learning process, agents can learn from other agents or learn about other agents if required to optimize their local behavior [104].

Figure 2.3 represents the general architecture (structure) of a learning agent [81]. This architecture can be conceptually divided into four components –

- **Learning Element**

  Component responsible for making improvements to the agents knowledge.

- **Performance Element**

  Component responsible for selecting external actions based on precept's from the environment.

- **Critic**

  The critic component acts as a evaluator which tells the agent how well it is doing with respect to the fixed performance standard.

- **Problem Generator**

  This component is responsible for suggesting exploratory actions that lead to new informative experiences.

2.3.1 Learning Techniques

It is not possible for programmers to anticipate all possible situations an agent might be in for non-trivial tasks. Additionally, as the environment is dynamic in most real-world situations the complexity grows exponentially. In some cases, it is impossible for programmers to figure out a solution themselves. A face-recognition and labelling system is one such task [81]. For these reasons amongst others, it is often a good idea to let intelligent agents learn the optimal behavior (actions) by themselves.

Learning techniques can be broadly classified into three types –

- **Unsupervised Learning**

  In this type of learning, the agent learns (or discovers) patterns from the input even though no explicit feedback is provided. This learning is used in situations when there is no historical failures to learn from. *Clustering* is the most common unsupervised learning task. For example, an agent predicting climate conditions might find groups of rainy days, windy days, sunny days, etc without labelled examples.

- **Supervised Learning**

  In supervised learning, the agent observes labeled examples i.e. input-output pairs

and learns a function that approximately maps the input to the output. The inputs are percepts observed that the agent and outputs are the relevant actions for those percepts.

- **Semi-Supervised Learning** This type of learning is a combination of *supervised* and *unsupervised* learning. The agent has access to some labelled examples and mostly unlabelled examples.

- **Reinforcement Learning** A reinforcement learning (RL) agent learns from a series of reinforcements i.e. rewards or punishments. It is inspired by behavioral psychology and uses a trail-and-error method to maximize its long term reward.

Conventionally, RL has been known to solve learning tasks with good convergence is various domains like robotics [44], finance [94], energy systems [62, 67], medicine [48, 53], cosmology, etc. As the motivation for this work lies in the energy domain and can be solved using RL, we discuss RL in further detail in the next section.

2.4 Reinforcement Learning

Reinforcement Learning (RL) is a machine learning approach that enables intelligent agents to learn the optimal behavior in a unknown environment via trail-and-error [91]. An agent learns the optimal behavior by analyzing the consequences of its actions in the form of a simple scalar signal (*reinforcement*) given by the environment [25]. This scalar signal can be a *reward* or a *penalty*. The goal of a RL agent is to maximize the reward received based on its interaction with the environment. The reward indicates what the agent needs to do without saying how to do it. It is common that along with immediate consequences of its actions (*immediate rewards*) an agent also needs to consider its impact on future situations (*delayed rewards*). Sometimes an action that is favorable in the immediate state might lead the agent to a state which is highly penalizing in the future and vice-verse. For example, a student applying for higher studies may find it immediately

penalizing in terms of money and time until graduation, but it is highly rewarding later on as it opens new and better career paths and a higher salary. Immediate reward obtained in a state along with consideration of delayed rewards from that state is called as the *utility* of that state.

To solve RL problems with delayed rewards, the interactions between the agent and the environment is often modeled as Markov Decision Processes (MDPs). MDPs provide a mathematical framework for modeling decision making situations. An MDP is a 5-tuple (S, A, P_a, R_a, γ), where –

- S - Finite set of states $\{s_t, s_{t+1}, ..., s_n\}$
- A - Finite set of actions $\{a_1, a_2, ..., a_n\}$
- $P_a(S, S')$ - Probability that an action a in state S at time t will lead to a state S' in time $t + 1$.
- $R_a(S, S')$ - Immediate reward received after transitioning from state S to state S'.
- $\gamma \in [0, 1]$ - Discount factor responsible for weighing future rewards against present rewards.

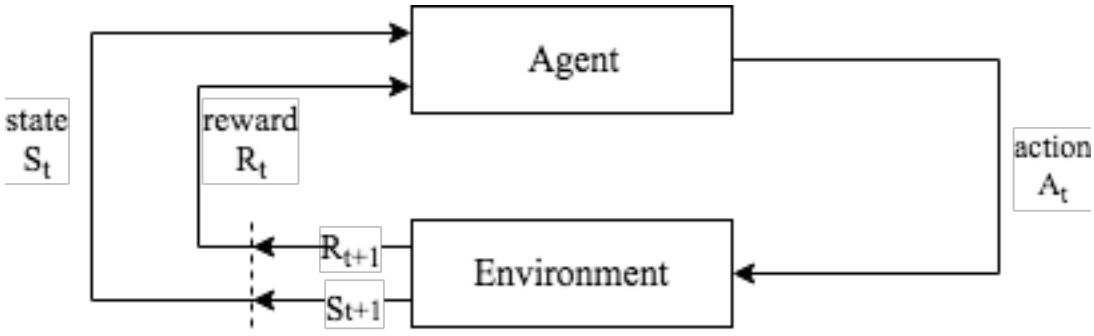

Figure 2.4: Reinforcement Learning process [91].

Figure 2.4 depicts a typical RL process where the agent takes an action A_t at time t and the environment returns a reward R_t. Markov models follow the Markovian property

[54] according to which state S_{t+1} is only dependent on state S_t and action A_t and no other actions or state before that.

In real-world scenarios often the transition probability P_a and R_a is unknown. In such cases, two different RL techniques come in to play - *model-based RL* and *model-free RL*. Model-based RL algorithms observe learn a model of how the environment works and then plan a solution (*policy*) using that model. Although effective, these algorithms encounter difficulties when the environment in which it operates is dynamic. Model-free RL algorithms can handle these issues of a dynamic environment. The most popular model-free algorithm - Q learning directly estimates the optimal *Q-values* for each (S, A) pair from a which a policy maybe derived by choosing the action with the highest Q-Value. The next sub-section discusses Q-learning in detail.

2.4.1 Q-Learning

Q-learning is form of model-free reinforcement learning [101]. It allows agents to learn to act optimally in Markovian domains by experiencing the consequences of their actions without building maps of the domains [100]. Q-learning is mathematically defined as –

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\Big(r + \gamma \max_{a'} Q(s^{'}, a^{'})\Big) \tag{2.1}$$

where,

- s is the current state and $s^{'}$ represents the next possible state.
- a is the current action and $a^{'}$ is the next possible action.
- α is the learning rate that ranges between $0 \leq \alpha \leq 1$. It determines the importance or weight of new experiences in the Q-value calculation.
- γ is the discount factor that ranges between $0 \leq \alpha \leq 1$. It determines weight of expected future experiences in the Q-value calculation.
- r is the reward an agent receives for taking an action a in state s.

The two important hyper-parameters α and γ determines how much the agent adapts or learns from new experiences and how much importance does it give to future experiences. A *alpha* value of 1 indicates that the agent simply forgets it previous experiences for the new one. Similarly, a γ value of 1 indicates that an agent gives equal importance to future experiences as much as it does for present ones. Q-learning algorithm uses a *Q-table* to store $Q(s, a)$ values and is very efficient in terms of memory because it does not store multiple instances of the same (s, a) pair. Using this Q-table, an agent selects the action with the highest Q-value for a state. An agent's goal is to maximize the long term reward by selecting the action that yields the best long term reward. However, to discover such actions, an agent needs to explore new actions that potentially might not yield high rewards. A number of action-selection policies exist. One such policy is called $\epsilon - greedy$ action-selection policy. Instead of choosing the action with the highest Q-value every time, we choose a random action with probability ϵ from the set of valid actions in any given state. Note that, it is possible that the randomly selected action can again be the action with the highest Q-value. A drawback of using $\epsilon - greedy$ action-selection policy is that it selects an action from the set of actions with equal probability and hence can select an action that leads to a state with high negative payoff. To overcome this, *softmax* action selection strategies are used that do not assign the same probability to all actions. The probabilities assigned to actions are based on their respective Q-values, so a action with higher Q-value has a higher probability of being selected. *Boltzman rule* is one such softmax action-selection policy that uses a parameter called *temperature* τ to enable a balance between *exploration* and *exploitation*. A high value of τ indicates more exploration and less exploitation i.e. the agent experiences a greater number of (s, a) pairs. As the exploration continues, the value of τ decreases over time until 0 and the agent only focuses on exploitation of the pre-learnt policy.

Even for simple problems, the number of (s, a) pairs can grow very large. Real world scenarios are often complex and the size of Q-table can grow immensely and working with such tables becomes unrealistic. Backgammon and Chess can approximately have 10^{20}

and 10^{40} states, respectively [81]. As an alternative, *function approximation* is used to represent the Q-function in equation 2.1. This is called as *approximate Q-learning* and is discussed in the next section.

2.4.2 Approximate Q-Learning

In approximate Q-learning we simply replace the Q-table with a approximation function. It is called as an approximation because the function might not return the *true* utility values as returned by the Q-table [81]. The state and action is represented as a set of features using a *feature function* $f(s, a)$ that returns a vector of feature values $f_1(s, a), f_2(s, a), ..., f_n(s, a)$. This feature vector is then fed into the approximation function where they are multiplied and summed up against their respective weights that finally returns a Q-value.

$$Q(s, a) = \sum_{i=1}^{n} f_i(s, a) w_i \tag{2.2}$$

Equation 2.2 represents a simple linear Q-value approximation function where each weight w_i is associated with a feature $f_i(s, a)$. Many other approximation functions exist for more complex scenarios when the learning behavior is not linear. Section 2.5 discusses using *artificial neural networks* as function approximators.

As the exploration progresses, instead of updating the Q-table we update the weights w_i by calculating the error using the formula –

$$w_i \leftarrow w_i + \alpha * difference * f_i(s, a) \tag{2.3}$$

where,

$$difference = \left(r + \gamma \max_{a^i} Q(s^{'}, a^{'})\right) - Q(s, a) \tag{2.4}$$

Previous research has proven that Q-learning in its tabular form (i.e. using Q-table) converges with probability 1 to an optimal policy[37, 98, 100]. However, Q-learning in its approximation form in certain cases is known to diverge [6, 11, 27]. [105] explores

the cause the of this divergence as sharing of parameters between states which causes increased Q-value estimation of states in a chain like fashion. Some works have proven that approximate q-learning converges to region instead of singular point [26, 71].

2.5 Deep Reinforcement Learning

Artificial Neural Networks (ANN) when used as function approximators in RL is called as Deep Reinforcement Learning (DRL). The word 'Deep' is derived from Deep Neural Networks (DNN) that is essentially an ANN with multiple hidden layers [15]. ANNs have been used as non-linear function approximators long before [90, 96, 97] they gained much popularity recently when a team of 7 from *DeepMind* (acquired by *Google* in 2014) demonstrated excellent results in their article *Playing Atari with Deep Reinforcement Learning* [57]. *TD-gammon* is another such example that uses a ANN at its heart comprising of a single hidden layer to train a reinforcement learning agent to play *Backgammon.* [97]. Although, many other non-linear function approximators exist like decision trees, localized basis functions, spline fitting trees, etc we focus on ANNs due to their recent success in solving complex problems.

2.5.1 Artificial Neural Networks

ANN is a supervised machine learning technique that utilizes a non-linear approach to predict outcomes. ANNs are inspired by nature and specifically by the working of the brain. ANNs are comprised of units called as *Neurons* that are connected to each other using *directed links.* These links help in propagating the *activation* between neurons. This is similar to how the human brain functions [81].

Figure 2.5 represents a simple neuron. A link from i to j propagates the activation from a_i to a_j. The links are associated with a scalar value called *weight* $w_{i,j}$ that determines the strength and the sign of the connection. Each neuron a_j firstly computes the sum of

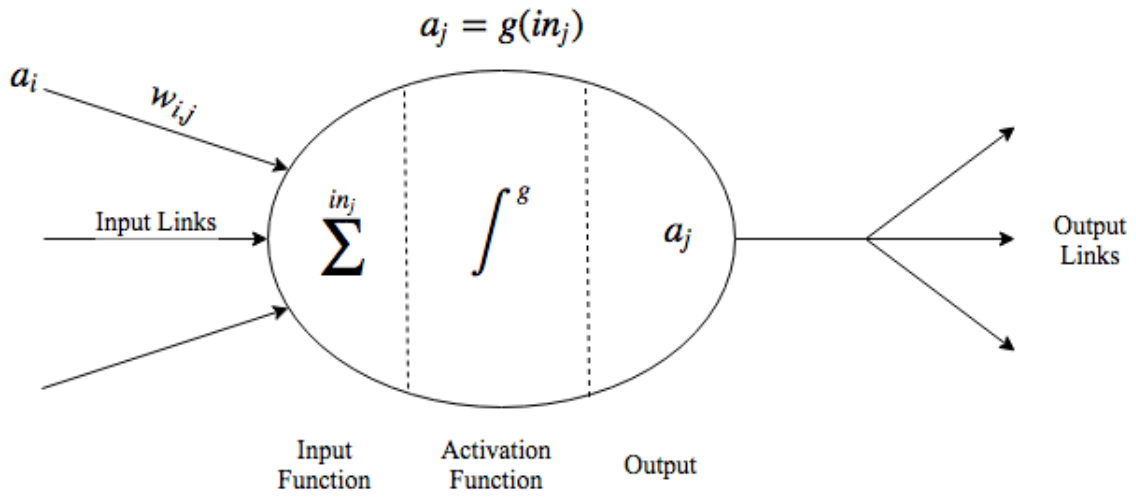

Figure 2.5: A simple mathematical model for a neuron [81].

its inputs multiplied with their respective weights as shown in equation 2.5.

$$in_j = \sum_{i=0}^{n} = w_{i,j}a_j \tag{2.5}$$

In the next step an *activation function* $g(x)$ is applied on in_j which gives the output value for this neuron as shown in equation 2.6. The activation function determines whether a neuron is active or not for the particular input. If a neuron has such hard threshold for activation it is called as a *perceptron*. Other non-linear activation functions like *sigmoid*, *tanh*, *ReLU*, *leaky ReLU*, etc exist. Non-linear activation functions ensure that the network can represent non-linear behaviors [81].

$$a_j = g(in_j) = g\Big(\sum_{i=0}^{n} w_{i,j}a_i\Big) \tag{2.6}$$

These neurons can then be connected to each other in two distinct ways[81] -

- **Feed-forward Network**

  This network has connections in only direction like a directed acyclic graph. Every neuron receives inputs from previous neurons, processes it, and then forwards it to the next neurons. There are no cycles present in such a network.

- **Recurrent Network**

  This network feeds its output back to its own input. This makes the network dynamic and able to support short term memory.

Neurons are arranged in layers as shown in figure 2.6 so that neurons in a particular layer receive inputs only from the previous layer. The figure represents a fully connected neural network, although different architectures exists which vary in the number of neurons per layer, number of hidden layers, connectivity of layers, number of output neurons, feedback to previous layers, etc. In the recent times, due to increased computational power neural network architectures with many hidden layers are becoming popular and is the basis of *deep learning* [15].

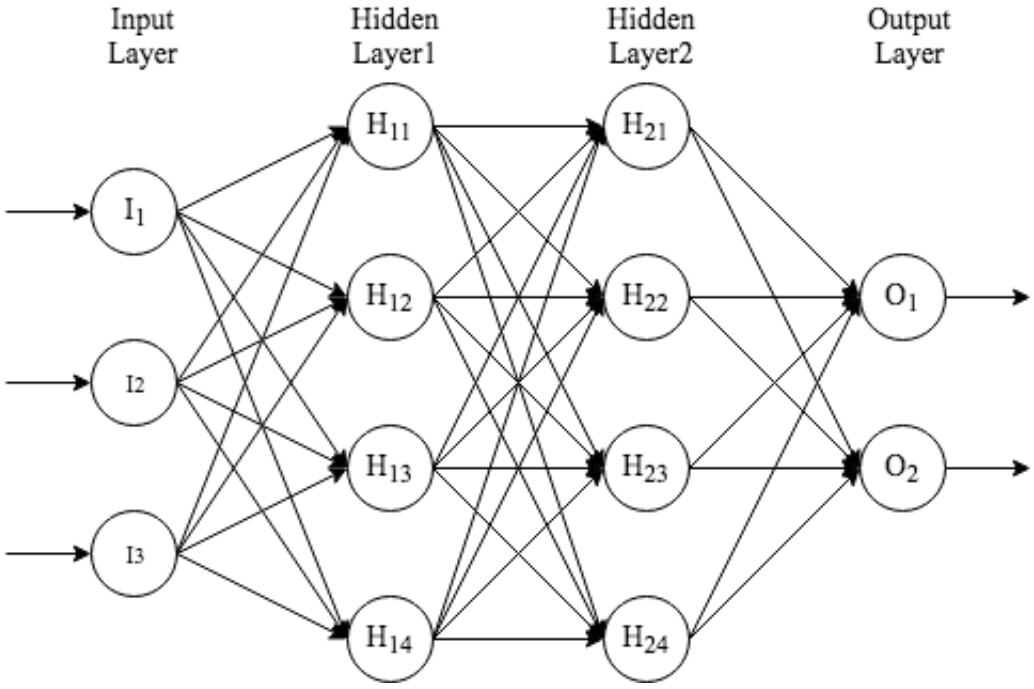

Figure 2.6: Typical architecture of a fully connected ANN.

The learning part of neural networks is achieved by computing the *gradient* of the *cost function*. Cost function is the measure of how good did the neural network perform with respect to the expected output. Simply put, its it the difference between the expected output and actual output. Several cost functions like *quadratic cost, cross-entropy*

cost, exponential cost, hellinger distance, etc exist. Gradient of the cost function is then computed using a suitable technique. On such famous technique is called as *Backpropagation* [33]. Earliest works in backpropagation were published in the 1970's but they only gained much recognition when a article *Learning representations by back-propagating errors* [80] was published in 1986. Essentially, backpropagation computes the derivate of the cost function w.r.t. to any weight and the bias in the network as show in equation 2.7. This computed gradient is then propagated back through the network and the weights are updated accordingly.

$$\frac{\partial C}{\partial w_{i,j}} \tag{2.7}$$

where,

- C is the cost function.
- $w_{i,j} \in W$ is any weight in the network.

In the naive implementation, a neural network with a single neuron in the output layer can be used to estimate the Q-values in Q-learning for every (s, a) pair. This is also known as *Deep Q Learning (DQL)*. Other strategies like Deep Policy Gradient (DPG) too exist that directly focuses on inferring the optimal policy.

2.5.2 Experience-Replay

It is known that approximation of Q-values using non-linear functions is not very stable. This leads to issues in convergence. *Experience replay* is a technique for generating uncorrelated data for online training of DRL systems [108]. It has proven to be very effective for convergence of such systems [1, 47, 57]. The idea behind it being simple, as the agent explores its environment, it stores all its experiences in a replay memory as a 4-tuple $(s, a, r, s^{'})$. During training, random minibatches from the replay-memory are

used instead of the most recent transition. This helps break the correlation between subsequent samples that would have otherwise driven the network to a local minimum [56]. This makes the training examples independent of each other just like any other supervised machine learning task.

