# Microgird Energy Trading Using Reinforcement Learning

A Case Study

**Moayad Hassan Mohamed Elamin**
**Fay Majid Ahmed Elhassan**

A thesis presented for the degree of
bachelor of engineeing

Electrical and Electronic Engineering
University of Khartoum
Sudan
August 2020

# Contents


**1 Introduction** **5**
- 1.1 Problem Statement . . . 5
- 1.2 MicroGrids . . . 6
- 1.3 Reinforcement Learning . . . 6
- 1.4 Research Objectives . . . 7

**2 Literature Review** **8**
- 2.1 Microgirds . . . 8
  - 2.1.1 Theoretical Background . . . 8
  - 2.1.2 Technical challenges of Micro-grids . . . 10
  - 2.1.3 Control hierarchy in micro-grids . . . 11
- 2.2 Reinforcement Learning . . . 15
  - 2.2.1 Machine Learning Introduction . . . 15
  - 2.2.2 Reinforcement Learning Introduction . . . 16
  - 2.2.3 Reinforcement Learning Solution Methods Introduction . . . 18
  - 2.2.4 Cross-Entropy Methods . . . 19
  - 2.2.5 Q-Leaning . . . 19
  - 2.2.6 Deep Reinforcement Learning . . . 20
  - 2.2.7 Policy Gradients Methods . . . 21
- 2.3 Related Work . . . 24

**3 Methodology** **28**
- 3.1 Microgrid Methodology . . . 28
- 3.2 Reinforcement Learning Methodology . . . 28
  - 3.2.1 Python and necessary libraries . . . 28
  - 3.2.2 Pytorch . . . 28
  - 3.2.3 OpenAI GYM . . . 29
  - 3.2.4 MicroGrid Enviroment . . . 30
  - 3.2.5 Continuous Action Space . . . 31
  - 3.2.6 Implementation of selected algorithms . . . 36
  - 3.2.7 The Data . . . 36

**A Code** **37**
- A.1 MATLAB code . . . 37
- A.2 Python code . . . 37

# List of Figures


2.1 A Simple Micro-Grid . . . . . 8
2.2 Hierarchy diagram . . . . . 12
2.3 A system with two voltage sources . . . . . 12
2.4 $P - w$ and $Q - V$ droop characteristics . . . . . 13
2.5 Machine Learning Sections . . . . . 15
2.6 Simple Neural Network . . . . . 16
2.7 Reinforcement Learning Parts . . . . . 17
2.8 A2C Actor and Critic networks . . . . . 22

3.1 DDPG Actor and Critic networks . . . . . 32

# Acronyms

Advance Actor Critic: **A2C**
Artificial Neural Network: **ANN**
Convolutional Neural Networks: **CNN**
Deep Deterministic Policy Gradient: **DDPG**
Deep Learning: **DL**
Deep Q-Learning: **DQN**
distributed energy resources: **DERs**
distributed generation units: **DG**
Distribution Network Operator: **DNO**
Energy Management System: **EMS**
Energy Storage System: **ESS**
Game Theory: **GT**
Local Controllers: **LC**
Machine Learning: **ML**
Markov Decision Process: **MDP**
Micro Grid: **MG**
Microgrid Central Controller: **MGCC**
Monte Carlo: **MC**
Natural Language Processing: **NLP**
Neural Network: **NN**
Proximal Policy Optimization: **PPO**
Recurrent Neural Networks: **RNN**
Reinforcement Learning: **RL**
Stochastic Gradient Descent: **SGD**
Supervised Learning: **SL**
Temporal Difference: **TD**
Transmission and distribution: **T&D**
Trust Region Policy Optimization: **TRPO**
Unsupervised Learning: **UL**
unit commitment: **UC**
Voltage source inverter: **VSI**

# Chapter 1

# Introduction

## 1.1 Problem Statement

As electricity is the backbone of all development and innovation, Sudan's electricity situation is a challenge that needs to be tackled with excellent efficiency and using innovative solutions. According to governmental sources, the national unified electricity grid covers less than 40% of Sudan; more than 60% of this is residential demand, 75% of power generated goes to the capital state; Khartoum state. Khartoum itself has a 66% connection percentage, while the 3rd and fourth states based on population density; South Darfur and North Kordofan have 2% and 5% connection percentage.

Electricity in Sudan has an average production cost of 18 cents; residential electricity is heavily subsidized and sold at a range of 1.8% to 19.4% of the production cost. Adding this to a usage increase of 10% annually, highest amongst neighboring countries, means that electricity inequality will only continue to grow. The top 1% of earners in Sudan use 49% of electricity produced while the lowest 49% use only 21% of electricity produced. This situation needs a re-evaluation of the system used in Sudan to achieve a fast recovery and adaptation to modern models.

Sudan's system is a conventional system of centralized generation; the government uses a single system to generate electricity at far-away stations or dams; then, it is distributed in the country. This system is dying in the world as the old equipment causes high maintenance cost and reliability problems; it is limited to the addition of new demand areas.

Old engineering, outdated systems of management, and efficiency concerns contribute to Sudan's astronomical loss rate of 25%. Combining this with its environmental concerns and lack of renewable energy utilization make it one to be changed.

The distributed generation where we generate where we will consume and have smaller grids, smaller generation and smaller transportation of electricity is an alternative worth considering. Here we a flexible system, easy to maintain, efficient, modular, reliable, economically efficient, and above all environmentally responsible because of the renewable energy cornerstone, it stands on.

When looking into Sudan and its high number of villages and lack of major cities, we can see that microgrids present a great model of distributed generation to Sudan's rural areas. Microgrids are a variation of smart grids where we create small scale grids to work on villages, islands, and small residential areas, which can work as a standalone islanded grid or be grid-connected. It uses distributed energy

sources and renewable energy sources to generate electricity locally and then fulfill the local demand and use storage units for night demand and fault cases. As it is a smart grid, then a grid management system is needed.

In the new wave of control, we use machine learning as a way to control different systems robustly through the full cycle from production to generations, and that is the reason we will be using its techniques in this project. Reinforcement learning is our technique of choice due to its usage in sequential control.

## 1.2 MicroGrids

A microgrid is a collection of energy production units and consumers (load) placed near each other to reduce transport and control costs. These grids use Renewable Energy Resources (RESs) as the production units, those include Solar Energy production units (Photovoltaic units PVs), Wind production units (Wind Turbines), Biomass units as well as Energy storage system. Loads supplied by the Microgrid can be critical loads (industrial factories, hospitals, schools)and non-critical loads (houses).

The supply between this type of loads merely depends on the mode of operation the Microgrid operates in; dual-mode operation on-grid where we connect the Microgrid to the primary utility grid in normal conditions. The other mode is off-grid or known as (islanded )mode, where the microgrid switch to be operating without the back up of the utility grid due to shortage or disturbance in the primary grid.

