

**MICROGRID ENERGY MANAGEMENT USING
DEEP Q-NETWORK REINFORCEMENT LEARNING**

BY

Mohammed Alabdullah

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

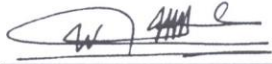
ELECTRICAL ENGINEERING

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN- 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by Mohammed Alabdullah under the direction of his thesis advisor and approved by his thesis committee, has been presented and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN ELECTRICAL ENGINEERING**.



Dr. Abdallah Al-Ahmari
Department Chairman



Dr. Suliman Al-Homidan
Dean of Graduate Studies

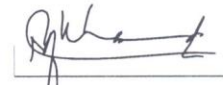
Date



Dr. Mohamed Abido
(Advisor)



Dr. Fahad Al-Ismail
(Member)



Dr. Muhammad Khalid
(Member)

© Mohammed Alabdullah

2021

[Dedication]

To my family

ACKNOWLEDGMENTS

First of all, I would like to thank my advisor Dr. Mohamed Abido for his continues support during the past two years. Without his guidance, this work would not have been possible. Second, I would like to extend my thanks and appreciations to the MSc committee, mainly Dr. Fahad Al-Ismail, and Dr. Muhammad Khalid, for their efforts in reviewing this work. Finally, I want to express my gratitude to my family members for providing me with unlimited support and continuous encouragement throughout my years of study.

TABLE OF CONTENTS

ACKNOWLEDGMENTS	V
TABLE OF CONTENTS.....	VI
LIST OF TABLES	IX
LIST OF FIGURES	X
LIST OF ABBREVIATIONS.....	XIII
NOMENCLATURE	XIV
ABSTRACT.....	XVII
ملخص الرسالة.....	XIX
1 CHAPTER 1 INTRODUCTION	1
1.1 Overview.....	1
1.2 Motivation & Objectives.....	3
1.3 Thesis Contributions	4
1.4 Thesis Organization	5
2 CHAPTER 2 LITERATURE REVIEW	7
2.1 Microgrid Energy Management Overview.....	7
2.2 Model-Based Vs. Model-Free Methodologies	10
2.3 Deep Reinforcement Learning	13
2.4 Summary	15
3 CHAPTER 3 PROPOSED METHODOLOGY	17
3.1 Machine Learning Methods and Deep Reinforcement Learning	17

3.1.1	Deep Learning.....	18
3.1.2	Reinforcement Learning	21
3.1.3	Deep Reinforcement Learning	24
3.2	Microgrid Environment Formulation.....	28
3.3	Energy Management Problem Formulation.....	31
3.4	Reward Function.....	32
4	CHAPTER 4 IMPLEMENTATION.....	34
4.1	Implementation Process	34
4.2	Pandapower.....	35
4.2.1	Electric Element Models.....	36
4.2.2	Power Flow Solver.....	38
4.2.3	Case Study	38
4.3	Deep Reinforcement Learning Toolkits.....	42
4.3.1	OpenAI Gym Library.....	42
4.3.2	Baselines	43
4.3.3	Case Study	43
4.4	Energy Management of a Microgrid.....	47
4.4.1	Energy Management of a Simplified Microgrid	47
4.4.2	Energy Management of a Complex Microgrid.....	53
5	CHAPTER 5 RESULTS & DISCUSSION	56
5.1	The Microgrid Environment	56
5.2	Computation Time	59
5.3	Hyperparameter Tuning.....	59
5.4	Energy Schedules.....	64
6	CHAPTER 6 CONCLUSION & FUTURE WORK.....	68

6.1	Conclusion	68
6.2	Recommended Future Work	69
7	APPENDIX.....	71
7.1	Case Study MG Network Parameters.....	71
7.2	Codes.....	73
7.2.1	Microgrid Environment.....	73
7.2.2	Agent Development.....	83
	REFERENCES	94
	VITAE.....	102

LIST OF TABLES

Table 4.1 Observation parameters	44
Table 4.2 Action space of agent.....	44
Table 4.3 Annual operating costs of DQN, and optimal values for different SOC	50
Table 4.4 Annual operating costs of DQN, and optimal values for different capacities ..	52
Table 7.1 Microgrid Line Parameters	71
Table 7.2 Transformers Information.....	71
Table 7.3 Load nominal values	71
Table 7.4 Switches Information.....	72
Table 7.5 DER information.....	72
Table 7.6 Energy Storage Devices Information.....	72

LIST OF FIGURES

Figure 2.1 Global solar manufacturing capacity vs. module price (2010-2020) [7].....	8
Figure 3.1 NN with one hidden layer.....	19
Figure 3.2 Convolution Neural Network (CNN) to classify handwritten digits [61]	20
Figure 3.3 Recurrent neural network structure [53].....	21
Figure 3.4 Reinforcement learning process	21
Figure 3.5 MDP illustration at each time step [53].....	22
Figure 3.6 DQN algorithm flowchart [53].....	25
Figure 3.7 Linear annealed exploration	28
Figure 3.8: Microgrid environment configuration	29
Figure 3.9 Architecture of DQN for microgrid energy management	32
Figure 4.1 DRL implementation process flow.....	35
Figure 4.2 Benchmark Medium Voltage Microgrid Configuration.....	39
Figure 4.3 Load profile in 24-hour period	40
Figure 4.4 DER generation profile in 24-hour period.....	40
Figure 4.5 Bus Voltage Magnitude	41
Figure 4.6: Line loading profile	41
Figure 4.7 DRL main components.....	42
Figure 4.8 Illustration of cartpole	44
Figure 4.9 MLP used in cartpole problem	45
Figure 4.10 Learning performance of DQN with different MLP structures.....	46
Figure 4.11 Schematic of microgrid	47
Figure 4.12 Load profile sample.....	48

Figure 4.13 PV power generation profile sample	48
Figure 4.14 Electricity prices sample.....	49
Figure 4.15 Illustration of NN used to map states to state-action value functions	50
Figure 4.16 SOE optimal solution with respect to grid electric prices	51
Figure 4.17 Optimal SOE vs SOE obtained from DQN agent	51
Figure 4.18 learning curve for of DQN agent given 24 hours of future state values.....	52
Figure 4.19 Learning curves for different NN structures	53
Figure 4.20 MV Microgrid	54
Figure 4.21 Framework to solve microgrid energy management	55
Figure 5.1 CIGRE MV Microgrid with DER	57
Figure 5.2 DQN training process with microgrid environment.....	58
Figure 5.3 Learning curve of NN with 2 hidden layers	60
Figure 5.4 Learning curve of NN with 3 hidden layers	60
Figure 5.5 Learning rate effect on training performance	61
Figure 5.6 Discount factor effect on training performance.....	62
Figure 5.7 Learning curve for different batch sizes	62
Figure 5.8 Learning curve for different fractions of exploration.....	63
Figure 5.9 Learning performance for different target network update rates.....	64
Figure 5.10 Energy storage hourly schedule with SOC.....	65
Figure 5.11 DG and energy storage hourly schedules with grid power exchange	65
Figure 5.12 Benchmarking DQN agent daily operating costs with optimal solution.....	66
Figure 5.13 Segment of the daily operating costs	66
Figure 5.14 Daily costs difference between optimal solution and DQN solution	67

Figure 5.15 Accumulated operating costs for the different solvers	67
---	----

LIST OF ABBREVIATIONS

NN	:	Neural Networks
DQN	:	Deep Q-Networks
DRL	:	Deep Reinforcement Learning
MDP	:	Markov Decision Process
PV	:	Photovoltaic
DER	:	Distribute Energy Resource
RL	:	Reinforcement learning
MPC	:	Model Predictive Control
EMS	:	Energy Management System
ESS	:	Energy Storage Systems

NOMENCLATURE

$P_{min}^{DG_d}$	:	The minimum active power output of generator d
$P_t^{DG_d}$	:	The active power output of generator d, at time t
$P_{max}^{DG_d}$	:	The maximum active power output of generator d
$S_{max}^{DG_d}$	:	The maximum apparent power output of generator d (kVA)
$Q_t^{DG_d}$	:	The reactive power output of generator d, at time t
$C_t^{DG_d}$	:	The cost of operating conventional generator at time t
Δt	:	Time period
P_t^E	:	Charging or discharging power of energy storage device at time t
P_{max}^E	:	The maximum charging/discharging power of energy storage device
E_t	:	Energy level of energy storage device or state of charge
E_{min}	:	Minimum energy level of energy storage device
E_{max}	:	Maximum energy level of energy storage device
η_{ch}	:	Charging efficiency of energy storage device
η_{dis}	:	Discharging efficiency of energy storage device

P_t^U	:	Active Power exchange of microgrid with grid at time t
P_{max}^U	:	Maximum active power exchange of microgrid with grid at time t
Q_t^U	:	Reactive power exchange of microgrid with grid at time t
S_{max}^U	:	Maximum complex power exchange of microgrid with grid at time t
C_t^U	:	Cost of purchase of electricity from grid at time t
R_t	:	Price of electricity from grid at time t
P_t^{PV}	:	Power output from solar PV at time t
P_t^W	:	Power output from wind at time t
θ	:	Neural Network parameters
α	:	Learning rate
∇	:	Gradient
$\mathcal{S}$	:	State Space
R	:	Reward function
γ	:	Discount factor
π	:	RL policy

$Q^\pi(s, a)$: Q value function that follows policy π , or state-action
value function

ABSTRACT

Full Name : Mohammed Alabdullah
Thesis Title : Microgrid Energy Management Using Deep Q-Networks
Reinforcement Learning
Major Field : Master of Science in Electrical Engineering
Date of Degree : April 2021

Microgrid energy management is of a high importance to achieve economical operation of the different distributed micro resources within a microgrid. Existing model-based approaches to solve the energy management problem depend on experts to construct the microgrid dynamics and parameters, which will incur costs with development and maintenance. Adding resources or loads will alter the architecture of the microgrid, which will change the probability distribution of the uncertain variables. This necessitates changing the predictor model, and possibly the solver to achieve the required scheduling performance.

Alternatively, model-free methods based on reinforcement learning consider the energy scheduling problem with no prior model of the microgrid, nor a predictor to estimate the uncertainty. Such methods depend on interactions with the system in order to find an optimal strategy to achieve the operation goals. Though, one main problem with such methods is that they deal with a high-dimensional state space input, that makes it difficult to extract valuable features to optimize value functions. Motivated by the recent advances of deep learning, deep q-network reinforcement learning is adopted as it can tackle high dimensional state spaces.

The thesis investigates the use of deep reinforcement learning to optimally manage the different energy resources in a microgrid. The methodology considers the stochastic behavior of different elements of a microgrid, including loads, generations, and electric prices. It also models different grid elements using equivalent models and considers the various power flow constraints in a realistic setting. The energy management problem is formulated as a finite horizon Markov Decision Process (MDP) by defining the state, action, reward, and objective functions, with no knowledge of the transition probabilities. Such formulation does not require explicit model of the microgrid, making use of the accumulated data and interaction with the system to derive the optimal policy. The microgrid environment is developed using OpenAI Gym open-source platform, to allow the experimentation of formulated model with different techniques. Deep reinforcement learning algorithms based on deep Q-networks is implemented to solve the developed formulation.

A case study based on a real microgrid is implemented to show the performance of the proposed methodology. The case study is based on a medium voltage microgrid from CIGRE where real historical generation and load profiles are considered. The proposed methodology demonstrated its capability to obtain comparable results to optimal values. Hence, using the proposed methodology can achieve online scheduling of the various energy resources within a microgrid with cost effective actions under stochastic conditions.

ملخص الرسالة

الاسم الكامل: محمد العبدالله

عنوان الرسالة: إدارة الطاقة في الشبكات الكهربائية الدقيقة باستخدام التعلم المدعم العميق المبنية بشبكات Q العميقة

التخصص: ماجستير العلوم في الهندسة الكهربائية

تاريخ الدرجة العلمية: أبريل 2021

تعد إدارة موارد الطاقة في الشبكات الكهربائية الدقيقة (Microgrids) ذات أهمية عالية لتحقيق التشغيل الاقتصادي الأمثل. تعتمد الأساليب الحالية لحل مشكلة إدارة الطاقة على نماذج مطورة من قبل خبراء تتكبد تكاليف التطوير والصيانة. حيث تؤدي إضافة الموارد أو الأحمال إلى تغيير بنية الشبكة المصغرة، مما يستلزم تغيير نموذج التوقع، بالإضافة إلى تغيير طريقة الحل للوصول للحل الأمثل لتحقيق أداء الجدولة المطلوب.

بدلاً من ذلك، تراعي الطرق البديلة التي لا تعتمد على النماذج مثل التعلم المعزز لحل مشكلة جدولة الطاقة بدون الاعتماد على وجود نموذج سابق للشبكة، ولا متنبئ لتقدير المتغيرات. تعتمد هذه الأساليب على التفاعل مع النظام ومحاولة إيجاد استراتيجية مثالية لتحقيق التشغيل الأفضل. ولكن إحدى المشكلات الرئيسية في مثل هذه الأساليب هي أنها تتعامل مع فضاء كثير الأبعاد، مما يجعل من الصعب استخراج ميزات قيمة لتحسين دالة القيمة. لهذا، ومع التطورات الحديثة في التعلم العميق، تم اعتماد التعلم المعزز العميق لأنه يمكن أن يتعامل مع الفضاءات عالية الأبعاد.

تبحث الأطروحة في استخدام التعلم المعزز العميق (DRL) لإدارة موارد الطاقة المختلفة على النحو الأمثل في الشبكات الكهربائية. تأخذ المنهجية في الاعتبار السلوك المتغير للعناصر المختلفة في الشبكات، بما في ذلك الأحمال ومصادر الطاقة وأسعار الكهرباء. كما تقوم أيضاً بنمذجة عناصر الشبكة المختلفة باستخدام نماذج مكافئة وتأخذ في الاعتبار قيود تدفق الطاقة المختلفة في بيئة واقعية. تمت صياغة مشكلة إدارة الطاقة على أنها عملية قرار ماركوف محدودة الأفق (MDP) من خلال تحديد الحالة، و القرار، و صياغة المكافأة، مع عدم وجود معرفة مسبقة باحتمالات الانتقال. لا تتطلب هذه الصيغة نموذجاً للمتغيرات في الشبكة الكهربائية، بل تعتمد على الاستفادة من البيانات المتراكمة والتفاعل مع النظام لاشتقاق السياسة المثلى. تم تطوير بيئة متكاملة للشبكة الكهربائية باستخدام OpenAI GYM ،

للسماح بتجربة النموذج المصاغ بتقنيات مختلفة. يتم تنفيذ خوارزميات تعلم التعزيز العميق القائمة على شبكات Q العميقة لحل الصيغة المطورة.

لإثبات فعالية الخوارزمية المقترحة، تقدم الأطروحة دراسة حالة تعتمد على شبكة كهربائية حقيقية لإظهار أداء المنهجية المقترحة. تعتمد دراسة الحالة على شبكة متوسطة الجهد من جمعية CIGRE حيث يتم النظر في البيانات التاريخية الحقيقية للأحمال ومصادر الطاقة. أثبتت المنهجية المقترحة قدرتها على الحصول على نتائج مقارنة للقيم المثلى. ختاماً، استخدام المنهجية المطروحة يمكن أن تحقق جدولة بشكل مباشر لموارد الطاقة المختلفة داخل الشبكات الكهربائية مع إجراءات فعالة من حيث التكلفة في ظل الظروف المتغيرة. |

CHAPTER 1

INTRODUCTION

1.1 Overview

The recent advances in renewable energy resources such as solar PV as well as energy storage technologies have changed the way the electrical grid is operated. Traditionally, power is generated centrally and delivered to the end customers through transmission and distribution networks. However, the steady decline of distributed generation prices, especially PV and wind, has paved the way for a new trend opposing the centralization of power generation and delivery.

The concept of microgrid has emerged to encompass the clustering of distributed generation resources of all types with different loads in small electrical grids. Such networks can operate in isolation of the main grid, or in grid-connected fashion thus enhancing the overall reliability of the system. The adoption of microgrid promises many benefits including sustainability, security, costs, and many others.

However, one major challenge with integrating microgrid energy resources is planning a stable and economical operation. This challenge is mainly driven by the intermittent and uncertain nature of renewable resources along with varying load profiles. Such stochastic nature of the micro-sources makes it difficult to accurately devise optimal operation schedules.

Using energy storage systems helps mitigating the uncertainty issues, though, energy management control systems are needed to efficiently operate such storage devices to achieve an optimization goal. Conventionally, the energy management problem is tackled using a model-based framework. Such methods rely on modelling the technical dynamics of the microgrid, then estimating the stochastic behavior of the uncertain variables using a predictor, and finally, solving the formulated optimization problem.

Model based approaches have been extensively studied in the literature. However, these methods depend on experts to construct the microgrid dynamics and parameters, which will incur costs with the development and maintenance. Adding resources, or loads will alter the architecture of the microgrid, which will change the probability distribution of the uncertain variables. This necessitates changing predictor model, and possibly the solver to achieve required scheduling performance.

Alternatively, learning based approaches that are model-free such as reinforcement learning consider the energy scheduling problem with no prior model of the microgrid, nor a predictor to estimate the uncertainty. They depend on interactions with the system and try to find an optimal strategy to achieve the operation goals. Though, one main problem with such methods is that they deal with a high-dimensional state space input, that makes it difficult to extract valuable features to optimize value functions.

To resolve this problem, deep reinforcement learning (DRL) is proposed as it makes use of deep neural network to learn from high dimensional state space. DRL techniques have been proposed in the artificial intelligence literature in recent years. Such methods proved their effectiveness to solve complex decision-making problems with high dimensional state

inputs. They achieved great success in complex games such as Go, Atari, Chess, and others, competing with professional players, and winning over them. Such great feat was made possible using reinforcement learning techniques with deep neural network to design an artificial agent that can devise optimal policies from high dimensional inputs.

1.2 Motivation & Objectives

The classical techniques that solve the microgrid energy management problem make use of a model-based framework to formulate the dynamics of the microgrid. Then, the uncertainties are estimated using a predictor, and the optimal schedule is obtained using an optimization solver. Such methods have been studied comprehensively and are widely known in different academic fields for their robustness.

Although such methods proved their effectiveness in the energy management domain, they highly depend on an expert to accurately model the dynamics of the microgrid. The scale, capacity, or dynamics of the microgrid could change with time, resulting in a change in the uncertainty profiles of generation or demand. Such result could lead to increasing the complexity and cost associated with implementing energy management systems for different microgrids.

Motivated by the recent achievements done by deep reinforcement learning methodologies, deep q-network reinforcement learning is implemented to solve the microgrid energy management problem. Existing similar work apply deep reinforcement learning approach on simplified networks, ignoring some critical constraints in the microgrid. The intention of our work is to extend the formulation of the problem to represent a realistic microgrid

system, accounting for the microgrid constraints and dynamic behavior. Mainly, the research work aims are:

- To develop the framework of microgrid dynamic behavior and constraints in the context of a Markov Decision Process, and reinforcement learning
- To develop state of the art deep reinforcement learning algorithm based on DQN for solving microgrid energy management problem
- To analyze the effect of using different neural network architectures with different hyperparameters on the performance of the algorithm

1.3 Thesis Contributions

The proposed thesis work attempts to close some of the gaps found in the current literature. The work investigates the use of deep reinforcement learning to optimally manage the different energy resources in a microgrid. The methodology considers the stochastic behavior of different elements of a microgrid, including loads, generations, and electric prices. It also models different grid elements using equivalent models and considers the various power flow constraints in a realistic setting. The energy management problem is formulated as a finite horizon Markov Decision Process (MDP) by defining the state, action, reward, and objective functions, with no knowledge of the transition probabilities. Such formulation does not require explicit model of the microgrid, making use of the accumulated data and interaction with the system to derive the optimal policy. The microgrid environment is developed using OpenAI GYM framework, to allow the experimentation of formulated model with different techniques. Deep reinforcement

learning algorithms based on deep Q-networks is implemented to solve the developed formulation.

The thesis main contributions can be summarized as follows:

- Novel implementation of deep q-network reinforcement learning framework to the microgrid energy management problem
- Designing the approach to be data driven, which considers the historical profiles of different elements of a microgrid (e.g. loads, generations, and electric prices)
- Consideration of microgrid different dynamics and power flow constraints
- Development of a flexible framework of microgrid environment that allows changing the microgrid configurations
- Thorough analysis of neural network hyperparameters on the performance of algorithm
- Benchmarking the methodology performance in a case study based on MV microgrid with real historical data

1.4 Thesis Organization

The thesis is organized as follows: Chapter 2 provides an overall literature survey that covers different energy management approaches. It specifically considers the energy management approach from a model-free perspective. It also provides a brief survey of machine learning methodologies based on reinforcement learning, which will be used to solve the MDP formulation of energy management problem.

Chapter 3 provides details on the proposed methodology and formulation of the microgrid energy management problem. The different aspects of deep reinforcement learning are explained, and further enhancements are presented. Additionally, the chapter provides the steps taken to develop the microgrid environment which consider the microgrid dynamic behavior and physical constraints. Moreover, the formulation of state space and action space for a given microgrid is elaborated. The chapter is concluded with a brief explanation of the reward function shaping used for the energy management problem.

Chapter 4 presents the different libraries used to implement proposed methodology based on deep reinforcement learning to conduct energy management in a given microgrid. Namely, Pandapower, OpenAI Gym, and Stable Baselines libraries are presented and discussed. Results of implementation are discussed in Chapter 5, in which different experiments are conducted to show the performance of the methodology and effect of varying different hyperparameters. In chapter 6, a conclusion to the thesis and recommended future work are provided.

CHAPTER 2

LITERATURE REVIEW

In this chapter, a literature review of relevant work to the topic of microgrid energy management and deep reinforcement learning is provided. Moreover, comparisons between model-free to model-based approaches to manage energy resources in a given microgrid are presented. Further emphasis is given to model free methodologies, in which more details are provided on machine learning approaches to solve the problem. Finally, a brief survey on deep reinforcement learning and recent advances in the field is provided.

2.1 Microgrid Energy Management Overview

A microgrid is defined as a group of loads and micro-sources operating under the control of one system [1], [2]. The microgrid could operate in parallel with the utility grid to optimally consume local power generation sources, or in islanded mode in case of failure in the main grid, thus enhancing overall reliability of local service. Many benefits are realized by adopting microgrids, including the reduction of greenhouse gases emission, improving voltage profiles, decentralization of power supply, reducing line losses [3]. It also allows customers to actively participate in the microgrid operation [4].

The decrease of renewable generation costs has driven the adoption of microgrid schemes. For example, the cost of manufacturing Solar PV has seen noticeable reduction over the past years, which was accompanied by huge increase in installation. According to [5], the global annual installation of solar PV grew more than six times within one decade, from

16 GW in 2010 to 105 GW in 2019. Figure 2.1 shows the solar manufacturing capacity globally versus the solar module price. Such trends are not only restricted to solar technologies. Similar trends are observed in other types of renewables, such as wind technologies which achieved a levelized cost of energy of less than 0.1 \$/kWh in 2019 [6].

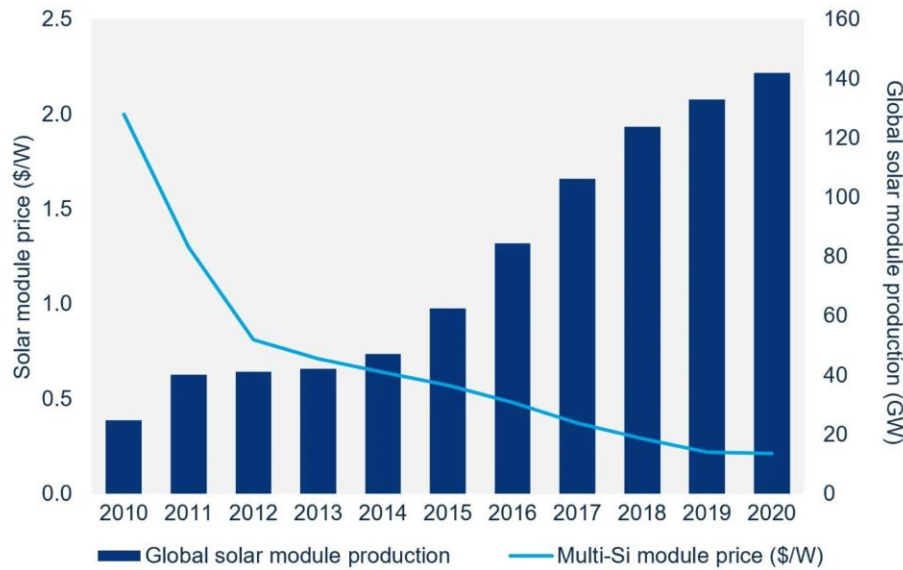


Figure 2.1 Global solar manufacturing capacity vs. module price (2010-2020) [7]

Nonetheless, challenges are faced when integrating distributed generation resources due to the uncertainties coming from the intermittent renewable generation, and varying load profiles. This makes it difficult to optimally schedule generation as in a conventional grid. Unforeseen power variations would necessitate commitment of expensive reserve or ancillary services, making microgrid operation uneconomical.

Many studies have been conducted to overcome such challenge. For instance, [8], [9] present a two-stage stochastic programming methodology to optimize the operation costs. Also, [10] considers different cost variations and adopted a risk-averse stochastic programming method. However, the aforementioned papers require information of the uncertainty statistical distribution beforehand. To work around the unavailability of

statistical distributions, [11], [12] use robust optimization to derive an optimal solution from possible worst-case conditions. Nonetheless, these studies derive fixed schedules that does not respond to unplanned variations in real time.