Zhang and Sutton [108] explore effects of the size of replay memory on the agent's performance on different tasks such as Grid World, Lunar Lander and Atari game - Pong. The authors find that a large replay buffer size significantly hurts the agent's performance in terms of convergence and propose an alternative solution $\mathcal{O}(1)$ as a remedy to this. The solution suggests to add the *latest transition* to the mini-batch of random samples and is called as *Combined Experience Replay (CER)*.

2.5.3 Energy solutions using DRL

DRL has been used previously to model solutions in the energy domain, mainly due its online nature and ability to discover solutions to complex problems. Efforts have been focused on optimizing energy usage of residential buildings by scheduling devices using DRL [58, 64, 102]. The authors of [58] propose an architecture which takes as input the energy profiles of the devices, energy price, temporal data and outputs a suitable energy schedule. Another work [23] from a group of scientists at University of Liege, Belgium uses DRL to model a solution that optimizes activation of energy storage devices considering the uncertainty of energy generation and consumption. The authors use 2 convolutional neural networks with 16 filters having size 2x1 and 2x2 respectively and stride 1.

[64] uses a DPG based solution to optimize the schedule of energy consuming devices in a dynamic energy pricing environment. The authors propose a nanogrid energy management controller empowered with two ANNs, each having 2 layers of 400 and 300 neurons respectively. They train the agents for 4 simulated days and are able to reduce the energy cost while considering the user discomfort.

2.6 Multi-Agent Reinforcement Learning

Multiple reinforcement learning agents in a multi-agent system is called as multi-agent reinforcement learning (MARL). Agents in such a system can either learn *independently* or *cooperatively* with other agents. It difficult to learn the optimal behavior in such environments due to interaction of multiple agents with the environment or with each other. This makes the environment *non-stationary*. To be explicit, non-stationarity can be caused due to two primary reasons –

- Multiple agents simultaneously interacting with each other and the environment to take actions that affect the state of the environment.
- The environment is self-evolving.

[93] evaluates the performance of cooperative vs independent reinforcement learning agents on a grid world problem. The terms *joint-action learners* and *independent learners* are also used sometimes to refer the same [14]. Cooperative learners or joint-action learners share the learned information with other agents and therefore also share their policies. Conversely, independent learners do not share their learned behavior with other agents but can learn other agent policies via their interaction with the environment. The author mentions that cooperative agents can learn faster and converge sooner than independent agents as they can share learned policies and solutions between themselves. Even though, there are advantages in cooperative learning, significant research in MARL shows that convergence guarantees are not practical [14]. Works that demonstrate convergence to equilibria, do so in a very constrained environment with highly restrictive assumptions [34, 49]. Authors of [103] present a NSCP (Non-stationary Converging Policies) learner that focuses on learning the *best-response policy* instead of reaching *Nash equilibria* [63] in general-sum stochastic games. It does so by inferring a accurate model of the opponents non-stationary policy and therefore creates a best-response policy for that strategy.

Many of these previous works that validate theories have been performed in confined and small scale environments and does not work well in real world situations where the scale is huge. Active research is being done to scale reinforcement learning techniques to tackle real world problems. Ordinal Sharing Learning(OSL) is a MARL method proposed for job scheduling problems, especially for load balancing in Grids [107]. It achieves this by employing a *utility-table* based learning approach and sharing locally learned information amongst agents. Another method, MARLIN-ATSC, employed for large scale traffic light control achieves considerable reduction in average intersection delay by employing agent learning in *independent mode* and *integrated mode* which is analogous to independent and cooperative learning [95]. The authors identify the key challenges as coordination and the curse of dimensionality. Although, join-action learning is effective, many tasks can be solved using independent learners and thereby avoiding the overhead incurred due to coordination. Moreover, independent learners are easier to implement.

2.6.1 Cooperation in Independent Learners

[78, 88] trains independent learners for the task of *RoboCup simulated soccer* to investigate cooperation between multiple independent agents learning simultaneously. They enable cooperation between agents by giving common or global rewards to agent teams. Similarly, another work [14] uses the locally obtained rewards by independent learners to generate *averaged rewards* that are distributed amongst the agents. Authors of [39] introduce a technique called as *Frequency Maximum Q Value (FMQ) heuristic* which is an extension of the standard Q learning algorithm. For each (s, a) pair it generates the expected value (EV) obtained after taking an action instead of the Q-value.

$$EV = Q(a) + c * freq\Big(maxR(a)\Big) * maxR(a) \tag{2.8}$$

where,

- $maxR(a)$ is the maximum reward obtained so far after choosing action a.

- $freq(maxR(a))$ is the fraction of times when $maxR(a)$ has been received as a reward for action a over the number of times action a has been executed.
- c is the weight that controls the importance of FMQ heuristic.

This work too distributes the same reward amongst all agents. A literature review of all these works strongly indicates that similar reward distribution amongst agents is central to enable coordination and cooperation amongst individual learners.

Chapter 3

Intelligent Control in Smart Grids

The previous chapter provide necessary and sufficient theoretical knowledge to review literature of intelligent control solutions in smart grids. This chapter evaluates and discusses the various works done in smart grids relevant to energy sharing and zero energy communities.

3.1 Energy Optimization in Smart Grids

3.1.1 Energy Usage Optimization in Single Buildings

High volume of data is being generated by *advance metering systems* in homes and buildings and can be utilized to benefit planning and operation. Energy optimization in buildings focuses on energy management of devices by controlling their schedules. There have been numerous works in this domain. Many of these works focus on scheduling - heating, ventilation and air conditioning (HVAC) [29, 52] as these are the major power consuming devices in homes. Apart from managing energy consuming devices, some works also focus on on-site energy generating devices to minimize the grid interaction as part of energy optimization [52]. The minimal requirement is that all of these devices are connected to the central controller in the house which is responsible for taking the decisions [89]. Vari-

ous techniques have been proposed to optimize energy schedules like linear and dynamic programming, heuristic methods, particle swarm optimization, game theory, fuzzy logic, heuristic methods and evolutionary algorithms [58, 60, 89].

The general problem with majority of the algorithms is that, for optimization they compute partial or the entire solution space to choose the best one, and hence are time consuming. [58] explores one such interesting approach that avoids computing the entire search space using DRL. The authors model the building environment using a Markov Decision Process (MDP). Two different RL algorithms - *Deep Policy Gradient(DPG)* and *Deep Q-Learning (DQL)* are being considered by this work. These algorithms prove to efficient, although initially it requires sufficient data to be trained.

3.1.2 Energy Optimization in Multiple Buildings

The concept of energy optimization in single buildings can be extended to multiple buildings. This means energy flow is managed between buildings using a central controller [24, 111] or in a distributed manner [83, 109, 110]. The energy might be generated on-site or can be requested from the central controller. Considering optimization between multiple buildings also introduces problems related to exchange of messages between buildings, fair distribution of energy, sharing of energy channel, transmission efficiency, grid stability and so on [110].

Majorly, works in energy usage optimization tend to optimize schedules of high energy consuming devices (HVAC) [4] without compromising user comfort [85] at the intra-building level. A review [4, 13, 58] of many works suggests that advance learning techniques like Evolutionary Algorithms, ANNs, XGBoost have been successful to solve design and operational problems in this domain. Such ideas have then been extended to multiple buildings to obtain greater reduction and optimization of energy usage. Although, the areas where optimization is performed, can be different, the ideology roots' from the former.

[24] employs a Genetic Algorithm based approach to estimate the amount of energy

consumed by individual buildings and Monte Carlo simulation to calculate the energy differences, control and coordinate the entire process. Another work shown in [111] uses a greedy matching algorithm to categorize the houses into subsets of Suppliers (S) and Consumers (D). The houses can then share energy with their nearest neighbours to minimize transmission losses. They then use a TDMA based algorithm to schedule the transfer of energy between houses.

3.2 Energy Sharing between Buildings

More efficient usage of energy can be done if the excess energy produced on-site can be shared with other homes that require it. This is beneficial in many ways. Firstly, it can benefit the seller economically. Secondly, sharing energy with nearby neighbours reduces the transmission losses that would have occurred otherwise if energy was requested from central grid. Thirdly, energy storage is not cheap and it is better to sell off energy as it reduces the negative environmental impact due to huge batteries.

[66] discusses a detailed review of centralized and distributed energy management systems. The authors define a *Central Energy Sharing (CES)* systems as the one that consists of a central controller that has information of all the distributed energy resources in the micro-grid as well as the forecasting systems, and schedules resources accordingly. This type of system allows broad observability but reduces the flexibility. The article states that *Distributed Energy Sharing (DES)* systems reflect a market environment, where each agent sends bid requests to the Central Microgrid Operator (CMO).

3.2.1 No Energy Sharing vs Centralized Energy Sharing

CES systems have proven to be very effective compared to No Energy Sharing (NES) systems. They have reduced energy losses as well as increased cost benefits for sellers. NES systems also employ huge batteries which is not cost efficient. Work in [111] introduces a CES system that classifies homes into sets of suppliers and consumers, and then uses a

greedy strategy to initiate energy exchange between homes. Although, the system proves to be better than no energy sharing systems, the authors mention that due to incorrect prediction the homes sometimes transfer more energy than they can actually harvest. This points out an important dimension in energy sharing i.e. precise prediction of energy generation. Also, this is difficult due to the arbitrary nature of renewable sources.

3.2.2 Centralized Energy Sharing Vs Distributed Energy Sharing

Another interesting system - IDES [110] uses a sophisticated distributed energy generation and sharing approach (DES). The authors have also introduced a novel pricing model, to incentivize energy sharing between homes. They have compared their results with the centralized system in [111] and have obtained better results (low energy losses and average failure time). Additionally, they point out that, a centralized system is more expensive and also has central point of failure.

3.2.3 Energy Sharing in nZEC

Exhaustive mining of the literature in nZEBs reveals very little work that considers energy sharing between nZEBs. This makes sense as a nZEB is supposed to be self-sustaining and therefore has no need to share energy. However, [40] argues that achieving a ZEB status without a grid is very difficult. The authors propose a solution by defining *energy communities* using a basic energy matching algorithm and conclude that energy sharing can help achieve ZEB status for individual buildings.

As discussed in section 1.1, existing definitions of nZECs consider a community of only nZEBs [51] and therefore has very little or no previous direct work that focuses on intra-nZEC energy sharing to the best of our knowledge. In section 1.1 we point out that a nZEC may comprise of a mix of buildings that may or may not produce energy or sufficient energy to be considered as a nZEB. Existing works in CES and DES can be extended to

optimize energy sharing in such a community, but they still lack consideration of the nZEC status. This is important, as previous works in energy sharing consider managing transactions between suppliers and consumers, with the focus primarily on maximizing individual gains, whereas, in an nZEC some buildings might have to set aside their selfish goals and incur minor losses for the greater good. These losses may be economical as buildings might have to sell energy at a lower cost during periods of low demand to neighboring buildings, whereas, they could profit more by storing energy and selling during periods of high demand.

3.3 Multi-agent Systems in Smart Grids

MAS has been described in great detail in section 2.2. MAS have been used in smart grids and especially microgrids to solve many operational problems [16] like distributed energy sharing, energy bidding, load balancing, etc. It allows autonomous and distributed control in smart grids. One such work is demonstrated in [50], that models separate components of a microgrid as individual agents using JADE. The authors conclude that, MAS can help build easy, scalable and high performant *distributed energy resources (DER) systems*. Many other works too claim DER-MAS systems to be more effective than centralized approaches [17, 42].

3.3.1 Non-cooperative vs Cooperative Strategies

[109] proposes a non-cooperative game with the purpose of introducing energy trading amongst interconnected microgrids. Energy trading has been formulated as a Nash game [63]. The fundamental concept used here is that, each agent takes a decision considering the other agents action in its utility function. A non-cooperative environment gives rise to competition between the agents [63]. This competition is desired to increase cost-benefits, reduce resource consumption with no common goal in mind. IDES [110] introduces a competitive market where individual homes compete against each other to sell or buy

energy from other homes. Energy status is broadcasted by all homes and energy prices are multicast. This multicast behaviour ensures privacy of energy-price data.

Cooperative strategies induce collaboration between agents as they generally have a common goal. [83] describes a *coalition game theory* based approach to introduce collaborative behaviour between home operating agents. The algorithm allows microgrids to operate autonomously and self-organize into disjoint partitions of buyers and sellers. Within the partition too, agents exchange energy between themselves and the central grid with the aim of minimizing the transmission loss.

Although, the reasons for adapting non-cooperative and cooperative strategies might differ based on the problem area, non-cooperative strategies can sometimes result in disastrous results. This is possible as agents in non-cooperative strategies do not consider the aggregate effect of their actions which can lead to sudden increase in demand and supply of energy and thus lead to de-stabilization of the energy grid [19]. Although, cooperative strategies are effective, they are difficult to implement [5, 21, 61, 111] as they have to carefully designed to learn the optimal behavior.

3.4 Summary

There has been extensive research in optimizing energy transactions within homes and between multiple buildings using state of the art techniques like linear programming, evolutionary algorithms, ANNs, XGBoost, etc. These transactions are handled using different strategies like CES and DES each having their own merits and demerits. To take advantage of modern programming techniques these systems are modeled as MAS, where each component of the system can be treated as a separate agent that is responsible to taking accurate decisions for that component. Literature suggests that cooperative MAS are suitable for modeling systems with a common goal but have to be carefully designed. A through investigation of previous works in nZEC reveals very little work that has been done in this domain, especially related to intra-nZEC energy sharing. This is because

previous definition of nZEC comprises of only nZEBs and as nZEBs are self-sustaining they do not require to share energy with other neighbouring buildings. Although this ideology is accurate, some works argue that this status is difficult to achieve in most cases and can only be achieved as a collective effort. We further point out that future energy communities until completely energy sustainable, might have a mix of buildings that may have different levels of energy generation and may be unable to suffice their own energy demand. Moreover, some buildings might not have energy generating devices due to physical constraints. We therefore introduce a new definition of nZEC which builds on the definition of nZEB and expands the previous definition of nZEC as, *"A micro-grid that has distributed generation, storage, delivery and consumption having net annual energy balance as nearly zero"* (previously defined in section 1.1). Keeping this definition in mind and to build a solution for this, we have investigated various modern programming techniques. We find that with the advent of technology and success of ANNs, DRL is a technique that is very popular amongst the reinforcement learning (RL) community. With all this information at hand, we define the research question, "*Can deep reinforcement learning (DRL) be used to optimize energy sharing between multiple buildings to enable it to be a zero-energy community?*"

The next chapter discusses the design of the proposed solution motivated by the research question.

Chapter 4

Design

In the previous chapter we reviewed state-of-the-art in the smart grids and narrowed down to zero energy communities. A thorough analysis of the literature revealed that little or no direct work has been done in modelling energy communities with a mixture of buildings having no, partial or complete energy generation via renewable sources to achieve zero energy status. These buildings maybe commercial or residential. Furthermore, we highlight the success of ANN in discovering non-linear solutions and its integration with reinforcement learning known as DRL. Now that we have highlighted the research area, we focus on the design considerations for building a general solution for this.

4.1 Requirements for a collaborative MARL based solution for nZEC

In the previous chapters we identify that, in general MAS are non-stationary primarily due to two reasons - agent-induced non-stationarity and environment induced non-stationarity. This holds true for MAS in the energy domain too. Energy systems are complex due to the stochastic nature of renewable generation caused by various environmental factors like wind, sky condition, man-made or natural obstructions, etc. Additionally, actions taken

by agents related to energy sharing too affect other agents by shifting the energy balance in the community. Decisions in such real world systems needs to taken quickly. Summarizing all that we have observed, we formally outline the following requirements –

1. The solution needs to be autonomous.
2. Time taken for decision making should be as small as possible.
3. The system should adapt to environmental changes i.e. it should learn and evolve continuously.
4. Enable agent collaboration between independent learners.

Autonomous solutions help overcome challenges posed by renewable energy generation sources' integration in power grids and reduces cost compared to conventional systems [20, 82]. They are 4 times more performant and help reduce 60% of the cost when compared to conventional systems according to an article by Siemens, Automation Company [82]. These systems need to operate in real time and therefore the decision taken should have minimal computation time. The complex and stochastic nature of energy systems suggests the need for a self-evolving behavior. Once, intelligent independent agents have been built they need to collaborate with each other to achieve the community goal of zero energy status. A design solution for these requirements is discussed in the next section.

4.2 System Design

Based on the requirements from the previous section we find that 2 components are central to building a solution for this –

- Agent Learning
- Hybrid Architecture

The agent learning component takes care of decision making, and self learning and therefore addresses the requirements 1, 2 & 3. A hybrid architecture is required to enable collaboration between independent agents by distributing global rewards and hence caters for requirement 4.

We discuss these two components in further detail in the next subsections.

4.2.1 Agent Learning

Agent learning is the basic requirement to build a autonomous solution. As discussed in previous chapters, RL agents can learn the optimal behavior and evolve with changing environment. Additionally, we have discussed that RL is preferred in decentralized systems to learn the optimal behavior. One of the other objectives of this thesis is to explore the effectiveness of DRL to optimize energy sharing between buildings to achieve nZEC. The deep learning component of DRL ensures smaller computation time compared to Q-table based approaches. For ease of implementation agents can be designed as independent learners instead of join-action learners (section 2.6). These design choices take care of requirements 1,2 and 3 outlined in section 4.1. A learning agent as shown in fig 2.3 is composed of three main components –

1. Perception
2. Learning
3. Actuators

At every timestep an agent receives energy consumption and generation information. It might also receive requests from other agents to share energy. Based on this perception and previous knowledge, the learning component of the agent decides the appropriate action to be taken. This action is then executed by the agent.

Design Choices in Reinforcement Learning

The learning component is built using DRL. As discussed in section 2.5 DRL has been able learn the optimal behavior in various complex scenarios. We have also discussed that approximate Q-learning using non-linear activation functions is not very stable. To overcome this upto some extent, a technique called experience replay is often used. Experience-replay generates uncorrelated data that helps to avoid learning any strict patterns during the training phase and thus helps to generalize better during the testing phase. A improvement of the traditional experience-replay technique called *combined-experience-replay* is used. This allows the most recent example to be used in the training batch and therefore helps in the stability of the learning algorithm. The entire agent learning process is formally shown in algorithm 1.

Apart from the standard process there are various hyper-parameters one needs to tune for both reinforcement learning as well as neural networks to achieve optimal performance. A RL agent often needs to decide between exploring new actions that might lead to new better or worse states and exploiting already known best actions. This is handled using $\epsilon - greedy$ action selection policy. An agent chooses a exploratory action with probability ϵ and exploits its existing knowledge of best known actions at all other times. This hyper-parameter ϵ is manually configured for the right balance between exploration and exploitation. As an alternative, an extension of this strategy called Boltzman rule is used that reduces the exploratory nature of the agent over time by by changing the parameter τ strategically. Although, such advanced techniques are effective for tuning the value of ϵ even simple strategies that decay the value of ϵ over time work well. In this design, we simply decay the value of ϵ as the training progresses over fixed number of training episodes and set this value to 0 in complete exploitation mode.