In some switching mode scenarios, the non-critical load is deactivated from the Microgrid until we reach a stable status. Microgrids components from a control unit, inverters, and batteries help determine the Microgrid's supply-demand profile. We connect The different Microgrids to a primary controller alongside the main grid. This controller receives the supply-demand information of each Microgrid and the mode, on-grid, or off-grid.

## 1.3 Reinforcement Learning

A reinforcement learning algorithm will handle the high-level control between the microgrids. It works as a black box that trades when needed deciding then with which Microgrid and with what price for our microgrids to achieve equilibrium.

Reinforcement Learning is a field where the problem it solves is an environment affected by an algorithm controlled agent. The agent takes an action that affects the environment, then the agent sees the environment as an observation, and receives a reward on the "goodness" of the action it last took.

We use this to optimize its policy, which decides which action to take based on which state we are in, and a state is a unique set of values that the observation returns that fully or partially describe our environment. We use the Markov decision process concept, where this state is sufficient to predict the future. Through time, the actions taken by the agent traverse the state space until we reach our goal state or goal reward. The optimal policy will do this in the best path possible as we will later introduce concepts that push towards a faster goal-reaching policy.

In the context of our project, we will create an environment that simulates several microgrids with full generation and load profiles. We will then create an agent that

will choose to buy or sell electricity when our generation and stored electricity are insufficient for our load. It will also choose the price of the transaction to achieve the maximum economic gain from trading situations.

## 1.4 Research Objectives

This thesis proposes working in Microgrid islanded mode and introduces a Reinforcement Learning algorithm to manage an energy trading process between neighboring grids to achieve optimality (equilibrium) in production. The research aims to propose a system for off-grid power management and trading between different microgrids using expected power production (supply), consumption (demand) forecasting, and power storage information. We will apply machine learning techniques in specific deep reinforcement learning to solve the trading process. We will test multiple algorithms on our created environment and find the best algorithm to solve our problem.

# Chapter 2

# Literature Review

## 2.1 Microgirds

### 2.1.1 Theoretical Background

The environmental and economic conditions, the need to provide a clean environment, and decrease the carbon emissions in the atmosphere and the need to decrease fossil fuels made technological advancements a need. Recent technological developments in micro-generation showed that micro-grids are the future of efficient and fast restoration of the power system.

#### Micro-grids

They can be defined as "A group of interconnected loads and distributed energy resources(DERs)with set electrical boundaries that act as a single controllable entity concerning the grid that can connect and disconnect itself from the grid based on the mode required."

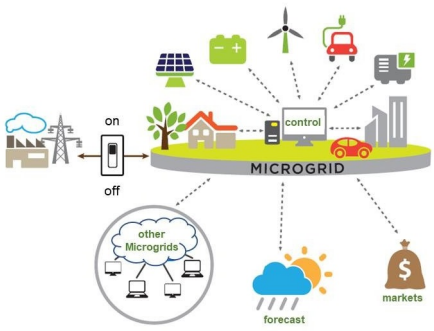


Figure 2.1: A Simple Micro-Grid

The term micro-grid dates back to 1882 when Edison installed 50 DC micro-grid before the operation of the utility grid. With the utilization of utility grid and benefiting from economic and increasing transmission process leading to fade away of micro-grids. Indeed, in the past years, with advancements in power electronics and DER technologies and more engagement with the electricity consumer, the micro-grid concept started seeing the light again.

There would be three different features if we compromised the DER installations could be considered as a micro-grid. There must be a master controller to control the system components as a single controllable entity. The installed generation capacity must exceed the peak critical load; thus, when we can disconnect from the grid and, most importantly, clearly defined electrical boundaries.

The characteristics mentioned earlier present the micro-grid as a small-scale power supply network for a small community; it allows the penetration of distributed generation into the system. One of its major advantages is its ability to work alone during utility grid disturbance or outage; it means that micro-grid can operate in two modes:

- ON-grid
- OFF-grid

The on-grid mode is when the microgrid is connected to the primary utility grid and work in synchronization with it. This mode enables bidirectional power flow, and if any disturbance happens to the primary grid, the micro-grid will switch to the off-grid mode or what is known as a standalone grid (islanded). In this mode, the microgrid acts as the primary provider to the specified geographical area, working autonomously with high-quality service by acting as local voltage and frequency regulator [1]. Micro-grid is not a backup generator; a backup generator has been around for quite a while, providing temporary supply to local loads when there is a disturbance in the main utility grid supply. However, micro-grids has a wide range of benefits and noticeably more flexible than a backup generator.

The Micro-grid main components include Loads, DERs, master controller, smart switches, protective devices, communication, control, and automation. The micro-grid load is known to be of two categories; critical and non-critical (fixed and flexible). Critical load (Fixed) must be satisfied at all conditions and is not altered. In contrast, the non-critical load (flexible) can differ and be adjusted based on the economic incentives or the status of the grid (islanded requirements).

DERs consist of distributed generation units(DG) and Energy Storage System (ESS) which can be installed on the utility or consumer premises. The distributed generation units are either dispatchable or non-dispatchable. Dispatchable units can be controlled by the central controller and are subjected to technical constraints depending on the unit type. Non-dispatchable cannot be controlled by the micro-grid controller as its input is changeable, and unrestrained such units are like Solar and wind, mainly renewable sources. The intermittency shows that generation is not always available. Simultaneously, unpredictability reveals that the generation tends to be unstable at different time scales. Those stated characteristics affect our non-dispatchable units negatively and usually increase the forecast error. The right solution is always to reinforce those units with an energy storage system (ESS).

As we know, electricity demand varies based on the time of day and time of year. While in the traditional power system, we are not capable of storing electricity, which leads to a gap between supply and demand. Micro-grid having a mixed power generation will allow as to fill in the mismatch as some generations have significant response times, and others have little flexibility. Some generations can start real quickly to provide more or less depending on demand. Provided the late reasons, the energy storage system is quite beneficial in managing such system .ESS synchronize

with DGs as an assurance to micro-grid generation capability. Its inclusion within the micro-grid system allows the excess energy generated to be stored or in the typical scenario that could be put into the utility grid.

The master controller in the micro-grid performs the scheduling in the microgrid's dual-mode based on economic and security considerations. Usually, the master controller is responsible for interaction with the utility grid, the decision to switch between on-grid and islanded.

With that been said micro-grids benefits are: improving reliability by introducing self-healing at local distribution network, managing local loads due to higher power quality, carbon emission reduction due to diversification usage in renewable energy sources, economically reducing the Transmission and Distribution (T&D) costs [look 2]

### 2.1.2 Technical challenges of Micro-grids

The integration of DERs units and micro-grid introduces several technical challenges that require addressing the control design and protection system to ensure the level of reliability is not affected. The potential benefits of DG are fully harnessed. Some of these challenges are stability issues arising while at transmission-level, and others are assumptions applied to distribution systems.

The most critical challenges in Protection and control are bidirectional power flow, stability issues, modeling, low inertia, uncertainty. [in 3].

Along with the above, the micro-grid must guarantee the reliable and economical operation of micro-grid while overcoming the challenges above. Henceforth, these are some of the required features in the control system: output control, power balance, DSM, economic dispatch, the transition between mode of operation [see 3].