The use of energy storage systems (ESS) can mitigate the issues of matching generation and demand variations. ESS allow the system operator to have more flexibility over the microgrid resource, and to shift the intermittent renewable generation to peak hours, thus earning from energy arbitrage [13]. Moreover, the costs associated with energy storage devices have declined over the years making them more affordable. According to [14], the average kWh cost of utility-scale battery storage in the US has decreased 70% from 2,152 \$/kWh in 2015 to 625 \$/kWh in 2018. Many other benefits can be realized by having ESS, which includes providing ancillary services such as load following, voltage support, frequency regulation [15]. Also, the use of energy storage systems helps increasing the reliability of power to customers during disturbance from supply side [16].

Though, ESS management is complex since it is not operated as in conventional generation by making economic dispatch or unit commitment. Efficient operation of ESS requires an energy management system (EMS) that optimally maximizes the operation benefits of distributed generation with energy storage and considers long-term time varying prices, demands, and renewable generations [17].

Optimizing the microgrid energy management can take different directions depending on user preferences, geographical area, regulation, tariff types, and several other possible considerations. The optimization function could consider environmental costs, capital and operational costs, energy storage costs, or any other type of penalties. For example, [18]

considers minimization of imported energy from the grid, [19] optimizes generation costs, and [20] considers grid costs. Additionally, multi-objective optimization is considered by several papers. For example, [21] considers power losses, carbon emission, and capital costs. Reference [22] minimizes generation and storage costs. In addition, paper [23] considers minimization of fuel consumption and the maximization of energy storage charge/discharge efficiency. Whereas reference [24] considers optimization of fuel cost, carbon emission, and energy loss.

In terms of constraints, several papers considered different elements of the microgrid. For example, [25] considers maximum and minimum limits of power generation from distributed generators. The paper also considered limits on the demand side. Additionally, [26] considers operation constraints in the microgrid as well as limits in renewable generation. Other papers looked into energy storage constraints such as charging and discharging rates, limits on electricity prices, or carbon emissions [27].

2.2 Model-Based Vs. Model-Free Methodologies

The traditional way of energy management in a microgrid makes use of a model-based framework to formulate the dynamics of the microgrid. Then, the uncertainties are estimated using a predictor, and the optimal schedule is obtained using an optimization problem solver [28], [29]. Examples include the use of rolling horizon method [30], in which for each decision step, a mixed integer optimization formulation is solved. Additionally, [31] develops a convex Model Predictive Control (MPC) methodology for dynamic optimal power flow in multiple battery storage systems in a microgrid.

Although such methods proved their effectiveness in the energy management domain, they highly depend on an expert to accurately model the dynamics of the microgrid. The scale, capacity, or dynamics of the microgrid could change with time, resulting in a change in the uncertainty profiles of generation or demand [32]. Additionally, solving a large scale microgrid energy problem using such methods on real time is difficult. Moreover, building a generalized framework using these methods for different environments is challenging [33]. This leads to increasing the complexity and cost associated with implementing EMS for different microgrids, which could hinder the adoption of such grid schemes.

To work around this problem, learning based methodologies have been proposed in the literature to resolve microgrid management difficulties. Such methods do not require explicit model of the microgrid, making use of the accumulated data and interaction with the system to derive the optimal control policy [34]. For example, [35] presents an energy management system using batch reinforcement learning. Additionally, [36] develops a solution to microgrid energy management using evolutionary adaptive dynamic programming and reinforcement learning. Also, [37] uses reinforcement learning framework to design a multiagent system that aims to optimize the microgrid energy management. Although, aforementioned work present novel ways to handle the energy management problem, it struggles with high dimensional state variables due the curse of dimensionality.

To work around the problem of state variable dimensionality, deep reinforcement learning (DRL) has been introduced in the literature in recent years [38]. It makes use of deep neural network to extract important features of the high dimensional state as the agent is exposed to experiences from the interaction with the environment, with no prior knowledge of the

system. The agent does not need a separate model to obtain a forecast or a probability distribution to the changing variables. Rather, the agent makes full use of the available set of collected data to learn the optimal policy that achieves optimal results. The deep reinforcement learning methodologies can be implemented to achieve autonomous real time control and decision making in different power system applications. For example, [39] applied DRL in strategic bidding and equilibrium analysis in electricity markets. Also, [40] adopted DRL to power management in distribution systems with bi-level power management of cooperatives that has several networked microgrids. Reference [41] applied DRL to optimize the demand side peak to average ratio in multi microgrids. Moreover, reference [42] developed an adaptive emergency control scheme based on DRL. Other applications include voltage control [43], control for multi energy systems [32] and others.

In terms of energy management applications, several works have been reported in the literature that makes use of DRL. For example, [44] formulates the microgrid management as a sequential decision-making problem under uncertainty. A convolutional NN is used to extract important features and output Q-values that are used by the reinforcement policy. The study proved effective in a case study, though, it only considered a simple network with no constraints such as power flows, or real time electricity prices. Additionally, [17] devises a method for real time economical operation of a microgrid using approximate dynamic programming. It considers system constraints and uncertainties in the load and generation as well as variable pricing of electricity. A deep recurrent neural network is used to predict the changing variables, which in turn used to obtain the value function approximation. However, the work is formulated to have a predictor to solve the formulation, which opposes the model-free paradigm. Reference [34] formulates the deep

Q-networks to manage energy resources in a low voltage network. Though, the work considers a simple objective of minimizing costs due grid exchanges and conventional generation. It also ignores power flow constraints in the network, which will likely result in violating network constraints.

2.3 Deep Reinforcement Learning

The field of Machine Learning has gone through great improvements in recent years, mainly due to advances in deep learning. Such advances can be linked to several factors including the increasing computation power using Graphics Processing Units and distributed computing, methodological breakthroughs in deep learning, and development of software ecosystems [45]. High performance have been achieved using state of art deep learning architectures to conduct classifications [46], pattern recognition [47]- [48], and generative tasks [49].

Such achievement also led to progressions in the domain of decision making, under reinforcement learning framework. Reinforcement Learning (RL) is one subcategory of machine learning and relates to sequential decision making. In this paradigm, an agent interacts with an environment so that cumulative reward signals are optimized. The interaction occurs in a trial-error basis, and the agent learns a good behavior from previous experiences [50].

In terms of formulating the RL framework, several approaches have been presented in the literature that can be classified as either value-based, or policy-based methods [51]. Q-learning is a value-based methodology that have been popularized recently due to breakthroughs in deep learning coupled with reinforcement learning [52]. In Q-learning,

the optimal Q-value function represents the expected discounted return in a given state, and following the action dictated by the optimal policy.

To obtain the Q-value function, Monte Carlo methods are used to estimate the values from many simulations following a given policy. Practically, this is not an efficient method to learn the value functions [53]. Another approach is to construct a lookup table for each state-action pair, and updating the values using the Bellman equation. However, such method is inapplicable with a high-dimensional state-action space. Alternatively, the value function can be parametrized using deep learning. This gives rise to variations of Q-learning algorithms such as Deep Q-network (DQN), which has proved its effectiveness and superior performance [38].

Deep learning provides several advantages in the context of reinforcement learning. Many real practical problems are complex with a high dimensional continuous state space, and so neural networks are well suited for high-dimensional inputs. They can also be trained incrementally and learn progressively as they are provided with more data [45].

Neural Fitted Q-learning [54] uses a neural network to parameterize the Q-values, in which the network parameters are updated using a variant of gradient descent. Such method was proved to reach the optimal Q-value after iterating enough times. However, the method suffers from slow convergence and possible instability, mainly due to propagated errors stemmed from neural network generalization property, and the fact that weights of target network are updated simultaneously [53].

To bridge the gaps found in aforementioned methodology, deep Q-learning network (DQN) presented by [55] makes use of a separate target network that is updated frequently

with the training network parameters. It also utilizes a replay memory to store the experienced trajectories, from which mini batches are sampled to perform gradient descent on the network parameters. This method results in preventing divergence and reducing instabilities of learnt Q values. Additionally, the proposed methodology makes use of a replay memory to keep a set of experiences that spans across the state action space, and thus have less variance when updated as opposed to making a single step update.

Nonetheless, overestimated values are likely to be selected when following this methodology. To overcome this issue, [56] proposed double DQN or DDQN methodology. In DDQN, the evaluation of greedy policy is done according to the online network, while the target network is used to estimate its value. This leads to reducing overestimation of the Q values and improving the stability. The method showed great performance on RL benchmarks, as demonstrated in [57].

The thesis considers the energy management problem in the context of deep reinforcement learning and provides novel implementation of DDQN algorithm. In addition, other heuristics to enhance the performance of DDQN are considered, including the use of replay memory, linear annealed epsilon greedy, and others. The approach is designed to be data driven, which considers the historical profiles of different elements of a microgrid. Moreover, the methodology considers microgrid different dynamics and power flow constraints in a flexible framework that allows changing microgrid configurations.

2.4 Summary

This chapter provided an overall literature survey of the main categories of energy management methodologies. Specifically, it looked at a comparison between the model-

free and model-based approaches. Additionally, an overview of deep q-network reinforcement learning methodologies is provided.

In summary, the following gaps are observed:

- Lack of comprehensive implementation of the microgrid environment when implementing model-free methodologies (e.g., no consideration to power flow constraints)
- Only basic Implementation of simple variant of deep reinforcement learning is considered
- Simple networks are considered when implementing sophisticated deep reinforcement learning methods (e.g., simple residential network with a few variables)

The proposed thesis work attempts to close some of the gaps found in the current literature. The work investigates the use of deep reinforcement learning to optimally manage the different energy resources in a realistic model of microgrids. The methodology considers the stochastic behavior of different elements of a microgrid, including loads, generations, and electric prices. It also models different grid elements using equivalent models and considers the various power flow constraints in a realistic setting.

CHAPTER 3

PROPOSED METHODOLOGY

In this chapter, the proposed methodology to solve the microgrid energy management is presented. First, the mathematical formulation of deep reinforcement learning is provided. Then, energy management problem is formulated as an MDP process. Furthermore, the different dynamic constraints, state space, and action space of the different microgrid elements are presented. Finally, the shaping of the reward function used in the MDP formulation is provided.

3.1 Machine Learning Methods and Deep Reinforcement Learning

Machine learning encompasses a set of methods that have the capability to recognize patterns in data through examples or environment interaction to accomplish a given task [58]. Such methods can be broadly classified into three types:

- Supervised learning, defined as the task of obtaining a function mapping, e.g. classification or regression, from labeled data.
- Unsupervised learning, defined as inferring patterns in datasets without given labels
- Reinforcement learning, defined as the task of learning by an agent by interacting with its environment aiming to maximize the accumulated rewards

Machine learning has gone through great improvements in recent years, mainly due to advances in deep learning. Such advances can be linked to various factors including the

increasing computation power using Graphics Processing Units (GPU) or distributed computing, methodological breakthroughs in deep learning, and development of software ecosystems [53].

3.1.1 Deep Learning

Deep learning in its simplest form can be described as function mapping: $f: X \rightarrow Y$, that is parameterized by θ :

$$y = f(x; \theta) \quad (3.1)$$

where $\theta \in \mathbb{R}^{n_\theta}$.

A deep neural network has multiple processing layers, each layer conducts a non-linear transformation that leads to learning different levels of abstraction. A neural network with one hidden layer is described as shown in **Figure 3.1**. The first layer of the network gets the input features x (with vector size n_x). The next layer transforms the input values by a non-linear function, which can be expressed as follows:

$$h = A(W_1 \cdot x + b_1) \quad (3.2)$$

where A is a non-linear activation function, W_1 is the weight matrix of size $(n_h \times n_x)$, b_1 is a bias vector of size n_h . The hidden layer of size n_h can be further transformed by more succeeding layers up to the final layer y , which gives the output values. In case of the network shown in **Figure 3.1**, the output is given as

$$y = (W_2 \cdot h + b_2) \quad (3.3)$$

where W_2 is a weight matrix of size $(n_y \times n_h)$, b_2 is a bias vector of size n_y .

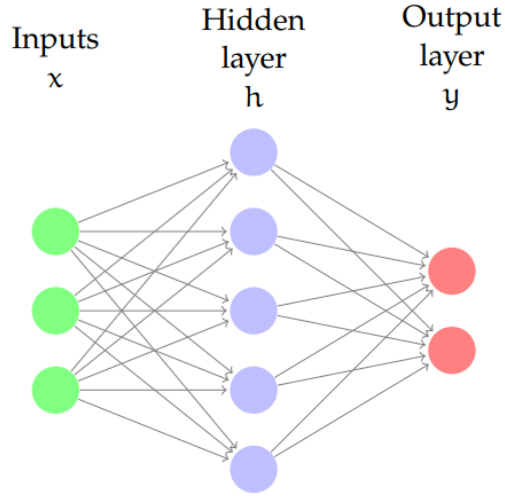


Figure 3.1 NN with one hidden layer

The network is trained to minimize the empirical error, most commonly through a gradient descent backpropagation algorithm. For example, one simple update algorithm of the network parameters is given as follows:

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} E[f] \quad (3.4)$$

where α is a learning rate, and $E[f]$ is an error function such as RMS or cross-entropy.

Beyond the simple structure of feedforward neural networks described above, many network variants have emerged with different sizes and layer operators to serve a specific application. The recent years saw an increasing number of hidden layers that exceeds a hundred such as in [59].

One architecture of interest is convolutional networks (CNNs) introduced by [60]. Such networks are well suited for pattern recognition in images and sequential time series. The network layers consist of filters or kernels that convolves the input into the next layer. Those filters are learned so that it activates when a certain feature is detected. An example of convolution neural network is shown in Figure 3.2, in which a CNN is used to classify

a 28x28 pixel image containing a handwritten digit. The CNN network consists of convolution filters, max pooling, and flattening operation to obtain features of value. These are then provided to a multilayer feedforward network that map the resultant output to the 10 digits.

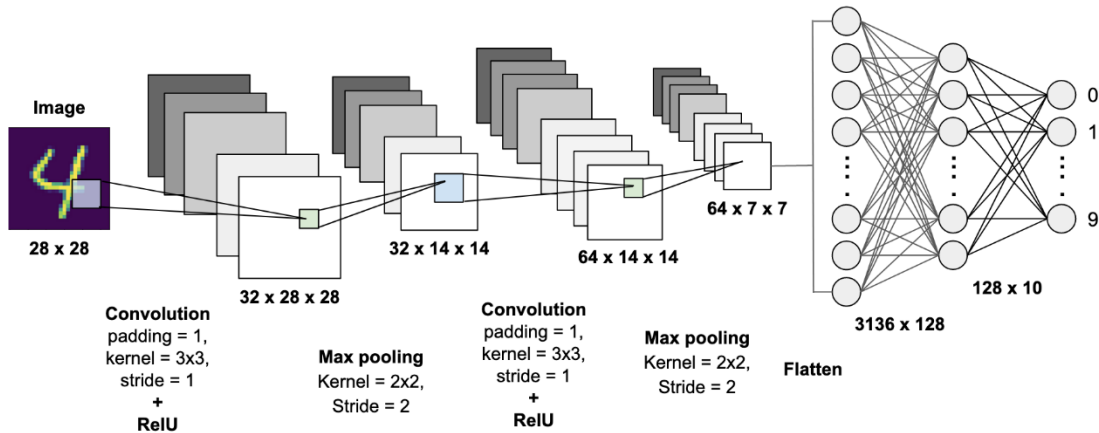


Figure 3.2 Convolution Neural Network (CNN) to classify handwritten digits [61]

Another important type of networks is recurrent neural networks (RNN). Such networks have a memory element and are capable of processing sequence of data. Figure 3.3 illustrates the structure of a simple recurrent neural network. In the left side, the middle layer with the letter 'h' denote a non-linear function which takes two values as an input and outputs two other values. The left side diagram is the compact version of the unfolded right side equivalent structure, which applies a recursive operator to pair (x,y) at each time step. Several variants of RNN have been devised over the years, one of which is the long-short memory network (LSTM). Such network is capable of encoding information from long sequences compared to a basic RNN [62].

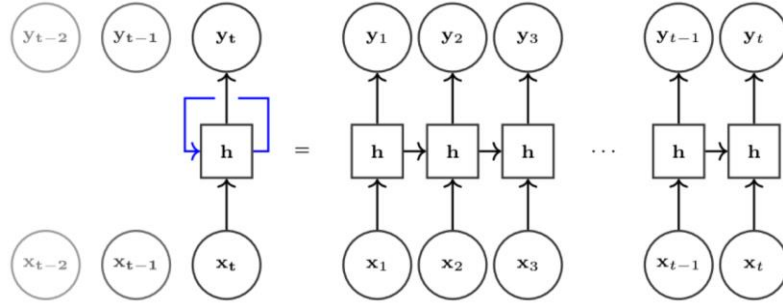


Figure 3.3 Recurrent neural network structure [53]

3.1.2 Reinforcement Learning

Reinforcement learning (RL) relates to sequential decision making in which an agent interact with an environment so that cumulative reward signals are optimized. The interaction occurs in a trial-error basis, and the agent learns a good behavior from previous experiences [50]. Formally, the reinforcement learning can be described by a discrete time stochastic control process. An agent starts in a state $s_0 \in \mathcal{S}$, and an initial observation $\omega_0 \in \Omega$. The agent makes an action at time step t , $a_t \in \mathcal{A}$. Then, the agent obtains a reward $r_t \in \mathcal{R}$, the state transitions to $s_{t+1} \in \mathcal{S}$, and a new observation is obtained $\omega_{t+1} \in \Omega$, as shown in Figure 3.4.

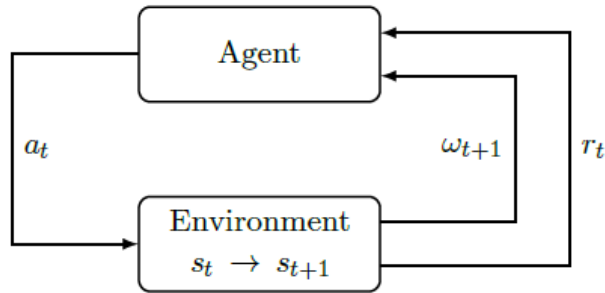


Figure 3.4 Reinforcement learning process

This formulation can be viewed as a Markov Decision Process (MDP) if the Markov property is satisfied. This entails that future states depend only on current observation, and

no full history is required. A fully observable MDP means that the observation matches the state, $\omega_t = s_t$. An MDP is described by the following:

$\mathcal{S}$: state space

$\mathcal{A}$: action space

T : transition function ($T: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$)

R : reward function ($R: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{R}$, where $\mathcal{R}$ is a continuous set of possible rewards)

γ : discount factor, $\gamma \in [0,1]$

Under this formulation, an agent selects an action dictated by policy π . Such policy can be either stationary (time dependent), or non-stationary (time independent). It can also be either deterministic or stochastic (gives a probability of taking an action).

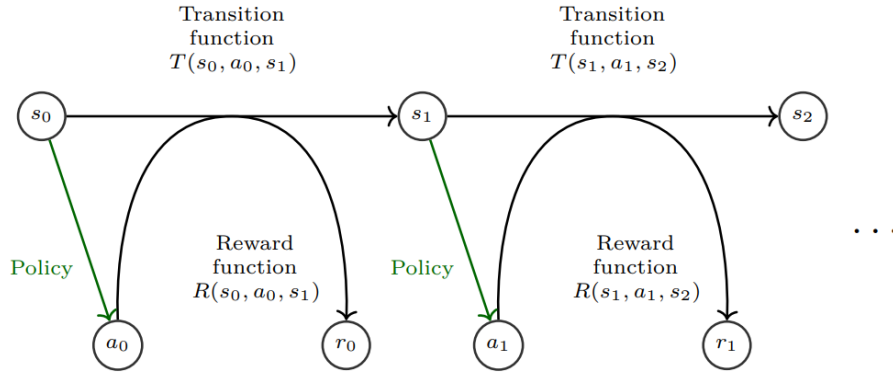


Figure 3.5 MDP illustration at each time step [53]

An RL agent aims to find a policy $\pi(s, a)$, so that the expected return is optimized. The return can be expressed as a Q-value function as follows:

$$Q^\pi(s, a) = \mathbb{E} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a, \pi \right] \quad (3.5)$$

where $r_t = \mathbb{E}[R(s_t, a, s_{t+1})]$.

This equation can be expressed recursively (using Bellman's equation) as:

$$Q^\pi(s, a) = \sum_{s' \in \mathcal{S}} T(s, a, s') (R(s, a, s') + \gamma Q^\pi(s', a = \pi(s'))) \quad (3.6)$$

The optimal Q-value function $Q^*(s, a)$, can be expressed as:

$$Q^*(s, a) = \max_{\pi \in \Pi} Q^\pi(s, a) \quad (3.7)$$

Additionally, the optimal policy can be defined as:

$$\pi^*(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} Q^*(s, a) \quad (3.8)$$

The optimal Q-value function $Q^*(s, a)$ represents the expected discounted return in state s , and following action a that is dictated by policy π^* . One straight way to obtain $Q^\pi(s, a)$ is to use Monte Carlo to estimate the values from many simulations following a given policy. Practically, this is not an efficient method to learn the value functions [53].

A simple Q-learning algorithm keeps a lookup table for each corresponding state and action. The optimal Q value is learned through the Bellman equation, but such method is inapplicable with a high-dimensional state-action space. Alternatively, the value function can be parametrized $Q(s, a; \theta)$, in which θ represent the parameters defining the Q values. This gives rise to variation of Q-learning algorithms such as DQN, which has proved its effectiveness and superior performance [38].

3.1.3 Deep Reinforcement Learning

Deep learning provides several advantages in the context of reinforcement learning. Many real practical problems are complex with a high dimensional continuous state space, and so neural networks are well suited for high-dimensional inputs. They can also be trained incrementally and learn progressively as they are provided with more data. An off-policy DRL methodology based on Q-learning is considered.

In Fitted Q-learning [54], a dataset D is constructed using past experiences in the form of (s, a, r, s') , in which reward r , and next state s' follow state-action (s, a) . The Q-value $Q(s, a; \theta_k)$ is updated at each iteration k to a target value Y_k , where:

$$Y_k = r + \gamma \max_{a'} Q(s', a'; \theta_k) \quad (3.9)$$

θ_k represent the parameters that define the Q-values at iteration k .

Such formulation can be applied in a neural network, where states can be provided as inputs to the neural network, and the outputs give the Q-values for each state-action. The network parameters θ_k are updated using methods such as stochastic gradient descent which minimizes the square error:

$$L = (Q(s, a; \theta_k) - Y_k)^2 \quad (3.10)$$

Hence, to update parameters at the next iteration, the following is applied:

$$\theta_{k+1} = \theta_k + \alpha (Y_k - Q(s, a; \theta_k)) \nabla_{\theta_k} Q(s, a; \theta_k) \quad (3.11)$$

where α is a scalar value defining the learning rate. Iterating through this update process should in hypothesis converge to an optimal solution. However, such formulation could

suffer from slow convergence and possible instability, mainly due to propagated errors stemmed from neural network generalization property, and the fact that weights of target network is updated simultaneously [53].

To bridge the gaps found in aforementioned Q-learning algorithms, deep Q-learning network (DQN) presented by [55] makes use of a separate target network with parameters θ_k^- , that are updated every C iterations with θ_k parameters. This method results in preventing divergence and reducing instabilities of learnt Q values.

$$Y_k = r + \gamma \max_{a'} Q(s', a'; \theta_k^-) \quad (3.12)$$

Additionally, a replay memory is used to keep the last N_{replay} steps experiences in the form (s, a, r, s') . Such that spans across the state action space, and have less variance as opposed to making a single step update. The algorithm is illustrated in Figure 3.6. To further illustrate the steps needed to implement DQN, **algorithm 1** is presented, which provides a pseudo code for DQN as implemented in [55].

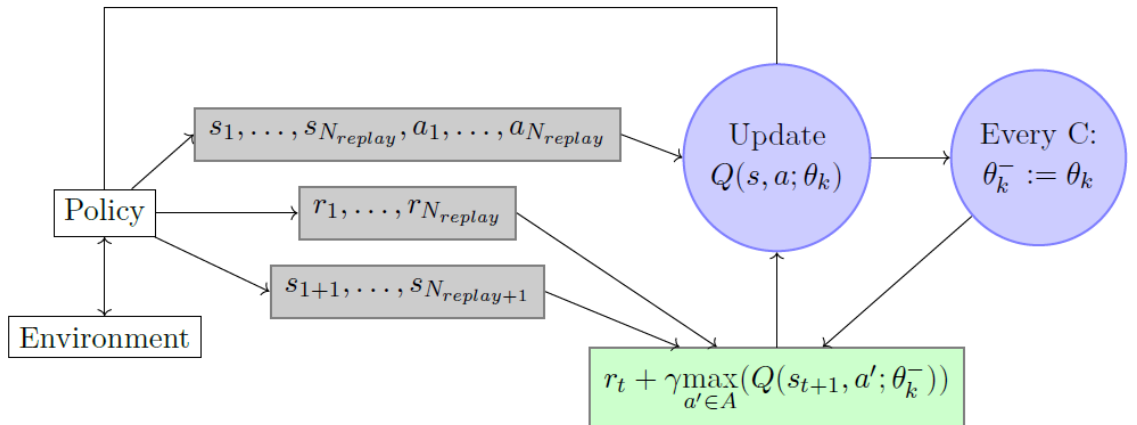


Figure 3.6 DQN algorithm flowchart [53]

Input: Microgrid observed states (e.g. SOC, Load MW, Generation MW, etc.)