Design Choices in Deep Learning

Building neural network model for a particular problem is not a trivial task. One has to consider many factors such as the depth of the network, number of neurons in each layer, the input size, activation function, learning rate, the optimization algorithm, number of training episodes, batch size, regularization techniques, etc. Several of these parameters are tuned after repeated experimentation and knowledge from previous works in the application area.

Number of layers and number of neurons in each of those layers represents the depth and width of the network. Adding more layers and neurons helps to model complex relationships but also increases the complexity of the network and the computation time. This is generally decided after repeated experimentation to strike the right balance between computational resources and performance. The activation function decides which neurons are active for the current set of inputs. To introduce non-linearity several activation functions like sigmoid, tanh, ReLU, and leaky ReLU exist, each of which have their own merits and demerits and are suitable for different problem areas. The simplest and the earliest known activation function is sigmoid that bounds the values between $[0, 1]$. The choice of activation function can strongly affect the final results and can again be selected based on experimentation and knowledge from previous works. A linear activation function is preferred for the output neuron that gives an estimate of the Q-value [23, 102]. The learning rate decides how fast the network adapts to new experiences. A very high learning rate can cause convergence issues due to oscillations of the loss function and even divergence in some cases. On the other hand, a very low learning rate can take a long time to converge. One can choose the optimal value of the initial learning rate based on experimentation and observation. Different automated learning rate decay strategies too exist like Reduce LR on plateau, a predefined schedule, etc. The batch size affects the training time and directly depends upon the memory available in the system.

Optimization algorithms help to minimize or maximize the objective function. In

neural networks our aim is to minimize the loss function i.e. the error computed as a difference between expected value and true value. This is can be done using a technique called gradient descent. Several gradient descent variants like batch gradient descent, mini-batch gradient descent, stochastic gradient descent are available [79]. *Stochastic gradient descent (SGD)* is the most popular optimization technique in systems that employ machine learning on a large scale [10, 113]. Different optimization exist that try to overcome the flaws SGD faces with convergence like Adelta, Adam, RMSProp, Adam, AdaMax, Nadam, AMSGrad [79]. Choice of optimization algorithm too depends on the problem domain. Thorough experimentation with different strategies suggests RMSProp to be more suitable for stability and convergence of our algorithm.

4.2.2 Combining Reinforcement Learning and Deep Learning

The choice of hyper-parameters and optimization algorithms in both reinforcement learning and deep learning is interlinked howsoever we try to separate them. We embed the neural network model that we have built as a function approximator for Q-value in the RL engine. However, this is not exactly done as it seems. For this, we use two separate neural networks for each agent. The first network estimates the Q-value and the second network estimates the target Q-value which is used to calculate the error between observed and expected Q-values. This helps with the stability of the algorithm that otherwise has problems with convergence [86, 99]. Each agent stores all the visited states in a *replay memory*, which then returns a sample of experiences for combined-experience-replay. The size of replay memory too affects the training and convergence of the algorithm [108]. For this simulation, we choose the replay memory size as 10000 after experimentation with different sizes.

Algorithm 1 below describes the agent learning process at an abstract level. The sequence of actions are similar to the illustration in fig 2.3.

Algorithm 1: Agent Learning Algorithm

```
1  env ← Environment()          /* env is an instance of the environment */
2  while true do
3      consumption ← env.preceptEnergyConsumption()
4      generation ← env.perceptEnergyGeneration()
5      state ← updateEnergyBalance(consumption, generation)
6      legalActions ← getLegalActions()
7      chance ← generateRandomNumber()
8      if chance ≤ ϵ then
9          action ← chooseRandomAction(legalActions)
10     else
11         action ← selectBestActionFromPolicy(legalActions)
12     end
13     nextState, reward ← env.takeAction(action)
14     updateAgentLearning(state, action, reward)
15 end
```

We further describe the agent learning process on line 14 - *updateAgentLearning(state, action, reward)* in greater detail as algorithm 2. As shown in algorithm 2 we initialize two separate deep learning networks - *qNetwork* and *targetQNetwork*. These two networks estimate the values for actual and expected Q-values. The *targetQNetwork* has a separate update frequency in correspondence to the update frequency of the *qNetwork* and is decided on the basis of experimentation. This helps with the stability of the algorithm.

Algorithm 2: Algorithm to update agent learning

```
1  batchSize = 64                    /* number of samples to use for training */
2  γ = 0.99                                                    /* discount */
3  learningThreshold = 1000        /* learning starts after 1000 experiences */
4  numOfQUpdates = 0             /* number of time Q-Network has been updated */
5  targetUpdateFrequency = 50    /* update frequency of the targetQNetwork */
6  qNetwork ← NNModel()                 /* instance of Neural Network model */
7  targetQNetwork ← NNModel()           /* instance of Neural Network model */
8  optimizer ← SGD(**params)                 /* Stochastic Gradient Descent */
9  r ← ReplayMemory()                   /* global instance of replay memory */
10 updateAgentLearning(state, action, reward)
11     r.store(state, action, reward)
12     if r.currentSize ≥ learningThreshold then
13         obsBatch, rewardBatch, nextObsBatch ← r.sample(batchSize)
14         qValues ← qNetwork(obsBatch)
15         targetQValues ← targetQNetwork(nextObsBatch)
16         bellmanErr ← -1.0 *( (reward + (γ * targetQValues)) - qValues)
17         optimizer.step(qNetwork, bellmanErr)   /* perform gradient descent */
18         numOfQUpdates++;
19         if numOfQUpdates % targetUpdateFrequency then
               /* update the targetQNetwork with the weights and
               configurations of the qNetwork                  */
20             targetQNetwork.update(qNetwork.internalState())
21         end
22     end
23 end
```

4.2.3 Hybrid Architecture

In the previous chapter we argue that the type of strategy used depends on the problem area. Agents in a nZEC have a common goal to achieve i.e. a zero energy status and therefore they need to do this in collaboration with the other agents. Literature in cooperative strategies (section 2.6.1) suggests the use of shared rewards or global rewards to enable cooperation between individual learners. Additionally, to learn, agents need to know the energy status of the community at any given moment. To enable this behavior we introduce a *Community Monitoring Service (CMS)*. The only purpose of this service is to collect individual energy status' from all agents at regular intervals and calculate the community energy status. Although this introduces a central point of failure, the computation is still distributed and is done individually by all agents. This makes the architecture *Hybrid* with minimal centralized computation. This design can be made completely decentralized with significant efforts and can be considered as a future work of this thesis. This service can be made fault-tolerant using various strategies that guarantees maximal availability at all times. To ensure privacy of data, information can be transmitted over HTTPS. This service ensures that all agents are aware of the community energy status before taking any decisions.

In the simplest form, a negative of the community energy status is used as a reward by an agent. Therefore, all agents receive similar rewards and adjust their behavior accordingly to maximize their reward. All agents are trained in an episodic manner and are rewarded at the end of each episode. Too delayed rewards can cause slower learning as the feedback for any action an agent takes is received after a many steps. For experimentation purposes we train the agents for a period of 3 simulated days for 1000 iterations. Every agent experiences a minimum of 144 states in each iteration. The number states experienced also depends on the number on interactions an agent has with other agents for energy sharing. Another alternative to this can be to train the agents in an online fashion and rewarding them at every timestep, hourly, daily, weekly or monthly.

4.3 System Architecture

Figure 4.1 represents the high level system architecture. Agents interact with each other using message passing mechanisms in in a peer to peer fashion to form a community. They initially register themselves with the community monitoring service which then informs the other agents of the new entrant. Agents contact the monitoring service at every timestep to get an approximate overview of the community energy status. We assume that, agents representing buildings may or may not be equipped with solar panels (180W Monocrystalline Solar Panels) to produce renewable energy and is the only source of energy production. Additionally, all buildings are equipped with batteries to store the energy generated by that building. The size and number of batteries depends on the type of building and its energy requirements. The exact configuration for agents is discussed in the next chapter.

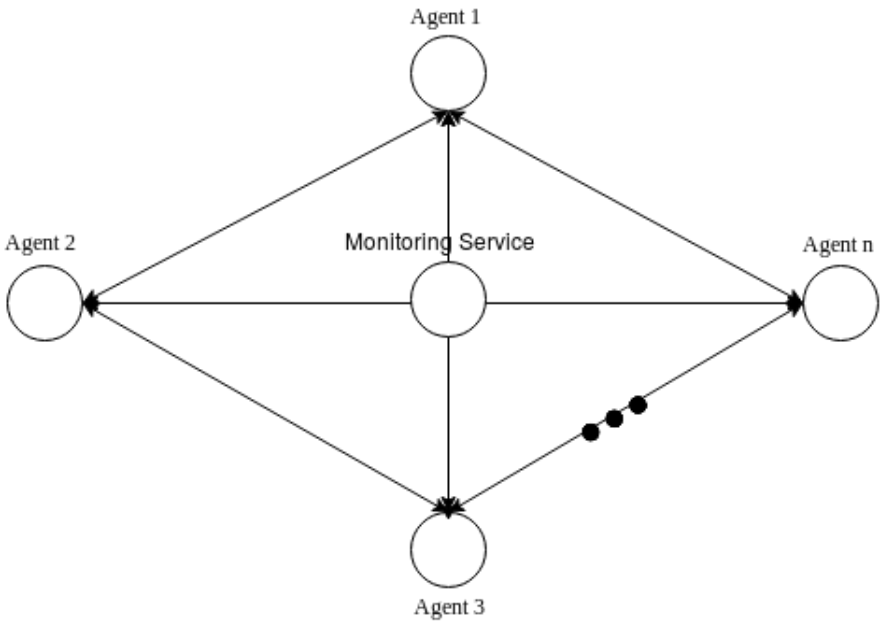

Figure 4.1: High level System Architecture

At any given time, an agent has 5 possible actions –

- Consume and store excess energy *(CONSUME_AND_STORE)*
- Request neighbour for additional energy *(REQUEST_NEIGHBOUR)*
- Request energy grid for additional energy *(REQUEST_GRID)*
- Grant energy request from ally *(GRANT_REQUEST)*
- Deny energy request from ally *(DENY_REQUEST)*

These actions are then filtered to generate a set of **physically possible** actions in that state. This means, the set of actions gets broken down into 3 exclusive sets –

1. $\{CONSUME\_AND\_STORE\}$
2. $\{REQUEST\_NEIGHBOUR, REQUEST\_GRID\}$
3. $\{GRANT\_REQUEST, DENY\_REQUEST\}$

One of these sets is then passed as a set of *legalActions* as shown in algorithm 1 to the decision making engine that either returns a random action with probability ϵ or an optimal action. In the first set as there no options available between the actions, the action is directly executed.

4.4 Summary

In this chapter we discussed the requirements for building a MARL based solution using DRL in the context of a zero energy community. We then address these requirements by designing a solution consisting of two major components. We also present the algorithm used for the agent learning process. In the next chapter we present the implementation of this design and the tools used.

Chapter 5

Implementation and Simulation

This chapter describes the implementation of DRL agents to enable energy sharing in a zero energy community. The environment is entirely simulated using a multi-agent framework. We describe in detail the implementation of DRL agents along with the communication mechanisms used by them. We further describe the mechanisms used to enable collaboration between the agents along with the implementation of the CMS.

5.1 Agent Implementation

For the development of this simulation agents have been implemented using a mix of technologies each suitable for a particular purpose. We use python 3.5, its multiprocessing libraries and frameworks based on them for our implementation.

5.1.1 OSBrain Framework

OSBrain is a general purpose multi-agent framework built in python that allows agents to run independently as separate processes and communicate with each other using *ZeroMQ*[36] message passing. ZeroMQ allows asynchronous communication between processes using different message passing mechanisms such as request-reply, push-pull, and

publish-subscribe. OSBrain additionally uses *Pyro4* that allows remote object invocation in python with ease.

On the highest level, agents have been built using this framework that allows basic agent group management and communication between them.

Agents use the following methods to communicate with each other –

- **def run_agent(name, nsaddr, serializer, transport)**
  This method instantiates an object of agent with a particular name, its host address, message serializer and transport protocol. If successful, it returns a Proxy of the agent.

- **def connect(server, alias)**
  This method establishes a connection with the remote agent using an instance of the client agent. The remote agent's unique name can be used instead of the address as an *alias*.

- **def send(alias, message)**
  This method send a message to the remote agent to which an connection has already been made.

- **def close(alias)**
  This method safely disconnects the client agent agent with the remote serving agent.

5.1.2 Messages

Agents communicate with each other using JSON messages. JSON messages are easy to parse and are human readable. A list of messages used by agents is shown below –

Energy Request

Whenever an agent representing a building faces a deficit of energy it sends a message to request for additional energy either to the energy grid or to a neighbouring building.

```
{
    'topic': 'ENERGY_REQUEST',
    'agentName':agent_name,
    'time': time,
    'energy': energy_amt
}
```

Energy Request Grant

If a building receives a request for energy transfer from a neighbouring building and decides to grant the energy request, it uses the following message format –

```
{
    'topic': 'ENERGY_REQUEST_ACCEPTED',
    'energy': energy_grant
}
```

Energy Request Deny

If a building receives a request for energy transfer from a neighbouring building and decides to reject the energy energy, it uses the following message format –

```
{
    'topic': 'ENERGY_REQUEST_DECLINE'
}
```

Additionally, to synchronize actions between all the agents, we have a separate process called *Synchronizer* that sends energy consumption data to all agents corresponding to the building they represent. All buildings used for the simulation are residential. This

is done purely for simulation purposes and can be easily removed in the actual system implementation.

Energy Consumption

This message is used by the Synchronizer process to send energy consumption data to all other agents at each timestep–

```
{
    'topic': 'END_OF_ITERATION',
    'time': 'YYYYmmdd HH:MM',
    'consumption' : consumption_value
}
```

5.1.3 RL Implementation

The RL process has been implemented as a pluggable module in the agent's learning and decision making process. The interaction between the agent and the environment is modelled as an MDP as explained in section 2.4. At the core of every MDP lies the state representation. Every agent has a *State* object that contains *AgentState* and *EnvironmentState* objects embedded in it. This helps an agent to keep track of its own internal state and its perspective of the world. At every time step these two objects are updated using the following methods –

- **state.agent_state.update(iteration, e_consumed, e_generated)**
  - used to update the consumed and generated energy by the agent.
- **state.environment_state.update(nzec_balance)**
  - used to keep track of the community zero energy status.

Apart from these, *State* also exposes two other methods to get a set of physically valid actions as described in the system architecture 4.3 and the reward for being in that state.

- **def get_possible_actions()**

  - returns a set of physically valid actions.

- **def get_reward()**

  - returns the reward for being in that state.

As discussed in the Design chapter 4, to manage the balance between agent exploration and exploitation, we simply decay the value of ϵ by 0.8 after every $(1/10_{th}) * number\_of\_training\_episodes$ starting with initial value of ϵ as 1. Decaying the value of ϵ by 0.8 strategically reduces the exploration overtime to 0.1 in the final set of episodes. This method is simple and proves to be very effective during training.

Q-function is a very important component, and its update lies at the core of the RL process. The Q-function is implemented as a separate python module and is plugged into the agent's RL process. It also caters for the *reward* and *discount* components of the MDP process. This is discussed in the next subsection.

5.1.4 Q-Update

The Q-update formula is provided in section 2.4 of chapter 2. It requires the following values –

- α - learning rate
- r - reward received
- γ - discount factor
- $Q(s, a)$ - Q function that estimates the Q-value for (s, a) pair.

α and γ are set at the start of the experiment, and the reward is received at each timestep from environment. The reward is generated only at the end of each episode and hence the rewards is 0 at all other times. The reward is simply the negative of the

nZEB status as discussed in section 4.2.3. We experimented with various values of α such as 0.2, 0.1, 0.0125, 0.00125 and 0.000125. Higher values of α led to divergence in agent policies. We kept on reducing the value of learning rate and performed the experiments until we finally chose the value 0.000125 that resulted in convergence. Similarly, we choose the value of γ to be 0.99 after experimenting with a range of values between 0.99 to 0.5. Discount factor decides the importance of future rewards and lower the value of γ the longer time its takes for the rewards to be propagated backwards.

The Q-function is implemented using *deep Q-Networks (DQN)* that uses two neural networks internally. The neural networks have been implemented using a deep learning library called *PyTorch*. One can simply create a ANN by inheriting the *torch.nn.Module* class from the PyTorch and overriding its *forward(self, input)* method. Increasing the number of layers and neurons per layer increased the computation time with no significant improvement in the nZEC status. A NN with fewer neurons and a single neuron had difficulty in converging to the optimal policy. We finally chose a NN with the following configuration -

- **Input Layer** with 63 neurons representing the encoded state and action.
- **2 x Hidden Layers** with 100 neurons each and the activation function as sigmoid. Experimentation with other activation functions like ReLU and Leaky ReLu had difficulties in convergence whereas using tanh didn't make a significant difference.
- **1 Output neuron** representing the Q-value with linear activation function.

DQN has two public methods –

- **def get_q_value(state, action)**
  - return an estimate for the (s, a) pair.
- **def update(state, action, reward)**
  - updates the Q-function based on the reward r received for taking action a in state s.

DQN internally uses a replay buffer that exposes the following functions –

- **def store_transition(state, action, reward)**
  - stores the state, action and reward as a tuple to enable combined-experience-replay.
- **def sample(batch_size)**
  - return a sample of random experiences and the most recent experience according to the batch_size.

Apart from this, the agent also has an API layer that it uses to communicate with the monitoring service. This is discussed in the detail in the next section. Figure 5.1 illustrates the class diagram for agent implementation.

5.2 Community Monitoring Service Implementation

The Community Monitoring Service (CMS) is responsible for maintaining the overall energy status of the energy community. It simply collects the energy consumption and generation data from all agents at every timestep. This information is then aggregated and returned to the agents whenever they request for the community net energy balance. This service has been built using Java 8, MySQL and supporting frameworks.

5.2.1 Spring Boot Framework

The monitoring service has been built using a REST API framework called *Spring Boot v2.0.4*. Spring Boot helps build standalone, production grade applications with minimum effort. It has an embedded *Tomcat* application server and a layer of security features that are enabled just by importing the plugin. APIs can be simply built by defining a class and a method and annotating them with *@RestController* and *@RequestMapping* Java annotations respectively. For further details the reader can refer the documentation provided on the official website [73].