Furthermore, we can summarize microgrid issues into three points.

1. Islanded mode

   This mode represents a future of interconnected grid with a high density of DG. The control strategies of islanding mode are quite essential for the micro-grid to operate in autonomous mode.

   We use two kinds of control strategies of islanding to operate the grid. The PQ inverter controls active and reactive power setpoint .furthermore, the VSI control maintains the voltage and frequency feeding the load.

   Henceforth the following issues occur within the islanded mode: As beginning as DG supply, the load demand equal sharing is required, but due to various unequal capacities of the DG load sharing tend to be impossible. Along with the harmonics and compensation effort for unbalance and nonlinearity of the load. Secondly, losing a DG in this mode allows the use of load shedding and battery unit to be explored to fulfill the critical load. Finally, guaranteeing Stability in islanded mode is quite challenging with the presence of non-linear load. (An overview on microgrid control strategy).

2. Stability

   Stability issues may arise in a micro-grid due to various causes such as islanding the micro-grid and grid reconnection, change in parameters, faults, mismatch

in the generation demand, an immediate connection of DG, or disconnection, and this leads to changes in the voltage and frequency of the system.

Henceforth usage of voltage and frequency controllers or regulators was suggested along with power electronic DGs to give the micro-grid flexibility. Along with ensuring both voltage and frequency are within predefined limit around setpoint values to adjust active and reactive power generated or consumed.

3. Protection

Certain conditions have to be taken into consideration when designing a micro-grid. Its ability to operate under unbalanced conditions such as spacing of overhead transmission and unbalanced impedance from three-phase load any fault within our power system. As the Protection of micro-grid is vital, a new scheme has been introduced that uses ABC-DQ transformation of the system voltage to detect any faults or short circuits. It achieves this by comparing measurements at different locations, thus associating with micro-grid network the faults varieties at different zones.

Unrestrained excess generation results in the voltage profile distortion in an islanded microgrid. Therefore, we should consider the characteristic difference between various DG to develop control strategies to regulate the power output. In cases where active power is not consumed, power oscillations can be used mainly in islanded mode.

### 2.1.3 Control hierarchy in micro-grids

To understand how the micro-grid is controlled and how it can operate in the two modes, on-grid, and island. Two opposite approaches are identified concerning the architecture of power system control, which is centralized and decentralized.

Centralized control is characterized by having one main central controller responsible for collecting all the required data for decision-making from the various DERs by performing the required calculations and concluding the control actions for each unit at this point.

On the other hand, we have the decentralized control in which we have a local controller for each DERs unit, receiving only local information without being aware of any other system activity.

An interrelated power system is usually characterized by covering large geographical areas. This characteristic means a fully centralized approach is entirely infeasible due to the computation needs and communication needed. Simultaneously, a decentralized approach is not possible either due to its need for a minimum level of coordination and cannot be achieved by using only local variables. Therefore, cooperation between centralized and decentralized control schemes is found in means of a hierarchical control scheme that consists of three control levels: primary, secondary, and tertiary. These control levels vary in their (i) speed of response, (ii) infrastructure requirements. see figure 2.2

1. Primary Control

In local control, it is the first level in our hierarchy featuring the fastest response. It is at the first level; its control is based on local measurements and does not need communication. Given the speed requirements and reliance on

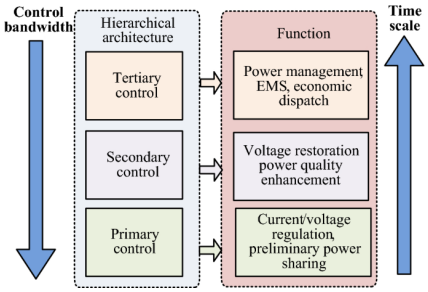


Figure 2.2: Hierarchy diagram

local measurements, islanding detection, inverter output control, and power-sharing balance are all at this level.

(a) Inverter Output Control This control usually contains the outer loop for voltage control and an inner loop for current regulation. Using PI controllers is the typical approach in designing the control loops supported with feed-forward compensation to enhance the current regulator performance; we will look at those control loops further in Chapter 3.

(b) Power Sharing Control A second stage within the primary control level is the power-sharing control concept, which we will cover in two indistinct theories:

i. PQ Control It is a public control that controls an inverter's voltage output by injecting the active and reactive power in cases the micro-grid cannot give voltage or frequency support .henceforth, and the micro-grid controller is not affected by the unstable voltage and frequency. Usually, when connected to the primary grid, it provides us by the reference frequency, unlike in private mode, it is given by another micro-grid operating on droop control.

ii. Droop Control The droop method is originally from the power balance of synchronous generators in interrelated power systems. A frequency and voltage deviation occur in our system when there is not inequity between the input mechanical power of the generator and output electrical active power, likely output reactive power.
Henceforth in this unit, if we drooped the frequency as a function of active output power, we can then share this power of total load among the various sources. Considering the relationship that dictates power transfer in a two inverter system, droop control applicability is apparent in figure 2.3

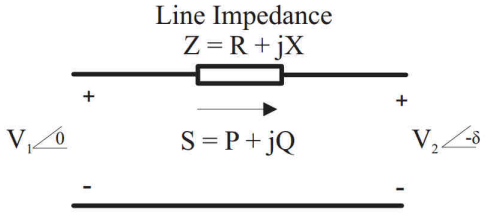


Figure 2.3: A system with two voltage sources

In droop control the relation between real power/frequency and reactive power/voltage can be expressed as:

$$W_0 = W^* - K_p(P_0 - P^*) \tag{2.1}$$

$$V_0 = V^* - Q_p(Q_0 - Q^*) \tag{2.2}$$

Where $w^*$ and $V^*$ are the angular frequency and voltage ,respectively ,and $w_0$ and $V_0$ are measured output frequency and voltage of DG system, respectively. The coefficient $K_P$ and $K_Q$ denote the droop coefficients and are determined by the following formulas:

$$K_P = \frac{\nabla f}{p} \tag{2.3}$$

$$K_Q = \frac{\nabla V}{Q} \tag{2.4}$$

$P-w$ droop characteristics are shown in figure 2.4a below while basic $Q-V$ droop characteristics is shown in figure 2.4b

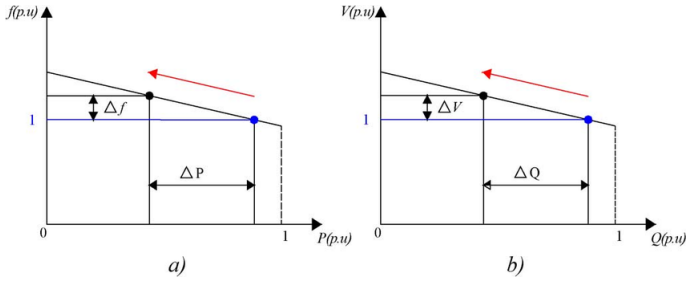


Figure 2.4: $P-w$ and $Q-V$ droop characteristics

Droop control eliminates the need for communication. Its control is based on local measurements, which is outstanding flexibility, in case we are guaranteed a balance between the supply and the demand there is not any need for local controllers. A further illustration will be conducted in Chapter 3

2. Secondary Control

   It is known as the Energy Management System (EMS) of the micro-grid, which is in charge of the security and reliability, and economic operation of the micro-grid in its dual-mode. This control level's performance gets more challenging as we switch to isolated mode(islanded) as there are high-variable energy sources, in which the unit dispatch command should be high at a rate enough to keep up with the unexpected changes of load and non-dispatchable DERs.