Output: Q action values that determine policy to select action (e.g. controlling energy resources)

Initializing replay memory D

Initialize Q-network with random weights θ

Initialize target Q-network with weights $\theta^- = \theta$

For $episode = 1$ to M **do**:

 Initialize sequence $s_1 = \{x_1\}$

For $t = 1$ to T **do**:

 While following ϵ -greedy policy, choose action

$$a_t = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \arg \max_a Q(s_t, a; \theta) & \text{otherwise} \end{cases}$$

 Execute action a_t in environment and observe reward r_t and next state s_{t+1}

 Store transition (s_t, a_t, r_t, s_{t+1}) in replay memory D

 Sample random batches of transitions s_k, a_k, r_k, s_{k+1} from D

 Set $y_k =$

$$\begin{cases} r_k & \text{if terminal state is reached} \\ r_k + \gamma \max_{a'} Q(s_{k+1}, a'; \theta^-) & \text{otherwise} \end{cases}$$

 Perform gradient descent step on $(y_k - Q(s_k, a_k; \theta))^2$ with respect to q-network parameters θ

 Set $\theta^- = \theta$ at every C steps

End

End

Algorithm 1: DQN algorithm implementation as per [55]

To further enhance the DQN algorithm convergence and stability, the following heuristics are considered.

A. Priority replay memory

Experience replay allows an RL agent to make use of past experiences. In previous DQN algorithm, sampling occur uniformly from the replay memory, while ignoring how important the sampled transitions. [63] developed a method to prioritize experience replays, so that significant transitions are more frequently replayed to learn more effectively. The method uses importance sampling based on temporal difference error to select significant transposition and thus enhance overall performance.

B. Double - DQN

In DQN algorithm discussed, the max operator is used to select and evaluate an action as detailed in algorithm 1. Consequently, overestimated values are likely to be selected. [56] proposed double DQN to solve this overestimate issue. To do so, paper proposed the evaluation of greedy policy according to the online network (with parameters θ_k) , while using the target network (θ_k^-) to estimate its value. The target value is then expressed as follows:

$$Y_k = r + \gamma Q \left(s', \arg \max_a Q(s', a; \theta_k); \theta_k^- \right) \quad (3.13)$$

C. Linear annealed exploration

As opposed to using a fixed low ϵ value while following an ϵ -greedy policy, the agent can start training using a high eps value, and then decrease linearly as the training continues, arriving to a low ϵ value. Such way ensures the agent makes a lot of exploration earlier, while exploiting the accumulated information at the end [50]. This is demonstrated in Figure 3.7, in which the value ϵ decreases linearly over time until it reaches a fixed value. Note that exploration fraction is a hyper parameter that can be adjusted to fine tune the agent performance.



Figure 3.7 Linear annealed exploration

3.2 Microgrid Environment Formulation

When it comes to developing the framework of the microgrid environment, there are plenty of options to consider. Several papers adopted different elements of the microgrid such as limitation on the demand side as well as constraints on the generation side [25]. Reference [26] considers operation constraints in the microgrid as well as limits in renewable generation. Other papers looked into energy storage constraints such as charging and discharging rates, limits on electricity prices, or carbon emissions [27]. The thesis considers the most critical constraints of microgrid, namely constraints on the power exchange with external grid, power output constraints from different distributed generation devices, and power flow constraints. The adopted model of microgrid is interconnected with the utility at PCC, and has some conventional distributed generators, energy storage systems, solar PV, wind turbines, and variable loads, as illustrated in Figure 3.8.

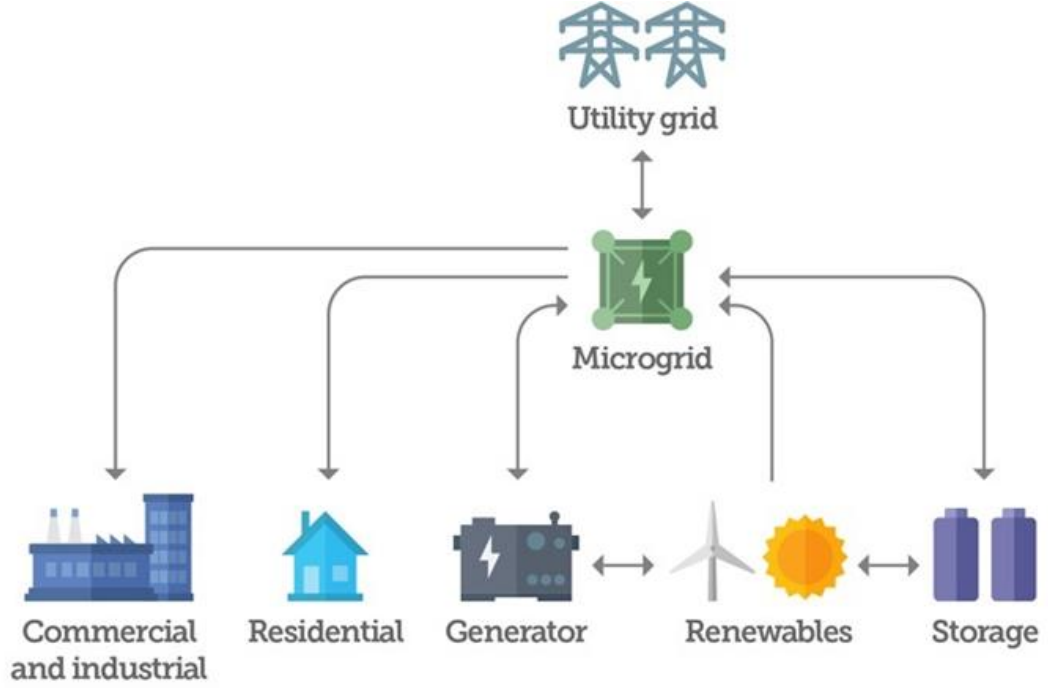


Figure 3.8: Microgrid environment configuration

The proposed formulation considers conventional generators that are constrained by the following equations:

$$P_{min}^{DG_d} \leq P_t^{DG_d} \leq P_{max}^{DG_d} \quad (3.14)$$

$$P_t^{DG_d^2} + Q_t^{DG_d^2} \leq S_{max}^{DG_d^2} \quad (3.15)$$

In which $P_t^{DG_d}$, $Q_t^{DG_d}$ represent the active and reactive power output of generator d, at time t. Rating of the generator is given by $S_{max}^{DG_d}$. The cost function of operating conventional generators is given by a quadratic function as shown in the following equation:

$$C_t^{DG_d} = \left[a_d (P_t^{DG_d})^2 + b_d P_t^{DG_d} + c_d \right] \Delta t \quad (3.16)$$

a_d , b_d , and c_d are constants.

Additionally, energy storage devices are constrained by the following equations:

$$0 \leq P_t^E \leq P_{max}^E \quad (3.17)$$

$$E_{min} \leq E_t \leq E_{max} \quad (3.18)$$

$$E_t = E_{t-1} + \eta_{ch} u_t P_t^E \Delta t - (1 - u_t) P_t^E \Delta t / \eta_{dis} \quad (3.19)$$

In which the charging or discharging power is represented by P_t^E , and the energy level (SOC) is given by E_t . A binary variable u_t is used to indicate if the ESS is charging or discharging. To represent the charging/discharging efficiencies, η_{ch} and η_{dis} are used accordingly. The time period of charging/discharging is represented by Δt .

The power exchange with the utility is considered, governed by the following equations:

$$-P_{max}^U \leq P_t^U \leq P_{max}^U, \forall t \quad (3.20)$$

$$P_t^{U^2} + Q_t^{U^2} \leq (S_{max}^U)^2 \quad (3.21)$$

In which P_t^U and Q_t^U represent the active/reactive power exchanges with the utility. The maximum exchange of complex power is given by S_{max}^U . The cost of purchasing power from the utility is given by the following equation, in which R_t is the real time price.

$$C_t^U = P_t^U \cdot R_t \cdot \Delta t \quad (3.22)$$

Additionally, power flow constraints are considered while controlling the energy resources.

This is represented power flow limits at each branch ij , as shown below:

$$P_t^{ij^2} + Q_t^{ij^2} \leq S_{max}^{ij^2} \quad (3.23)$$

To ensure voltage limits are within standard values, the voltage is constrained as follows:

$$|V^i|_{min} \leq |V_t^i| \leq |V^i|_{max} \quad (3.24)$$

where V_t^i is the voltage at bus i at time t is limited by given minimum and maximum values.

3.3 Energy Management Problem Formulation

Using the aforementioned model of a microgrid, MDP is formulated as follows. The state variables at each time step t is given by $(P_t^{PV}, P_t^W, P_t^D, Q_t^D, R_t, E_t)$, in which P_t^{PV} is the power output from the solar PV, P_t^W is the power output of the wind turbine, P_t^D and Q_t^D are the load active and reactive powers, R_t is the real time price of electricity, and E_t is the energy level of the energy storage system. The model can be formulated to take only current state variables, or past historical variables [17]. The choice of which data time period to be provided to the model is later discussed at the implementation section.

The action space is given by $(P_t^{DGd}, Q_t^{DGd}, P_t^E)$, in which P_t^{DGd}, Q_t^{DGd} are the active/reactive power from conventional generators, and P_t^E is the charging/discharging power of the energy storage system. The action space is developed to be discrete in order to be compatible with DQN algorithm. Other DRL algorithms can accept continues space such as deep deterministic policy gradient [64], though we are only concerned with the implementation of DQN.

The interaction of agent with the environment, and the overall flow of training is illustrated in Figure 3.9. The state variables of the microgrid are provided to the q-network, which in turn makes a decision based on ϵ -greedy policy (as explained in 2.3.3) that maximizes the

q-value. The interactions between the environment and the agents are collected and saved in a replay memory, to be used for training the online network, and the target network. This process continues until episodes end, or termination conditions are met.

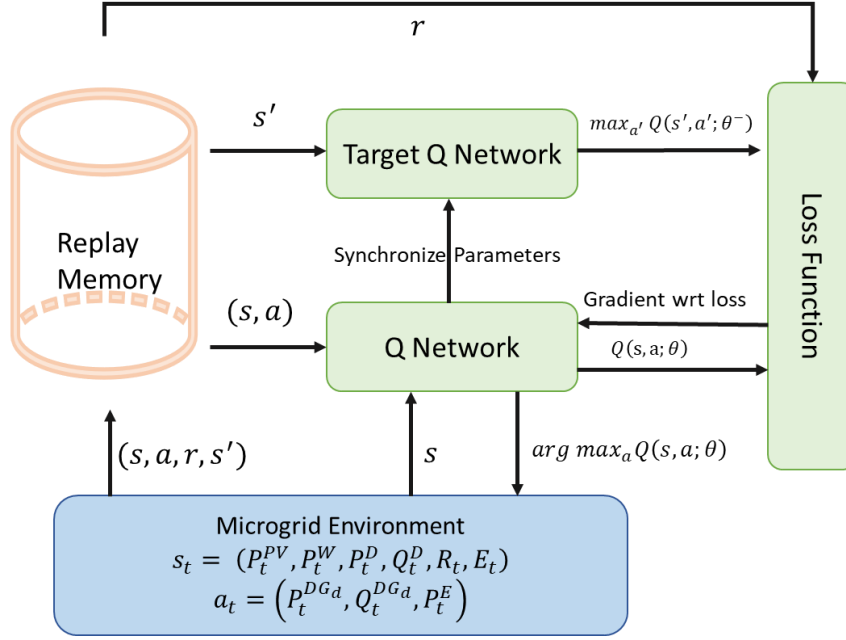


Figure 3.9 Architecture of DQN for microgrid energy management

3.4 Reward Function

Designing a reward mechanism for an RL agent is a critical step in a successful implementation, and often requires an expert's knowledge of the subject problem. In a simple game, the reward could be a positive number correlating with the numerical score, or final episode result. However, for a complex task such as energy management of a microgrid, the reward function could combine multiple objectives in a network and requires thorough understanding of different dynamic constraints.

The reward function in the energy management problem can take different approaches depending on use case, regulation, cost of electricity and several other factors. The reward

function could consider environmental costs, capital and operational costs, energy storage costs, or any other type of penalties. Examples of variables considered in the reward function include imported energy from the external grid [18], distributed generation costs [19], and grid costs [20].

In this thesis, the main objective is to optimize the operational costs while adhering to system constraints. Hence, the reward function is formulated to correlate with the microgrid operational costs. These include the costs of operating the conventional generators, and power purchases from utility. Equation (3.25) provides a simple formulation of the reward function at each time step.

$$r_t = - \left(\sum C_t^{DGd} + C_t^U \right) \quad (3.25)$$

In addition, the reward function considers any violation to the power flow constraints, by providing a negative reward, and terminating the training episode. It also has a negative reward whenever an action violates energy storage constraints, so that violating actions are discouraged.

CHAPTER 4

IMPLEMENTATION

This chapter details the implementation process to solve the microgrid energy management problem using deep reinforcement learning. The different libraries used to implement the proposed methodology are explained. The microgrid environment is first developed using OpenAI gym library framework, that interacts with Pandapower library that checks power flow simulations. Additionally, the DRL agent is developed using Stable Baselines library. Finally, case studies are provided demonstrating the implementation process.

4.1 Implementation Process

To implement DRL methodologies to solve the microgrid energy management, [44] designed a novel DQN agent that interacts with a residential microgrid environment. The environment is developed in isolation from the agent, though the agent can observe some states as governed by the methodology. The agent is allowed to take discrete actions on two types of energy storage devices, either charging, discharging or doing nothing. Nonetheless, the developed environment lacks many considerations of larger microgrids.

The thesis adapts the environment considerations implemented in [17], which uses more comprehensive microgrid dynamics and constraints. In addition, the thesis implements the DRL methodologies using state of art practices and adopts robust open source libraries. The implementation process of the proposed methodology is shown in Figure 4.1. The microgrid environment is implemented following the API requirements of the Gym library,

and Pandapower is run simultaneously to ensure system constraints are not violated. The DQN agent, developed using Stable Baselines requirements, takes actions while interacting the microgrid environment. Further details on these libraries are provided in the next sections.

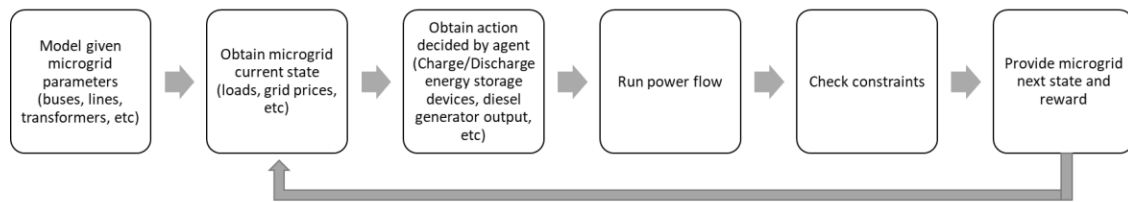


Figure 4.1 DRL implementation process flow

4.2 Pandapower

Pandapower [65] is an open-source power system analysis library based on Python. It provides many functions to conduct variety of studies including power flow, state estimation, short circuit calculations, and others. It can also model variety of electrical elements given nameplate parameters which are then evaluated using equivalent circuit models. Data in Pandapower models is structured in Python Pandas data frames, allowing ease of data manipulation and edition.

Since our thesis deals with energy management in a microgrid, it is essential that any action taken does not violate power flow constraints. Hence, power flow is simulated using

Pandapower at every time step to ensure compliance of bus voltage levels, and line loadings.

In this section, the different electric models available within Pandapower are explained. In addition, a case study of using Pandapower in a given microgrid is presented.

4.2.1 Electric Element Models

Pandapower adopts element-based models to model different types of electric grids. In such models, every element is connected to one or more buses and is defined with nameplate parameters. As opposed to generic branch models, there are separate models for lines, transformers, etc. This provides flexibility in designing a network with nameplate parameters, such as a line length or its relative impedance.

Pandapower has several electric models, represented by their equivalent circuits, which will be discussed briefly below. [66] can be used to get through details of the different formulas to calculate the electric parameters.

1. Bus

A bus represents the nodes of the grid, defined by a nominal voltage in kV. The rated power for a given bus can be specified with parameter `net.sn_kva`. After conducting analysis, voltage magnitudes and voltage angles can be evaluated.

2. Line

A line is modelled with π -equivalent circuit [67]. Different parameters are specified when modelling a line, including line length, longitudinal impedance which is represented by the resistance and reactance, and shunt admittance. After conducting analysis, the toolbox provides the line current `res_line.i_ka`, which is basically the

maximum current across the line, and the line loadings *res_line.loading_percent* which is the ratio of line current *res_line.i_ka* to the maximum thermal line current.

3. Transformer

Pandapower can model two or three winding transformers. Equivalent models can be based on T equivalent circuit or π -equivalent circuit [67]. Different parameters are specified when defining a transformer, including voltage levels at the high and low voltage sides. Additionally, different transformer losses such as copper losses, active power losses and other parameters can be defined. Moreover, the model allows for other features such as changing transformer ratio by defining a tap changer.

4. Load

To model power consumption, loads are used, which are defined by active and reactive power. Different models are available including constant power, constant current, or constant impedance. Additionally, Pandapower allows to specifying other parameters such as scaling, percent of load constant current.

5. Static Generator

To represent the injection of constant power by any type of generation, static generators are used. The value of injected power is specified with active power *sgen.p_kw*, and reactive power *sgen.q_kvar*. Similar to other electric models, the toolbox allows specifying different parameters such as scaling, to provide more flexibility.

6. External grid

This element models an external grid represented by voltage magnitude and voltage angles. Positive P&Q values represent importing power from external grid, while negative values represent exporting power from microgrid to external grid. Different parameters of external grid can be specified, such as maximum and minimum injected active and reactive powers.

7. Energy Storage

Pandapower provides an element for energy storage, specified by P&Q values. However, Pandapower does not update SOC during simulation, as it is time independent. Hence, energy storage is modelled using a sperate method designed specifically for our grid. The method will update energy storage SOC, and account for different efficiency losses.

4.2.2 Power Flow Solver

The power flow solver in Pandapower uses a method based on Newton Raphosn [68]. Different parameters such as node type are done automatically by the library. Additionally, different initialization methods are offered such as using DC power flow solution, or flat initialization. Pandapower power provide an efficient implantation of the solver that allows better solution convergence, as compared to other libraries (e.g. Matpower).

4.2.3 Case Study

To evaluate the use of Pandapower, a benchmark medium voltage microgrid is implemented [69]. The MG configuration is shown in Figure 4.2. The microgrid consists of two sub-networks comprising two feeders operating at 20 kV. It is connected to an external grid rated at 110 kV through two step-down transformers. Various variations of the network can be achieved by configuring switches S1, S2, and S3. For example, when

all switches are open, the feeders operate in a radial manner. Additionally, to create a loop in feeder 1, S2 and S3 are closed.

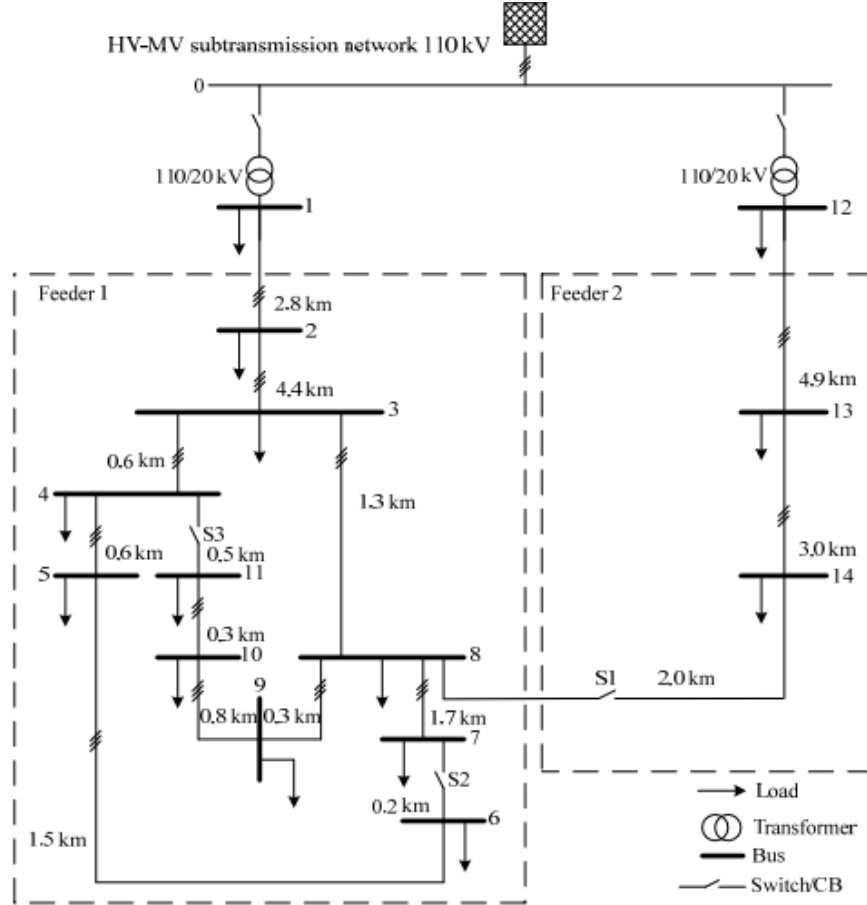


Figure 4.2 Benchmark Medium Voltage Microgrid Configuration

To implement this microgrid, the different parameters of each element need to be specified. Table 7.1 in the appendix, provides the line parameters including parameters such as lengths, resistance, and reactance. Table 7.2 shows the main transformer parameters used for T-model of a two-winding transformer. Table 7.3 provides load nominal data.

To test power flow analysis in Pandapower, time series data for loads and distributed generation are synthesized using typical profiles, as shown in Figure 4.3 & Figure 4.4.

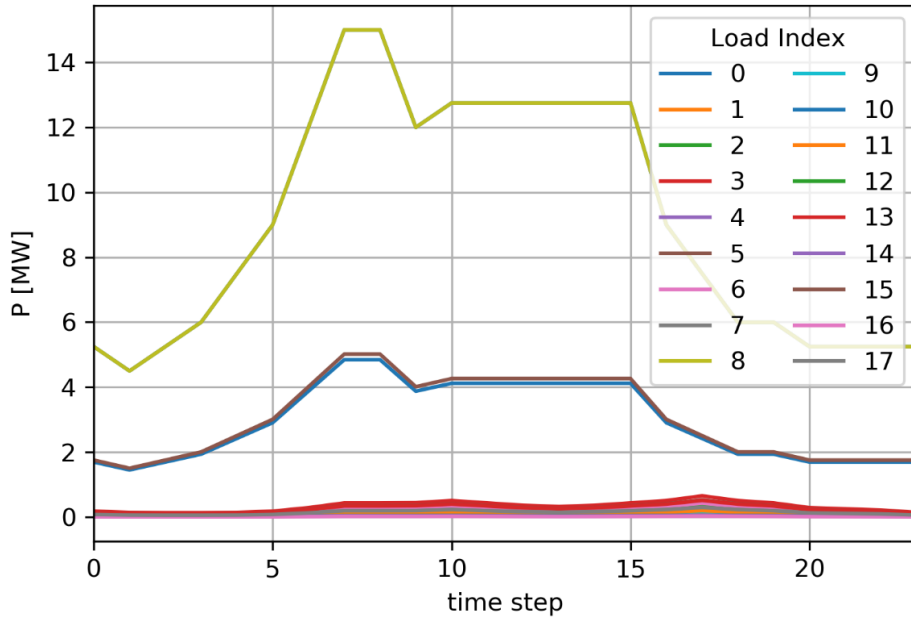


Figure 4.3 Load profile in 24-hour period

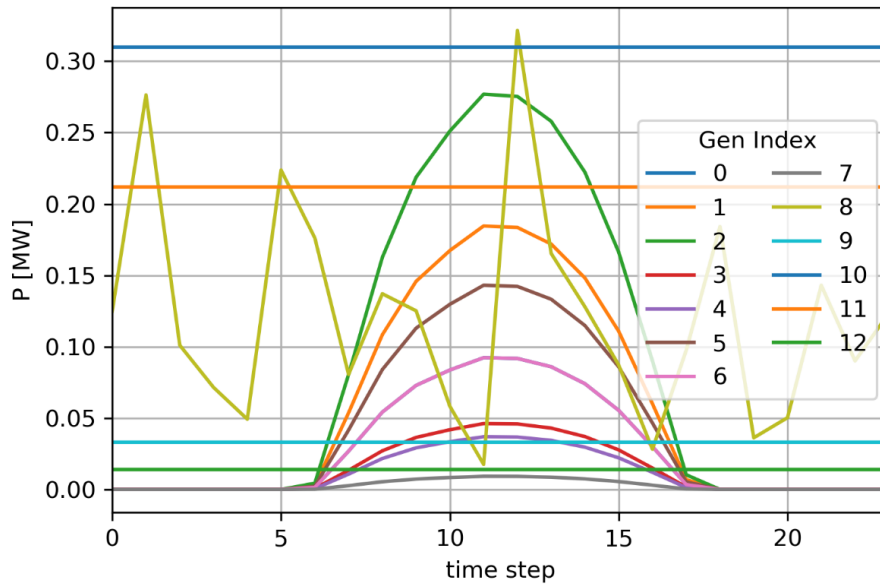


Figure 4.4 DER generation profile in 24-hour period

After conducting power flow at each time step, below results are obtained. Figure 4.5 show voltage magnitudes of each bus during the specified period. The voltage levels at the external grid bus are fixed at 1.03pu, whereas the rest of buses vary between 0.96 & 1.02pu. The voltage levels remain within acceptable values during this period.