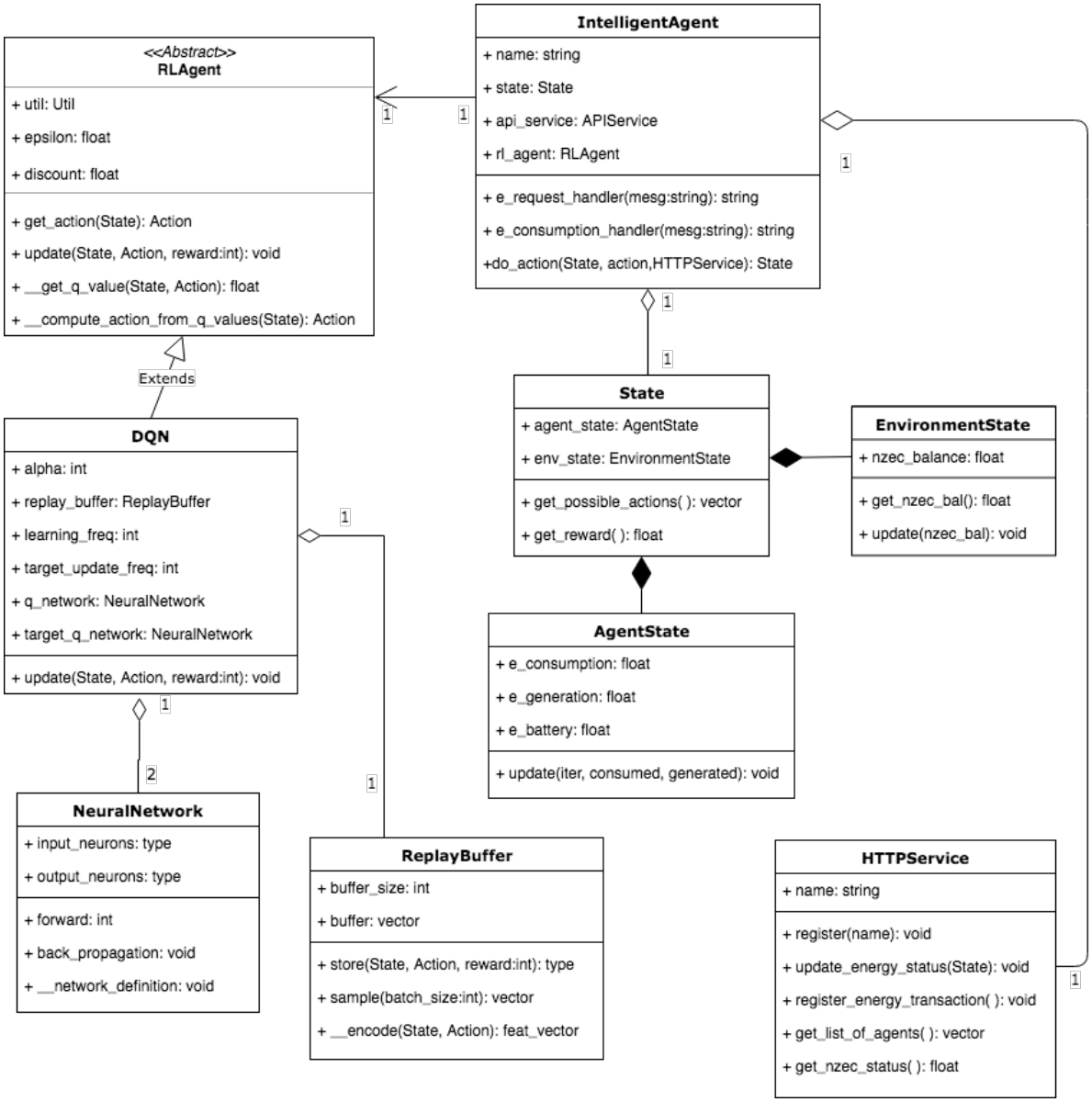

Figure 5.1: Class diagram for Intelligent Agent implementation.

The following APIs are exposed as a service to the intelligent agents discussed in the previous section –

- **/agent/register**

  Request Method - POST

  Allows agents to register with the monitoring service.

- **/agent/ping**

  Request Method - POST

  Notifies the monitoring service that the calling agent is active.

- **/agent/active**

  Request Method - GET

  Returns a list of active agents with the addresses.

- **/energy/status/grid/{iteration}**

  Request Method - GET

  Retrieves energy status of the community for a particular training iteration.

- **/energy/status**

  Request Method - PUT

  Used by an agent to update the energy balance of the community.

- **/energy/transaction**

  Request Method - POST

  Logs a energy transaction made by agent with another agent or the energy grid.

5.2.2 Database

The information gathered by the APIs in the previous subsection is stored in a relational database called MySQL. The overall architecture of the database is shown in figure 5.2.

This database design is trivial and hence we do not discuss it in detail. There are 4 tables namely –

- **iteration_log**

  - this contains the total energy borrowed from neighbours, the central grid, and energy generated & consumed by a particular house during one training cycle.

- **energy_transaction**

  - this contains all the energy transactions done for sharing energy between houses.

- **agent**

  - this contains a list of agents and a flag which indicates if they are active or not.

- **summary**

  - this contains information similar to the *iteration_log* table but only for the currently active episode.

5.3 Dataset

5.3.1 Energy Consumption Data

The energy consumption data used in this simulation for training DRL agents has been generated using *Load Profile Generator (LPG)* [72]. LPG generates residential energy consumption curves by performing behavioral simulation of the people in the households. We have generated data for three different types of household for a period of 1 year from 1-January-2014 to 31-December-2014.

The residents of the three independent households are –

1. Family with 1 child and both parents at work (Code - CHH01).

2. Retired Couple with no work (Code - CHS02).

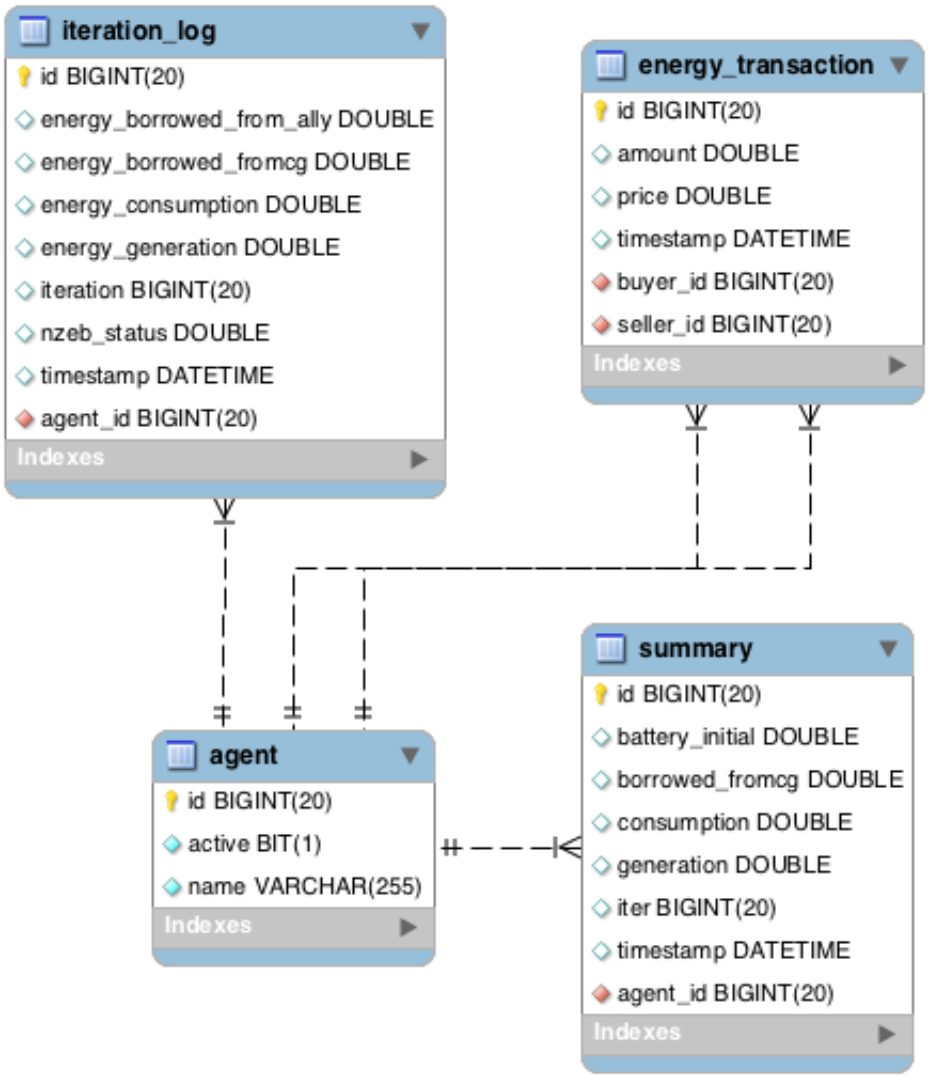

Figure 5.2: Database Schema for Monitoring Service implementation.

3. Students sharing a house (Code - CHR52).

We use energy consumption data of three weekdays during both winter and summer from these datasets to train our agents. Energy consumption graphs for these datasets are shown in figures 5.3, 5.4, 5.5, 5.6, 5.7 & 5.8. The timescale on the X-axis is in (MM DD
YYYY) format.

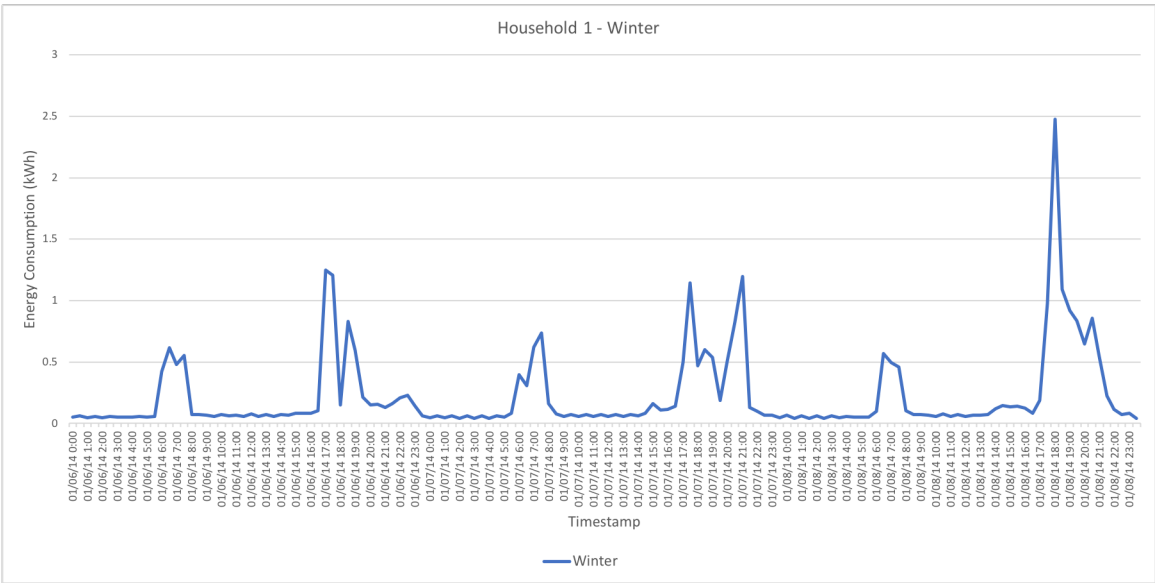

Figure 5.3: Household 1 - energy consumption during winter.

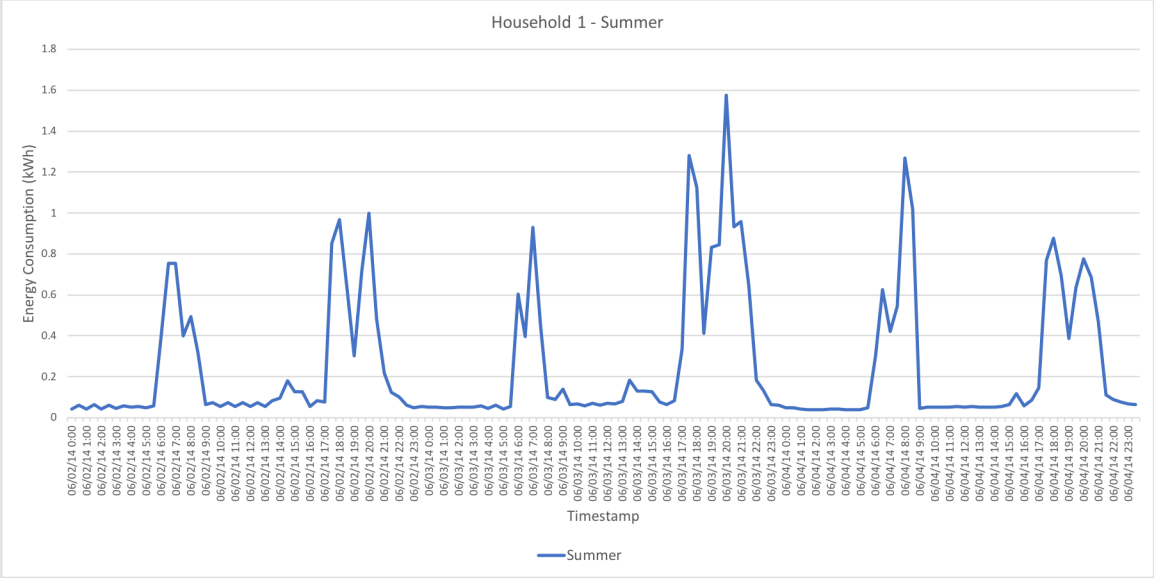

Figure 5.4: Household 1 - energy consumption during summer.

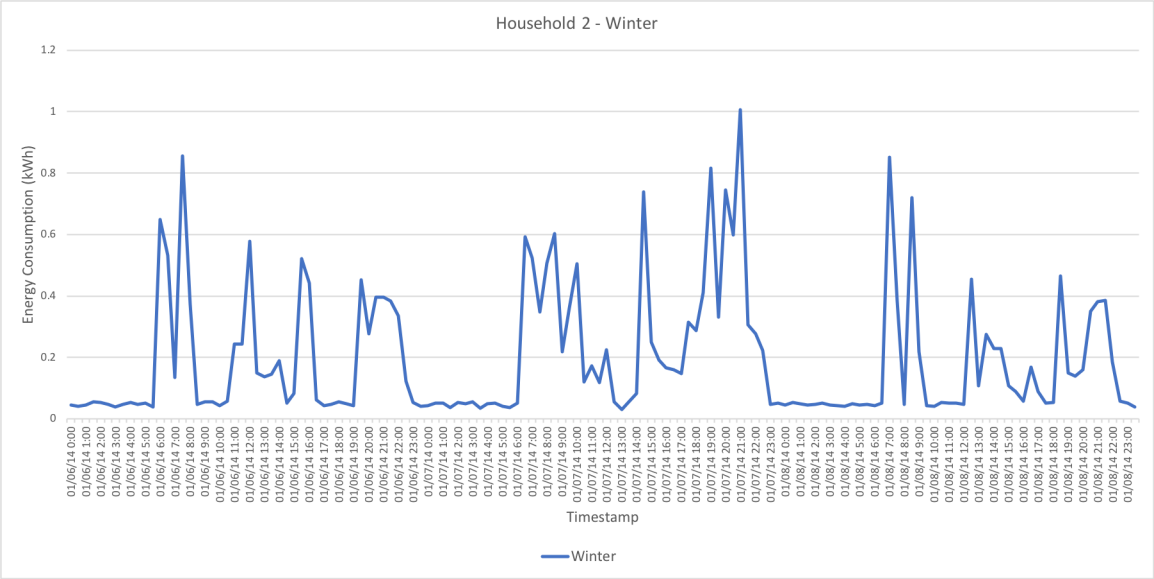

Figure 5.5: Household 2 - energy consumption during winter.

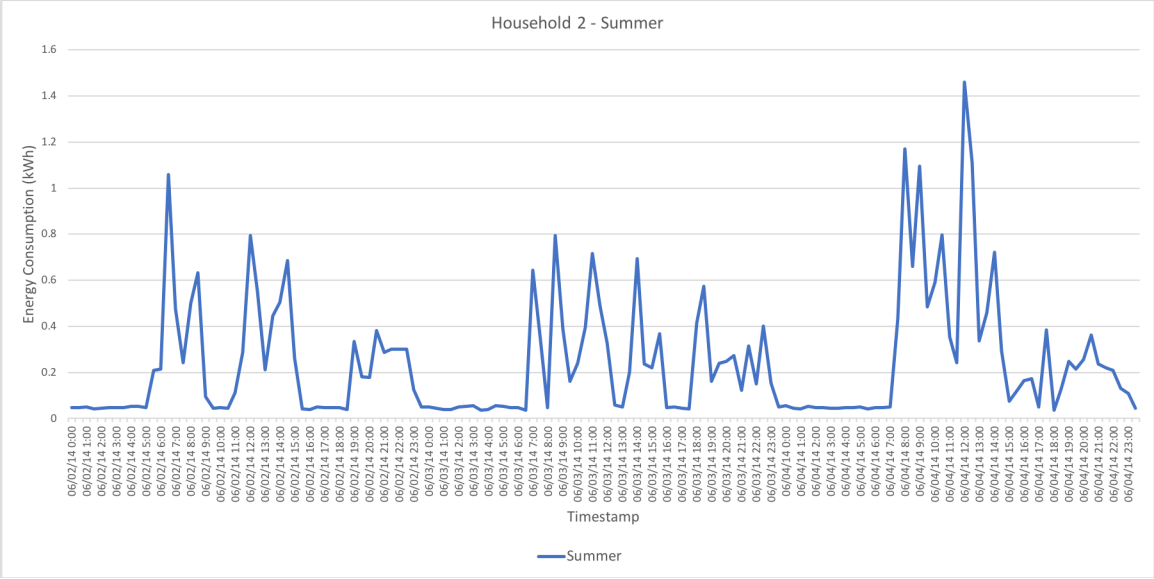

Figure 5.6: Household 2 - energy consumption during summer.

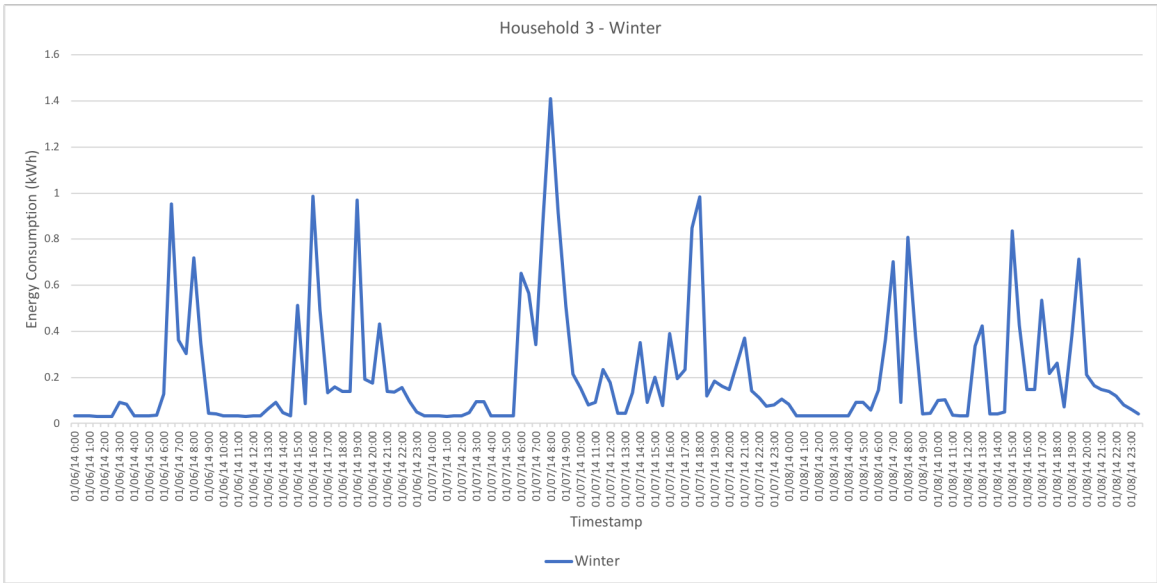

Figure 5.7: Household 3 - energy consumption during winter.