   The EMS works on finding the optimal and unit commitment (UC) and dispatch available DER units; its architecture has two main approaches: centralized and decentralized. With that being said, this level tends to be the highest level of control in the hierarchy for standalone micro-grids.

   The centralized approach's architecture contains a central controller that is enriched with the information of every DER and load in the microgrid and

network itself as well as forecasting system information. This central controller makes decisions using either online calculation of optimal operation or databases continuously updated and pre-built with information on proper operation.

Solving energy management related problems while guaranteeing a high level of autonomy for load and DER is one of the decentralized approach benefits. This autonomy is achieved through three levels: Distribution Network Operator (DNO), Microgrid Central Controller(MGCC), and Local Controllers(LC).

DNO controls the communication between the micro-grid and the distribution network and other microgrids, making it part of the tertiary control.MGCC supervise the operation of DERs and load within a micro-grid and in charge of their reliable and economical operation. At the same time, LC control DER units in decentralized architecture, an LC can communicate with MGCC and other LC to share knowledge.

3. Tertiary Control

   This control is the highest point in our hierarchical control level, and it works on setting the optimal setpoint based on the power system. It is usually in charge of coordinating multiple micro-grids interacting with one another within the same system and communicating the needs from the primary or host grid.

   It works by providing a signal to the secondary level at micro-grid and subsystems forming the full system. On the contrary, the secondary control coordinates internal primary control leading the primary control to function autonomously and react in predefined ways to identified signals [3].

## 2.2 Reinforcement Learning

### 2.2.1 Machine Learning Introduction

Gaining popularity in research and being the centre of technological advancements in all domains, machine learning is a field that focuses on providing data-driven, intelligent answers to research questions. We generally divide it into Supervised Learning, Unsupervised Learning, and Reinforcement Learning.

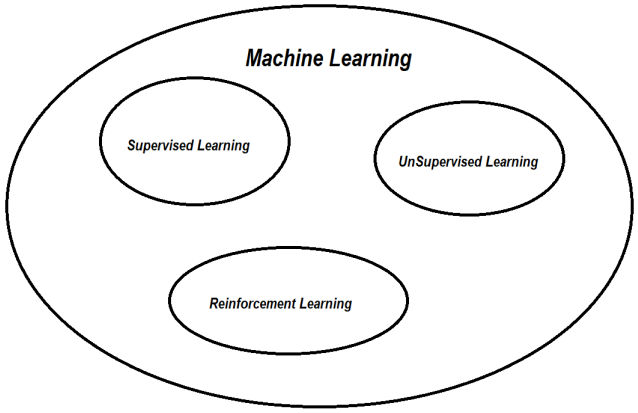


Figure 2.5: Machine Learning Sections

Supervised Learning (SL) is the domain where we try to map inputs to outputs using labeled data. The SL task is either a classification task or a regression task.

In classification, we use a particular object's features to predict its class (i.e., classifying a malignant or benign tumor), meanwhile in regression, where we use features of an object to try to predict another feature of that object, i.e., predicting the price of a house using features of size, location.

Unsupervised Learning (UL) uses unlabeled data to divide it into an unknown set of classes and extract specific structures from the data, ...etc. We divide UL into Parametric UL, where we assume probabilistic distribution of the data based on specific parameters. The mission is to try and get those parameters so that we can predict the future.

The other field of UL is Non-Parametric UL. Here, we make no assumptions about the data, and we only group the data into clusters with resembling features.

Though several problems are solvable using these two methods of learning, the real power and research start when we start talking about the opportunities Artificial Neural Networks ANNs provide. These are trials of modeling the process of thinking that lies inside a human brain.

A neural net is a collection of small processing units called neurons that receive input. It uses its predefined function to calculate a result and then output this result to another neuron or the user. Collecting a group of neurons, we create a neural net that can take multiple inputs and use complex functions to get a single output. We call each level of neurons a layer; we have an input layer, output, hidden layers.

Deep Learning DL is an application on ANNs with many levels of hidden layers typically more than three. We use this architecture to solve problems with increasing levels of complexity and individual requirements. The most famous divisions of deep

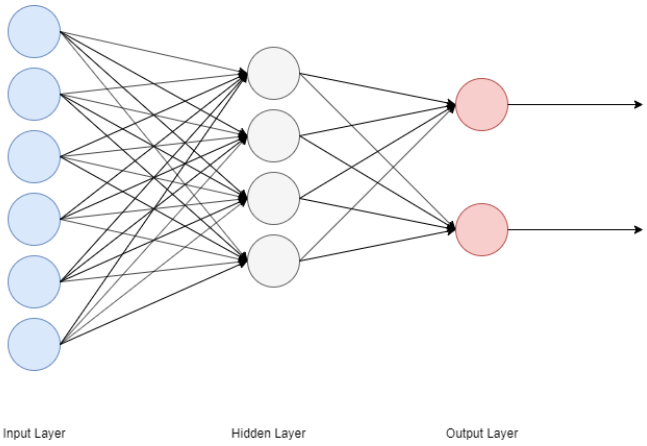


Figure 2.6: Simple Neural Network

learning are Convolutional Neural Networks CNNs and Recurrent Neural Networks RNNs.

CNNs use filters to tackle problems concerning pictures and images. These problems include object detections, image classification, and much more in-depth domain problems.

The focus of RNNs is time-varying data, i.e., data that require a time-based context to solve. The most important application of this is Natural Language Processing NLP, which needs text context data to understand general speech.

### 2.2.2 Reinforcement Learning Introduction

Reinforcement Learning RL is the third type of machine learning domain. It is learning what to do so to maximize a digital reward signal [see 4]. The problem is as one where an agent (i.e., a player in a game) is traversing an environment, and he takes actions and collects rewards as he goes.

We can describe the whole problem as an environment *env* that can be described as a state-space $S$ that consists of states $s$ that describe fully the world that can affect or be affected by the agent's decisions. The agent can take action $a$ from an action space where $a \in A$ that will change its state and receive a reward $r$ where $r \in R$.

This Mathematical representation means that an RL problem can be described as a Markov Decision Process MDP. In an MDP, we have the concepts of a reward function $R(s)$, which can map states to rewards achieved when reaching that state.

In an episodic process (one with a clear starting state and a final state), the total reward is the accumulation of each reward received through the journey traversing the environment until reaching the final state. This total reward is the value we are trying to maximize in our RL problem.