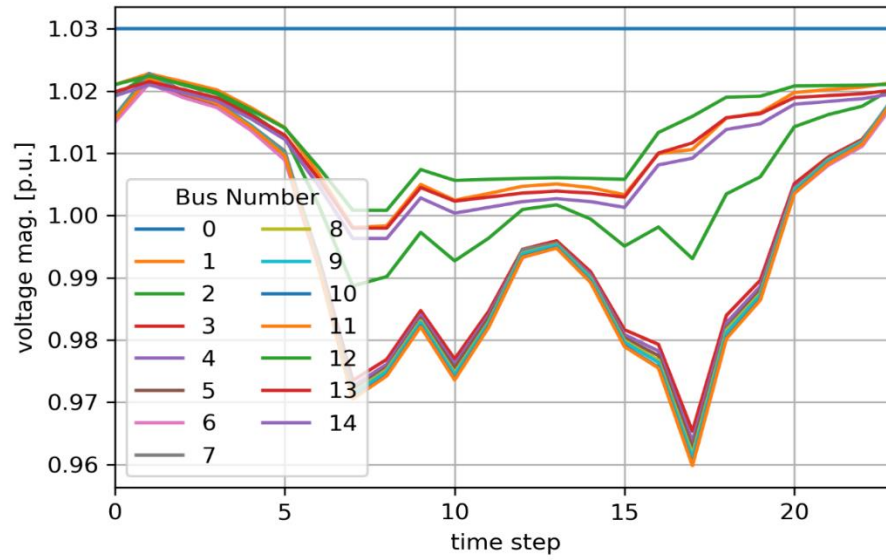


Figure 4.5 Bus Voltage Magnitude

Line loadings is shown in Figure 4.6 in percentages for all lines. Only lines 1 & 10 reach high values since they are the main lines supplying power to the rest of microgrid.

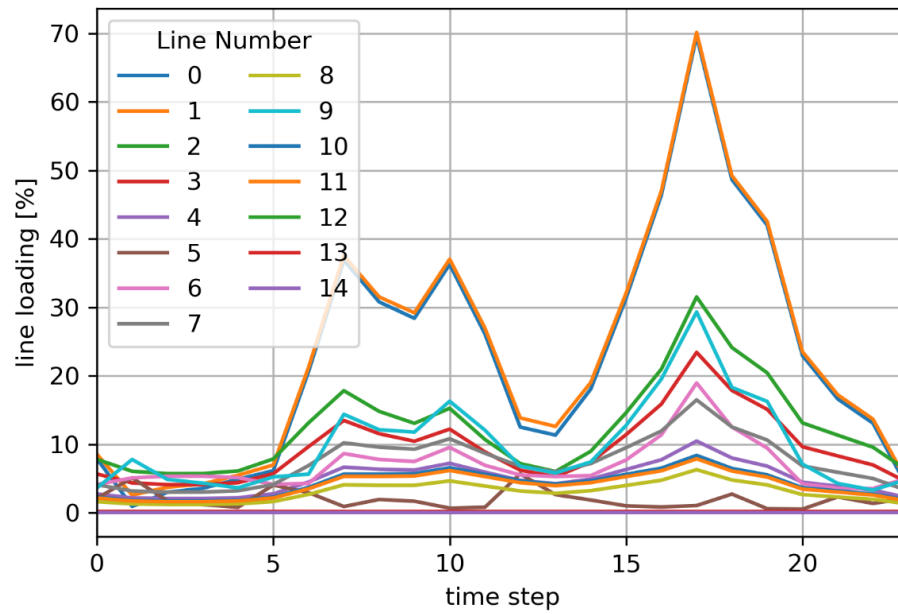


Figure 4.6: Line loading profile

4.3 Deep Reinforcement Learning Toolkits

In this section, two main toolkits that are used to formulate the deep reinforcement learning problem are discussed. They are both designed by OpenAI team and are proven to be state of art framework for implementing and solving DRL related problems.

4.3.1 OpenAI Gym Library

Gym [70] is an open-source toolkit developed by OpenAI to conduct research in the field of reinforcement learning. It provides an abstraction for the environment in a RL framework as discussed in section Figure 4.7, and assumes no structure of the agent. At each time step, the environment takes an action dictated by the agent, and returns an observation, and reward. A reset function is called whenever the process is started, which provides the initial observation.

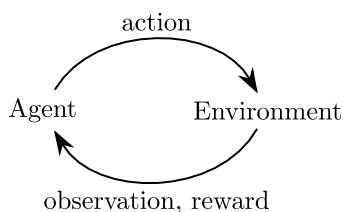


Figure 4.7 DRL main components

Gym library provides a long list of environments with a common interface, to allow for benchmarking differing agents. Available environments include classical control problems, algorithmic based tasks, 2D and 3D robotic control and others. It also provides a framework to create custom environment, in which the different environment constraints and dynamics can be implemented, as well as the reward mechanism. Later in the thesis, a custom

environment for a microgrid is implemented, and is used to conduct experiments for deep learning agents.

4.3.2 Baselines

Baselines [71] is an opensource toolkit developed by OpenAI and has state of art implementations of several reinforcement learning algorithms. Due to ongoing development, lack of unified structure, and weak documentation, a forked version of baselines called ‘stable baselines’ is used [72]. Stable baselines provide better documentation, and better monitoring functions used for analyzing the behavior of different implemented agents.

4.3.3 Case Study

To illustrate the working mechanism of Gym environment along with an agent by Stable baselines, a simple experiment based on the cart pole control problem is implemented. This problem consists of a pole that is attached to a cart, that moves in a frictionless track. The formulas governing the dynamics of the pole is implemented in the Gym using [73]. The pole is unstable but can be controlled by applying forces to the cart appropriately. Initially, the pole starts upright, as shown in, and the aim is to prevent it from falling. For every timestep the pole stays upright, a reward of +1 is given. The environment is terminated when the pole falls more than 12 degrees from the vertical axis, the cart moves more than 2.4 units away from the center, or the episode length exceeds 200.

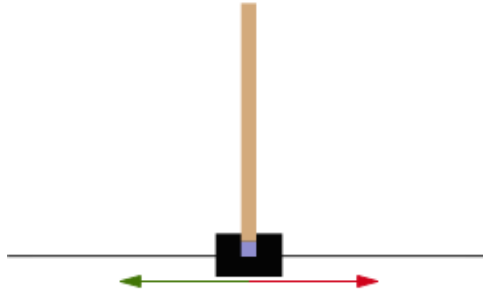


Figure 4.8 Illustration of cartpole

The environment observations are initiated with continues variables, constrained between minimum and maximum values as indicated in Table 4.1. Additionally, the agent can take one of two actions at each time step as detailed in Table 4.2. The problem is considered solved when the average reward is greater than or equal to 195 over 100 consecutive trials.

Table 4.1 Observation parameters

Observation	Min	Max
Cart Position	-4.8	4.8
Cart Velocity	-Inf	Inf
Pole Angle	-24 deg	24 deg
Pole Velocity at tip	-Inf	Inf

Table 4.2 Action space of agent

Action	Description
0	Push cart to the left
1	Push cart to the right

An algorithm based on DQN is used with a multi-layer perceptron with 2 hidden layers, each with a number of nodes with a rectified linear activation function. The input layer contains the 4 states observed from the environment, whereas the output returns the approximated Q-values for each action as shown in Figure 4.9.

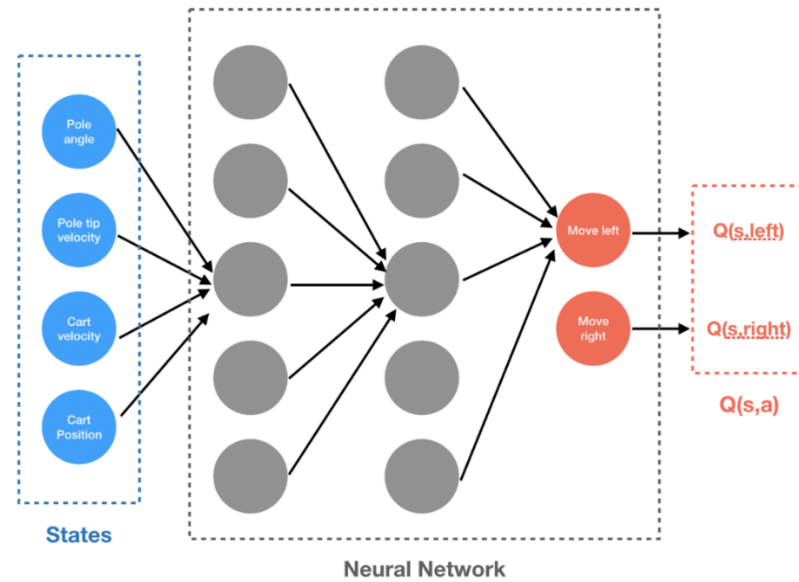


Figure 4.9 MLP used in cartpole problem

The training is done for 10,000 steps in this experiment. The learning performance for a number of different models each with different number of hidden nodes is shown in **Figure 4.10**. The best model can be saved during training, as the performance fluctuate as agent explore/exploit the environment. Overall, 3 of the experimented models achieved a reward over 200, with the best reward achieved by the MLP with 2 hidden layers each with 64 nodes.

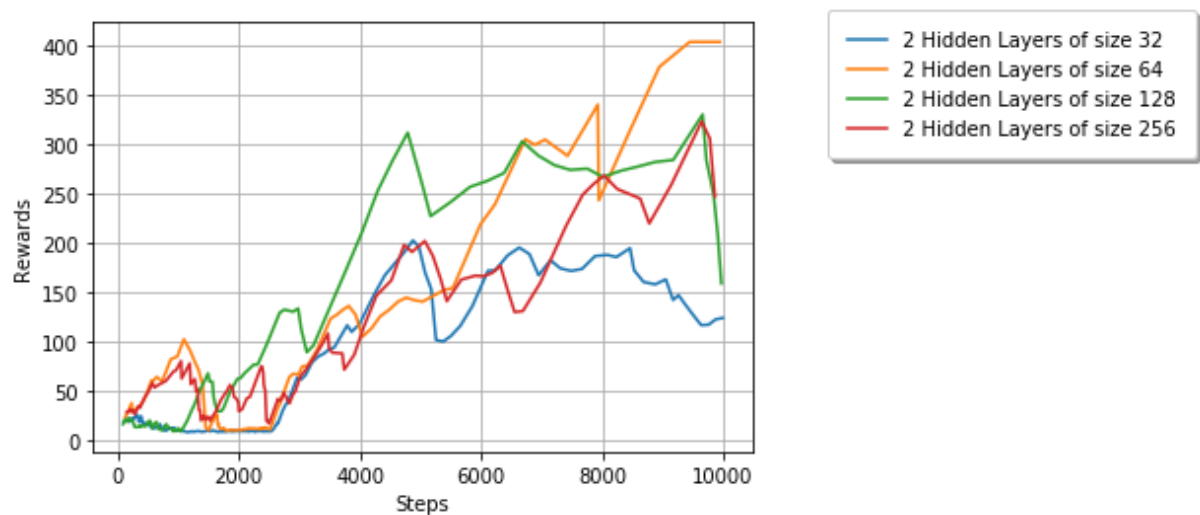


Figure 4.10 Learning performance of DQN with different MLP structures

Further hyperparameter tuning could lead to better learning and better results overall. Benchmark studies comparing model-based methodologies such as MPC versus model free methods based on reinforcement learning have been presented literature. For example, [74] showed that model-free RL agent can outperform nonlinear MPC in a similar experiment when the model is not accurately designed. Additionally, [75] compared the different reinforcement methodologies at the cartpole experiment as well as many other classical environments.

4.4 Energy Management of a Microgrid

In this section, the implementation of the proposed methodology to solve the energy management problem in a given microgrid is discussed. The subject methodology is first implemented for a simplified microgrid consisting of a single load, with PV, and energy storage. Then, a generic framework is developed, in which a more complex microgrid is implemented.

4.4.1 Energy Management of a Simplified Microgrid

The methodology is initially implemented in a small microgrid, consisting of one load, one solar PV, and one energy storage as shown in Figure 4.11. These are interfaced with the grid through a single coupling point. The energy is transferred back and forth between the grid and the microgrid, allowing the exchange of energy according to varying electric prices.

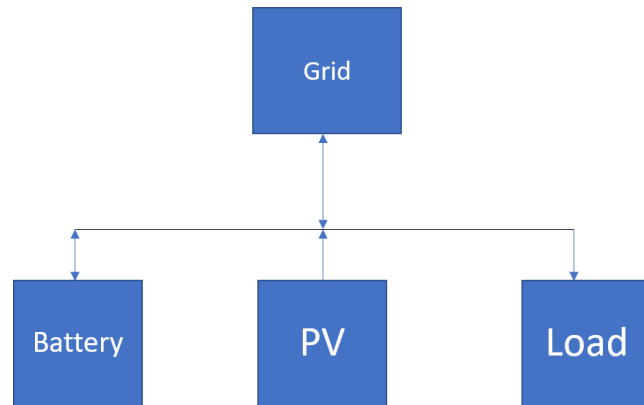


Figure 4.11 Schematic of microgrid

A residential load with nameplate peak of 10kW is adopted. A sample load profile is shown in Figure 4.12. Solar PV is connected in the load side, with a nameplate capacity of 5kW.

A sample PV generation is shown in Figure 4.13. The electric tariffs are varying according to market conditions, with a sample shown in Figure 4.14.

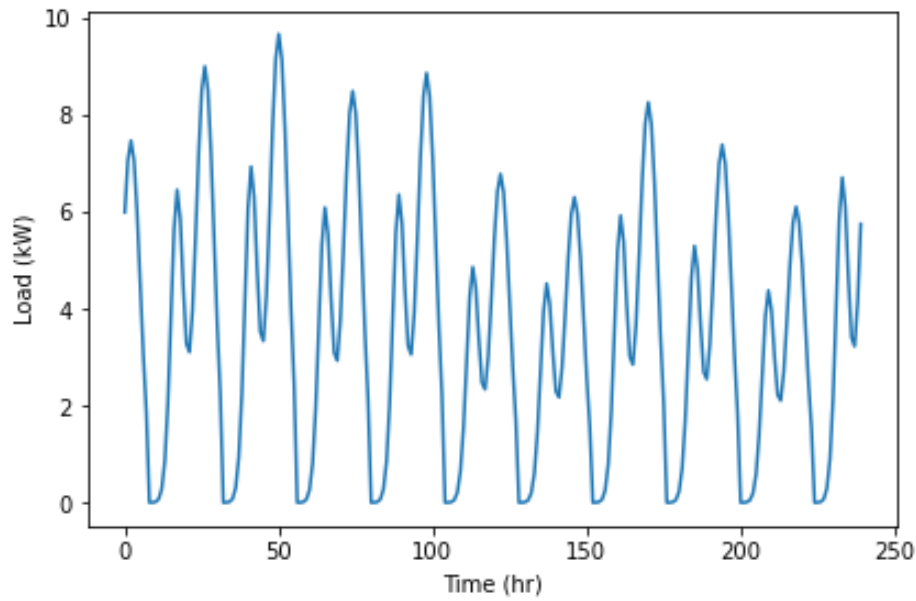


Figure 4.12 Load profile sample

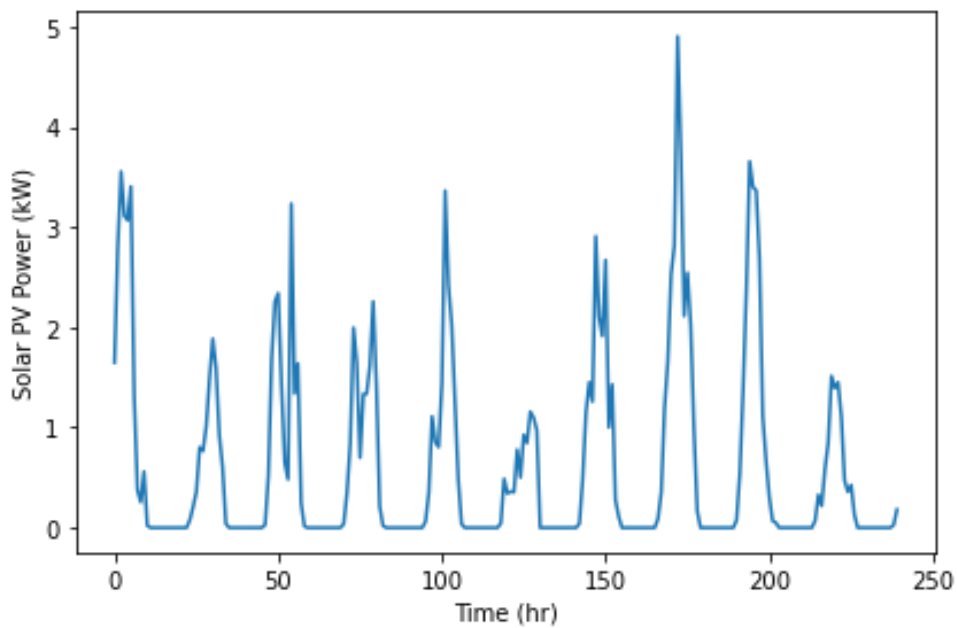


Figure 4.13 PV power generation profile sample

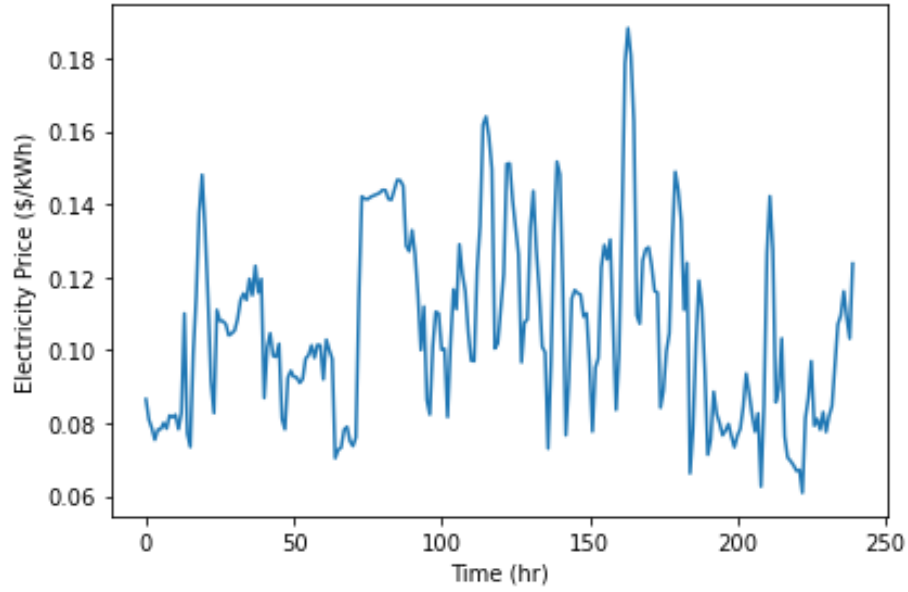


Figure 4.14 Electricity prices sample

The objective is to optimize the operation of the energy storage so that to maximize the costs paid to the grid. This can be done by taking benefit of the varying electric prices along with varying load and PV generation.

To illustrate the capabilities of DRL based on DQN to solve such types of problems, an experiment is conducted in which information about state variables is provided to the network. The objective of this experiment is to show that our methodology can approach optimal values if given information of future variables. A feedforward NN is used to map the different states at any given time to their corresponding state action value function for each possible action. The network takes an input of future 24 hours values of the different variables (PV, load, grid prices), and outputs the Q-Value for each possible action (e.g. charging/discharging) at current state. This is illustrated in Figure 4.15, in which a NN with two hidden layers is used with rectified linear activation functions.

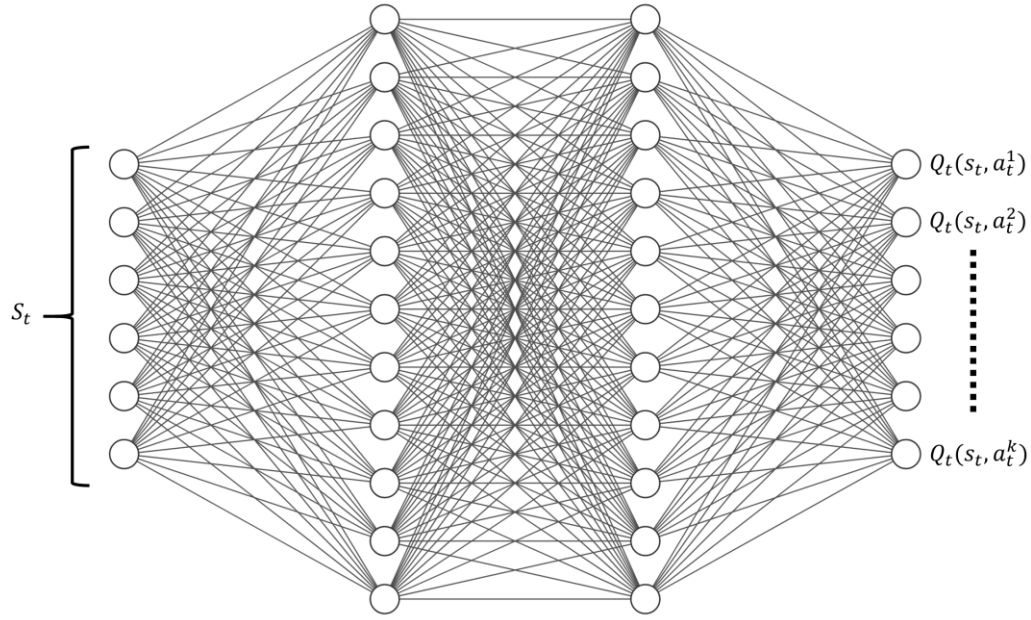


Figure 4.15 Illustration of NN used to map states to state-action value functions

To show that the methodology is robust in finding optimal operation schedule for different energy storage parameters, the initial SOC is varied for the 5kWh energy storage. The obtain results are compared with those obtained a MILP solver, that takes the state variables for one full year. The results clearly show that proposed methodology can optimally solve energy management if given correct prediction.

Table 4.3 Annual operating costs of DQN, and optimal values for different SOC

Initial SOC	Costs (\$) - DQN	Costs (\$) - Optimum
0%	3100	3103
0.25%	3101	3102.2
0.5%	3099	3102
0.75%	3098.3	3101.9
100%	3098.5	3101

Figure 4.16 shows the optimal resultant SOE for the energy storage compared to corresponding electric prices. Additionally, Figure 4.17 shows a comparison of the resultant SOE (DQN agent) with optimal solution. Clearly, both solutions act in similar manner across the time period.

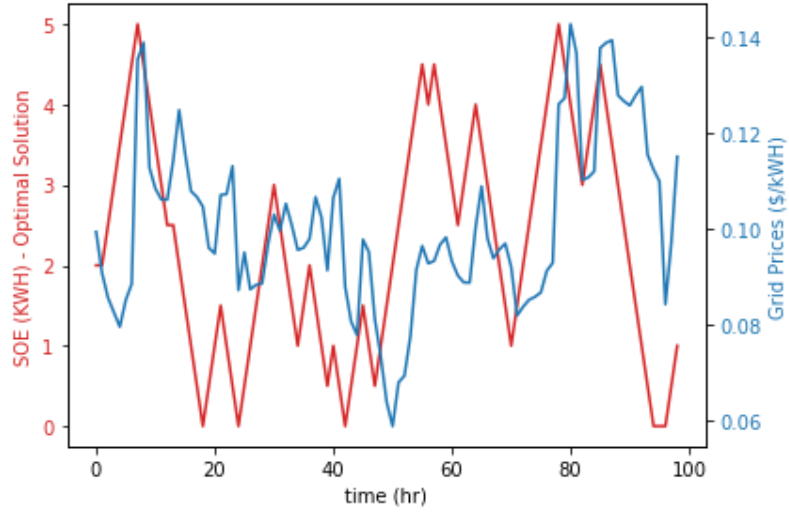


Figure 4.16 SOE optimal solution with respect to grid electric prices

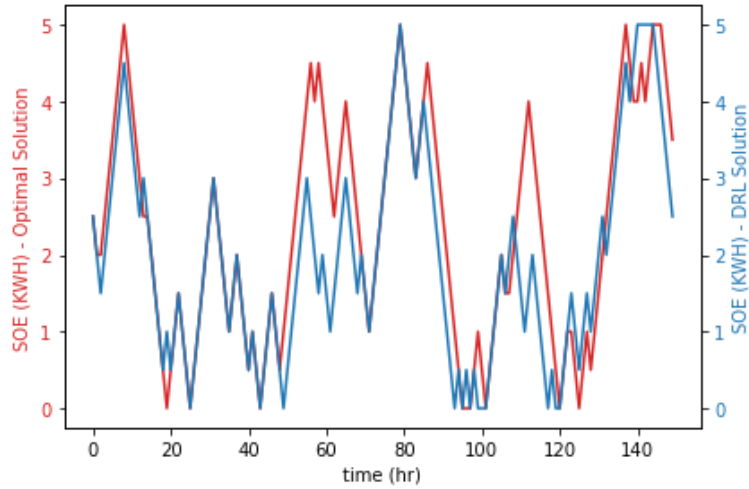


Figure 4.17 Optimal SOE vs SOE obtained from DQN agent

The learning curve of the DQN agent is provided in Figure 4.18, showing a smooth learning progress with time. Additionally, the performance of the developed algorithm is tested with different energy storage capacities. Table 4.4 provides a comparison between results obtained from DQN and optimal case for different energy storages. It is observed that DQN was able to reach optimal values in different scenarios.