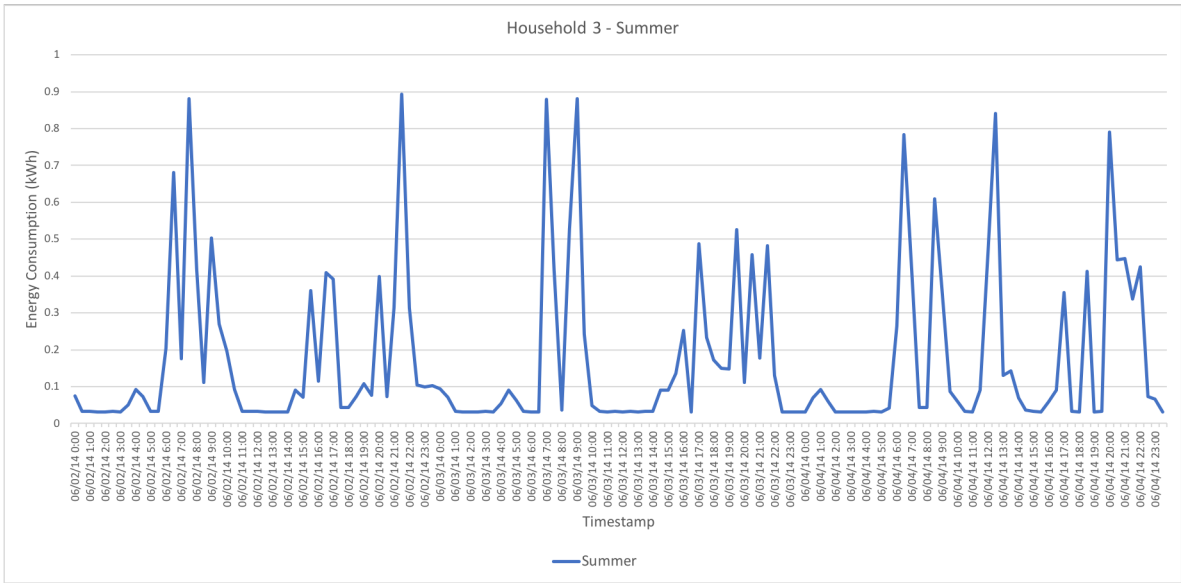

Figure 5.8: Household 3 - energy consumption during summer.

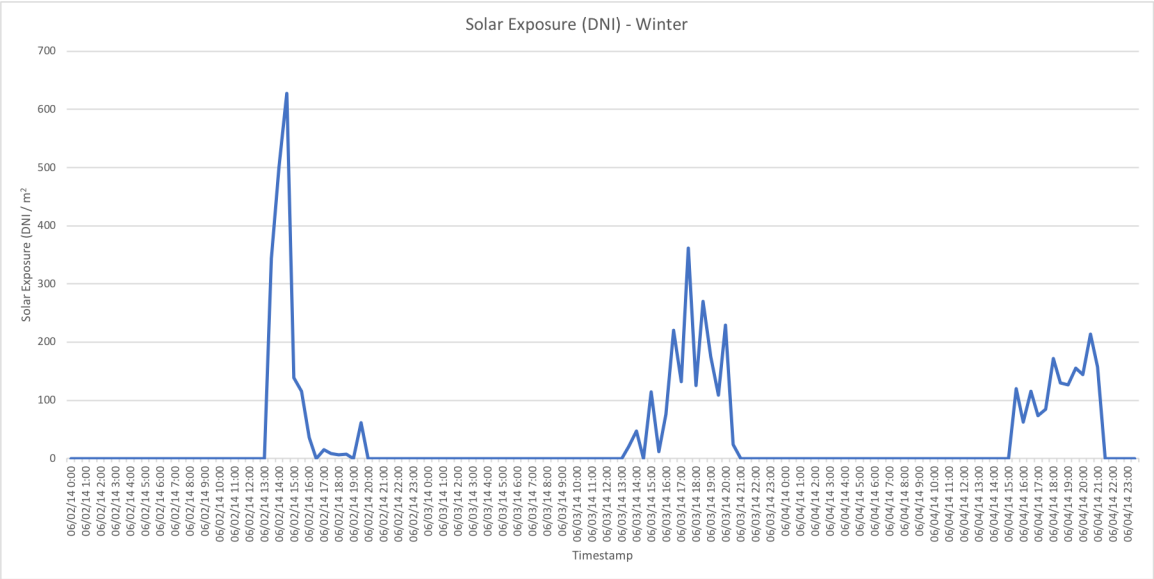

Figure 5.9: Solar exposure during Winter.

As the households are present in the same locality, we assume that they receive the same solar exposure to generate energy from solar panels. For this simulation we use solar irradiance data of Toronto city extracted from the NSRDB database [65]. A plot of solar exposure has been shown for the same days in figures 5.9 & 5.10 during summer and winter as used for energy consumption.

5.4 Simulation

Once we have these individual components ready, we configure them to communicate with each other by providing each others' addresses. Agents join the community after successful registration with the CMS. The Synchronizer process then emits consumption data to all the agents, and the agents then choose the appropriate action accordingly. Whenever an house faces a energy deficit an agent can take two possible action i.e. it can request for additional energy either to the supply grid or a neighbouring house. The selection of action depends $\epsilon - greedy$ policy and the learned optimal policy. Similarly, if a house receives a energy request from a neighbour, it can choose between granting or rejecting that energy

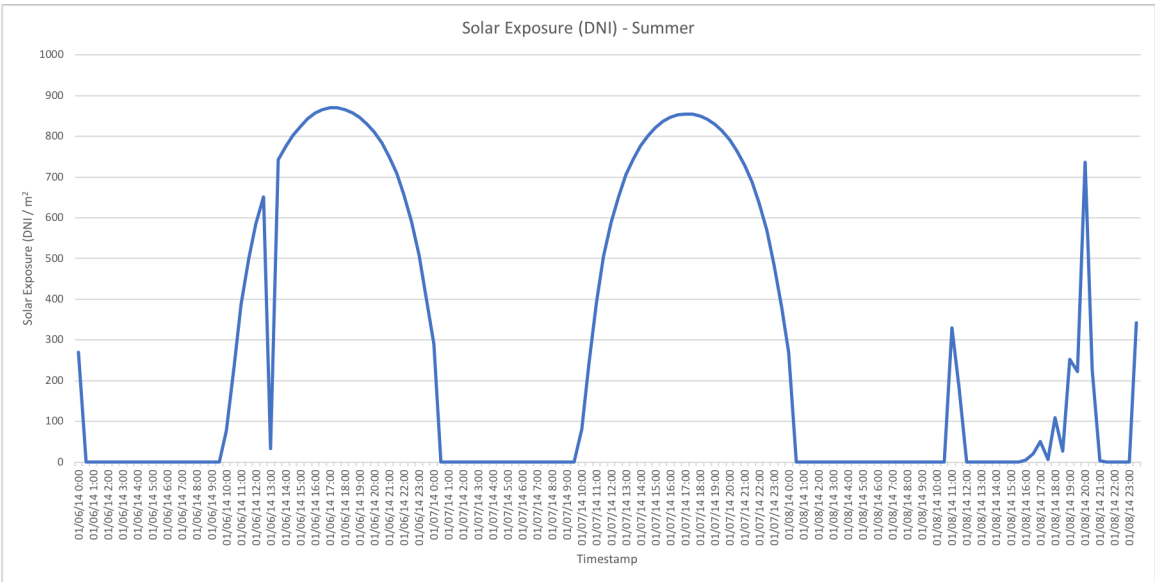

Figure 5.10: Solar exposure during Summer.

request.

In every timestep, an agent can make only 1 successful energy transaction with a neighbour. If the neighbour is unable to completely suffice the energy request, the remainder is borrowed from the supply grid. Agents use a *random neighbour selection* strategy. If a energy request is rejected by a neighbouring house, another house is selected at random and this continues till the list of available houses is exhausted.

We train all the agents for 500 episodes over 3 simulated days during Summer and Winter and evaluate their results.

Chapter 6

Evaluation

In this chapter we present an evaluation of the intelligent energy sharing solution we have designed and implemented to enable zero energy communities. We firstly present the evaluation objectives and then the evaluation metrics that measures the performance of our solution. We discuss in detail the experiments performed for evaluation and present their analysis.

6.1 Evaluation Objectives

The goal of this chapter is to evaluate how well does the Design in chapter 4 address the requirements specified in section 4.1. The main objectives of this intelligent energy sharing solution is to provide an autonomous, self-evolving, real-time solution that allows collaboration with other agents to enable a zero energy community. We have further discussed in chapter 4 how the design addresses these requirements, however, the success of this solution also depends on its performance in different evaluation scenarios.

Formally, the intelligent energy sharing solution this thesis presents is said to have succeeded if it satisfies the following conditions –

1. Agents are able to operate autonomously. Autonomous solutions are much more per-

formant and cost effective when compared with traditional solutions. Agents should initiate and complete energy transactions with minimal or no human intervention.

2. As these systems operate in real time, the time taken for decision making should be as small as possible.

3. Intelligent agents should be able to adapt to changing environmental conditions. Distributed generation and consumption of energy along with changing climatic conditions introduces high uncertainty of energy production. Similarly, actions taken by other agents affect the environment and make it dynamic. An agent should be able to adapt to these changes and make optimal choices to reach the net zero energy goal.

4. Intelligent agents should learn to collaborate with each other and their actions should lead to a zero energy community.

5. The system should scale well. Energy communities may have a few houses to hundreds of houses. Agents should still be able to learn the optimal behavior and converge to a net zero energy community.

6.2 Experiments

We evaluate the effectiveness of this energy sharing solution in different simulated scenarios. The agent learning parameters remain the same for all experiments as discussed in the implementation section. Here, we only change the the environment and house related parameters and observe the effects on the agent learning process.

6.2.1 Experimental Setup

Before starting with the evaluation, we describe a few terminologies and configurations that are common to all evaluation scenarios below. Agents representing houses have names

for ease of reference. Agents representing House 1, 2, 3, and 4 have names Alice, Bob, Charlie, and Dave respectively. These houses are equipped with solar panels and is the only source of on-site renewable energy. A detailed configuration for each scenario is provided in the tables 6.1 -6.4 below. The code in **Consumption Prof.** column corresponds to the house code in sub-section 5.3.1. The 4th column represents the number of solar cells installed by the house to generate energy. These number of cells might not be relevant in the real world and are purely used to simulate conditions to evaluate the performance of agents, as solar panels might have standard number of solar cells. We begin training our agents at 00:00 hrs i.e. during midnight and as there is no solar exposure during that time all agents are forced to borrow energy from the supply grid. This introduces a bias that leads to problems in convergence. To overcome this bias, we provide agents with an initial battery charge as shown in the 5^{th} column. The batteries used in this simulation are have a maximum power voltage of 12V and a charge rate of 100 amps for 20 hours. We convert this into kWh for ease of calculation (calculates to 1.2 kWh). Each house is equipped with 6 such batteries and has a total storage capacity of (1.2 * 6 =) **7.2 kWh**.

6.2.2 3 Houses Having Varying Generation Capacity

We perform this experiment with 3 homes having different consumption and generation profiles. The configuration used in table 6.1 has been has been used to evaluate the agent's learning performance during summer and winter. We train the agents for 3 days during both times and evaluate the obtained results.

House No.	Agent	Consumption Prof.	n*(SCells)	Batt. Init (kWh)
1	Alice	CHH01	72	7.2
2	Bob	CHS02	54	2.5
3	Charlie	CHH01	12	5.0

Table 6.1: Configuration for 3 houses having different generation capacity.

Evaluation during Winter

Figure 6.1 shows a comparison of strategies used to achieve nZEC. The *Learned Behaviour* label in the figure indicates the learning behaviour of our intelligent energy sharing solution. Similarly, *Always Share, Random, and Never Share* labels indicate always-share-energy strategy, random-action-selection strategy, and no-energy-sharing strategy respectively. This is true for the other images too in the following sections. The worst performing strategy is a no-energy-sharing environment where houses focus on optimizing their own gains without consideration of others. This is similar to the previous definition of nZEC where no energy sharing was involved with the only difference that houses were *assumed* to be completely self-sustainable. In this scenario, to achieve zero energy status as a community, the optimal behavior is to always share energy with neighbours as there are no other components (like a energy pricing model) that will affect the decisions taken by agents. As seen in figure 6.1 as agents train, they learn the optimal behavior (denoted by the Learned Behaviour label that is represented by a blue line) to always share energy (grey line).

To achieve this optimal behavior agents learn to borrow more energy from their neighbours and less energy from the supply grid. This is evident from figures 6.2 and 6.3. However, we also observe that (fig. 6.3) 2 agents (Alice & Bob) learn to request for additional energy to the supply grid. This is because, agents learn to operate selflessly in order to support the third agent (Charlie) that has lesser renewable generation. This helps to improve the net zero energy balance of the community.

Additionally, due to an indirect effect energy sharing, wastage of renewable energy produced in the community is reduced. As houses share energy with each other, a deficit is introduced their locally available batteries and they can now store the generated surplus energy which would otherwise have been wasted due to completely charged batteries. This behavior can be observed in figure 6.4 where two houses (represented by agents Alice & Bob) generate slightly more energy over time. Although, the difference is very minor this

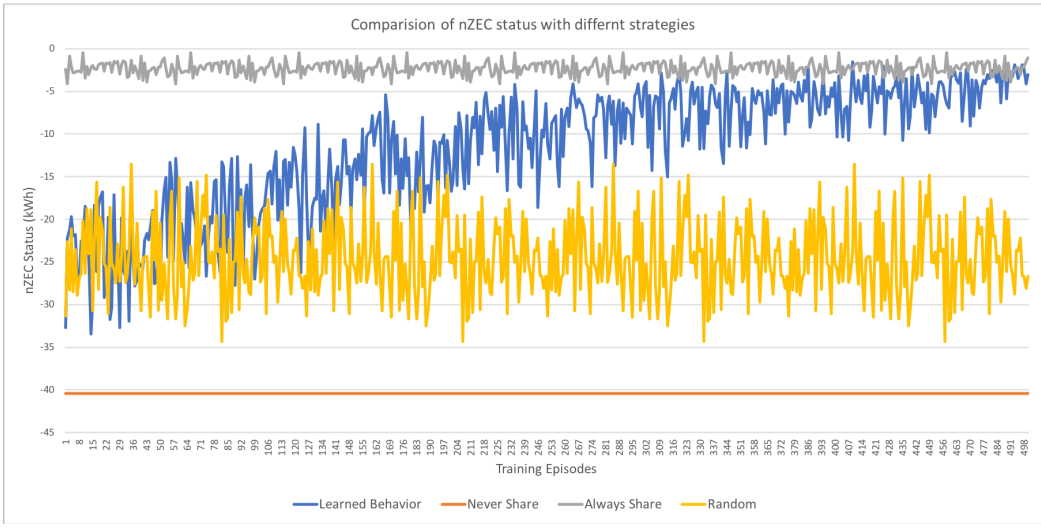

Figure 6.1: Comparison of nZEC status using different strategies with 3 houses during Winter.

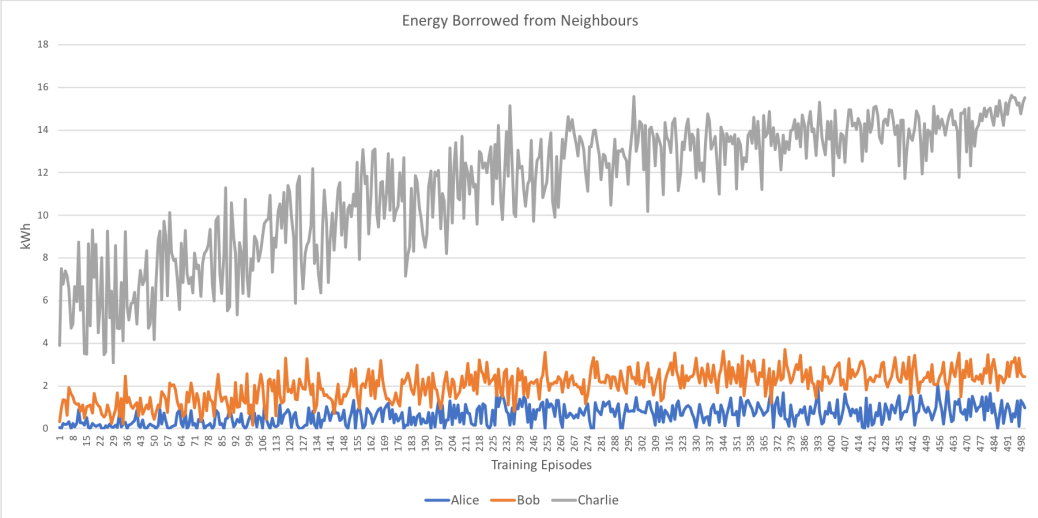

Figure 6.2: Agents learning to borrow energy from neighbours during Winter in a simulation of 3 houses.

does have a positive impact on the community.

Figure 6.5 shows the individual nZEB status' achieved by the houses. This too shows a positive trend.

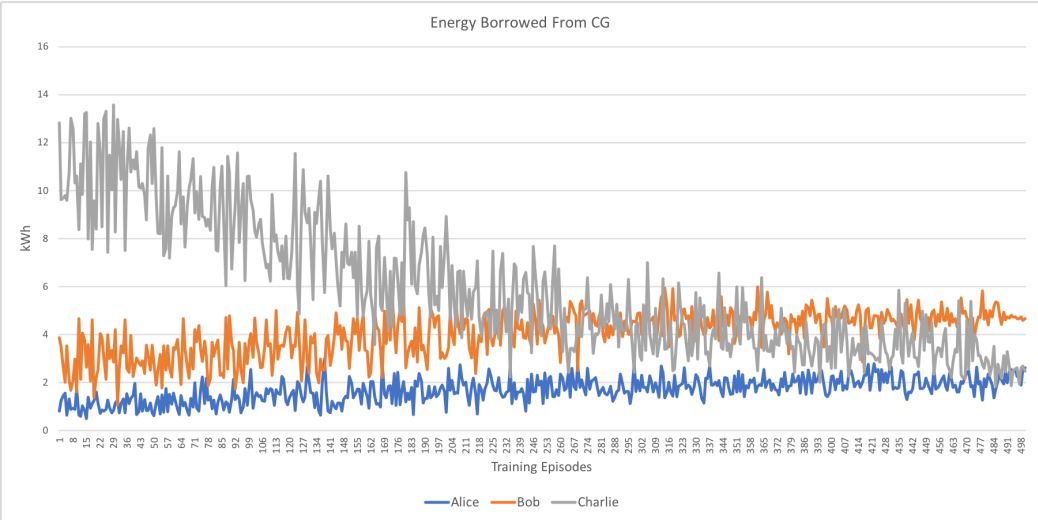

Figure 6.3: Agents learning to avoid borrowing energy from the supply grid during Winter in a simulation of 3 houses.