Another concept apparent in MDPs is the concept of a policy $\pi$, which defines the path (also known as trajectory $\tau$) that the agent will take during the episode. The policy $\pi(s)$ maps state-action pairs, i.e., what action to take when an agent is in state $s$. There are two types of policies, either a deterministic policy, one that the agent is told precisely what action to make when arriving at state s where $a = \pi(s)$. The other type of policy is a stochastic policy; here the agent is told in probabilistic values the probability of taking action a when in-state $s$ where

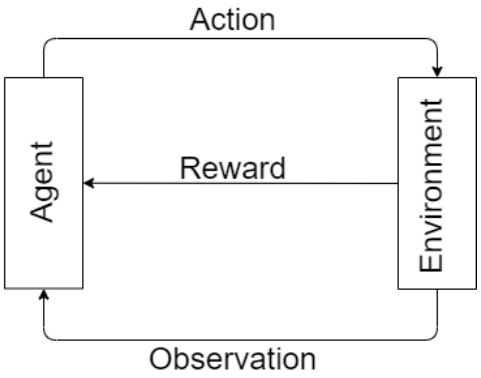


Figure 2.7: Reinforcement Learning Parts

$$\pi(a|s) = P[A_t = a|S_t = s] \tag{2.5}$$

Achieving the maximum reward possible implies that we will take the best possible policy to give the maximum reward at each step. This policy is called the optimal policy $\pi^*$, and accordingly, we can say that the main target of RL is to achieve an optimal policy for the environment in which we work. Finding this optimal policy requires a way to measure the optimality or goodness of a certain policy, and this can be done using a Value Function $V(s)$ which is the reward to accumulate over the future starting at current state $s$ [see 4]. This can be expressed as:

$$V_\pi(s) = E_\pi[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + ...|S_t = s] \tag{2.6}$$

where $V_\pi(s)$ is the set of rewards expected to be accumulated starting at state s and following the policy $\pi$. Using the definition of return $G_t$ which is given by:

$$G_t = R_t + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... + \gamma^{T-1} R_T \tag{2.7}$$

We could re-write $V_\pi(s)$ as:

$$V(s) = E[R_t + \gamma G_t|S_t = s] \tag{2.8}$$

Representing the environment is the model, which is a mapping of what the environment behaviour in response to the agent's action with $P^a_{ss'}$ describing the probability of arriving at state $s'$ when taking action a at state $s$ and $R^a_s$ being the expected reward for arriving at state $s$ taking action $a$ described as follows:

$$P^a_{ss'} = P[S_{t+1} = s'|S_t = s, A_t = a] \tag{2.9}$$

and

$$R^a_s = E[R_{t+1}|S_t = s, A_t = a] \tag{2.10}$$

Following, the optimal value function of a state $s$ following a policy $\pi$ in a number of steps $H$ is given by:

$$V^*_\pi = max_\pi E[\sum_{t=0}^{H} \gamma^t R^a_{s_t}|S_0 = s] \tag{2.11}$$

Moreover, finding the best possible value function of a given policy and then updating the policy to find the best policy is called the policy iteration. It can be given by the algorithm (Value Update or Bellman Update Backup):

1. Start with $V_0^*(s) = 0$
2. For $k = 1, 2, \ldots, H$:
   For all states $s$ in $S$:

$$V_k^*(s) \leftarrow max_a \sum_{s'} P_{ss'}^a (R(s, a, s') + \gamma V_{k-1}^*(s'))) \tag{2.12}$$

$$\pi_k^*(s) \leftarrow argmax_a \sum_{s'} P_{ss'}^a (R(s, a, s') + \gamma V_{k-1}^*(s') \tag{2.13}$$

One alternative to using these dynamic programming methods of value iteration and policy iteration is to introduce a way to learn as the episode is going, using an entire set of visits over states to come up with the value function of each state. These methods are called Monte-Carlo methods and their basic concept is that the value of a state $V(s)$ is calculated using a number of visits to the state in the episode $N(s)$ and total return on that state $S(s)$ by the following algorithm:

To evaluate $s$
1. Timestep $t$ when $s$ is visited in an episode:
2. Increment counter $N(s) \leftarrow N(s) + 1$
3. Increment total return $S(s) \leftarrow S(s) + G_t$
4. Value is given by $V(s) = S(s)/N(s)$
5. With $V(s) \rightarrow V_\pi$ as $N(s) \rightarrow \infty$

using this method and generalizing over all episodes:

For all states $S_t$ with return $G_t$:

$$N(S_t) \leftarrow N(S_t) + 1 \tag{2.14}$$

$$V(S_t) = V(S_t) + (G_t - V(S_t))/N(S_t) \tag{2.15}$$

For non stationary environments:

$$V(S_t) = V(S_t) + \alpha(G_t - V(S_t)) \tag{2.16}$$

### 2.2.3 Reinforcement Learning Solution Methods Introduction

Before delving into the specific solution method for RL, we need to specify the types of solution methods based on a couple of factors:

- Model-Free or Model-Based
- Value-Based or Policy-Based
- On-Policy or Off-Policy

- Discrete-Action or Continuous-Action

Model-Free methods do not make any assumptions about the environment nor the reward, meaning it receives an observation and takes action without any other processing than what is needed to get the action to take. A model-based method will try to create a model of the environment, predicting the next observation and reward after it takes action, it does multiple steps forward predictions to get the best action to take at this point and is mostly used on deterministic environments.

Policy-based directly optimize the policy to make the agent carry out the best action at every step. A probability distribution over available actions represents the policy. Value-based methods make the agent calculate the value of every action and take the best action at each step.

Off-policy methods can learn from historical data and previous observations and transitions, making the usage of a replay-buffer to store transitions viable, while on-policy do not have that ability.

### 2.2.4 Cross-Entropy Methods

One of the most basic solution methods when talking RL problems is cross-entropy methods. Their idea is pretty simple and builds on the intuition gained from looking at the RL main problem.

The idea of gaining as much reward as possible used as we replace all the agent complications with a non-linear trainable NN function, with the input being the observations from the environment $s$ and the output being the policy $\pi$. In practice, we represent the policy as a probability distribution over the actions $a$, making the problem a classification problem. The algorithm describing the method is:

1. Play N number of episodes with initial NN model and environment.
2. Calculate the total reward for each episode and set a reward boundary, usually at the $70^{th}$ percentile of rewards.
3. Discard all rewards below the boundary.
4. Train the NN model on elite episodes using state s as input and used actions as a target.
5. Repeat 1 until the target mean reward is reached.