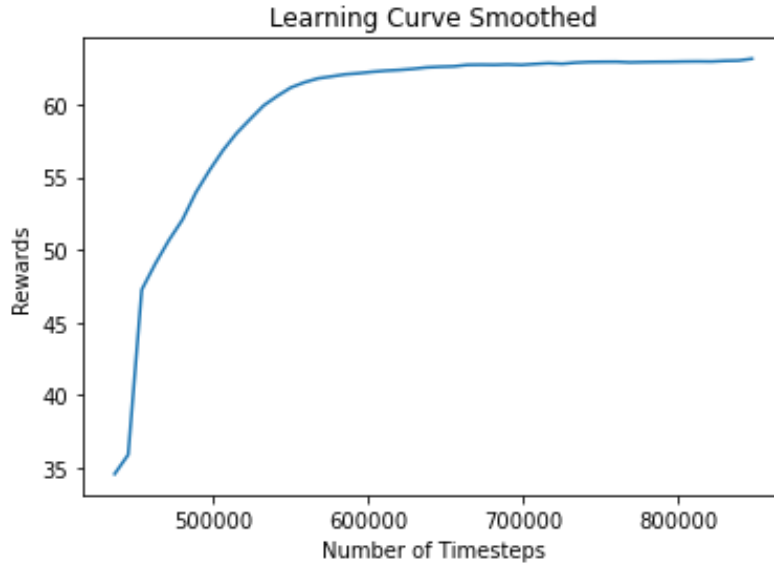


Figure 4.18 learning curve for of DQN agent given 24 hours of future state values

Table 4.4 Annual operating costs of DQN, and optimal values for different capacities

Battery Capacity (kWh)	Costs (\$) - DRL	Costs (\$) - Optimum
2	3120	3123
4	3103	3107
6	3094	3098
8	3086	3091
10	3084	3087

Now that DQN is shown to reach near optimal values when provided with actual future values, an experiment is conducted with providing historical values to the agent. In this experiment, a real case scenario is simulated, in which the model only observes the past 24 hours values of the variables. The model tries to predict the best action to optimize rewards.

The training is run for different DQN architectures. **Figure 4.9** shows an example of learning performance for different NN structures. Similarly, different hyperparameters are tuned to achieve better results. The effect of varying hyperparameters is discussed in next chapter. In this experiment, the DQN agent could achieve up to \$3050 of operational costs

compared to the optimal case of \$3102. The performance of DQN in managing energy resources is further analyzed in the next chapter.

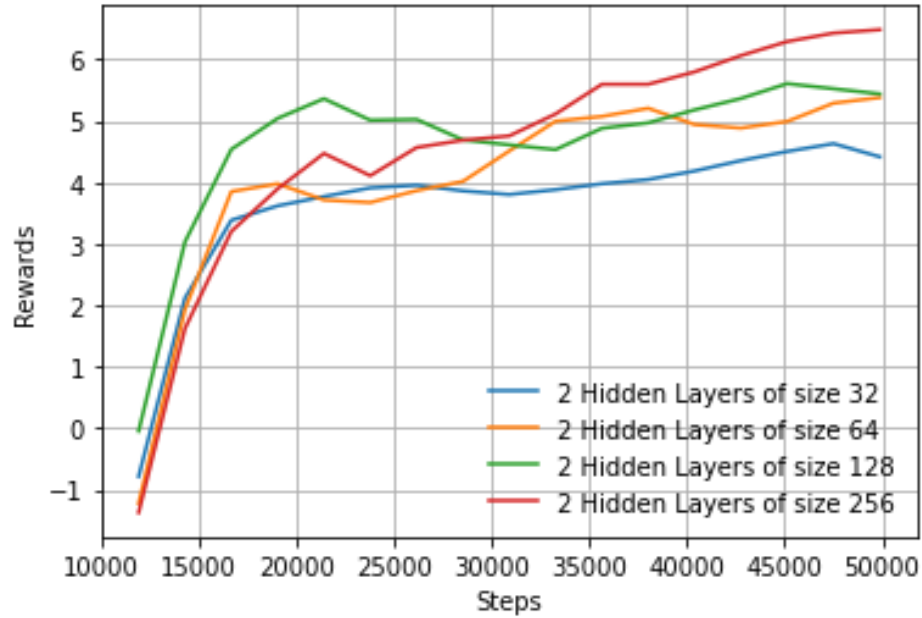


Figure 4.19 Learning curves for different NN structures

4.4.2 Energy Management of a Complex Microgrid

In the previous section, it was shown how DQN can solve energy management problem for a simple network and provide near optimal values. However, the implementation ignores network constraints, and does not fully integrate other complex devices usually found a microgrid.

We develop a framework that implements a generic microgrid, such as the one shown in Figure 4.20. The different parameters used to implement such networks can be specified to be simulated and checked. Different types of resources can be incorporated in the grid, be it conventional or non-conventional distributed power generation units including PV, wind,

diesel generators, and energy storage units. It is also possible to connect the microgrid to an external grid through a PCC.

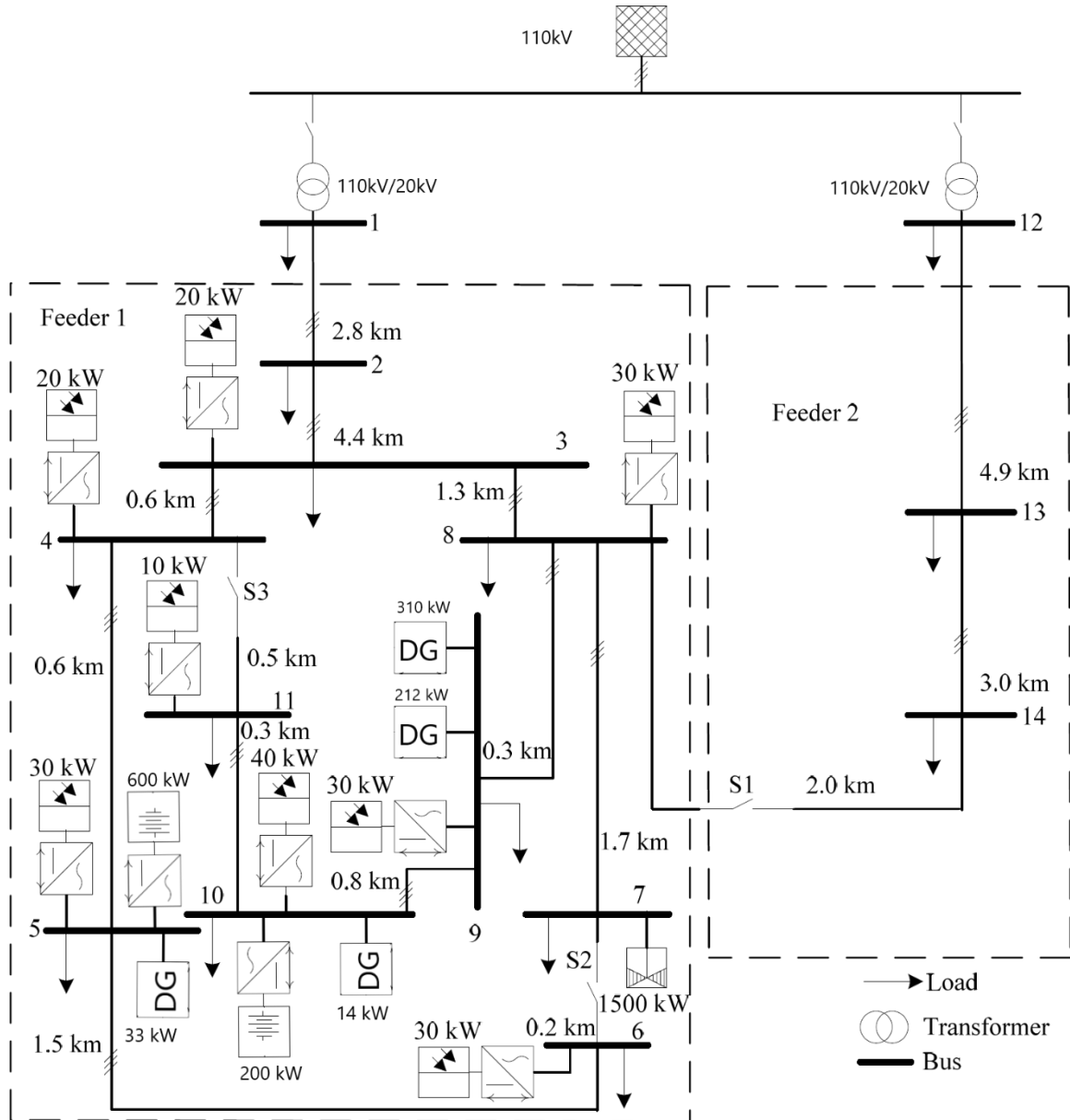


Figure 4.20 MV Microgrid

The framework is developed while considering openAI Gym interface requirements. Additionally, Pandapower toolbox is used to run power flow calculations and ensure our

constraints are not violated. Finally, Stable baselines is used to implement the DQN agent with extensions to solve the set-up environment.

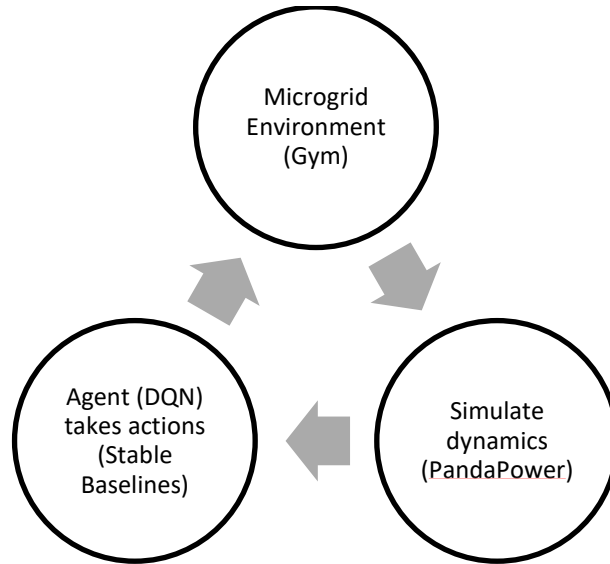


Figure 4.21 Framework to solve microgrid energy management

The agent performance in taking optimal actions depends on the hyperparameters chosen during the training stage. In the next chapter, different experiments are conducted to demonstrate the hyperparameter tuning process of the agent.

CHAPTER 5

RESULTS & DISCUSSION

This chapter provides details and results of the hyperparameters tuning of proposed DQN model which is used to solve the energy management problem. Such parameters include the learning rate, the discount factor, the size of hidden layers, the number of hidden layers, and others. To train the developed agent to solve the energy management problem, training data consisting of historical data are provided. The performance of the agent is then evaluated using a testing environment that uses a separate set of historical data. Results of energy scheduling simulations are provided and benchmarked with the optimal solution that has full knowledge of future variables.

5.1 The Microgrid Environment

The proposed methodology is conducted on the CIGRE MV microgrid shown in Figure 5.1, which contains various types of distributed energy resources, as well as different types of loads. The microgrid interfaces with an external grid at 110kV voltage level and is operating at 20kV level. The different microgrid parameters for the lines, buses, switches, transformers, generators, and loads are provided in section 7.1. Overall, the microgrid contains 18 loads, 8 PV plants, 1 wind turbine, 4 diesel generators, and 2 batteries.

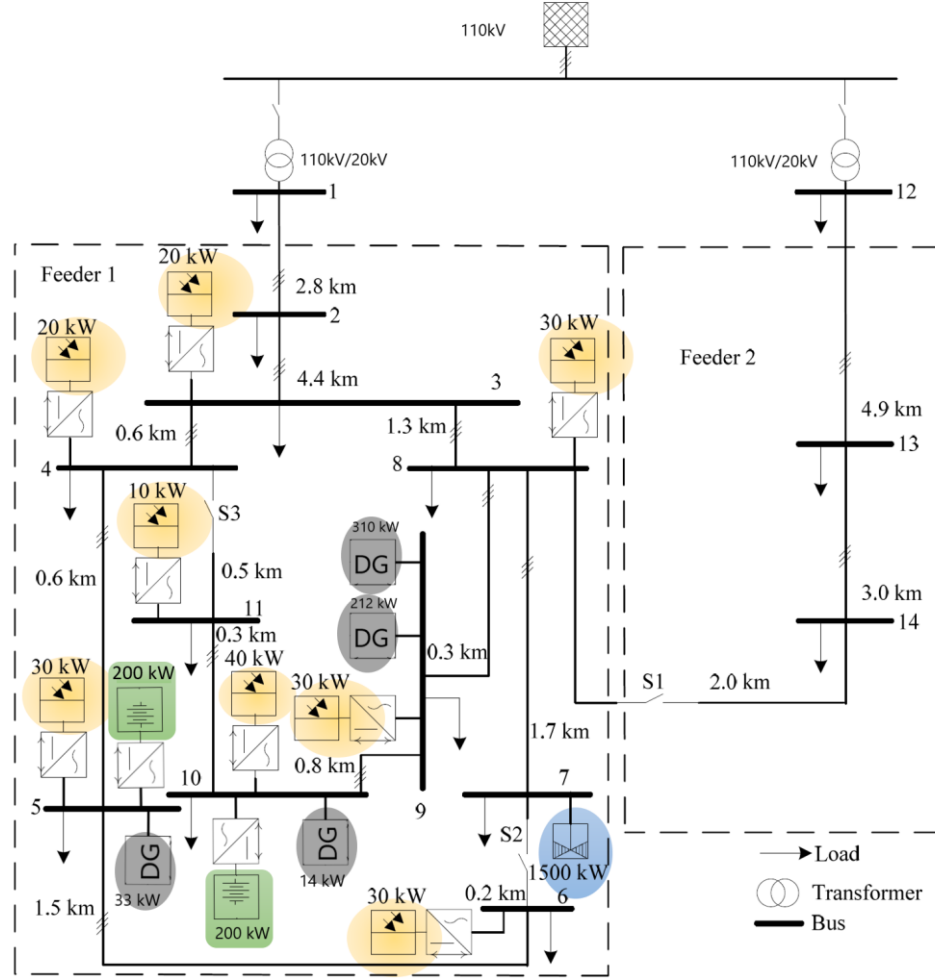


Figure 5.1 CIGRE MV Microgrid with DER

A training Q-network is constructed with a feedforward NN that is used to map the different states at any given time to their corresponding state-action value function for each action a_t^k . The network takes an input of the past 24 hours values of the different variables (PV, wind, grid prices, loads), and outputs the Q-Value for each possible action (charging/discharging batteries, running/stopping DG) at current state. Figure 5.2 illustrates the interaction of the training Q network with the environment, and the target Q network in double DQN configuration. The training Q network is MLP NN with 678 inputs corresponding to 24 hours past values of each load, DER as well as electric prices and

batteries SOC. The output consists of 144 units corresponding to the possible state-action Q-value.

The target network is constructed with a similar structure to the training Q-network and is updated frequently. A replay memory of size 50,000 is constructed to store (s, a, r, s') transition at each step. Initially, the agent is allowed to take random actions for the first 1000 steps before the learning start, as the model still lacks experience. The performance of the training network is logged, with parameters being save periodically. The parameters of the best performing network are saved to be used for testing.

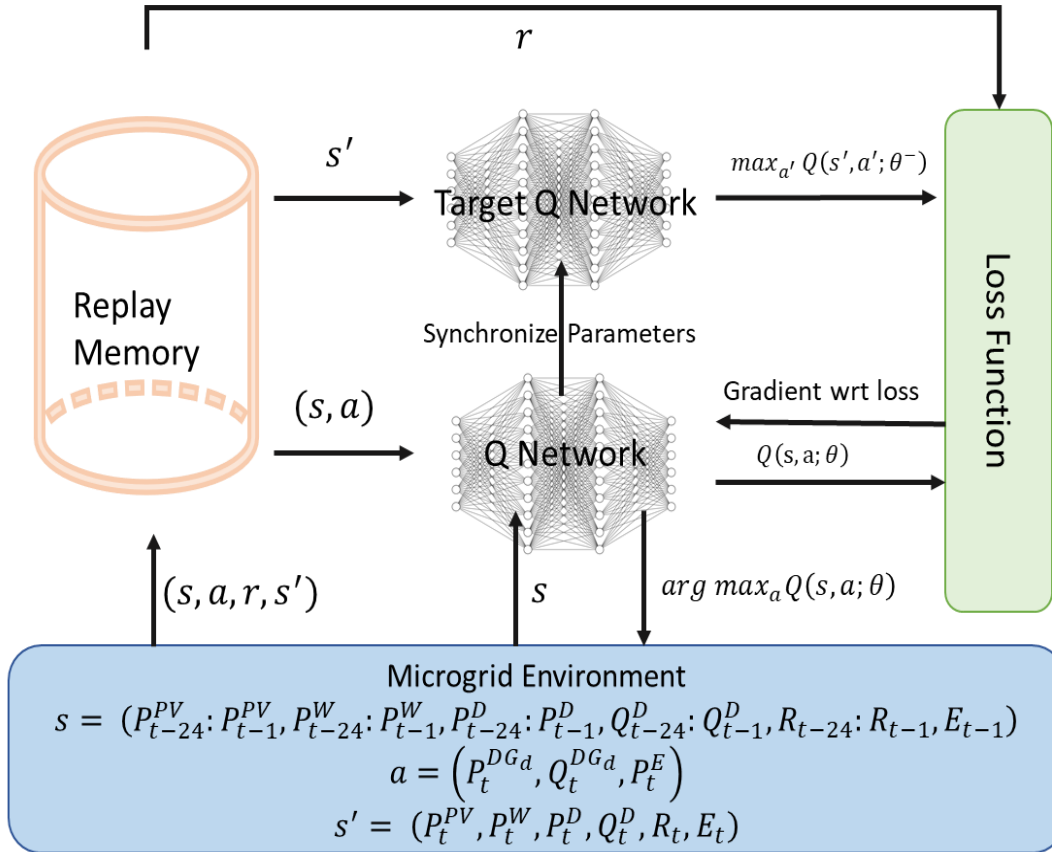


Figure 5.2 DQN training process with microgrid environment

5.2 Computation Time

Several experiments are conducted to fine tune the NN different hyperparameters. Since training computation can take long hours (it takes around 24 hours to run the training of 1 million steps with I7-8700K CPU @ 3.7GHz with 16 GB RAM), the various parameters are tuned manually, guided by intellectual intuition and literature findings. This shall not be an obstacle in real deployments, since advances in GPU and cloud computing can enhance this training time significantly, given enough hardware resources are allocated.

Additionally, the data is divided in two sets: training (75%) and testing (25%). The agent is allowed to train in the developed framework for 500K time steps in each network configuration. The results are then benchmarked to those obtained in the optimal case. While running the simulation on the test data, each time step takes around 0.01 seconds. Overall, it takes around 22 seconds for the 2160 hours simulation steps. This demonstrates the ability of such method to conduct real time scheduling of energy resources.

5.3 Hyperparameter Tuning

To find a suitable NN structure, simulations are conducted for different layers and number of nodes per hidden layers. It is observed that having two layers with at least 256 is efficient to reach acceptable reward levels. Figure 5.3 and Figure 5.4 show the learning curve for various number of NNs with two & three hidden layers. Using 3 layers is observed to have no advantage over 2 layers, and it might occasionally perform worse than two hidden layers. A larger network will result in less weight sharing, giving more learning stability.

However, a larger network could result in more weights that needs to be changed, leading to deteriorating the policy performance.

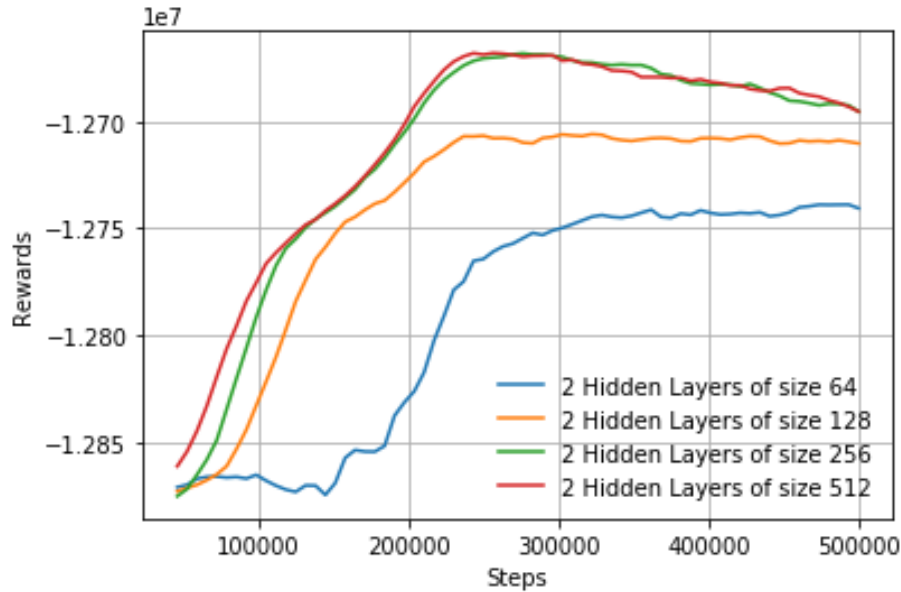


Figure 5.3 Learning curve of NN with 2 hidden layers

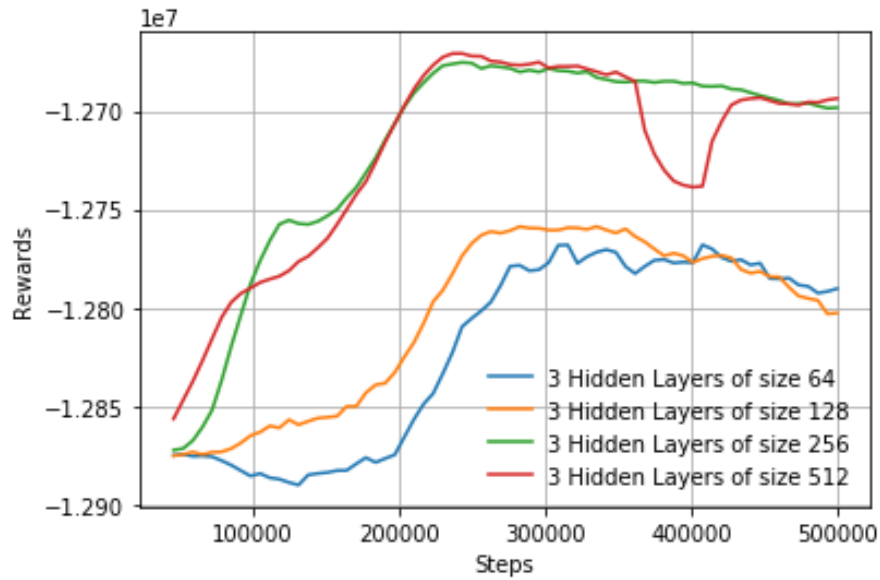


Figure 5.4 Learning curve of NN with 3 hidden layers

Additionally, the use of different learning rates in the training performance is investigated.

Using a learning rate between 0.001 and 0.0001 is observed to achieve best results. Figure

5.5 illustrate the training performance for different learning rates. Having larger learning rates causes a severe deuteriation of the policy since it clearly fails to learn better policies.

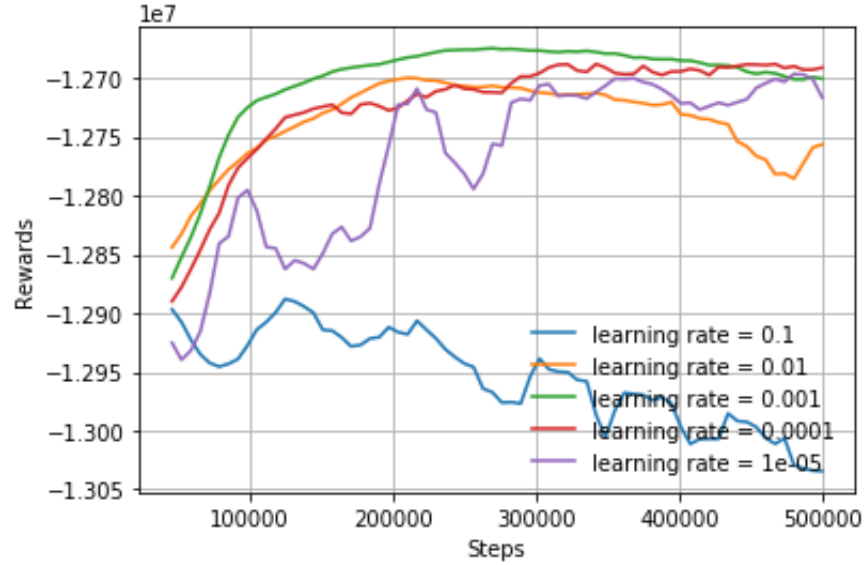


Figure 5.5 Learning rate effect on training performance

Moreover, the effect of the discount factor (γ) used in updating the target network value is analyzed. The discount factor has a direct effect on the objective of the agent performance. Since a main part of the reward function depends on the stochastic electricity prices, which makes predicting the future harder. Hence, lowering discount factor is more beneficial in optimizing the reward accumulation. Figure 5.6 shows the learning curves for different agents with different discount factors. Having gamma values below 0.5 performed much better than higher values. This means that current actions only affect shorter time periods in the future.