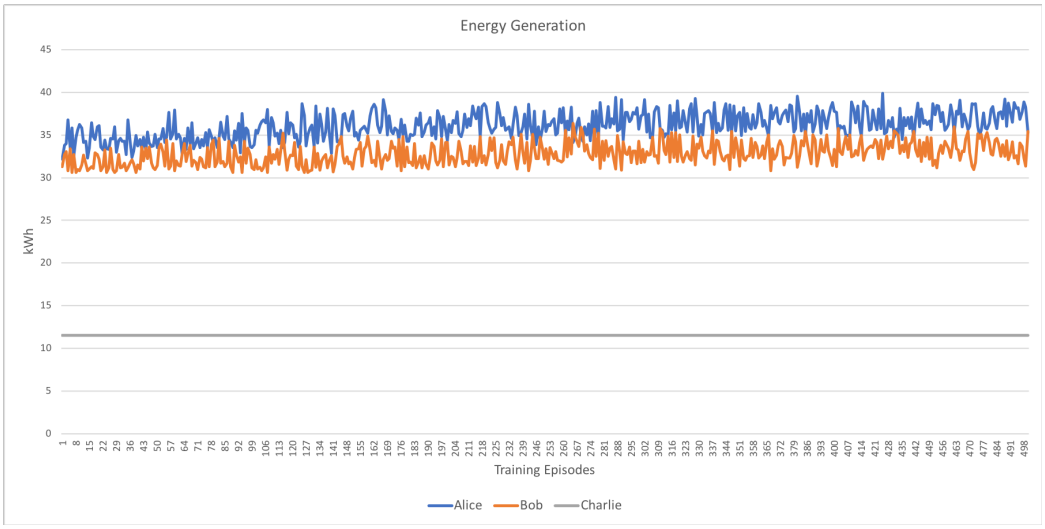

Figure 6.4: On-site generated energy comparison during Winter in a simulation of 3 houses.

Evaluation during Summer

Figure 6.6 shows a comparison of strategies used to achieve nZEC during Summer. The lines overlap with each other completely and therefore only a single line is visible. With the configuration in table 6.1, during summer houses are entirely self-sustainable and therefore have no need to share energy with each other. As there is no energy sharing involved,

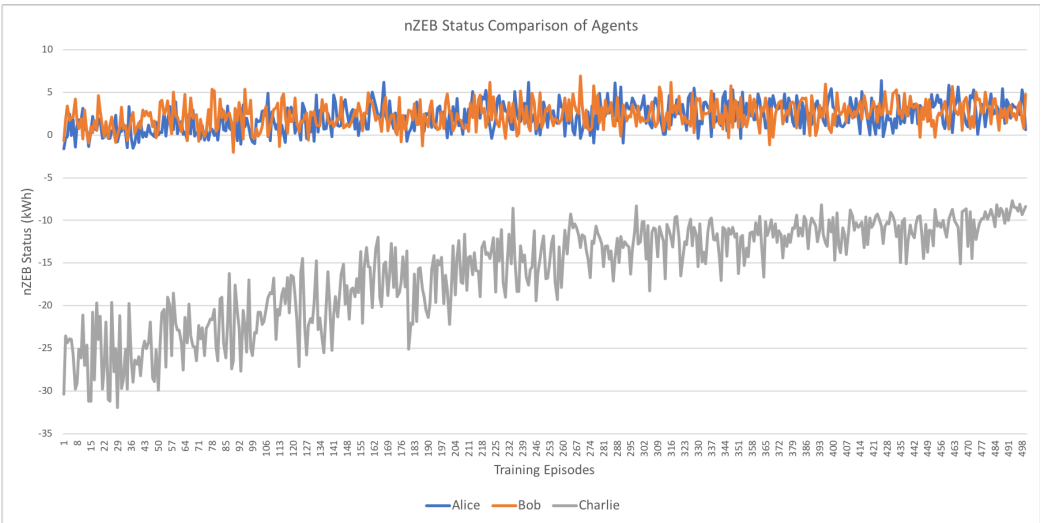

Figure 6.5: Individual nZEB status comparison of houses during Winter in a simulation of 3 houses.

no decisions are made regarding the choice of borrowing energy from grid or neighbours. Therefore, all strategies have similar nZEC status of **-1.72 kWh**. This minor negative status of **-1.72 kWh** is due to the initial bias introduced by the training data as agents start training at midnight and have no solar exposure to produce energy during that time.

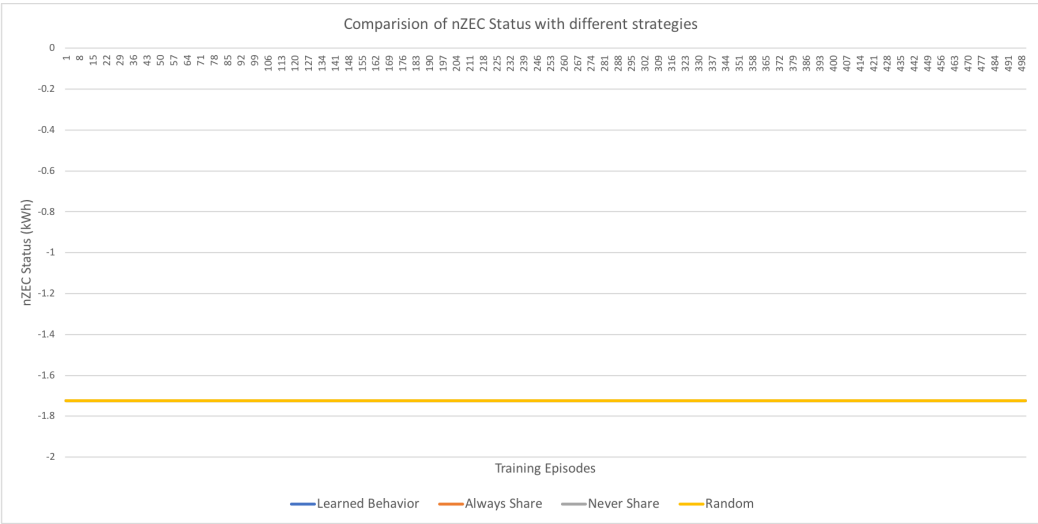

Figure 6.6: Comparison of nZEC status using different strategies with 3 houses during Summer.

6.2.3 4 Houses Having Varying Generation Capacity

In the previous scenario (evaluation of 3 houses during summer), all houses were self-sustainable and therefore no learning behaviour could be observed. To evaluate the adaptive behavior of agents in different climatic conditions, we introduce a fourth house with no source of renewable energy generation. We firstly evaluate this setup during Summer and then in Winter. This also helps simulate an environment with different levels of energy generation including no energy generation. We argued that such energy communities can exist (section 1.1) and hence the need for energy sharing in such communities. A detailed configuration for these houses is provided in table 6.2.

House No.	Agent	Consumption Prof.	n*(SCells)	Batt. Init (kWh)
1	Alice	CHH01	72	7.2
2	Bob	CHS02	54	2.5
3	Charlie	CHH01	12	5.0
4	Dave	CHR52	0	0

Table 6.2: Configuration for 4 houses with varying generation capacity.

Evaluation during Summer

Figure 6.7 shows a comparison of nZEC status' achieved using different strategies with 4 houses during Summer. In comparison with the optimal behavior i.e. always share energy, the learned behavior performs well. There is a difference of approximately 7 kWh between the two over 3 days in terms of performance. This is not as performant as the previous case with 3 houses, but at-least guarantees a lower bound (the worst case) on the nZEC status. When compared with the no-energy sharing scenario the agents perform very well (difference of 64 kWh).

Figures 6.8 and 6.9 show how the agents learn to borrow energy from the neighbours instead of the supply grid.

Figure 6.10 shows the energy generated by individual houses during summer and fig. 6.11 shows the individual nZEB status' attained by these houses.

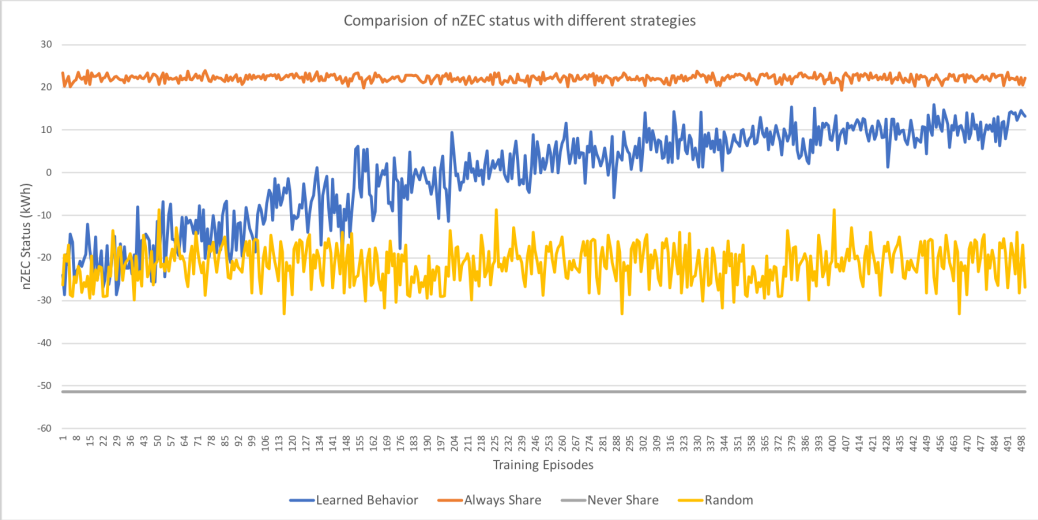

Figure 6.7: Comparison of nZEC status using different strategies with 4 houses during Summer.

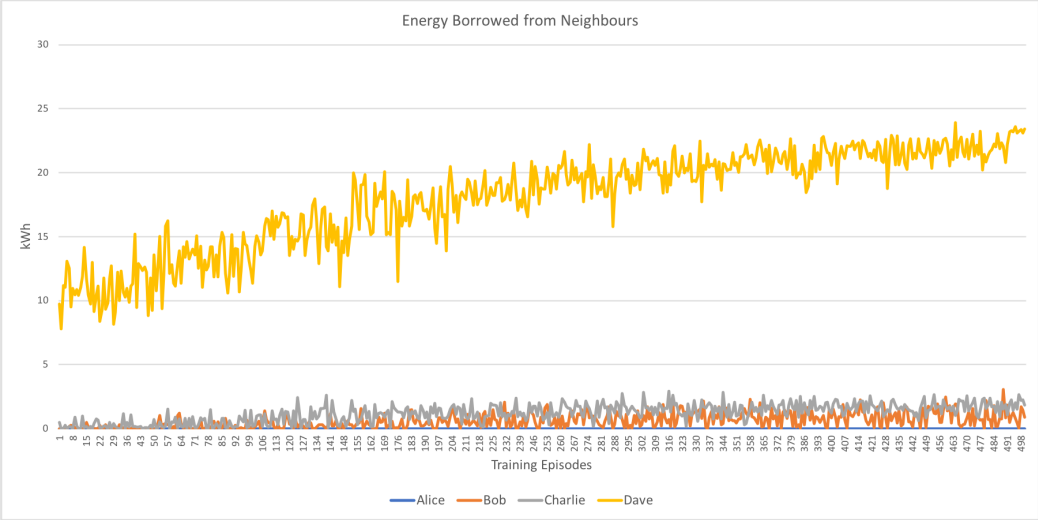

Figure 6.8: Agents learning to borrow energy from neighbours during Summer in a simulation of 4 houses.

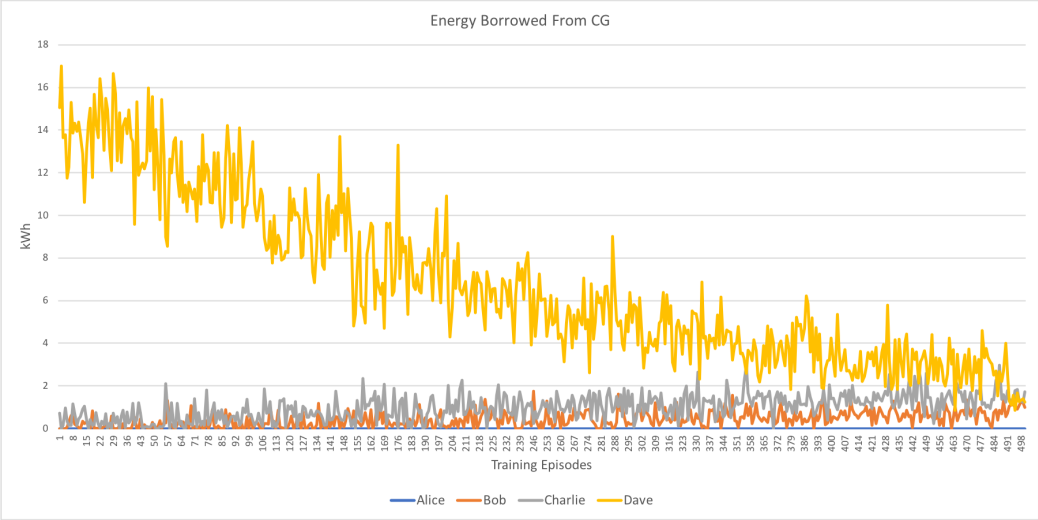

Figure 6.9: Agents learning to avoid borrowing energy from the supply grid during Summer in a simulation of 4 houses.

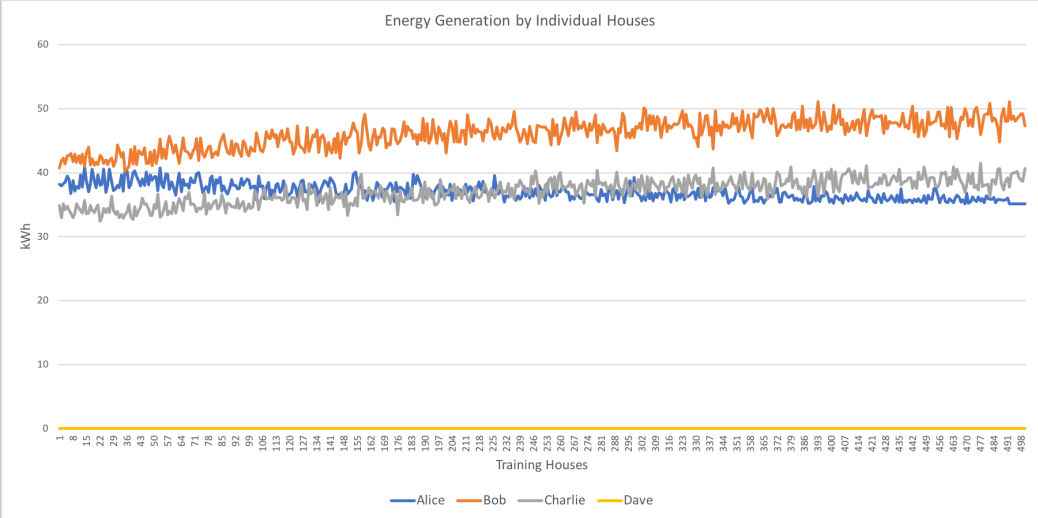

Figure 6.10: On-site generated energy comparison during Summer in a simulation of 4 houses.

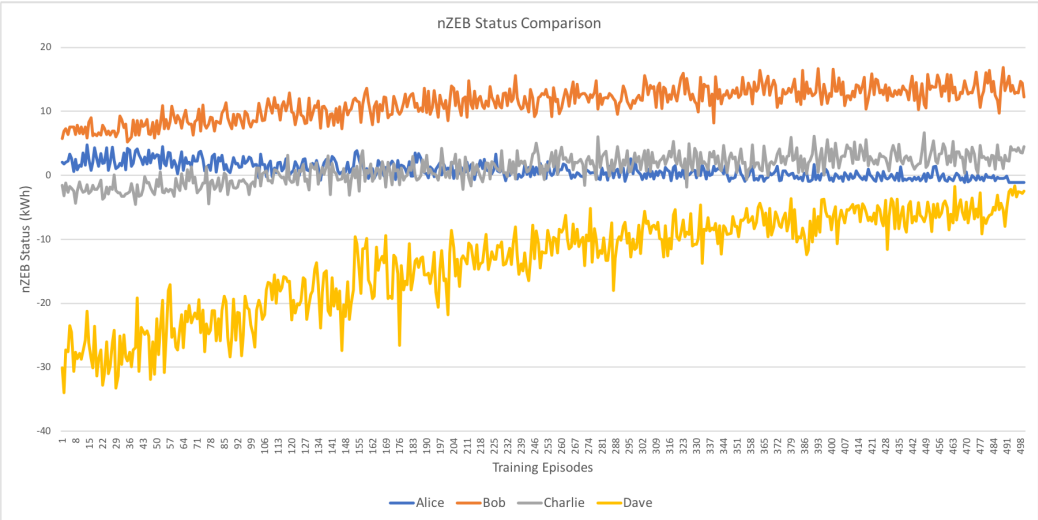

Figure 6.11: Individual nZEB status comparison of houses during Summer in a simulation of 4 houses.

Evaluation during Winter

Similar to the previous section, we evaluate the performance of the intelligent energy sharing solution during Winter as well as its ability to adapt to changes in the environment. Figure 6.12 shows a comparison of nZEC status' achieved using different strategies with 4 houses during Winter. Here too, the results of the learned behaviour are much better than the no-energy-sharing and random-action-selection strategy but comparable to an always-share-energy strategy.

6.2.4 4 Houses Having Same Generation Capacity

In the previous section we have evaluated houses having different or no energy generation capacities. In this section we evaluate the performance of our intelligent energy sharing solution in a scenario when all houses have similar configuration i.e. same number of panels and initial battery charge.

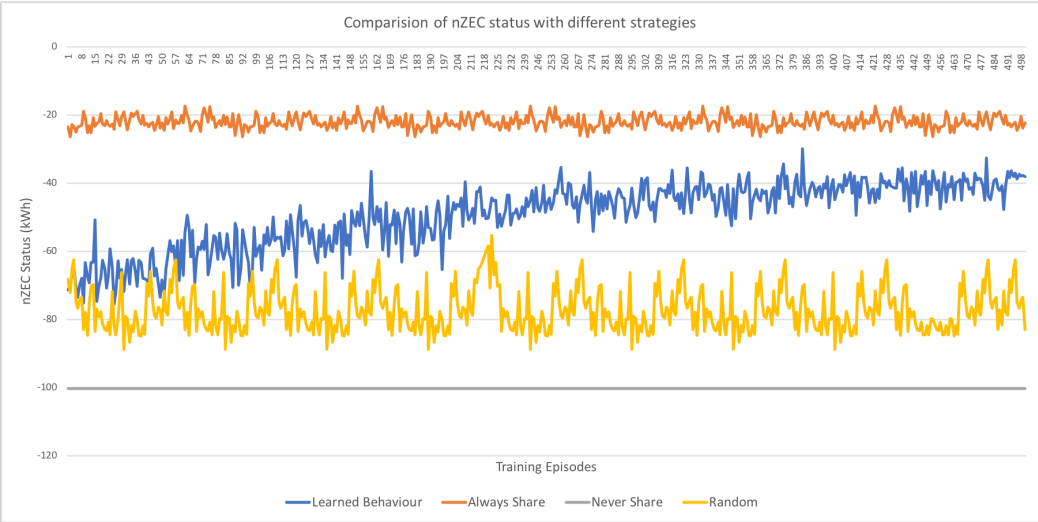

Figure 6.12: Comparison of nZEC status using different strategies with 4 houses during Winter.