### 2.2.5 Q-Leaning

Another value that's important do describe an RL environment is the Q-value $Q(a, s)$ which describes the quality of taking a certain action $a$ when in state $s$ following policy $\pi$ and is given by:

$$Q_\pi(s, a) = \sum_{s'} P^a_{ss'}(R^a_s + \gamma Q_\pi(s', a)) \tag{2.17}$$

Finding a solution to this value is inherently a temporal difference problem, which can be defined as a combination between Monte Carlo and Dynamic Programming in which we can learn directly from experience without needing a dynamics model (MC) and we update estimates based in part by other learned without waiting for final estimates. Working from equation 2.16, TD doesn't wait for a whole episode to

update $V(S_t)$ , it only waits for the next time step $t+1$ to update the value function using both $V(S_{t+1})$ and $R_{t+1}$ updating on the transition to $S_{t+1}$ using the equation:

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) \tag{2.18}$$

This is called one-step $TD$ or $TD(0)$, a special version of the complete $TD(\lambda)$ or $n$-step $TD$. The algorithm for implementing it is:

1. **Loop** for each episode:
   2. Initialize $S$:
   3. **Loop** for each step of the episode:
   4. $A \leftarrow$ action taken by $\pi$ for $S$
      Take action $A$, obseve $R, S$

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

$$S \leftarrow S'$$

   5. Untill $S$ is terminal

Q-learning is an off-policy $TD$ algorithm [first introduced 5] which made a major breakthrough in RL in which we use the update 2.19 in the normal $TD(0)$ algorithm.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_t, a) - Q(S_t, A_t)) \tag{2.19}$$

### 2.2.6 Deep Reinforcement Learning

These techniques work on using neural networks to approximate either policy parameters or to get $V$, $Q$ and $A$. In value function Deep-RL methods, more specifically $DQN$s [introduced in 6] we can look at the problem as one of regression, using a deep NN to approximate the value of $Q$ function, updating our $Q$ value using the tabular $Q$ learning update:

$$Q(S, A) \leftarrow (1 - \alpha)(Q(S, A) + \alpha(r + \gamma max_{a' \in A}(S', A'))) \tag{2.20}$$

When using this update in practice, we face some problems that limit the usability of the method. The problem of acting randomly or using the $Q$ approximation is solved using an epsilon-greedy method. We start with a completely random action and decaying the randomness with probability $\epsilon$ till a small value of 2% randomness.

Another problem is the requirement of SGD on training data, as it requires independent and identically distributed data $i.i.d$; this is not satisfied. We solve this problem by using a large replay buffer, adding new experiments, and pushing old ones out, allowing for independent data that are fresh.

The last problem is the similarity between states after each other, not allowing our NN to distinguish between them. We will solve this by using a target network, keeping a copy of the NN parameters using it for $Q(s', a')$. This network is updated periodically, making the network stable.

Our algorithm for $DQN$ is:

1.Initialize $Q(s,a), Q'(s,a)$ with random weights, $\epsilon = 1.0$ , and an empty replay buffer

2.Choose random action $a$ with probability $\epsilon$ ;otherwise, $a = argmax_a Q(s,a)$

3.Execute action $a$ , observe next state $s'$ and reward $r$

4.Store transition $(s,a,r,s')$ in replay buffer

5.Sample a random mini-batch of transitions from the replay buffer

6.For each entry in the buffer $(Transition)$, calculate target $y = r$ if the episode has ended at this step and $y = r + \gamma max_{a' \in A} \hat{Q}(s',a')$ otherwise

7.Calculate loss $L = (Q(s,a) - r)^2$

8.Update $Q(s,a)$ using $SGD$

9.Every N steps copy parameters from $Q$ to $\hat{Q}$

10.Repeat 2 till convergence

### 2.2.7 Policy Gradients Methods

These can learn a parametrized policy $\pi_\theta$ that can select an action without returning to a value function. It can be used to learn policy parameters but not to select exact actions. Updating the policy parameters can be achieved either using gradient-based or gradient-free methods to maximize expected return. Policy gradients update the policy parameter on each step in the direction of an estimate of the gradient of performance compared to the policy parameter. Given the trajectory $\tau$, which is the set of state action reward sequences, we can define a policy parameter performance:

$$J(\theta) = E[R(\tau)] \tag{2.21}$$

It's obvious that we need to maximize $J$ in order to find the optimal $\theta^*$. Using gradient descent on this problem (the most basic machine learning idea) we find that:

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t) \tag{2.22}$$

The far right of the equation can be approximated to:

$$\nabla E_{\pi_\theta}[R(\tau)] = E_{\pi_\theta}[R(\tau) \nabla_\theta \log \pi_\theta(\tau)] \tag{2.23}$$

Using an expansion for $\pi_\theta(\tau)$ we can arrive at:

$$\nabla E_{\pi_\theta}[R(\tau)] = E_{\pi_\theta}[(G_t - b)(\sum_{t-1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t))] \tag{2.24}$$

The equation tells us that to successfully update the parameters, and we do not need a model, we can only use state-action rewards to make a useful update. We can solve the expectation in the equation by sampling a large number of trajectories while replacing $R(\tau)$ with $G_t$ removes the variance that appears in the standard equation. The baseline $b$ is used to reduce the estimate's bias, $b$ must be independent of the parameters, and a good baseline makes use of the state-value current state. Equation 2.24, without adding a baseline is called $REINFORCE$, which is the classic policy gradient algorithm.

The steps for $REINFORCE$ are as follows:

1. Initialize Neural Network with random parameters.
2. Play $N$ episodes saving transitions $(s, a, r, s')$.
3. For each timestep, $t$, in every episode, $k$, calculate the discounted total reward for that step $Q_{k,t} = \sum_{i=0} \gamma^i r_i$.
4. Calculate the loss function for all transitions $\mathcal{L} = -\sum_{k,t} Q_{k,t} \log(\pi(s_{k,t}, a_{k,t}))$.
5. Perform $SGD$ update of weights minimizing the loss.
6. Repeat 2 until converged

Actor-critic methods combine policy gradients with model fitting, and we use an actor to model the policy and a critic to model the value function $V$. By introduce a critic, we reduce the number of samples to collect for each policy update; we do not collect all samples until the end of an episode. The idea of variance reduction is essential to Actor-critic methods, and we will look at the Advantage Actor Critic ($A2C$) method.

In Actor-critic the idea is making the baseline state-dependent, but as we know that $Q(s, a) = V(s) + A(s, a)$ with $A(s, a)$ being the advantage, we use $V(s)$ as the baseline and add a network to estimate it's value as shown in figure 2.8

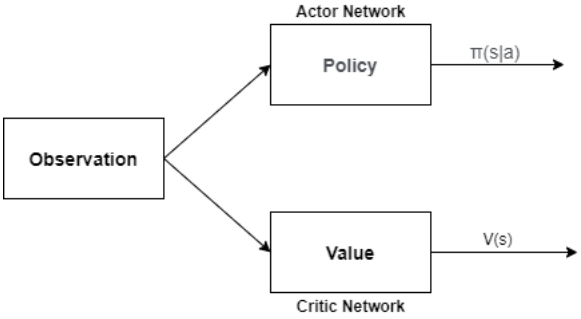


Figure 2.8: A2C Actor and Critic networks

The steps for $A2C$ are as follows:

1. Initialize network parameters $\theta$ with random values.
2. Play $N$ steps in the environment using the policy $\pi_\theta$ and saving $s_t, A_t$ and $r_t$.
3. $R = 0$ if the end of the episode is reached or $V_\theta(s_t)$.
4. For $i = t - 1....t_start$(note that steps are processed backwards):

   $R \leftarrow r_i + \gamma R$

   Accumulate the policy gradients: $\partial\theta_\pi \leftarrow \partial\theta_\pi + \nabla_\theta \log \pi_\theta(a_i|s_i)(R - V_\theta(s_i))$.