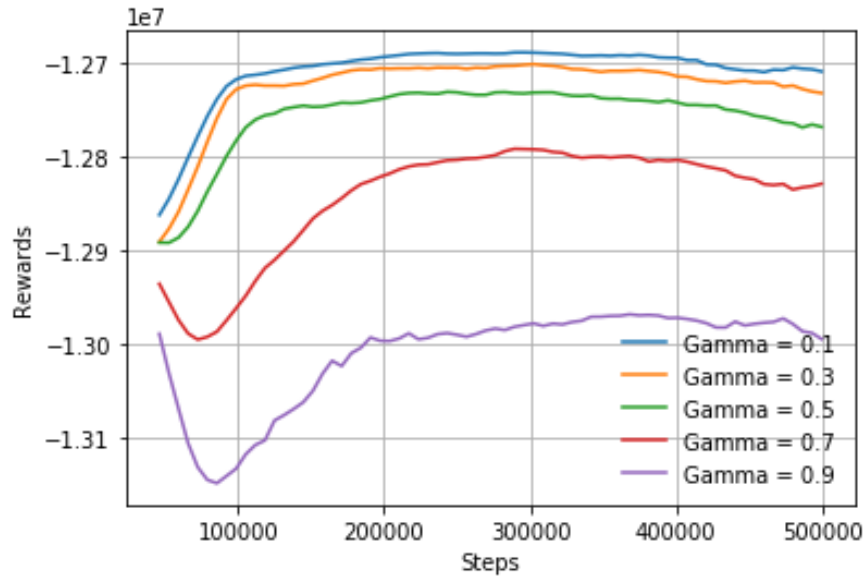


Figure 5.6 Discount factor effect on training performance

The size of batches used to apply gradient descent and update the training network weights is investigated. Figure 5.7 shows the learning curves for different batch sizes. Overall, it is observed that larger the batch size the better learning performance, though batch sizes larger than 256 had no significant improvements.

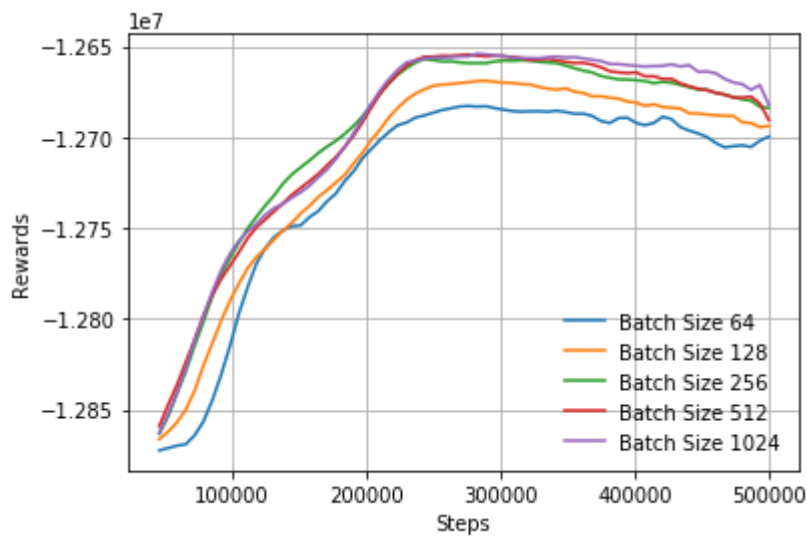


Figure 5.7 Learning curve for different batch sizes

Also, the effect of using different exploration fractions during training on the performance of the agent is studied. Overall, it is observed that higher exploration fractions led to slower learning in the initial stages, but later arrived at comparable values similar to lower exploration fraction values. An exploration fraction of 0.4 led to best performance amongst tested values.

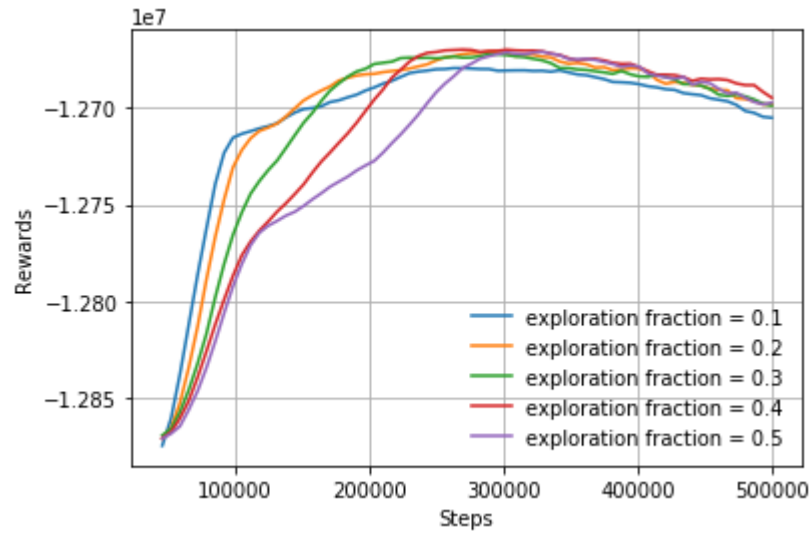


Figure 5.8 Learning curve for different fractions of exploration

Furthermore, the target network is updated frequently, to match the training network. Different updating frequencies is investigated as shown in Figure 5.9. It is observed that the update frequency does not critically affect the performance of our agent.

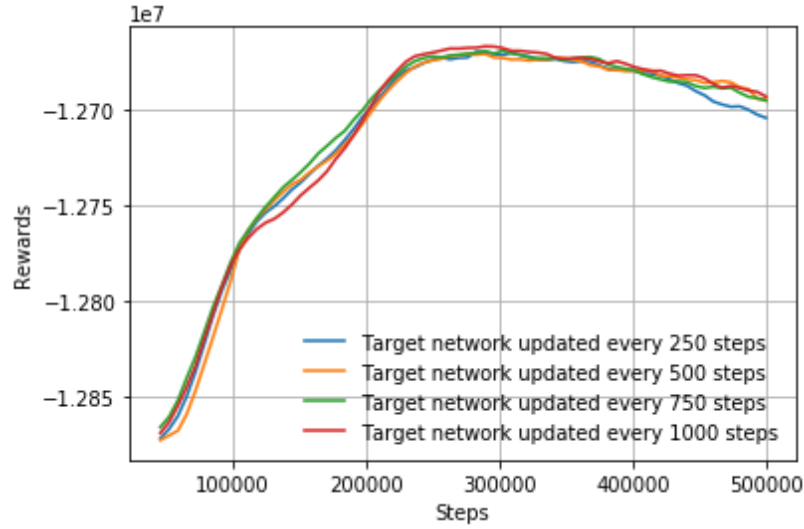


Figure 5.9 Learning performance for different target network update rates

5.4 Energy Schedules

In order to examine the performance of the developed agent, scheduling is simulated for the energy resources in the microgrid. The simulation is done in a test environment that uses historical data for 3 months. Figure 5.10 shows total charging/discharging schedules and SOC for the energy storage devices. The agent is allowed to charge/discharge the energy storage devices, as long as it doesn't violate specified dynamic constraints of the energy storage. The agent tries to maximize the benefits of controlling the energy storage by accounting for different variables including external grid electric prices, and available resources withing the microgrid. Figure 5.11 shows the generation schedules for the diesel generators and energy storage devices, and external grids. Similar to energy storages, the agent tries to optimize the operation of the DG units taking into account the costs of operating the units, as well as other state variables.

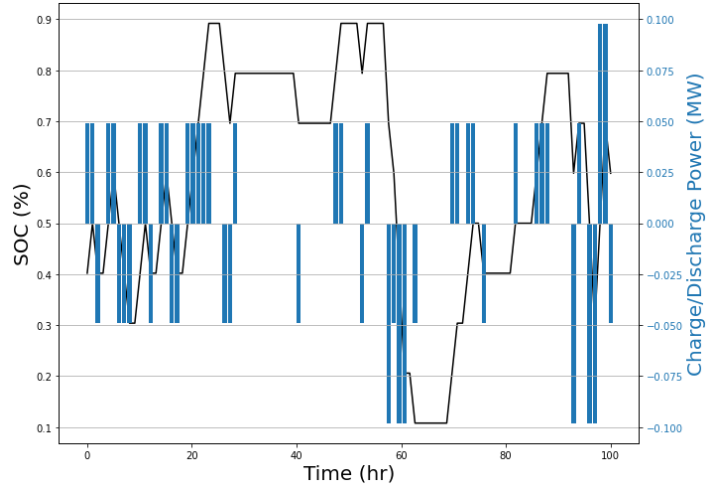


Figure 5.10 Energy storage hourly schedule with SOC

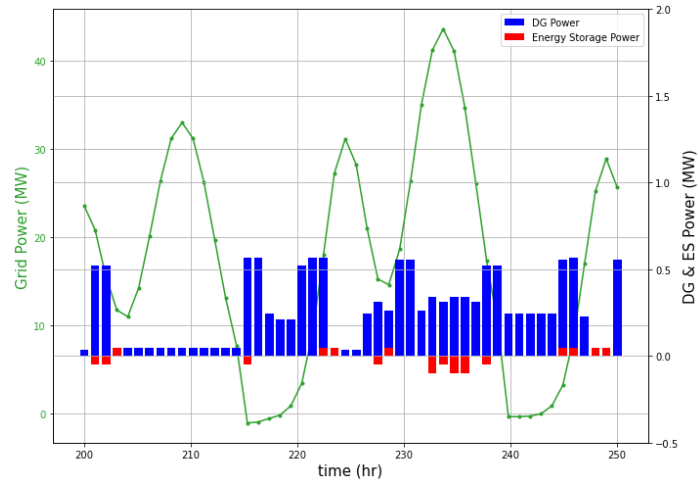


Figure 5.11 DG and energy storage hourly schedules with grid power exchange

To benchmark the performance of the developed agent, the daily costs incurred by following the agent actions are compared with the optimal solution. Figure 5.13 shows the daily incurred costs for both solutions during the test period. The adjacent zoomed in figure shows the variations of the daily costs. To clearly show the variation between the two solutions, Figure 5.14 shows the difference in daily costs between the optimal and DQN solution.

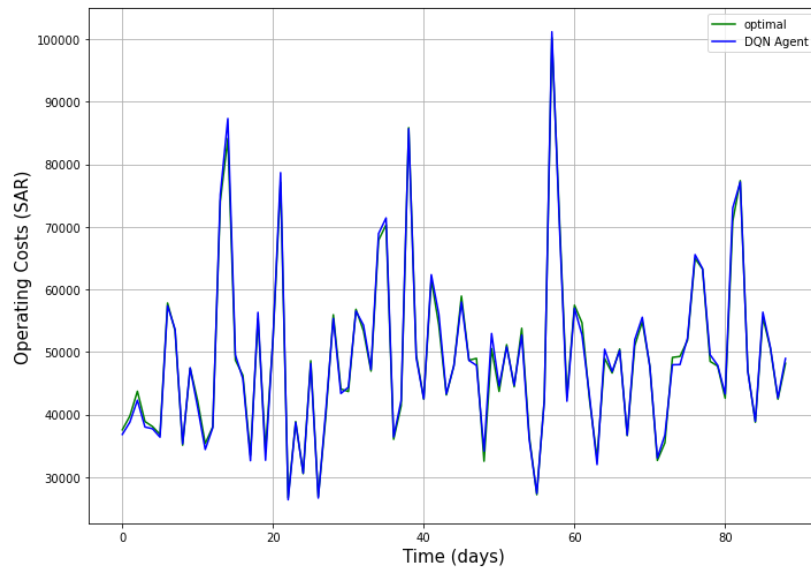


Figure 5.12 Benchmarking DQN agent daily operating costs with optimal solution

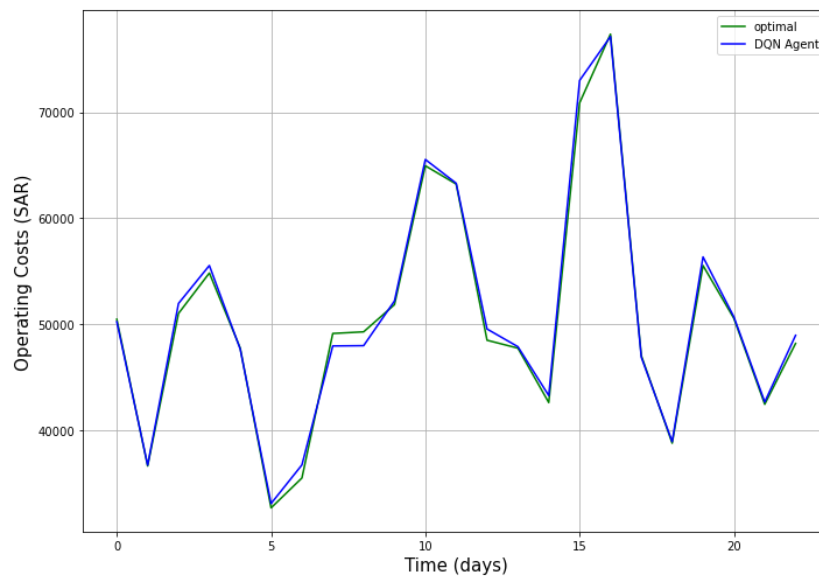


Figure 5.13 Segment of the daily operating costs

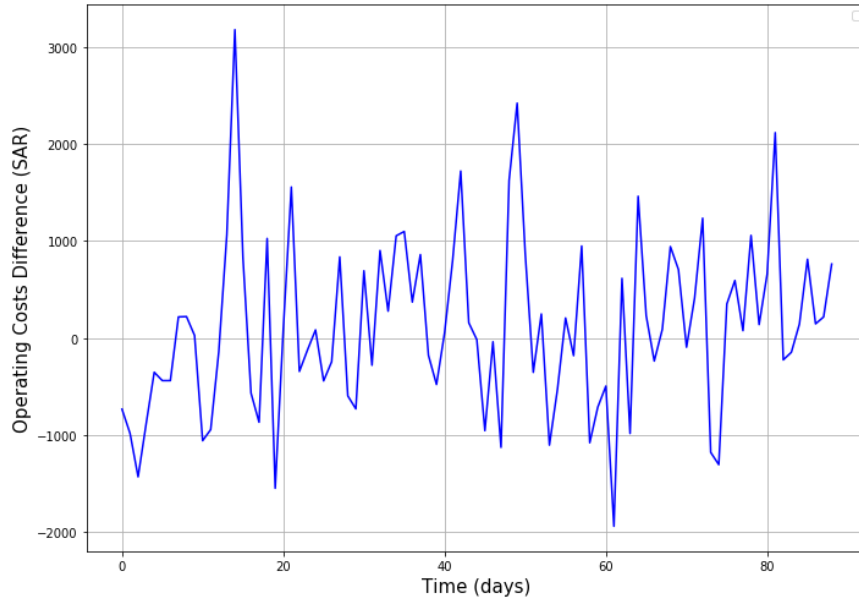


Figure 5.14 Daily costs difference between optimal solution and DQN solution

To illustrate the performance of developed DQN, the accumulated costs along the test period in are plotted as shown Figure 5.15. Two DQN agents are provided next to the optimal solution, one with untuned hyperparameters, and another with tuned hyperparameters.

Overall, DQN showed its capability to conduct energy management, with results that are comparable to those obtained in the optimal case.

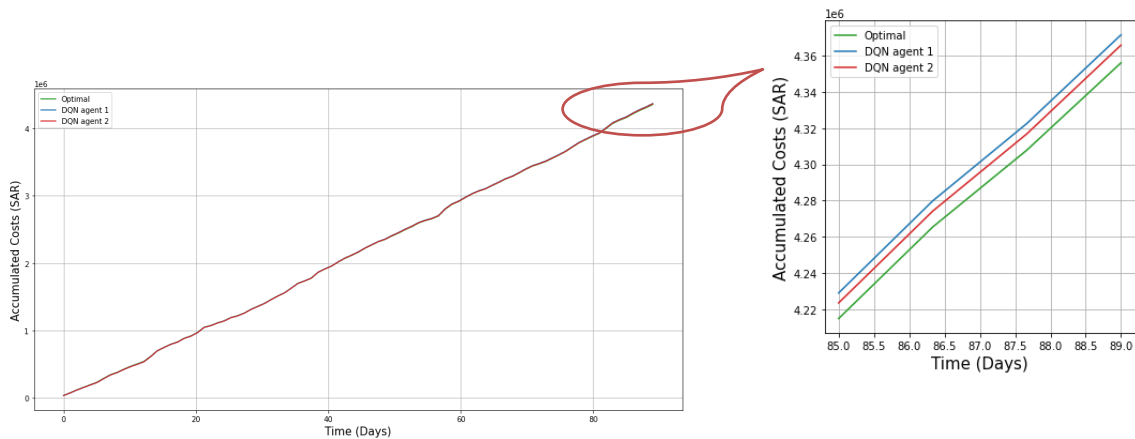


Figure 5.15 Accumulated operating costs for the different solvers

CHAPTER 6

CONCLUSION & FUTURE WORK

6.1 Conclusion

This thesis proposed the use of deep reinforcement learning methodology based on deep Q network algorithm to solve the energy problem formulation of a given microgrid. The methodology considered the stochastic behavior of different elements of a microgrid, including loads, generations, and electric prices. It also modeled different grid elements using equivalent models and considered the various power flow constraints in a realistic setting. The problem is implemented in an MDP formulation and solved utilizing state of art algorithm based on DQN with further enhancements. The thesis also studied the effect of different hyperparameter on the performance of the proposed algorithm. The methodology is implemented in OpenAI Gym framework, with the possibility to implement different tools available in Stable Baseline to devise different solvers. Pandapower is also integrated in the framework to model the electric elements, and conduct power flow calculations at each time step. The developed framework can be adjusted for different microgrid architectures providing flexibility to test the set-up for different types of electrical grids.

In conclusion, the thesis demonstrated that the deep reinforcement learning methodologies as implemented in the thesis can obtain near optimal results. The methodology can conduct online scheduling of the various energy resources within a grid and make cost effective

actions under stochastic conditions. The results were benchmarked with the optimal results obtained by MILP solver that had full knowledge of the different stochastic variables. The methodology did not require a model of the uncertainties, rather it only interacted with the environment to devise the best policy to manage the microgrid resources.

Such methodology can form the basis of a generalized framework of microgrid energy management, which can be used to devise real time schedules of the different energy resources. It can also be used to conduct planning analysis of the use of distributed energy resources and to evaluate their associated costs along a given time horizon.

6.2 Recommended Future Work

Further enhancements can be implemented to the proposed methodology. For example, further work can investigate the use of alternative networks to map states to q-values using convolutional or recurrent neural networks (e.g. LSTM). These types of network proved their effectiveness in other setups and might as well suit the energy management problem. Additionally, other studies can investigate the use of policy optimization methods as opposed to value-based optimization which includes DQN. These types of methods directly optimize the network parameters to learn an optimal policy, and hence provide more stable learning (at the cost of sample inefficiency).

Other types of studies could look at defining the energy management problem from the perspective of the owner and apply DRL for each participant in the microgrid. Moreover, DRL could be applied at multiple microgrids from the perspective of the system operator. Such architecture could make use of cooperating microgrids, to allow optimal use of resources.

The advances of deep learning and reinforcement learning has made it possible to change the norms in operating the grids. Such promising success could lead to further adaptation of AI technologies in traditional fields like power systems.

Appendix

7.1 Case Study MG Network Parameters

Table 7.1 Microgrid Line Parameters

Line #	Name	from bus	To bus	length (km)	R (ohm/km)	X (ohm/km)	max current (kA)
0	Line 1-2	1	2	2.82	0.501	0.716	0.145
1	Line 2-3	2	3	4.42	0.501	0.716	0.145
2	Line 3-4	3	4	0.61	0.501	0.716	0.145
3	Line 4-5	4	5	0.56	0.501	0.716	0.145
4	Line 5-6	5	6	1.54	0.501	0.716	0.145
5	Line 7-8	7	8	1.67	0.501	0.716	0.145
6	Line 8-9	8	9	0.32	0.501	0.716	0.145
7	Line 9-10	9	10	0.77	0.501	0.716	0.145
8	Line 10-11	10	11	0.33	0.501	0.716	0.145
9	Line 3-8	3	8	1.3	0.501	0.716	0.145
10	Line 12-13	12	13	4.89	0.51	0.366	0.195
11	Line 13-14	13	14	2.99	0.51	0.366	0.195
12	Line 6-7	6	7	0.24	0.501	0.716	0.145
13	Line 11-4	11	4	0.49	0.501	0.716	0.145
14	Line 14-8	14	8	2	0.51	0.366	0.195

Table 7.2 Transformers Information

XFR Number	Name	HV Bus	LV Bus	MVA Rating	Primary Voltage (kV)	Secondary Voltage (kV)
0	Trafo 0-1	0	1	25	110	20
1	Trafo 0-12	0	12	25	110	20

Table 7.3 Load nominal values

Load #	Name	Bus	P (MW)	Q (MVAR)
0	Load R1	1	14.994	3.044662
1	Load R3	3	0.27645	0.069285
2	Load R4	4	0.43165	0.108182

3	Load R5	5	0.7275	0.182329
4	Load R6	6	0.54805	0.137354
5	Load R8	8	0.58685	0.147078
6	Load R10	10	0.4753	0.119121
7	Load R11	11	0.3298	0.082656
8	Load R12	12	14.994	3.044662
9	Load R14	14	0.20855	0.052268
10	Load CI1	1	4.845	1.592474
11	Load CI3	3	0.22525	0.139597
12	Load CI7	7	0.0765	0.04741
13	Load CI9	9	0.57375	0.355578
14	Load CI10	10	0.068	0.042143
15	Load CI12	12	5.016	1.648679
16	Load CI13	13	0.034	0.021071
17	Load CI14	14	0.3315	0.205445

Table 7.4 Switches Information

Switch #	Bus	Element	Close Statue	Name
0	6	12	TRUE	None
1	7	12	FALSE	S2
2	4	13	FALSE	S3
3	11	13	TRUE	None
4	8	14	FALSE	S1
5	14	14	TRUE	None
6	0	0	TRUE	None
7	0	1	TRUE	None

Table 7.5 DER information

DER #	Name	Bus	Capacity P (MW)	Q (MVAR)
0	PV 1	3	0.02	0
1	PV 2	4	0.02	0
2	PV 3	5	0.03	0
3	PV 4	6	0.03	0
4	PV 5	8	0.03	0
5	PV 6	9	0.03	0
6	PV 7	10	0.04	0
7	PV 8	11	0.01	0
8	WT 1	7	1.5	0
9	DG 1	5	0.033	0
10	DG 2	9	0.31	0
11	DG 3	9	0.212	0
12	DG 4	10	0.014	0

Table 7.6 Energy Storage Devices Information

#	Name	Bus	Capacity (MWH)	Efficiency	Max SOC	Min SOC	Max Charging (MW)	Max Discharging (MW)
0	Battery 1	5	0.25	0.98	0.9	0.1	-0.2	0.2
1	Battery 2	10	0.25	0.98	0.9	0.1	-0.2	0.2

7.2 Codes

7.2.1 Microgrid Environment

```
#importing required libraries
import numpy as np
import pandas as pd
import gym
from gym import spaces, logger
from gym.utils import seeding
import pandapower as pp
from pandapower import ppException
import pandapower.networks as pn
from numpy import genfromtxt
from pandapower.plotting.plotly import pf_res_plotly
from pandapower.plotting.plotly import simple_plotly
from sklearn import preprocessing

class microgrid(gym.Env):
    metadata = {'render.modes': ['human']}

    def __init__(self):
        super(microgrid, self).__init__()

        self.days = 275
        self.duration = self.days*24
        self.look_back = 24

        #loading historical data
        #look up numpy reading functions
        self.df = pd.read_excel("data/spotmarket_data_2007-2013.xls")
        self.grid_prices = self.df.to_numpy()/10
        self.grid_price = self.grid_prices.flatten()[0:1*self.duration]
        min_max_scaler= preprocessing.MinMaxScaler()
        n = self.grid_price.reshape(-1, 1)
```

```

n()
    self.grid_price_normalized = min_max_scaler.fit_transform(n).flatten()
    self.load_normalized = np.load("data/Load.npy")[0:1*self.duration]
    self.pv_normalized = np.load("data/PV.npy")[0:1*self.duration]
    self.wp_df = pd.read_excel("data/wp.xls")
    self.wp_normalized = self.wp_df.to_numpy()[0:1*self.duration].flatten()
ten()

```

```

# create network
self.net = self.create_net()
self.net_f = self.net

self.battery_power = 0.05 #0.05MW
self.battery_capacity = 0.25 #0.25MWh
self.efficiency_ratio = 0.98
self.soc_start = 0.5
self.soc_max = 1
self.soc_min = 0