House No.	Agent	Consumption Prof.	n*(SCells)	Batt. Init (kWh)
1	Alice	CHH01	24	2.5
2	Bob	CHS02	24	2.5
3	Charlie	CHH01	24	2.5
4	Dave	CHR52	24	2.5

Table 6.3: Configuration for 4 houses having same generation capacities.

Evaluation during Winter

Figure 6.13 shows a comparison of different strategies with 4 houses having the same configuration but different consumption patterns. As seen in the figure, the learned behaviour curve tends towards the optimal strategy i.e. always-share energy but has similar results to that of a random-action-selection strategy. However, it provides a least performance boundary that is better than a no-energy-sharing strategy.

Evaluation during Summer

Figure 6.14 shows a comparison of agent learning with different strategies during summer. We observe *divergence* in the nZEC status using the DRL based strategy and it performs worse than a random strategy. Agents learn over time to not share energy. Agents learn

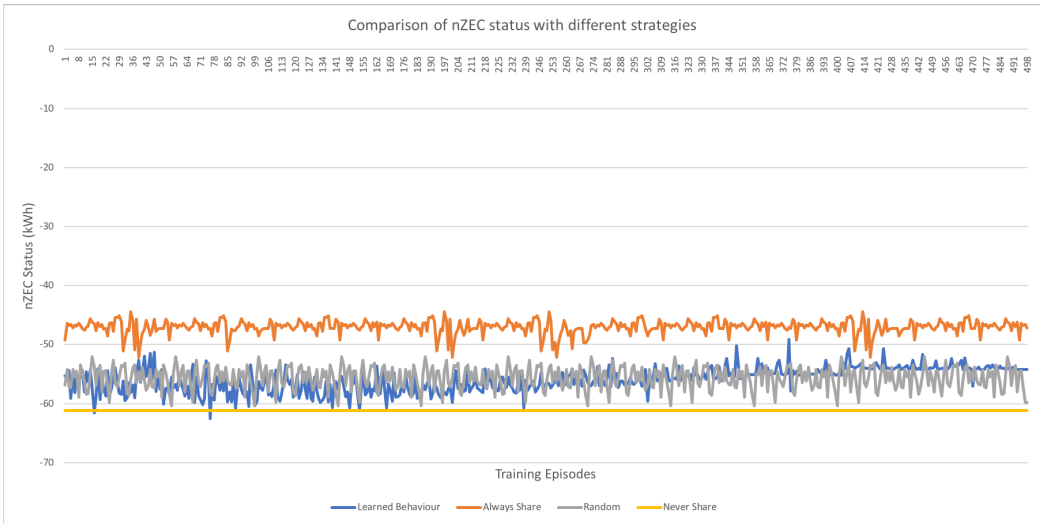

Figure 6.13: Comparison of nZEC status using different strategies with 4 houses during Winter, where all houses have similar configuration.

to act selfishly and focus on optimizing their own nZEB status instead of the nZEC status. Tuning the reward function further that incentives energy sharing may help with convergence.

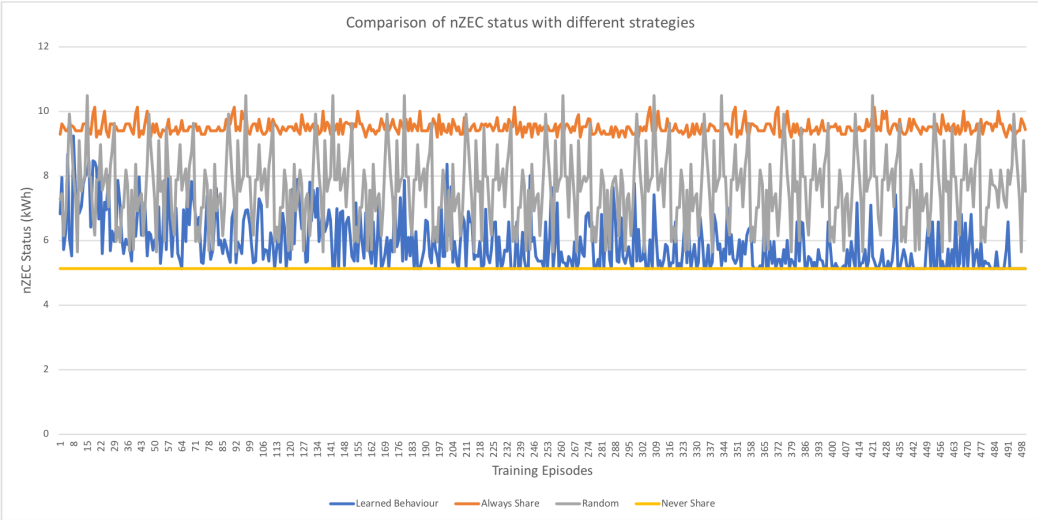

Figure 6.14: Comparison of nZEC status using different strategies with 4 houses during Summer, where all houses have similar configuration.

6.2.5 10 Houses Having Mixed Generation Capacity

Previous sections have evaluated our DRL based energy sharing solution in environments where houses have varying as well as similar energy generation capacities. We were able to successfully test its ability to adapt to different climatic conditions. In this section, we evaluate its ability to learn the optimal solution when the number of agents grow. For this, we have simulated 10 houses with varying generation capacities as shown in table 6.4.

House No.	Agent	Consumption Prof.	n*(SCells)	Batt. Init (kWh)
1	A1	CHH01	72	7.2
2	A2	CHS02	54	2.5
3	A3	CHH01	12	5.0
4	A4	CHR52	0	0
5	A5	CHH01	12	5.0
6	A6	CHH01	72	7.2
7	A7	CHS02	54	2.5
8	A8	CHH01	12	5.0
9	A9	CHR52	0	0
10	A10	CHH01	12	5.0

Table 6.4: Configuration for 10 houses having varying generation capacity.

Figures 6.15 and 6.16 show a comparison of nZEC status with different strategies during Winter and Summer respectively. In both cases, we observe that our solution performs better than a random-action-selection strategy and a no-energy-sharing strategy. During Winter, the distinction between always-share-energy strategy, learned strategy, and random-action-selection strategy is very minor, however, we still can observe that our solution tends towards the optimal strategy.

6.3 Evaluation Summary

In this chapter, we have evaluated our DRL-based energy sharing solution in 2 environments, 3 configurations and 3 different scales. A combination of these have been incorpo-

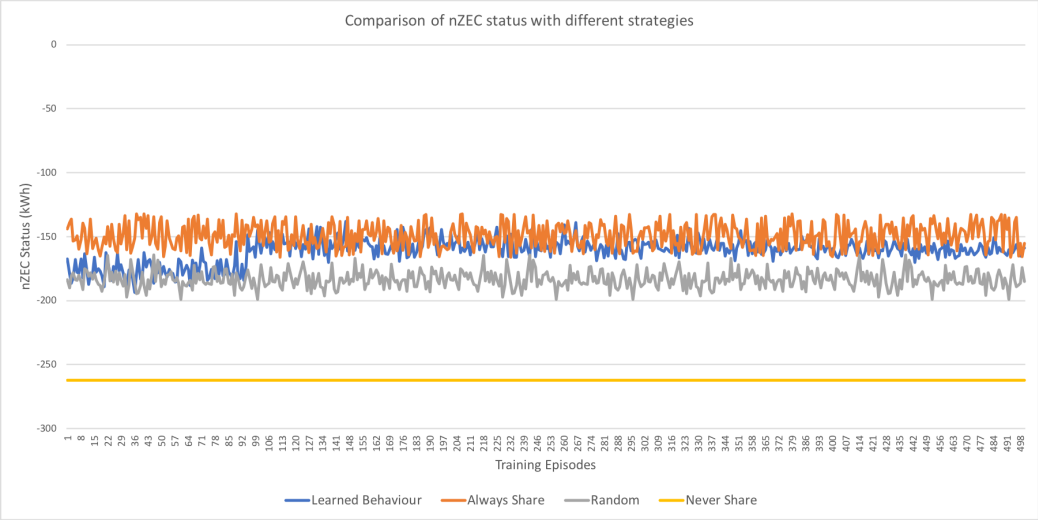

Figure 6.15: Comparison of nZEC status using different strategies with 10 houses during Winter.

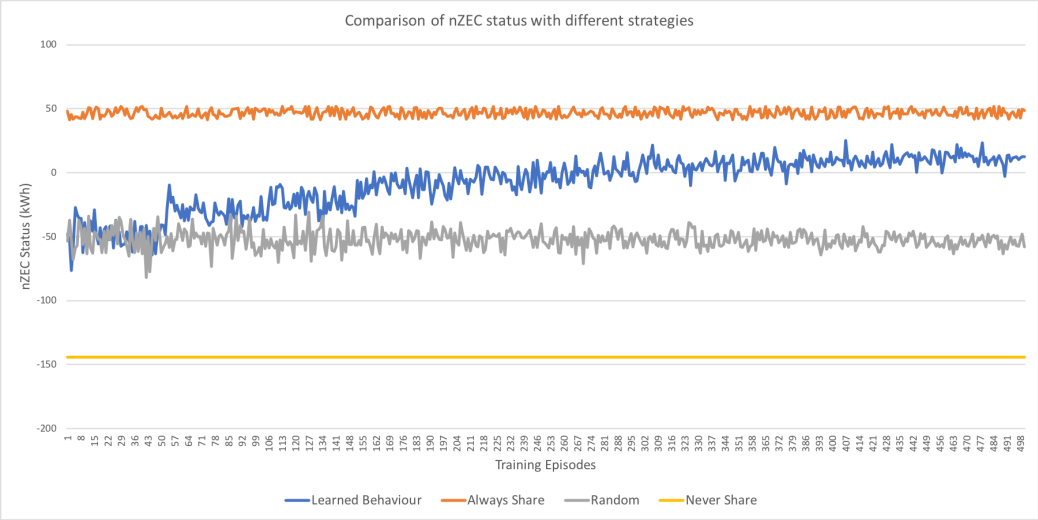

Figure 6.16: Comparison of nZEC status using different strategies with 10 houses during Summer.

rated by 4 major evaluation scenarios. Agents operate autonomously and select the best action in any given situation by exploiting the already learn't policy (Objective #1 as listed in section 6.1).

Agents were trained for 3 simulated days with 4 houses in an episodic manner for 500 episodes which took around 40 hours. This 40 hours also includes the time required for synchronizing the actions and exchanging messages between the agents. We have trained our agents on a single Linux machine with 4 cores and 15GB main memory on Google Cloud Platform (GCP). As the number of agents grow, the training time also increases due to limitations on the physical resources available. However, once the agents are trained, the time taken for decision making is very small (approx 100 ms) . This time can change depending on the communication latency and hardware used. A standard Q-learning algorithm uses a Q-function that internally uses a Q-table. As the number of states grow the size of this Q-table grows drastically and therefore time taken for decision making also increases accordingly. DRL completely avoid this by replacing the Q-function with deep learning based function approximator that has consistently much lower inference time in problems with huge state spaces such as this. This ensures that the time taken for decision making is as small as possible (Objective #2).

Evaluations indicate that our solution improves the nZEC status drastically when compared to a no-energy-sharing strategy and random-action-selection strategy in most cases (except with 4 houses with same configuration during Summer). An improvement of 40 kWh with 3 houses during winter and over 60 kWh with 4 houses during summer over 3 days in the overall community's energy balance was found when compared to a no-energy-sharing strategy. Similarly, with 10 houses a improvement of 97 kWh during Winter and 156 kWh during Summer was found. Our solution also adapts well to changing seasons i.e. summer and winter as well as with varying levels of energy generation (Objective #3). Results indicate that agents learn to share energy by collaborating with each other and thereby reducing the energy requested from the supply grid (Objective #4). Additionally, as an indirect effect of energy sharing, houses were able to produce more energy

that consequently increased the flow of energy generated from renewable sources in the community.

We train the agents over 3 different scales i.e. with 3, 4, and 10 agents. Evaluation indicates that with increasing number of agents the performance of the solution slowly decreases but still is able to perform better than an no-energy sharing strategy and random-action-selection strategy in most cases (Objective #5). We were not able to entirely evaluate the scalability aspect of our solution with hundreds of agents due to issues related to the framework and hardware constraints. The average time taken for training 4 agents for 3 simulated days and 500 iterations was 40 hours and with 10 agents for the same setup increased upto 70 hours.

Chapter 7

Conclusion

In this chapter we summarize the thesis and highlight our main contributions. We then conclude with a discussion of open research issues and future work.

7.1 Thesis Contribution

This thesis addresses the issue of energy sharing in zero energy communities.

Chapter 1 motivated the work and outlined the issues of energy sharing in zero energy communities. We argued that the previous definition of nZEC was incomplete and therefore introduced a new definition that encompasses a mixture of buildings with varying levels of energy generation. We then discussed that an intelligent collaborative behaviour between buildings is required to enable such ZECs. We also discussed the complexity of energy sharing in such communities and the suitability of DRL to handle this.

In Chapter 2 we discussed the basic concepts of Smart Grids and MAS. We reviewed different types of MAS architectures and frameworks. We then discussed the RL using standard Q-table and approximate Q-learning. We also reviewed DRL and its application in modelling intelligent energy based solutions. We finally reviewed MARL based systems and enabling cooperation between independent learners in such systems.

In Chapter 3 we thoroughly reviewed literature in intelligent energy systems. We

argued that little or no direct work has been done on energy sharing in nZEC and discuss the complexity involved in doing so. We also reviewed cooperative and non-cooperative strategies for MAS in the energy domain.

In Chapter 4 we derived a set of requirements for intelligent energy sharing solution in zero energy communities. We argued that the solution needs to have autonomy; minimal decision making time i.e. it should operate in real-time; learn to adapt to environmental changes; and collaborate with other house agents. We then presented the design of two major components i.e. agent learning and a hybrid architecture that cater for these requirements. We also present the agent learning algorithm when discussing the agent learning process. Finally, we present the overall system architecture along with action choices each agent faces at each time-step.

In Chapter 5 we presented and discussed the implementation of the intelligent agent, Community Monitoring Service (CMS) and the dataset used. We list the JSON messages used by agents to communicate with each other and discuss the choice of hyper-parameters for the agent learning process. Lastly, we merge all of these components and discuss the Simulation using the Synchronizer process.

In Chapter 6 we evaluated the intelligent energy sharing solution in zero energy communities. We evaluated the adaptability and performance of our DRL based solution in a self-made simulation consisting of 2 seasons, 3 community configurations, and 3 different scales. In all evaluation scenarios agents were trained for 3 simulated days, for 500 episodes. Results show that agents learn to operate autonomously and adapt well to different seasons and configurations. Results also indicate that our solution performs much better than a no-energy-sharing strategy and a random-action-selection strategy in most cases. We evaluate our solution for scalability by comparing the results obtained with 3, 4 and 10 agents. It was observed that the difference between the nZEC status using the optimal strategy and our DRL based solution increases with increasing number of agents. This could be due to the increased non-stationarity introduced by the increasing number of agents and their actions. However, the achieved nZEC status is still much lower than

no-energy-sharing and random-action-selection strategy in most of the cases. We were not able to entirely evaluate the scalability aspect of our solution with hundreds of agents due to issues related to the framework and hardware constraints. We trained all the agents on a single machine as separate processes and the average time taken for training 4 agents was 40 hours, and for 10 agents was 70 hours.

7.2 Open Research Issues

While designing and evaluating our DRL based energy sharing solution we have made a few assumptions to simplify the design and have identified several areas for potential future research. Also, due to issues related to the framework and hardware constraints we were not able to fully evaluate the potential of our solution. We have outlined these issues and further evaluations that could be done to test the behaviour and performance of the system.

In our design, if an agent decides to request a neighbour for additional energy it randomly selects a neighbouring house. If its energy request is rejected then it selects another house at random and so on until it finds a house that accepts its energy request. If the list of all available houses is exhausted it finally requests the supply grid for additional energy. As an alternative, an agent can learn to select a particular neighbour or learn to rank houses based on whether they can suffice its request or not. This will help reduce the number of messages transmitted as well as the time taken for the energy transaction to complete. Additional parameters such as transmission efficiency and energy price too can used for ranking neighbouring houses. Optimizing further, instead of learning to select a neighbour from all the houses available in a community, agents could form learn to form sub-communities and transact energy within those sub-communities. This will enable a more stable agent-learning process and reduced non-stationarity due to the limit on the number of agents involved in the sub-community. This will also help scale better with hundreds of houses.

Our design allows an agent to perform only one successful energy transaction with neighbours in one timestep. This means that, if its energy request is not completely satisfied by one house it will request for the remaining energy to the supply grid. This affects the nZEC status negatively but constraints the number of transactions that can take place in one timestep. This helps with the stability of the grid by limiting the physical flows of electricity. It is possible to use advanced techniques that can optimize this to allow more transactions in a single timestep without destabilizing the grid.

Our design is simple and the focus is mainly on motivating agents to share energy without any energy pricing involved. However, in real-world systems, energy is distributed at a price and this can vary depending on the seller and time of the day. A possible future research could be to embed a pricing model in this solution along with global and local rewards.

We have trained our agents in an episodic manner and hence agents might experience the same state many times. In many real-world systems like stock-trading, agents are trained in a online fashion where they do not experience the same state twice as it is a time-series data. This is true for energy systems too that are affected by time. Learning the optimal behaviour in such online systems is much more difficult. A potential future research can be to modify the system to allow agent training in an online manner and observe its behaviour. Additionally, one can also embed an energy consumption forecasting and energy generation forecasting model to improve the accuracy of the system.

As our approach is based on DRL there are several hyper-parameters one can tune to achieve greater performance. Moreover, there are several optimization techniques that one can experiment with to achieve faster convergence. We assume that all agents implement the same agent learning algorithm whereas in real-world systems this might not be true. In the real-world, agents can have heterogeneous policies and convergence in such systems is difficult.

We have trained all our agents on a single Linux machine with 4 cores and 15 GB main memory. As the number of agents grow, the time taken for training also increases

drastically (upto 70 hours with 10 agents for 3 simulated days). Increasing the number of agents further resulted in the CPU running at max capacity and eventually led to crashes. As an alternative, with few modifications in the original source code one could train these agents in an distributed manner i.e. each agent on a separate machine. Additionally, one could use GPU to further accelerate the training process. This will allow better scalability testing with hundreds of houses.

Our energy sharing solution is able to converge to a policy that is comparable to the optimal policy and is able to provide the least performance boundaries of the system behaviour in most of the cases we have evaluated for. However, one cannot guarantee convergence and optimality using approximation techniques like DRL.

Bibliography

[1] Marcin Andrychowicz et al. "Hindsight experience replay". In: *Advances in Neural Information Processing Systems*. 2017, pp. 5048–5058.

[2] Anonymous. *EPRI Smart Grid Resource Center*. 2012. URL: http://smartgrid.epri.com/ (visited on 07/08/2018).

[3] Canadian Electricity Association. *The smart grid*. URL: https://electricity.ca/wp-content/uploads/2017/05/SmartGridpaperEN.pdf (visited on 06/27/2018).