   Accumulate the value gradients $\partial\theta_v = \partial\theta_v + \partial(R - V_\theta(s_i))^2/\partial\theta_v$.

5. Update the network parameters using the accumulated gradients, moving in the direction of the policy gradients, $\partial\theta_\pi$, and in the opposite direction of the value gradients, $\partial\theta_V$.

6. Repeat 2 until convergence.

Here we have talked generally about most discrete action methods that are relevant to our work; We will talk about continuous action methods whether they are only for continuous actions or can work in both types of actions in the next chapter when we explain our project methodology.

## 2.3 Related Work

We mentioned earlier that the electricity demand is increasing and that it should balance with the constant need to limit global warming, that is why we introduced microgrids. The unidirectional design of national power grids did not allow for surplus microgrid generated power to be redistributed and sold to the primary national grid, so the process of energy trading was devised to accommodate for this problem in low voltage networks between microgrids.

In some cases, the microgrid demand is satisfied through the traditional utility grid when the connection is on-grid. Nevertheless, that means we will be paying an extra cost for them to overcome that shortage. Microgrids stand for economic optimality, which leads us to have efficient backups at reasonable and profitable cost and solve some of the technical problems we mentioned. These problems include better power quality, reduced voltage fluctuations, and a reliable system that is not affected by the utility grid outage through energy trading.

Understanding the problems facing microgrids takes us to the core of our research, the "energy transition between microgrids." The term energy trading is understood as the importing and exporting of energy in a market of retailers, producers, and vendors, including the large industrial consumers in the utility grid. It was then reinterpreted as the local energy trading that happens amongst users in the microgrid.

Energy trading can be approached from different perspectives. We can refer to it as an optimization problem where we can look at it as we look to the microgrid control either through centralized or decentralized approaches. In centralized, we have one central controller responsible for solving this problem, where it looks into a way to minimize the generation and the transportation cost of the microgrid. In contrast, we have a decentralized approach that looks into studying all the participants and the benefits that fall for each inclination.

In energy trading, what affects one affects all that is any action performed by any participant in the market will affect all. That is why we got introduced to the Game Theory (GT) technique in energy trading that in itself has a different concept to cover or try to solve the energy trading problem. GT acts in competitive situations where it takes into account that one's strategy will affect all the other strategies. We mostly use it on the decentralized structure of our microgrid, which makes it easier to check for each one's behavior. GT games are classified into Direct and Indirect games where one aims to find the ideal policy, and the other is concerned with planning a game that satisfies particular objectives. The former does not have any effect on energy trading so far. We will focus on direct games such as non-cooperative games in which each individual is looking for their benefit.

The work of Pilz [7] covers many studies regarding GT energy trading with its different properties. We find one study taking energy storage with the background of the schedule and reducing the peak to the average ratio where they plan a cost function under certain conditions that lead the system to be balanced. We then assume that each consumer's total load is the sum of the external power delivered from the primary grid. After setting up, he then proposed two different approaches, one is a static non-cooperative game where the utility sets a cost function, and the player plays a scheduled game in order for him to minimize the respective cost. Here we have a user with the advantage of selling energy back to the utility grid,

known as a reverse peak. A second game was introduced that takes the utility grid as part of the game (participant) and adjusts the prices and schedules the trade a typical leader-follower structure defined by the "Stackelberg game." which proves that Stackelberg equilibrium is equivalent to minimizing the peak-to-average ratio.

Another study takes a look at situations where the traditional power station could not meet the high demand at some point, so it buys the needed energy from energy consumers (electric vehicles, renewable energy farms, and any participants involved with the central power station as an individual). The researchers proposed a non-cooperative Stackelberg game where we do not deal with each component alone. Instead, they operated a solution that serves the social benefit assuring that each component benefits from participating in energy trading. They introduced a price model where the price can differ for different energy consumers. Henceforth, the authors applied an iterative algorithm to minimize the cost for the central power station and, at the same time, maximize the sum of utility functions of energy consumers.

Another research covered the transition of energy among the MGs. The trade does not happen directly with each other but instead tries to trade surplus energy with the market and request the deficiency as well. This multileader-multifollower Stackelberg game proposed, the sellers act as leaders and the buyers as followers in which the surplus energy is proposed by the leaders to the followers proportionally to the bids each buyer has placed. This method leads us to know that the best solution for this scenario depends on bids given and the number of players in the game. Because of the expanding rivalry between the purchasers, the worth monotonically diminishes when the quantity of purchasers increases. Simultaneously, the aggregate of the utility qualities for the dealers' increases, since more costumers permit them to sell more.

Another Stackelberg seller-buyer structure among MGs was taken into consideration as the ones before. However, to make the model more expressive, the author encompasses the known structure to the Bayesian game. In this type of game, our knowledge is incomplete, and we do not have full awareness of the game aspects and players' states, meaning that each player is private about their information. In this case, we take the players as normal or abnormal, the emergency state in which the sellers are less profound to sell energy and value the stored energy. From the buyers' point of view, they tend to bid more to ensure the requested energy delivery. We build on the last study by proposing a communication link between respective MGs where a weighting variable is used to express the relation between them. Precisely the conditional probability distribution over the condition of the player is classified as a two-stage technique. In stage one, each Mg estimates the state based on the players' given messages; the second stage updates the estimates based on information gathered from the close neighbors in the structure, looking into increasing the trust within the network showing and the partial trusted information. In the end, a debate is held questioning whether this will increase the power quality but was left for further work.

Unlike the late researches here, the central unit does not only communicate, but it works as a distributor or gatherer for the energy that is traded among the MGs. Also, no scheduling scheme is proposed to pay the sellers. By providing energy to the system, the respective MG collects points that increase the contribution value. If this MG runs into a deficit of energy, its high contribution value will give it a more

significant chunk of energy given by the rest. The distributor sets that in order to maximize the social welfare function. Knowing the distribution mechanism, the game here deals with the remark of how much energy to request directly proportional to this and inversely proportional to the contribution value it gets. Furthermore, each buyer is given a stage in the queue in which h should try to be served earlier to minimize what is requested from the utility grid. In this case, not enough surplus energy to serve, we have nash equilibrium property that even if participants deviate from it, the other does not be impacted negatively.