```

```

variables_number = self.look_back*4 + 6

```

```

high = np.ones(variables_number)
low = np.zeros(variables_number)
self.observation_space = spaces.Box(low, high, dtype=np.float32)

```

```

self.action_space = spaces.Discrete(144)
# get mapping to be used to map action to multi action
self.mapping_array = genfromtxt('array.csv', delimiter=',')

```

```

self.grid_exchange_cost = None
self.grid_power_state = None

```

```

self.state = None

```

```

# pv_initial = np.outer(self.net.sgen.p_mw.loc[self.net.sgen.type =
= 'PV'], self.pv_normalized[0:self.look_back]).flatten()
pv_initial = self.pv_normalized[0:self.look_back]
load_initial = self.load_normalized[0:self.look_back]
wind_initial = self.wp_normalized[0:self.look_back]

```

```

        soc_initial = np.ones(self.net.storage.p_mw.shape[0]) * self.soc_
start
        diesel_initial = np.ones(self.net.sgen.type.loc[self.net.sgen.type
== 'DG'].shape[0]) * 0
        price_initial = self.grid_price_normalized[0:self.look_back]
        self.state = np.concatenate((soc_initial, diesel_initial, price_init
ial, wind_initial, pv_initial, load_initial), axis = 0)

        self.counter = 1 + self.look_back
        self.grid_exchange_cost = 0

        return np.array(self.state).astype(np.float32)

def _get_obs(self):
    return np.copy(self.state)

def step(self, action):
    err_msg = "%r (%s) invalid" % (action, type(action))
    assert self.action_space.contains(action), err_msg

    soc_state = self.state[0:2] #* self.net.storage.p_mw
    dg_state = self.state[2:6]
    #dont make use immediately
    grid_price_state = self.state[6:6+self.look_back]* self.price_max
    wt_state = self.state[6+self.look_back:6+2*self.look_back]
    pv_state = self.state[6+2*self.look_back:6+3*self.look_back]
    load_state = self.state[6+3*self.look_back:6+4*self.look_back]

    reward = 0
    mapped_action = self.map_action(action)
    bat_action = mapped_action[0:2]
    dg_action = mapped_action[2:6]
    charging_power = bat_action * self.efficiency_ratio * self.battery_
power
    next_soc = soc_state + charging_power/self.battery_capacity
    #check next soe is within limits
    for i in range(len(next_soc)):
        if next_soc[i] >= self.soc_min and next_soc[i] <= self.soc_max
:
            soc_state[i] = next_soc[i]
            #reward += -1*charging_power*self.grid_price[self.counter]
        else:
            charging_power[i] = 0
            reward -= 100

```

```

        #self.net.load.p_mw= self.load_normalized[self.counter] * self.net
        _f.load.p_mw
        #self.net.load.q_mvar = self.load_normalized[self.counter] * self.
        net_f.load.q_mvar
        #self.net.sgen.p_mw.loc[self.net.sgen.type == 'PV'] = self.pv_norm
        alized[self.counter] * self.net_f.sgen.p_mw.loc[self.net_f.sgen.type == 'P
        V']

        self.net.load.scaling = self.load_normalized[self.counter]
        self.net.sgen.scaling.loc[self.net.sgen.type == 'PV'] = self.pv_no
        rmalized[self.counter]
        self.net.sgen.scaling.loc[self.net.sgen.type == 'WP'] = self.wp_no
        rmalized[self.counter]
        #self.net.sgen.p_mw.loc[self.net.sgen.type == 'WP'] = self.wp_norm
        alized[self.counter] * self.net_f.sgen.p_mw.loc[self.net_f.sgen.type == 'W
        P']

        #self.net.sgen.p_mw.loc[self.net.sgen.type == 'DG'] = dg_power
        self.net.sgen.scaling.loc[self.net.sgen.type == 'DG'] = dg_action
        self.net.storage.p_mw = charging_power

    #
    try:
        pp.runpp(self.net)
        power_flow_flag = False
    except ppException:
        power_flow_flag = True

    if power_flow_flag:
        done = True
        reward -= 1*100
    else:
        done = bool(self.counter>=self.duration-1)

    dg_power = self.net.res_sgen.p_mw[9:]
    #np.array(dg_action * self.net_f.sgen.p_mw.loc[self.net_f.sgen.typ
    e == 'DG'])
    #for i in range(len(dg_power)):
    reward -= self.thermal_power_cost(np.sum(dg_power))

    #self.grid_power_state = self.net.res_ext_grid['p_mw']

    dg_state = dg_action

```

```

        grid_price_state = self.grid_price_normalized[self.counter-
self.look_back:self.counter]
        wt_state = self.wp_normalized[self.counter-
self.look_back:self.counter]
        pv_state = self.pv_normalized[self.counter-
self.look_back:self.counter]
        load_state = self.load_normalized[self.counter-
self.look_back:self.counter]
        self.state = np.concatenate((soc_state,dg_state,grid_price_state,w
t_state,pv_state,load_state), axis = 0)

        grid_exchange = self.net.res_ext_grid.p_mw[0]# + self.net.res_ext_
grid.q_mvar
        self.grid_exchange_cost = grid_exchange*self.grid_price[self.count
er]
        if not done:
            reward += -1*grid_exchange*self.grid_price[self.counter]
#            reward += -
1*grid_exchange*self.grid_price_normalized[self.counter]

        ## monitor
        ...
        print('load mw')
        print(np.sum(self.net.load.p_mw))
        print('generation mw')
        print(np.sum(self.net.sgen.p_mw))
        print('bat')
        print(np.sum(self.net.storage.p_mw))
        print('sum')
        print(np.sum(self.net.load.p_mw)-
np.sum(self.net.sgen.p_mw)+np.sum(self.net.storage.p_mw))
        print('grid mw')
        print(np.sum(self.net.res_ext_grid.p_mw))
        print('mvar')
        print(np.sum(self.net.load.q_mvar)-
np.sum(self.net.sgen.q_mvar)+np.sum(self.net.storage.q_mvar))
        print(np.sum(self.net.res_ext_grid.q_mvar))
        ...

        self.counter+=1

```

```

        return np.array(self.state).astype(np.float32), reward, done, {}#n
p.array([self.grid_exchange_cost])

def render(self):
    #pf_res_plotly(self.net)
    print(self.net.res_storage)
    print(self.net.res_sgen)
    print(self.net.res_load)
    print('')
    print('')

    print('sum')
    print(np.sum(self.net.res_load.p_mw)-
np.sum(self.net.res_sgen.p_mw)+np.sum(self.net.res_storage.p_mw))
    print('grid mw')
    print(np.sum(self.net.res_ext_grid.p_mw))
    print('mvar sum')
    print(np.sum(self.net.res_load.q_mvar))
    print('grid mvar')
    print(np.sum(self.net.res_ext_grid.q_mvar))

    print('grid exchange cost')
    print(self.grid_exchange_cost)

    print('grid exchange cost')
    print(self.grid_exchange_cost)

def close(self):
    pass

def create_net(self):
    net = pn.create_cigre_network_mv(with_der="all")
    #create standard CIGRE MV network
    for i in range(4):
        net.sgen.name[i+9] = 'DG {}'.format(i+1)
        net.sgen.type[i+9] = 'DG'
    for i in range(8):
        net.sgen.name[i] = 'PV {}'.format(i+1)
        net.sgen.type[i] = 'PV'
    net.sgen.name[8] = 'WT 1'
    return net

def thermal_power_cost(self,power):
    A = 0.1
    B = 71.6

```

```

        C = 46.15
        cost = A*power**2+B*power + C
        return cost
    def map_action(self, action):
        #mapping one dimension action space, to multi dimension action space
        mapped = self.mapping_array[action,:]
        return mapped

from pandas import read_json
from numpy import nan
import pandapower as pp
"""
Create the CIGRE MV microgrid
"""

def create_cigre_network_mv(with_der=True):

    net_cigre_mv = pp.create_empty_network()

    # Linedata
    line_data = {'c_nf_per_km': 151.1749, 'r_ohm_per_km': 0.501,
                 'x_ohm_per_km': 0.716, 'max_i_ka': 0.145,
                 'type': 'cs'}
    pp.create_std_type(net_cigre_mv, line_data, name='CABLE_CIGRE_MV', element='line')

    line_data = {'c_nf_per_km': 10.09679, 'r_ohm_per_km': 0.510,
                 'x_ohm_per_km': 0.366, 'max_i_ka': 0.195,
                 'type': 'ol'}
    pp.create_std_type(net_cigre_mv, line_data, name='OHL_CIGRE_MV', element='line')

    # Busses
    bus0 = pp.create_bus(net_cigre_mv, name='Bus 0', vn_kv=110, type='b',
                        zone='CIGRE_MV')
    buses = pp.create_buses(net_cigre_mv, 14, name=['Bus %i' % i for i in
                        range(1, 15)], vn_kv=20,
                        type='b', zone='CIGRE_MV')

    # Lines
    pp.create_line(net_cigre_mv, buses[0], buses[1], length_km=2.82,
                  std_type='CABLE_CIGRE_MV', name='Line 1-2')

```

```

pp.create_line(net_cigre_mv, buses[1], buses[2], length_km=4.42,
               std_type='CABLE_CIGRE_MV', name='Line 2-3')
pp.create_line(net_cigre_mv, buses[2], buses[3], length_km=0.61,
               std_type='CABLE_CIGRE_MV', name='Line 3-4')
pp.create_line(net_cigre_mv, buses[3], buses[4], length_km=0.56,
               std_type='CABLE_CIGRE_MV', name='Line 4-5')
pp.create_line(net_cigre_mv, buses[4], buses[5], length_km=1.54,
               std_type='CABLE_CIGRE_MV', name='Line 5-6')
pp.create_line(net_cigre_mv, buses[6], buses[7], length_km=1.67,
               std_type='CABLE_CIGRE_MV', name='Line 7-8')
pp.create_line(net_cigre_mv, buses[7], buses[8], length_km=0.32,
               std_type='CABLE_CIGRE_MV', name='Line 8-9')
pp.create_line(net_cigre_mv, buses[8], buses[9], length_km=0.77,
               std_type='CABLE_CIGRE_MV', name='Line 9-10')
pp.create_line(net_cigre_mv, buses[9], buses[10], length_km=0.33,
               std_type='CABLE_CIGRE_MV', name='Line 10-11')
pp.create_line(net_cigre_mv, buses[2], buses[7], length_km=1.3,
               std_type='CABLE_CIGRE_MV', name='Line 3-8')
pp.create_line(net_cigre_mv, buses[11], buses[12], length_km=4.89,
               std_type='OHL_CIGRE_MV', name='Line 12-13')
pp.create_line(net_cigre_mv, buses[12], buses[13], length_km=2.99,
               std_type='OHL_CIGRE_MV', name='Line 13-14')

line6_7 = pp.create_line(net_cigre_mv, buses[5], buses[6], length_km=0
.24,
                        std_type='CABLE_CIGRE_MV', name='Line 6-7')
line4_11 = pp.create_line(net_cigre_mv, buses[10], buses[3], length_km
=0.49,
                        std_type='CABLE_CIGRE_MV', name='Line 11-4')
line8_14 = pp.create_line(net_cigre_mv, buses[13], buses[7], length_km
=2.,
                        std_type='OHL_CIGRE_MV', name='Line 14-8')

# Ext-Grid
pp.create_ext_grid(net_cigre_mv, bus0, vm_pu=1.03, va_degree=0.,
                  s_sc_max_mva=5000, s_sc_min_mva=5000, rx_max=0.1, r
x_min=0.1)

# Trafos
trafo0 = pp.create_transformer_from_parameters(net_cigre_mv, bus0, bus
es[0], sn_mva=25,
                                              vn_hv_kv=110, vn_lv_kv=
20, vkr_percent=0.16,
                                              vk_percent=12.00107, pf
e_kw=0, i0_percent=0,

```

```

shift_degree=30.0, name
='Trafo 0-1')
    trafo1 = pp.create_transformer_from_parameters(net_cigre_mv, bus0, bus
es[11], sn_mva=25,
vn_hv_kv=110, vn_lv_kv=
20, vkr_percent=0.16,
vk_percent=12.00107, pf
e_kw=0, i0_percent=0,
shift_degree=30.0, name
='Trafo 0-12')

    # Switches
    # S2
    pp.create_switch(net_cigre_mv, buses[5], line6_7, et='1', closed=True,
type='LBS')
    pp.create_switch(net_cigre_mv, buses[6], line6_7, et='1', closed=False
, type='LBS', name='S2')
    # S3
    pp.create_switch(net_cigre_mv, buses[3], line4_11, et='1', closed=False
e, type='LBS', name='S3')
    pp.create_switch(net_cigre_mv, buses[10], line4_11, et='1', closed=True
e, type='LBS')
    # S1
    pp.create_switch(net_cigre_mv, buses[7], line8_14, et='1', closed=False
e, type='LBS', name='S1')
    pp.create_switch(net_cigre_mv, buses[13], line8_14, et='1', closed=True
e, type='LBS')
    # trafos
    pp.create_switch(net_cigre_mv, bus0, trafo0, et='t', closed=True, type
='CB')
    pp.create_switch(net_cigre_mv, bus0, trafo1, et='t', closed=True, type
='CB')

    # Loads
    # Residential
    pp.create_load_from_cosphi(net_cigre_mv, buses[0], 15.3, 0.98, "ind",
name='Load R1')
    pp.create_load_from_cosphi(net_cigre_mv, buses[2], 0.285, 0.97, "ind",
name='Load R3')
    pp.create_load_from_cosphi(net_cigre_mv, buses[3], 0.445, 0.97, "ind",
name='Load R4')
    pp.create_load_from_cosphi(net_cigre_mv, buses[4], 0.750, 0.97, "ind",
name='Load R5')
    pp.create_load_from_cosphi(net_cigre_mv, buses[5], 0.565, 0.97, "ind",
name='Load R6')

```

```

    pp.create_load_from_cosphi(net_cigre_mv, buses[7], 0.605, 0.97, "ind",
name='Load R8')
    pp.create_load_from_cosphi(net_cigre_mv, buses[9], 0.490, 0.97, "ind",
name='Load R10')
    pp.create_load_from_cosphi(net_cigre_mv, buses[10], 0.340, 0.97, "ind"
, name='Load R11')
    pp.create_load_from_cosphi(net_cigre_mv, buses[11], 15.3, 0.98, "ind",
name='Load R12')
    pp.create_load_from_cosphi(net_cigre_mv, buses[13], 0.215, 0.97, "ind"
, name='Load R14')

    # Commercial / Industrial
    pp.create_load_from_cosphi(net_cigre_mv, buses[0], 5.1, 0.95, "ind", n
ame='Load CI1')
    pp.create_load_from_cosphi(net_cigre_mv, buses[2], 0.265, 0.85, "ind",
name='Load CI3')
    pp.create_load_from_cosphi(net_cigre_mv, buses[6], 0.090, 0.85, "ind",
name='Load CI7')
    pp.create_load_from_cosphi(net_cigre_mv, buses[8], 0.675, 0.85, "ind",
name='Load CI9')
    pp.create_load_from_cosphi(net_cigre_mv, buses[9], 0.080, 0.85, "ind",
name='Load CI10')
    pp.create_load_from_cosphi(net_cigre_mv, buses[11], 5.28, 0.95, "ind",
name='Load CI12')
    pp.create_load_from_cosphi(net_cigre_mv, buses[12], 0.04, 0.85, "ind",
name='Load CI13')
    pp.create_load_from_cosphi(net_cigre_mv, buses[13], 0.390, 0.85, "ind"
, name='Load CI14')

    # distributed energy resources

    pp.create_sgen(net_cigre_mv, buses[2], 0.02, q_mvar=0, sn_mva=0.02, na
me='PV 3', type='PV')
    pp.create_sgen(net_cigre_mv, buses[3], 0.020, q_mvar=0, sn_mva=0.02, n
ame='PV 4', type='PV')
    pp.create_sgen(net_cigre_mv, buses[4], 0.030, q_mvar=0, sn_mva=0.03, n
ame='PV 5', type='PV')
    pp.create_sgen(net_cigre_mv, buses[5], 0.030, q_mvar=0, sn_mva=0.03, n
ame='PV 6', type='PV')
    pp.create_sgen(net_cigre_mv, buses[7], 0.030, q_mvar=0, sn_mva=0.03, n
ame='PV 8', type='PV')
    pp.create_sgen(net_cigre_mv, buses[8], 0.030, q_mvar=0, sn_mva=0.03, n
ame='PV 9', type='PV')
    pp.create_sgen(net_cigre_mv, buses[9], 0.040, q_mvar=0, sn_mva=0.04, n
ame='PV 10', type='PV')

```

```

    pp.create_sgen(net_cigre_mv, buses[10], 0.010, q_mvar=0, sn_mva=0.01,
name='PV 11', type='PV')
    pp.create_sgen(net_cigre_mv, buses[6], 1.5, q_mvar=0, sn_mva=1.5, name
='WT 7',
                    type='WP')

    pp.create_storage(net_cigre_mv, bus=buses[4], p_mw=0.6, max_e_mwh=nan,
sn_mva=0.2,
                    name='Battery 1', type='Battery', max_p_mw=0
.6, min_p_mw=-0.6)
    pp.create_sgen(net_cigre_mv, bus=buses[4], p_mw=0.033, sn_mva=0.033,
                    name='DG 1', type='CHP diesel')
    pp.create_sgen(net_cigre_mv, bus=buses[8], p_mw=0.310, sn_mva=0.31, na
me='DG 2',
                    type='CHP diesel')
    pp.create_sgen(net_cigre_mv, bus=buses[8], p_mw=0.212, sn_mva=0.212, n
ame='DG 3',
                    type='CHP diesel')
    pp.create_storage(net_cigre_mv, bus=buses[9], p_mw=0.200, max_e_mwh=na
n, sn_mva=0.2,
                    name='Battery 2', type='Battery', max_p_mw=0
.2, min_p_mw=-0.2)
    pp.create_sgen(net_cigre_mv, bus=buses[9], p_mw=0.014, sn_mva=.014,
                    name='DG 4', type='CHP diesel')

# Bus geo data
net_cigre_mv.bus_geodata = read_json(
    """"{"x":{"0":7.0,"1":4.0,"2":4.0,"3":4.0,"4":2.5,"5":1.0,"6":1.0,"
7":8.0,"8":8.0,"9":6.0,
        "10":4.0,"11":4.0,"12":10.0,"13":10.0,"14":10.0},"y":{"0":16,"1":1
5,"2":13,"3":11,"4":9,
        "5":7,"6":3,"7":3,"8":5,"9":5,"10":5,"11":7,"12":15,"13":11,"14":5
}}""")
# Match bus.index
net_cigre_mv.bus_geodata = net_cigre_mv.bus_geodata.loc[net_cigre_mv.b
us.index]
return net_cigre_mv

```

7.2.2 Agent Development

```

import stable_baselines
import os

```

```

import matplotlib.pyplot as plt
import time
from stable_baselines import DQN
from stable_baselines.common.cmd_util import make_vec_env
from stable_baselines.common.policies import MlpPolicy
from stable_baselines.deepq import LnMlpPolicy
from stable_baselines.common.evaluation import evaluate_policy
from stable_baselines.bench import Monitor
from stable_baselines.common.vec_env import DummyVecEnv
from stable_baselines.deepq.policies import FeedForwardPolicy
from stable_baselines.results_plotter import load_results, ts2xy
from stable_baselines.common.noise import AdaptiveParamNoiseSpec, NormalAc
tionNoise
from stable_baselines import results_plotter
from stable_baselines.common.callbacks import BaseCallback
import tensorflow as tf

```

```

class PlottingCallback(BaseCallback):
    """
    Callback for plotting the performance in realtime.
    """
    def __init__(self, verbose=1):
        super(PlottingCallback, self).__init__(verbose)
        self._plot = None

    def _on_step(self) -> bool:
        # get the monitor's data
        x, y = ts2xy(load_results(log_dir), 'timesteps')
        if self._plot is None: # make the plot
            plt.ion()
            fig = plt.figure(figsize=(6,3))
            ax = fig.add_subplot(111)
            line, = ax.plot(x, y)
            self._plot = (line, ax, fig)
            plt.show()
        else: # update and rescale the plot
            self._plot[0].set_data(x, y)
            self._plot[-2].relim()
            self._plot[-2].set_xlim([self.locals["total_timesteps"] * -
0.02,
                                self.locals["total_timesteps"] * 1.02])
            self._plot[-2].autoscale_view(True, True, True)
            self._plot[-1].canvas.draw()

```

```

def moving_average(values, window):
    """
    Smooth values by doing a moving average
    """
    weights = np.repeat(1.0, window) / window
    return np.convolve(values, weights, 'valid')

def plot_results(log_folder, title='Learning Curve'):
    """
    plot the results
    """
    x, y = ts2xy(load_results(log_folder), 'timesteps')
    y = moving_average(y, window=1)
    # Truncate x
    x = x[len(x) - len(y):]

    fig = plt.figure(title)
    plt.plot(x, y)
    plt.xlabel('Number of Timesteps')
    plt.ylabel('Rewards')
    plt.title(title + " Smoothed")
    plt.show()

def get_xy(log_folder):
    x, y = ts2xy(load_results(log_folder), 'timesteps')
    y = moving_average(y, window=7)
    # Truncate x
    x = x[len(x) - len(y):]
    return x,y

#EXP: Learning Rate Effect
learning_rates = np.array([0.1, 0.01, 0.001, 0.0001, 0.00001])
timesteps=int(5e6)
start1 = time.time()
for i in range(5):
    env = microgrid()
    env.reset()
    # Create log dir
    log_dir = "C:/Users/Homad/Dropbox/Thesis/codes/October_work/Cigre_modi
fied/Results/learning_rate2_" + str(learning_rates[i])

```

```

    os.makedirs(log_dir, exist_ok=True)
    env = Monitor(env, log_dir)
    env = DummyVecEnv([lambda: env])
    kwargs = {'gamma': 0.1, 'learning_rate': learning_rates[i], 'batch_size': 128, 'double_q': True, 'prioritized_replay': True}
    policy_kwargs = dict(act_fun=tf.nn.relu, layers=[256, 256], dueling=False)
    model = DQN("MlpPolicy", env, policy_kwargs=policy_kwargs, verbose=1, **kwargs)
    model.learn(total_timesteps=timesteps)
    model.save("learning_rate2_" + str(learning_rates[i]))
    del model
end1 = time.time()
# total time taken
print(f" {end1 - start1}")
fig, ax = plt.subplots()
ax.set_xlabel('Steps')
ax.set_ylabel('Rewards')
ax.tick_params(axis='y')
for i in range(5):
    log_dir = "C:/Users/Homad/Dropbox/Thesis/codes/October_work/Cigre_modified/Results/learning_rate2_" + str(learning_rates[i])
    x, y = get_xy(log_dir)
    label = "learning rate = " + str(learning_rates[i])
    ax.plot(x, y, label=label)
plt.grid()
leg = ax.legend();
ax.legend(loc='lower right', frameon=False)

#Exp: epsilon effect
start = time.time()
timesteps=int(5e6)
eps = np.array([0.1, 0.2, 0.3, 0.4, 0.5])
for i in range(5):
    env = microgrid()
    env.reset()
    # Create log dir
    log_dir = "C:/Users/Homad/Dropbox/Thesis/codes/October_work/Cigre_modified/Results/eps_" + str(eps[i])
    os.makedirs(log_dir, exist_ok=True)
    env = Monitor(env, log_dir)
    env = DummyVecEnv([lambda: env])

```

```

    kwargs = {'gamma': 0.1, 'learning_rate': 0.001, 'batch_size': 128, 'double_q': True, 'prioritized_replay': True, 'exploration_fraction': eps[i]}
    policy_kwargs = dict(act_fun=tf.nn.relu, layers=[256, 256], dueling=False)
    model = DQN("MlpPolicy", env, policy_kwargs=policy_kwargs, verbose=1, **kwargs)
    model.learn(total_timesteps=timesteps)
    model.save("eps_" + str(eps[i]))
    del model
end = time.time()
# total time taken
print(f" {end - start}")
fig, ax = plt.subplots()
ax.set_xlabel('Steps')
ax.set_ylabel('Rewards')
ax.tick_params(axis='y')
for i in range(5):
    log_dir = "C:/Users/Homad/Dropbox/Thesis/codes/October_work/Cigre_modified/Results/eps_" + str(eps[i])
    x, y = get_xy(log_dir)
    label = "exploration fraction = " + str(eps[i])
    ax.plot(x, y, label = label)
plt.grid()
leg = ax.legend();
ax.legend(loc='lower right', frameon=False)

#EXP: MLP hidden Layer effect (2 layers)
timesteps=int(5e5)
start2 = time.time()

for i in range(4):
    env = microgrid()
    env.reset()
    # Create log dir
    log_dir = "C:/Users/Homad/Dropbox/Thesis/codes/October_work/Cigre_modified/Results/MLP_2hidden_corrected_" + str(2**i*64)
    os.makedirs(log_dir, exist_ok=True)
    env = Monitor(env, log_dir)
    env = DummyVecEnv([lambda: env])
    kwargs = {'gamma': 0.1, 'learning_rate': 0.001, 'batch_size': 128, 'double_q': True, 'prioritized_replay': True, 'exploration_fraction': 0.4}
    policy_kwargs = dict(act_fun=tf.nn.relu, layers=[2**i*64, 2**i*64], dueling=False)