[4] Shady Attia et al. "Assessing gaps and needs for integrating building performance optimization tools in net zero energy buildings design". In: *Energy and Buildings* 60 (2013), pp. 110–124.

[5] Robert Axelrod and William D Hamilton. "The Evolution of Cooperation The Evolution of Cooperation". In: *Evolution* 211.4489 (1981), pp. 1390–1396. ISSN: 00368075. DOI: 10.1086/383541. eprint: t8jd4qr3m (13960). URL: http://www.jstor.org.elib.tcd.ie/stable/pdf/1313150.pdf.

[6] Leemon Baird. "Residual algorithms: Reinforcement learning with function approximation". In: *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 30–37.

[7] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. "Developing multi-agent systems with a FIPA-compliant agent framework". In: *Software: Practice and Experience* 31.2 (2001), pp. 103–128.

[8] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. "JADE–A FIPA-compliant agent framework". In: *Proceedings of PAAM*. Vol. 99. 97-108. London. 1999, p. 33.

[9] Rafael H Bordini et al. "A survey of programming languages and platforms for multi-agent systems". In: *Informatica* 30.1 (2006).

[10] Léon Bottou. "Large-scale machine learning with stochastic gradient descent". In: *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[11] Justin A Boyan and Andrew W Moore. "Generalization in reinforcement learning: Safely approximating the value function". In: *Advances in neural information processing systems*. 1995, pp. 369–376.

[12] Lars Braubach, Alexander Pokahr, and Winfried Lamersdorf. "A universal criteria catalog for evaluation of heterogeneous agent development artifacts". In: *From Agent Theory to Agent Implementation (AT2AI-6)* (2008), pp. 19–28.

[13] Debaditya Chakraborty and Hazem Elzarka. "Advanced machine learning techniques for building performance simulation: a comparative analysis". In: *Journal of Building Performance Simulation* (2018), pp. 1–15.

[14] Caroline Claus and Craig Boutilier. "The dynamics of reinforcement learning in cooperative multiagent systems". In: *AAAI/IAAI* 1998 (1998), pp. 746–752.

[15] Li Deng, Dong Yu, et al. "Deep learning: methods and applications". In: *Foundations and Trends® in Signal Processing* 7.3–4 (2014), pp. 197–387.

[16] Aris Dimeas and Nikos Hatziargyriou. "A multi-agent system for microgrids". In: *Hellenic Conference on Artificial Intelligence*. Springer. 2004, pp. 447–455.

[17] Chun-xia Dou et al. "MAS-based solution to energy management strategy of distributed generation system". In: *International Journal of Electrical Power & Energy Systems* 69 (2015), pp. 354–366.

[18] Edmund H Durfee. "Practically coordinating". In: *AI Magazine* 20.1 (1999), p. 99.

[19] Ivana Dusparic et al. “Multi-agent residential demand response based on load forecasting”. In: *Technologies for Sustainability (SusTech), 2013 1st IEEE Conference on*. IEEE. 2013, pp. 90–96.

[20] Festo. *Energy-autonomous automation*. 2014. URL: https://www.festo.com/net/SupportPortal/Files/424514/Energieautarke_Automatisierung_en_M.pdf (visited on 07/15/2018).

[21] Jakob Foerster et al. “Learning with opponent-learning awareness”. In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2018, pp. 122–130.

[22] *Foundation for Intelligent Physical Agents. Specifications 1997*. URL: http://www.fipa.org (visited on 07/08/2018).

[23] Vincent Francois-Lavet et al. “Deep reinforcement learning solutions for energy microgrids management”. In: *European Workshop on Reinforcement Learning (EWRL 2016)*. 2016.

[24] Samira Garshasbi, Jarek Kurnitski, and Yousef Mohammadi. “A hybrid Genetic Algorithm and Monte Carlo simulation approach to predict hourly energy consumption and generation by a cluster of Net Zero Energy Buildings”. In: *Applied Energy* 179 (2016), pp. 626–637. ISSN: 03062619. DOI: 10.1016/j.apenergy.2016.07.033. URL: http://dx.doi.org/10.1016/j.apenergy.2016.07.033.

[25] Pierre Yves Glorennec. “Reinforcement learning: An overview”. In: *Proc. of ESIT*. Vol. 2000. Citeseer. 2000, pp. 17–35.

[26] Geoffrey J Gordon. “Reinforcement learning with function approximation converges to a region”. In: *Advances in neural information processing systems*. 2001, pp. 1040–1046.

[27] Geoffrey J Gordon. "Stable function approximation in dynamic programming". In: *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 261–268.

[28] Ritu Gupta and Gaurav Kansal. "A survey on comparative study of mobile agent platforms". In: *International Journal of Engineering Science and Technology* 3.3 (2011).

[29] Monika Hall and Achim Geissler. "Optimization of concurrency of PV-generation and energy demand for a heat pump–comparison of a monitored building and simulation data". In: *CISBAT 2015* (2015).

[30] Mohamed E El-Hawary. "The smart grid – state-of-the-art and future trends". In: *Electric Power Components and Systems* 42.3-4 (2014), pp. 239–250.

[31] Barbara Hayes-Roth, Lee Brownston, and Robert van Gent. "Multiagent Collaboration in Directed Improvisation." In: *ICMAS*. 1995, pp. 148–154.

[32] Dong He, Qingyu Xiong, and Xin Shi. "Modeling of the renewable energy system of an net zero energy community". In: *Control And Decision Conference (CCDC), 2017 29th Chinese*. IEEE. 2017, pp. 3590–3595.

[33] Robert Hecht-Nielsen. "Theory of the backpropagation neural network". In: *Neural networks for perception*. Elsevier, 1992, pp. 65–93.

[34] Junling Hu and Michael P Wellman. "Nash Q-learning for general-sum stochastic games". In: *Journal of machine learning research* 4.Nov (2003), pp. 1039–1069.

[35] Zhichuan Huang et al. "Minimizing electricity costs by sharing energy in sustainable microgrids". In: *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*. ACM. 2014, pp. 120–129.

[36] iMatix. *ZeroMQ Distributed Messaging*. 2017. URL: `http://zeromq.org/` (visited on 07/15/2018).

[37] Tommi Jaakkola, Michael I Jordan, and Satinder P Singh. "Convergence of stochastic iterative dynamic programming algorithms". In: *Advances in neural information processing systems*. 1994, pp. 703–710.

[38] Nick R Jennings. "Commitments and conventions: The foundation of coordination in multi-agent systems". In: *The knowledge engineering review* 8.3 (1993), pp. 223–250.

[39] Spiros Kapetanakis and Daniel Kudenko. "Reinforcement learning of coordination in cooperative multi-agent systems". In: *AAAI/IAAI* 2002 (2002), pp. 326–331.

[40] Genku Kayo, Ala Hasan, and Kai Siren. "Energy sharing and matching in different combinations of buildings, CHP capacities and operation strategy". In: *Energy and Buildings* 82 (2014), pp. 685–695.

[41] David Keil and Dina Goldin. "Indirect interaction in environments for multi-agent systems". In: *International Workshop on Environments for Multi-Agent Systems*. Springer. 2005, pp. 68–87.

[42] M Reyasudin Basir Khan, Razali Jidin, and Jagadeesh Pasupuleti. "Multi-agent based distributed control architecture for microgrid energy management and optimization". In: *Energy Conversion and Management* 112 (2016), pp. 288–307.

[43] M Venkata Kirthiga, S Arul Daniel, and S Gurunathan. "A methodology for transforming an existing distribution network into a sustainable autonomous microgrid". In: *IEEE Transactions on Sustainable Energy* 4.1 (2013), pp. 31–41.

[44] Jens Kober and Jan Peters. "Reinforcement learning in robotics: A survey". In: *Reinforcement Learning*. Springer, 2012, pp. 579–610.

[45] Sarit Kraus. "Negotiation and cooperation in multi-agent environments". In: *Artificial intelligence* 94.1-2 (1997), pp. 79–97.

[46] Kalliopi Kravari and Nick Bassiliades. "A survey of agent platforms". In: *Journal of Artificial Societies and Social Simulation* 18.1 (2015), p. 11.

[47] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[48] Geert Litjens et al. "A survey on deep learning in medical image analysis". In: *Medical image analysis* 42 (2017), pp. 60–88.

[49] Michael L Littman. "Friend-or-foe Q-learning in general-sum games". In: *ICML*. Vol. 1. 2001, pp. 322–328.

[50] Thillainathan Logenthiran, Dipti Srinivasan, and David Wong. "Multi-agent coordination for DER in MicroGrid". In: *Sustainable Energy Technologies, 2008. ICSET 2008. IEEE International Conference on*. IEEE. 2008, pp. 77–82.

[51] Rui Amaral Lopes et al. "A cooperative net zero energy community to improve load matching". In: *Renewable Energy* 93 (2016), pp. 1–13.

[52] Lu Lu et al. "HVAC system optimization in-building section". In: *Energy and Buildings* 37.1 (2005), pp. 11–22.

[53] Mahdi Mahfouf, Maysam F. Abbod, and Derek A. Linkens. "A survey of fuzzy logic monitoring and control utilisation in medicine". In: *Artificial intelligence in medicine* 21.1-3 (2001), pp. 27–42.

[54] Andrei Andreevich Markov. "The theory of algorithms". In: *Trudy Matematicheskogo Instituta Imeni VA Steklova* 42 (1954), pp. 3–375.

[55] Anna Joanna Marszal et al. "On-site or off-site renewable energy supply options? Life cycle cost analysis of a Net Zero Energy Building in Denmark". In: *Renewable energy* 44 (2012), pp. 154–165.

[56] Tambet Matiisen. *Demystifying Deep Reinforcement Learning*. 2018. URL: `https://ai.intel.com/demystifying-deep-reinforcement-learning/` (visited on 07/15/2018).

[57] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[58] Elena Mocanu et al. “On-line Building Energy Optimization using Deep Reinforcement Learning”. In: (2017), pp. 1–9. arXiv: 1707.05878. URL: http://arxiv.org/abs/1707.05878.

[59] Elena Mocanu et al. “On-line building energy optimization using deep reinforcement learning”. In: *IEEE Transactions on Smart Grid* (2018).

[60] Amir-Hamed Mohsenian-Rad and Alberto Leon-Garcia. “Optimal residential load control with price prediction in real-time electricity pricing environments”. In: *IEEE transactions on Smart Grid* 1.2 (2010), pp. 120–133.

[61] R Murray et al. “Consensus and cooperation in multi-agent networked systems”. In: *Proceedings of IEEE* (2007), pp. 215–233.

[62] Vishnuteja Nanduri and Tapas K Das. “A reinforcement learning model to assess market power under auction-based energy pricing”. In: *IEEE transactions on Power Systems* 22.1 (2007), pp. 85–95.

[63] John Nash. “Non-cooperative games”. In: *Annals of mathematics* (1951), pp. 286–295.

[64] Minh NH Nguyen et al. “Deep Reinforcement Learning based Smart Building Energy Management”. In: *Korea Software Conference Proceedings* (2017), pp. 871–873.

[65] NREL. *National Solar Radiation Database (NSRDB)*. 2018. URL: https://nsrdb.nrel.gov/nsrdb-viewer (visited on 07/15/2018).

[66] Daniel E Olivares, Claudio A Cañizares, Mehrdad Kazerani, et al. “A centralized optimal energy management system for microgrids”. In: *2011 IEEE Power and Energy Society General Meeting*. Citeseer. 2011, pp. 1–6.

[67] Daniel O’Neill et al. “Residential demand response using reinforcement learning”. In: *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*. IEEE. 2010, pp. 409–414.

[68] OpenSistemas. *OSBrain*. URL: https://osbrain.readthedocs.io/en/stable/about.html (visited on 07/08/2018).

[69] Eiichi Osawa. "A scheme for agent collaboration in open multiagent environments". In: *IJCAI*. Vol. 93. 1993, pp. 352–359.

[70] Liviu Panait and Sean Luke. "Cooperative multi-agent learning: The state of the art". In: *Autonomous agents and multi-agent systems* 11.3 (2005), pp. 387–434.

[71] Vassilis A Papavassiliou and Stuart Russell. "Convergence of reinforcement learning with general function approximators". In: *IJCAI*. 1999, pp. 748–757.

[72] Noah Pflugradt. *Load Profile Generator*. 2018. URL: https://www.loadprofilegenerator.de/ (visited on 06/27/2018).

[73] Pivotal. *Spring Boot v2.0.4*. 2018. URL: https://spring.io/projects/spring-boot (visited on 07/15/2018).

[74] Alexander Pokahr, Lars Braubach, and Winfried Lamersdorf. "Jadex: A BDI reasoning engine". In: *Multi-agent programming*. Springer, 2005, pp. 149–174.

[75] Ricardo J Rabelo and LM Camarinha-Matos. "Negotiation in multi-agent based dynamic scheduling". In: *Robotics and Computer-Integrated Manufacturing* 11.4 (1994), pp. 303–309.

[76] Sarvapali D Ramchurn, Dong Huynh, and Nicholas R Jennings. "Trust in multi-agent systems". In: *The Knowledge Engineering Review* 19.1 (2004), pp. 1–25.

[77] EPBD Recast. "Directive 2010/31/EU of the European Parliament and of the Council of 19 May 2010 on the energy performance of buildings (recast)". In: *Official Journal of the European Union* 18.06 (2010), p. 2010.

[78] Martin Riedmiller et al. "Karlsruhe brainstormers-a reinforcement learning approach to robotic soccer". In: *Robot Soccer World Cup*. Springer. 2000, pp. 367–372.

[79] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[80] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), p. 533.

[81] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson Education Limited, 2016.

[82] Christine Ruth. *Opening the Door to Autonomous Network Management*. 2016. URL: `https://www.siemens.com/innovation/en/home/pictures-of-the-future/digitalization-and-software/autonomous-systems-smart-grids.html` (visited on 07/15/2018).

[83] Walid Saad, Zhu Han, and H Vincent Poor. "Coalitional game theory for cooperative micro-grid distribution networks". In: *Communications Workshops (ICC), 2011 IEEE International Conference on*. IEEE. 2011, pp. 1–5.

[84] Igor Sartori et al. "Criteria for definition of net zero energy buildings". In: *the Proceedings of EuroSun* (2010).

[85] Pervez Hameed Shaikh et al. "A review on optimized control systems for building energy and comfort management of smart sustainable buildings". In: *Renewable and Sustainable Energy Reviews* 34 (2014), pp. 409–429.

[86] Neil Slater. *Why does DQN require two different networks?* 2018. URL: `https://ai.stackexchange.com/a/6983/11584` (visited on 07/07/2018).

[87] Jarosław Stańczak, Weronika Radziszewska, and Zbigniew Nahorski. "Dynamic pricing and balancing mechanism for a microgrid electricity market". In: *Intelligent Systems' 2014*. Springer, 2015, pp. 793–806.

[88] Peter Stone, Richard S Sutton, and Gregory Kuhlmann. "Reinforcement learning for robocup soccer keepaway". In: *Adaptive Behavior* 13.3 (2005), pp. 165–188.

[89] Huo-Ching Sun and Yann-Chang Huang. "Optimization of power scheduling for energy management in smart homes". In: *Procedia engineering* 38 (2012), pp. 1822–1827.

[90] Richard S Sutton. "Learning to predict by the methods of temporal differences". In: *Machine learning* 3.1 (1988), pp. 9–44.

[91] RS Sutton and AG Barto. "Introduction to Reinforcement Learning. MIT Press". In: *Cambridge, MA* (1998).

[92] G Swaminathan et al. "Investigations of microgrid stability and optimum power sharing using robust control of grid tie pv inverter". In: *Advances in Smart Grid and Renewable Energy*. Springer, 2018, pp. 379–387.

[93] Ming Tan. "Multi-agent reinforcement learning: Independent vs. cooperative agents". In: *Proceedings of the tenth international conference on machine learning*. 1993, pp. 330–337.

[94] Zhiyong Tan, Chai Quek, and Philip YK Cheng. "Stock trading with cycles: A financial application of ANFIS and reinforcement learning". In: *Expert Systems with Applications* 38.5 (2011), pp. 4741–4755.

[95] Samah El-Tantawy, Baher Abdulhai, and Hossam Abdelgawad. "Multiagent reinforcement learning for integrated network of adaptive traffic signal controllers (MARLIN-ATSC): methodology and large-scale application on downtown Toronto". In: *IEEE Transactions on Intelligent Transportation Systems* 14.3 (2013), pp. 1140–1150.

[96] Gerald Tesauro. "Neurogammon wins computer olympiad". In: *Neural Computation* 1.3 (1989), pp. 321–323.

[97] Gerald Tesauro. "Temporal difference learning and TD-Gammon". In: *Communications of the ACM* 38.3 (1995), pp. 58–68.

[98] John N Tsitsiklis. "Asynchronous stochastic approximation and Q-learning". In: *Machine learning* 16.3 (1994), pp. 185–202.

[99] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-Learning." In: *AAAI*. Vol. 2. Phoenix, AZ. 2016, p. 5.

[100] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[101] Christopher John Cornish Hellaby Watkins. "Learning from delayed rewards". PhD thesis. King's College, Cambridge, 1989.

[102] Tianshu Wei, Yanzhi Wang, and Qi Zhu. "Deep reinforcement learning for building hvac control". In: *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM. 2017, p. 22.

[103] Michael Weinberg and Jeffrey S Rosenschein. "Best-response multiagent learning in non-stationary environments". In: *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*. IEEE Computer Society. 2004, pp. 506–513.

[104] Gerhard Weiss. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press, 1999.

[105] Marco A Wiering. "Convergence and divergence in standard and averaging reinforcement learning". In: *European Conference on Machine Learning*. Springer. 2004, pp. 477–488.

[106] Uri Wilensky. *NetLogo*. URL: `https://ccl.northwestern.edu/netlogo/` (visited on 07/08/2018).

[107] Jun Wu et al. "A novel multi-agent reinforcement learning approach for job scheduling in Grid computing". In: *Future Generation Computer Systems* 27.5 (2011), pp. 430–439.

[108] Shangtong Zhang and Richard S Sutton. "A Deeper Look at Experience Replay". In: *arXiv preprint arXiv:1712.01275* (2017).

[109] Tianyang Zhao et al. "Game theory based distributed energy trading for microgrids parks". In: *Energy, Power and Transportation Electrification (ACEPT), 2017 Asian Conference on*. IEEE. 2017, pp. 1–7.

[110] Weigang Zhong et al. "IDES: Incentive-driven distributed energy sharing in sustainable microgrids". In: *2014 International Green Computing Conference, IGCC 2014* (2015). DOI: 10.1109/IGCC.2014.7039166.

[111] Ting Zhu et al. "Sharing renewable energy in smart microgrids". In: *Cyber-Physical Systems (ICCPS), 2013 ACM/IEEE International Conference on*. IEEE. 2013, pp. 219–228.

[112] LIU Zifa et al. "Distributed reinforcement learning to coordinate current sharing and voltage restoration for islanded DC microgrid". In: *Journal of Modern Power Systems and Clean Energy* 6.2 (2018), pp. 364–374.

[113] Martin Zinkevich et al. "Parallelized stochastic gradient descent". In: *Advances in neural information processing systems*. 2010, pp. 2595–2603.