```

```

    model = DQN("MlpPolicy", env, policy_kwargs=policy_kwargs, verbose=1,
**kwargs)
    model.learn(total_timesteps=timesteps)
    model.save("mlp_2hidden_" + str(2**i*64))
    del model

end2 = time.time()
fig, ax = plt.subplots()
ax.set_xlabel('Steps')
ax.set_ylabel('Rewards')
ax.tick_params(axis='y')
for i in range(4):
    log_dir = "C:/Users/Homad/Dropbox/Thesis/codes/October_work/Cigre_modi
fied/Results/MLP_2hidden_corrected_" + str(2**i*64)
    x,y = get_xy(log_dir)
    label = "2 Hidden Layers of size " + str(2**i*64)
    ax.plot(x,y, label = label)
plt.grid()
leg = ax.legend();
ax.legend(loc='lower right', frameon=False)

# EXP: MLP hidden Layer effect (3 layers)
timesteps=int(5e5)
start3 = time.time()
for i in range(4):
    env = microgrid()
    env.reset()
    # Create log dir
    log_dir = "C:/Users/Homad/Dropbox/Thesis/codes/October_work/Cigre_modi
fied/Results/MLP_3hidden_" + str(2**i*64)
    os.makedirs(log_dir, exist_ok=True)
    env = Monitor(env, log_dir)
    env = DummyVecEnv([lambda: env])
    kwargs = {'gamma':0.1, 'learning_rate': 0.001, 'batch_size': 128, 'do
uble_q': True, 'prioritized_replay': True, 'exploration_fraction':0.4}
    policy_kwargs = dict(act_fun=tf.nn.relu, layers=[2**i*64, 2**i*64, 2**
i*64], dueling=False)
    #model = DQN(mlp_return(i), env, policy_kwargs=policy_kwargs, verbose=
1, **kwargs)
    model = DQN("MlpPolicy", env, policy_kwargs=policy_kwargs, verbose=1,
**kwargs)
    model.learn(total_timesteps=timesteps)
    model.save("mlp_3hidden_" + str(2**i*64))
    del model

```

```

end3 = time.time()
# total time taken
fig, ax = plt.subplots()
ax.set_xlabel('Steps')
ax.set_ylabel('Rewards')
ax.tick_params(axis='y')
for i in range(4):
    log_dir = "C:/Users/Homad/Dropbox/Thesis/codes/October_work/Cigre_modi
fied/Results/MLP_3hidden_" + str(2**i*64)
    x,y = get_xy(log_dir)
    label = "3 Hidden Layers of size " + str(2**i*64)
    ax.plot(x,y, label = label)

plt.grid()
leg = ax.legend();
ax.legend(loc='lower right', frameon=False)

# EXP: batch size effect
start4 = time.time()
batches = np.array([64, 128, 256, 512])
for i in range(4):
    env = microgrid()
    env.reset()
    # Create log dir
    log_dir = "C:/Users/Homad/Dropbox/Thesis/codes/October_work/Cigre_modi
fied/Results/batch_size_" + str(2**i*64)
    os.makedirs(log_dir, exist_ok=True)
    # Logs will be saved in log_dir/monitor.csv
    env = Monitor(env, log_dir)
    env = DummyVecEnv([lambda: env])
    policy_kwargs = dict(act_fun=tf.nn.relu, layers=[256, 256], dueling=False)
    kwargs = {'gamma': 0.99, 'learning_rate': 0.0001, 'batch_size': batches[i], 'double_q': False, 'prioritized_replay': False}
    model = DQN("MlpPolicy", env, policy_kwargs=policy_kwargs, verbose=1, **kwargs)
    # model = DQN('MlpPolicy', env, learning_rate=learning_r, batch_size = batches[i], verbose=1)
    model.learn(total_timesteps=timesteps)
    model.save("batch_size_" + str(2**i*64))
    del model
end4 = time.time()
# total time taken

```

```

print(f" {end4 - start4}")
fig, ax = plt.subplots()
ax.set_xlabel('Steps')
ax.set_ylabel('Rewards')
ax.tick_params(axis='y')
for i in range(4):
    x,y = get_xy("C:/Users/Homad/Dropbox/Thesis/codes/October_work/Cigre_m
odified/Results/batch_size_" +str(2**i*64))
    label = "Batch Size " + str(2**i*64)
    ax.plot(x,y, label = label)

plt.grid()
leg = ax.legend();
ax.legend(loc='lower right', frameon=False)

```

#other variations

```

env = microgrid()
env.reset()
# Create log dir
log_dir = "C:/Users/Homad/Dropbox/Thesis/codes/October_work/Cigre_modified
/Results/MLP_variation_base"
os.makedirs(log_dir, exist_ok=True)
env = Monitor(env, log_dir)
env = DummyVecEnv([lambda: env])
kwargs = {'gamma':0.99, 'learning_rate': 0.0001, 'batch_size': 256, 'doub
le_q': False, 'prioritized_replay': False}
policy_kwargs = dict(act_fun=tf.nn.relu, layers=[256, 256], dueling=False)
#model = DQN(mlp_return(i), env, policy_kwargs=policy_kwargs, verbose=1, *
*kwargs)
model = DQN("MlpPolicy", env, policy_kwargs=policy_kwargs, verbose=1, **kw
args)
model.learn(total_timesteps=timesteps)
model.save("mlp_variation_base")
del model

```

```

env = microgrid()
env.reset()
# Create log dir
log_dir = "C:/Users/Homad/Dropbox/Thesis/codes/October_work/Cigre_modified
/Results/MLP_variation_replay"
os.makedirs(log_dir, exist_ok=True)
env = Monitor(env, log_dir)

```

```

env = DummyVecEnv([lambda: env])
kwargs = {'gamma': 0.99, 'learning_rate': 0.0001, 'batch_size': 256, 'double_q': False, 'prioritized_replay': True}
policy_kwargs = dict(act_fun=tf.nn.relu, layers=[256, 256], dueling=False)
#model = DQN(mlp_return(i), env, policy_kwargs=policy_kwargs, verbose=1, **kwargs)
model = DQN("MlpPolicy", env, policy_kwargs=policy_kwargs, verbose=1, **kwargs)
model.learn(total_timesteps=timesteps)
model.save("MLP_variation_replay")
del model

env = microgrid()
env.reset()
# Create log dir
log_dir = "C:/Users/Homad/Dropbox/Thesis/codes/October_work/Cigre_modified/Results/MLP_variation_all_step5e6"
os.makedirs(log_dir, exist_ok=True)
env = Monitor(env, log_dir)
env = DummyVecEnv([lambda: env])
kwargs = {'gamma': 0.99, 'learning_rate': 0.001, 'batch_size': 128, 'double_q': True, 'prioritized_replay': True}
policy_kwargs = dict(act_fun=tf.nn.relu, layers=[256, 256], dueling=True)
model = DQN("MlpPolicy", env, policy_kwargs=policy_kwargs, verbose=1, **kwargs)
model.learn(total_timesteps=timesteps)
model.save("MLP_variation_all_moresteps")
del model

fig, ax = plt.subplots()
ax.set_xlabel('Steps')
ax.set_ylabel('Rewards')
ax.tick_params(axis='y')

x,y = get_xy("C:/Users/Homad/Dropbox/Thesis/codes/October_work/Cigre_modified/Results/MLP_variation_base")
label = "Base DQN"
ax.plot(x[10:],y[10:], label = label)

x,y = get_xy("C:/Users/Homad/Dropbox/Thesis/codes/October_work/Cigre_modified/Results/MLP_variation_replay")
label = "DQN with prioritized replay "
ax.plot(x[10:],y[10:], label = label)

```

```

x,y = get_xy("C:/Users/Homad/Dropbox/Thesis/codes/October_work/Cigre_modif
ied/Results/MLP_variation_all")
label = "Double DQN"
ax.plot(x[10:],y[10:], label = label)

plt.grid()
leg = ax.legend();
ax.legend(loc='lower right', frameon=False)

# EXP: gamma exp
start7 = time.time()
gammas = np.array([0.1, 0.3,0.5,0.7,0.9])
timesteps=int(5e5)

for i in range(5):
    env = microgrid()
    env.reset()
    log_dir = "C:/Users/Homad/Dropbox/Thesis/codes/October_work/Cigre_modi
fied/Results/MLP_gamma_" + str(gammas[i])
    os.makedirs(log_dir, exist_ok=True)
    env = Monitor(env, log_dir)
    env = DummyVecEnv([lambda: env])
    kwargs = {'gamma':gammas[i], 'learning_rate': 0.001, 'batch_size': 12
8, 'double_q': False, 'prioritized_replay': True}
    policy_kwargs = dict(act_fun=tf.nn.relu, layers=[256, 256], dueling=Fa
lse)
    model = DQN("MlpPolicy", env, policy_kwargs=policy_kwargs, verbose=1,
**kwargs)
    model.learn(total_timesteps=timesteps)
    model.save("MLP_gamma_" + str(gammas[i]))

    del model
    print(start7)
end7 = time.time()
# total time taken
print(f" {end7 - start7}")

fig, ax = plt.subplots()
ax.set_xlabel('Steps')
ax.set_ylabel('Rewards')
ax.tick_params(axis='y')
for i in range(5):

```

```
log_dir = "C:/Users/Homad/Dropbox/Thesis/codes/October_work/Cigre_modi  
fied/Results/MLP_gamma_" + str(gammas[i])  
x,y = get_xy(log_dir)  
label = "Gamma = " + str(gammas[i])  
ax.plot(x,y, label = label)  
  
plt.grid()  
leg = ax.legend();  
ax.legend(loc='lower right', frameon=False)
```

References

- [1] R. Lasseter, *MicroGrids, IEEE Power Engineering Society Winter Meeting Conference Proceedings*, 2002.
- [2] R. H. Lasseter, "Smart distribution: Coupled microgrids," *Proceedings of the IEEE*, vol. 99, pp. 1074-1082, 2011.
- [3] M. F. Zia, E. Elbouchikhi and M. Benbouzid, "Microgrids energy management systems: A critical review on methods, solutions, and prospects," *Applied energy*, vol. 222, pp. 1033-1055, 2018.
- [4] J. E. Stamp, "Microgrid Modeling to Support the Design Process.," 2012.
- [5] "Solar Module Supply Chain Service," Wood Mackenzie, 2020. [Online]. Available: <https://www.woodmac.com/research/products/power-and-renewables/pv-supply-chain-technology-service/>. [Accessed 25 3 2021].
- [6] "Wind Costs," IRENA, 2020. [Online]. Available: <https://www.irena.org/Statistics/View-Data-by-Topic/Costs/Wind-Costs>. [Accessed 25 3 2021].
- [7] X. SUN, "Solar Technology Got Cheaper and Better in the 2010s. Now What?," GTM, 17 12 2019. [Online]. Available: <https://www.greentechmedia.com/articles/read/solar-pv-has-become-cheaper-and-better-in-the-2010s-now-what>. [Accessed 3 2021].
- [8] W. Su, J. Wang and J. Roh, "Stochastic energy scheduling in microgrids with intermittent renewable energy resources," *IEEE Transactions on Smart Grid*, vol. 5, pp. 1876-1883, 2013.
- [9] Z. Wang, B. Chen, J. Wang, M. M. Begovic and C. Chen, "Coordinated energy management of networked microgrids in distribution systems," *IEEE Transactions on Smart Grid*, vol. 6, pp. 45-53, 2014.
- [10] F. Farzan, M. A. Jafari, R. Masiello and Y. Lu, "Toward optimal day-ahead scheduling and operation control of microgrids under uncertainty," *IEEE Transactions on Smart Grid*, vol. 6, pp. 499-507, 2014.

- [11] Y. Zhang, N. Gatsis and G. B. Giannakis, "Robust energy management for microgrids with high-penetration renewables," *IEEE Transactions on Sustainable Energy*, vol. 4, pp. 944-953, 2013.
- [12] F. Valencia, D. Sáez, J. Collado, F. Ávila, A. Marquez and J. J. Espinosa, "Robust energy management system based on interval fuzzy models," *IEEE Transactions on Control Systems Technology*, vol. 24, pp. 140-157, 2015.
- [13] X. Tan, Q. Li and H. Wang, "Advances and trends of energy storage technology in microgrid," *International Journal of Electrical Power & Energy Systems*, vol. 44, p. 179–191, 2013.
- [14] "Utility-scale battery storage costs decreased nearly 70% between 2015 and 2018," EIA, 23 10 2020. [Online]. Available: <https://www.eia.gov/todayinenergy/detail.php?id=45596>. [Accessed 25 3 2021].
- [15] M. Faisal, M. A. Hannan, P. J. Ker, A. Hussain, M. B. Mansor and F. Blaabjerg, "Review of energy storage system technologies in microgrid applications: Issues and challenges," *Ieee Access*, vol. 6, p. 35143–35164, 2018.
- [16] M. K. Kiptoo, M. E. Lotfy, O. B. Adewuyi, A. Conteh, A. M. Howlader and T. Senjyu, "Integrated approach for optimal techno-economic planning for high renewable energy-based isolated microgrid considering cost of energy storage and demand response strategies," *Energy Conversion and Management*, vol. 215, p. 112917, 2020.
- [17] P. Zeng, H. Li, H. He and S. Li, "Dynamic energy management of a microgrid using approximate dynamic programming and deep recurrent neural network learning," *IEEE Transactions on Smart Grid*, 2018.
- [18] M. Erol-Kantarci, B. Kantarci and H. T. Mouftah, "Cost-aware smart microgrid network design for a sustainable smart grid," in *2011 IEEE GLOBECOM Workshops (GC Wkshps)*, 2011.
- [19] E. Alvarez, A. C. Lopez, J. Gómez-Aleixandre and N. Abajo, "On-line minimization of running costs, greenhouse gas emissions and the impact of distributed generation using microgrids on the electrical system," in *2009 IEEE PES/IAS Conference on Sustainable Alternative Energy (SAE)*, 2009.

- [20] G. E. Asimakopoulou, A. L. Dimeas and N. D. Hatziaargyriou, "Leader-follower strategies for energy management of multi-microgrids," *IEEE transactions on smart grid*, vol. 4, pp. 1909-1916, 2013.
- [21] M. L. Di Silvestre, G. Graditi and E. R. Sanseverino, "A generalized framework for optimal sizing of distributed energy resources in micro-grids using an indicator-based swarm approach," *IEEE Transactions on Industrial Informatics*, vol. 10, pp. 152-162, 2013.
- [22] A. Hooshmand, B. Asghari and R. Sharma, "A novel cost-aware multi-objective energy management method for microgrids," in *2013 IEEE PES Innovative Smart Grid Technologies Conference (ISGT)*, 2013.
- [23] T. A. Nguyen and M. L. Crow, "Optimization in energy and power management for renewable-diesel microgrids using dynamic programming algorithm," in *2012 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*, 2012.
- [24] G. Corso, M. L. Di Silvestre, M. G. Ippolito, E. R. Sanseverino and G. Zizzo, "Multi-objective long term optimal dispatch of distributed energy resources in micro-grids," in *45th International Universities Power Engineering Conference UPEC2010*, 2010.
- [25] H. S. V. S. K. Nunna and S. Doolla, "Energy management in microgrids using demand response and distributed storage_A multiagent approach," *IEEE Transactions on Power Delivery*, vol. 28, pp. 939-947, 2013.
- [26] Q. Jiang, M. Xue and G. Geng, "Energy management of microgrid in grid-connected and stand-alone modes," *IEEE transactions on power systems*, vol. 28, pp. 3380-3389, 2013.
- [27] A. A. Khan, M. Naeem, M. Iqbal, S. Qaisar and A. Anpalagan, "A compendium of optimization objectives, constraints, tools and algorithms for energy management in microgrids," *Renewable and Sustainable Energy Reviews*, vol. 58, pp. 1664-1683, 2016.
- [28] M. Patterson, N. F. Macia and A. M. Kannan, "Hybrid microgrid model based on solar photovoltaic battery fuel cell system for intermittent load applications," *IEEE Transactions on Energy Conversion*, vol. 30, pp. 359-366, 2014.
- [29] K. Thirugnanam, S. K. Kerk, C. Yuen, N. Liu and M. Zhang, "Energy management for renewable microgrid in reducing diesel generators usage with multiple types of

- battery," *IEEE Transactions on Industrial Electronics*, vol. 65, pp. 6772-6786, 2018.
- [30] R. Palma-Behnke, C. Benavides, F. Lanas, B. Severino, L. Reyes, J. Llanos and D. Sáez, "A microgrid energy management system based on the rolling horizon strategy," *IEEE Transactions on Smart Grid*, vol. 4, pp. 996-1006, 2013.
- [31] T. Morstyn, B. Hredzak, R. P. Aguilera and V. G. Agelidis, "Model predictive control for distributed microgrid battery energy storage systems," *IEEE Transactions on Control Systems Technology*, vol. 26, pp. 1107-1114, 2017.
- [32] Y. Ye, D. Qiu, X. Wu, G. Strbac and J. Ward, "Model-free real-time autonomous control for a residential multi-energy system using deep reinforcement learning," *IEEE Transactions on Smart Grid*, vol. 11, p. 3068–3082, 2020.
- [33] L. Yu, S. Qin, M. Zhang, C. Shen, T. Jiang and X. Guan, "Deep Reinforcement Learning for Smart Building Energy Management: A Survey," *arXiv preprint arXiv:2008.05074*, 2020.
- [34] Y. Ji, J. Wang, J. Xu, X. Fang and H. Zhang, "Real-Time Energy Management of a Microgrid Using Deep Reinforcement Learning," *Energies*, vol. 12, p. 2291, 2019.
- [35] B. Mbuwir, F. Ruelens, F. Spiessens and G. Deconinck, "Battery energy management in a microgrid using batch reinforcement learning," *Energies*, vol. 10, p. 1846, 2017.
- [36] G. K. Venayagamoorthy, R. K. Sharma, P. K. Gautam and A. Ahmadi, "Dynamic energy management system for a smart microgrid," *IEEE transactions on neural networks and learning systems*, vol. 27, pp. 1643-1656, 2016.
- [37] E. Foruzan, L.-K. Soh and S. Asgarpour, "Reinforcement learning approach for optimal distributed energy management in a microgrid," *IEEE Transactions on Power Systems*, vol. 33, pp. 5749-5758, 2018.
- [38] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski and others, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, p. 529, 2015.
- [39] Y. Ye, D. Qiu, M. Sun, D. Papadaskalopoulos and G. Strbac, "Deep reinforcement learning for strategic bidding in electricity markets," *IEEE Transactions on Smart Grid*, vol. 11, p. 1343–1355, 2020.

- [40] Q. Zhang, K. Dehghanpour, Z. Wang and Q. Huang, "A learning-based power management method for networked microgrids under incomplete information," *IEEE Transactions on Smart Grid*, vol. 11, p. 1193–1204, 2020.
- [41] Y. Du and F. Li, "Intelligent multi-microgrid energy management based on deep neural network and model-free reinforcement learning," *IEEE Transactions on Smart Grid*, vol. 11, p. 1066–1076, 2020.
- [42] Q. Huang, R. Huang, W. Hao, J. Tan, R. Fan and Z. Huang, "Adaptive power system emergency control using deep reinforcement learning," *IEEE Transactions on Smart Grid*, vol. 11, p. 1171–1182, 2020.
- [43] R. Diao, Z. Wang, D. Shi, Q. Chang, J. Duan and X. Zhang, "Autonomous voltage control for grid operation using deep reinforcement learning," in *2020 IEEE Power & Energy Society General Meeting (PESGM)*, 2020.
- [44] V. François-Lavet, D. Taralla, D. Ernst and R. Fonteneau, "Deep reinforcement learning solutions for energy microgrids management," in *European Workshop on Reinforcement Learning (EWRL 2016)*, 2016.
- [45] J. D. Kelleher, *Deep learning*, MIT press, 2019.
- [46] A. Khan, A. Sohail, U. Zahoor and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artificial Intelligence Review*, vol. 53, p. 5455–5516, 2020.
- [47] A. Wong, M. Famouri, M. Pavlova and S. Surana, "TinySpeech: Attention condensers for deep speech recognition neural networks on edge devices," *arXiv preprint arXiv:2008.04245*, 2020.
- [48] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh and K. Shaalan, "Speech recognition using deep neural networks: A systematic review," *IEEE access*, vol. 7, p. 19143–19165, 2019.
- [49] J. Gui, Z. Sun, Y. Wen, D. Tao and J. Ye, "A review on generative adversarial networks: Algorithms, theory, and applications," *arXiv preprint arXiv:2001.06937*, 2020.
- [50] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, MIT press, 2018.

- [51] J. Viquerat, R. Duvigneau, P. Meliga, A. Kuhnle and E. Hachem, "Policy-based optimization: single-step policy gradient method seen as an evolution strategy," *arXiv preprint arXiv:2104.06175*, 2021.
- [52] K. Arulkumaran, M. P. Deisenroth, M. Brundage and A. A. Bharath, "A brief survey of deep reinforcement learning," *arXiv preprint arXiv:1708.05866*, 2017.
- [53] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, J. Pineau and others, "An introduction to deep reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 11, pp. 219-354, 2018.
- [54] M. Riedmiller, "Neural fitted Q iteration--first experiences with a data efficient neural reinforcement learning method," in *European Conference on Machine Learning*, 2005.
- [55] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.
- [56] H. Van Hasselt, A. Guez and D. Silver, "Deep reinforcement learning with double q-learning," *arXiv preprint arXiv:1509.06461*, 2015.
- [57] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.
- [58] C. M. Bishop, Pattern recognition and machine learning, Springer Science+ Business Media, 2006.
- [59] C. Szegedy, S. Ioffe, V. Vanhoucke and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.
- [60] Y. LeCun, Y. Bengio and others, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, p. 1995, 1995.
- [61] K. Patel, "MNIST Handwritten Digits Classification using a Convolutional Neural Network (CNN)," Towards Data Science, 7 9 2019. [Online]. Available:

<https://towardsdatascience.com/mnist-handwritten-digits-classification-using-a-convolutional-neural-network-cnn-af5fafbc35e9>. [Accessed 24 3 2021].

- [62] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735-1780, 1997.
- [63] T. Schaul, J. Quan, I. Antonoglou and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.
- [64] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.
- [65] L. Thurner, A. Scheidler, F. Schäfer, J.-H. Menke, J. Dollichon, F. Meier, S. Meinecke and M. Braun, "pandapower_an open-source python tool for convenient modeling, analysis, and optimization of electric power systems," *IEEE Transactions on Power Systems*, vol. 33, pp. 6510-6521, 2018.
- [66] L. Thurner, A. Scheidler and F. Schäfer, "Pandapower," Energy Management and Power System Operation, University of Kassel, 2020. [Online]. Available: <https://pandapower.readthedocs.io/>. [Accessed 12 August 2020].
- [67] IEEE, IEEE recommended practice for industrial and commercial power systems analysis (brown book), 1998.
- [68] J. J. Grainger, W. D. Stevenson, W. D. Stevenson and others, Power system analysis, 2003.
- [69] K. Strunz, E. Abbasi, C. Abbey, C. Andrieu, F. Gao, T. Gaunt, A. Gole, N. Hatziaargyriou and R. Iravani, "Benchmark systems for network integration of renewable and distributed energy resources," *Cigre Task Force C*, vol. 6, p. 78, 2009.
- [70] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba, *OpenAI Gym*, 2016.
- [71] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu and P. Zhokhov, *OpenAI Baselines*, GitHub, 2017.
- [72] A. Hill, A. Raffin, M. Ernestus, A. Gleave, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert and others, "Stable baselines," *GitHub repository*, 2018.

- [73] A. G. Barto, R. S. Sutton and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE transactions on systems, man, and cybernetics*, pp. 834-846, 1983.
- [74] I. Koryakovskiy, M. Kudruss, R. Babuška, W. Caarls, C. Kirches, K. Mombaur, J. P. Schlöder and H. Vallery, "Benchmarking model-free and model-based optimal control," *Robotics and Autonomous Systems*, vol. 92, pp. 81-90, 2017.
- [75] Y. Duan, X. Chen, R. Houthooft, J. Schulman and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*, 2016.
- [76] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner and others, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278-2324, 1998.
- [77] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor and Y. Wu, *Stable Baselines*, GitHub, 2018.
- [78] Y. Bengio, D.-H. Lee, J. Bornschein, T. Mesnard and Z. Lin, "Towards biologically plausible deep learning," *arXiv preprint arXiv:1502.04156*, 2015.
- [79] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu and P. Zhokhov, "Openai baselines (2017)," URL <https://github.com/openai/baselines>, 2016.
- [80] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.
- [81] M. Roderick, J. MacGlashan and S. Tellex, "Implementing the deep q-network," *arXiv preprint arXiv:1711.07478*, 2017.
- [82] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

Vitae

Name :Mohammed Alabdullah |

Nationality :Saudi Arabia |

Date of Birth :3/1/1991|

Email :alabmh0d@gmail.com|

Address :Dhahran, Saudi Arabia|

Background :Mohammed completed his BEng in Electronics & Electrical Engineering from University College London (UCL) in 2014. He then joined Saudi Aramco as a Power Systems Engineer, overseeing different aspects of power infrastructure within the oil and gas industry. He successfully led several critical power systems projects. Currently he is working in completing his MSc degree with KFUPM in Electrical Engineering.]