For security reasons, all correspondences are composed of the central unit. Seen from any of the MGs, this prompts a fragmented data game, as no one thinks about the systems and settlements of the others. All the more specifically, the author divides the MGs into merchants and purchasers and plans a two-phase Stackelberg game in which every one of these gatherings attempts to find their best activities by methods or reinforcement learning algorithms. The same classification and giving principle based on proportionality are added; this implies there are two utility capacities, one for each group of buyers and sellers without the knowledge of the other players. It shows that the learning algorithm here converges to the best reply, which is the same as the solution to the sellers' and buyers' optimization, respectively. In comparison, the iteration solutions earlier this take 100 times more to converge to Nash Equilibrium.

If we looked at the energy exchange proficiencies, we find some focused on selling the energy back to the conventional grid. At the same time, others took into account two types of participants, the sellers and the buyers. In most cases, the energy transition happens in secondary structure as it does not happen between individuals but happens through an operator, a third party that leads us to not fully decentralized scenario.

On the other hand, most of the utility functions taken by the games are focused on the monetary function perspectives from looking into the cost of storing the energy to the cost of transmission of energy between different parties. On the other hand, the utility function did not look into the price function but instead looked into the ratio between allocated energy and requested energy. Other approaches acted upon an auction algorithm where the buyers and the sellers are balanced in the market.

In all scenarios, the customers were referred to as sellers or buyers despite having surplus energy or deficit. In the above models, there was a shortage in models that combine a high-quality demand analysis with the RE generation in energy trading. Most of this researcher proposed what they call blue-sky approaches with "reinforcement learning" and "contribution-based" energy trading. Furthermore, all those authors lacked in the long-term assessable suggestions opposing the merely one-day ahead analyses in energy trading. We use a reinforcement learning algorithm that works on solving the situation without prior information about the microgrid. As achieving Supply-Demand equilibrium is complicated when considering the non-formality of the RES's and many studies were proposed in market-based energy trading among microgrids to utilize DERs across the network fully [8]

The idea of microgrids replacing conventional power grids in rural areas has been the subject of research. B. M. Sivapriya et al. [9] worked with the problem of microgrid design using the center of moment approach to the placement of PV panels on the network providing case studies for their designs on villages in India.

Murenzi et al. [10] worked in Africa, introducing Microgrids as a viable method to electrify sub-Saharan Africa. They showed that in a typical Rwandan village, the installation of a microgrid with PV, batteries, and a micro-hydro is a better financial alternative than extending the national power grid transmission to reach the village.

Applications of Reinforcement Learning in smart grids and microgrids vary, A smart building energy management algorithm [11] that uses a Markov decision process to model the smart building. The algorithm controlled included interactions with the utility grid and internal RES. The algorithm used Q-Learning to make decisions on energy dispatch actions achieved better energy costs in the building against multiple pricing policies. Mocanu et al. [12] created a deep belief network that improved the performance of standard reinforcement learning algorithms. They namely worked on SARSA and Q-learning, in the context of predicting energy in a smart building, the algorithm can generalize a learned behavior model into any other building without any specific history of that building. Leo Raju et al.[13] proposed a model-free reinforcement learning algorithm (Q-learning) to solve the optimal dispatch problem, which concerns finding the best combination of available power resources to provide the required load with minimal cost. Their algorithm converged to the optimal solution and provided adaptability in dynamic situations and unforeseen load management.

Fabrice et al.[14] proposed an algorithm to fully control power flow between a multi-storage Microgrid mapping it as a Multi-Agent System (MAS) and using Multi-Agent Reinforcement Learning to solve the problem. They produced results showing that a centralized control; unit for the microgrid is not needed. The algorithm can achieve the minimal cost of drawing power from the primary grid and achieve most grid independence. Finally, Xiao et al.[15] proposed an energy trading game between different microgrids intending to achieve the Nash Equilibrium without knowing the generation and load demand of the other microgrids using a DQN-based energy trading strategy achieving an improvement of 22.3% in the utility of the microgrid.

# Chapter 3

# Methodology

## 3.1 Microgrid Methodology

## 3.2 Reinforcement Learning Methodology

Working based on the Microgrid simulation, and due to computation constraints and compatibility limitations with Matlab, we create a different simulation for our microgrid. We will use the same configuration and data in an environment built using Python programming language. This environment is more comfortable to work in and friendly to RL algorithm implementations.

### 3.2.1 Python and necessary libraries

*Python* is the leading programming language in machine learning; it is high flexibility, speed, and support make it a perfect language for research in this field. *Python* has several libraries designed for mathematics, array manipulation, and data processing. Two of the most important libraries are $Numpy$ and $Pandas$. $Numpy$ is a library used for linear algebra and array calculus. It supports several functions that are optimized for the highest performance and speed in array operations. $Pandas$ is a library for data manipulation. $Pandas$ works with the concept of a $DataFrame$, which represents a table of different types of data; it supports operations, analysis, and modification of data, which is helpful for large data files, which are the cornerstone of machine learning.

### 3.2.2 Pytorch

When working with deep learning problems, the concept of a $Tensor$ becomes essential; it is the building stone for all DL libraries. A *tensor* is simply a multidimensional array or an array of any shape. *Pytorch* is one of the most popular DL libraries and is more or less a tensor operations library. *Pytorch* has *Cuda-GPU* support, which, as mentioned before, the advancements in gaming and $GPUs$ paved the way for DL to become widespread and usable since 2012. *Pytorch* can perform its operations either on $CPU$ or on $GPU$ for higher-level tensors. Transferring the $CPU$ tensors to $GPU$ is done using the $.to(device)$ command, it is used to transfer between both devices changing *device* to *cuda* or *cpu* depending on usage.

*Pytorch* supports dynamic graphs when creating NNs, which gives the programmer freedom when creating their networks; it also supports automatic gradient calculations, which is vital to optimizing NNs. *Pythorch* will calculate the network gradients, backpropagate the values, and automatically apply the new parameters to the network. *Pytorch* also has built-in loss calculation methods, which include Mean Squared Error Loss ($MSELoss$), Binary Cross-Entropy Loss ($BCELoss$), and Cross-Entropy Loss. It also has multiple optimizers, including Stochastic Gradient Descent $SGD$, $RMSprop$, $Adagragd$, and the famous $Adam$ optimizer.

### 3.2.3 OpenAI GYM

This library is the most central library for RL researchers and practitioners; it provides a complete set of classes and functionalities to create, test, and evaluate RL algorithms using its built-in functions. It provides a set of working environments for learning and testing algorithms and the ability to create custom environments that make use of $GYM$'s functionalities.

The concept of spaces is essential in $GYM$ as it describes the set of values that our environment or action can take; we have three different types of spaces:

1. Discrete space: this space a mutually exclusive set of numbers for an item, if we declare a Discrete space with six values, then these values are zero through 5.

2. Box: an N-dimensional tensor of rational numbers, we set a lower value, higher value, and the shape of the tensor and its data type, and the values will take any number of values between high and low with the provided shape.

3. Tuple: a space that combines both types of spaces, we can have a box of discrete and Box type subspaces

$GYM$ provides us with building block for an environment that must include the following:

1. Action space: the set of value that the action can take, either a discrete, box or Tuple space

2. Observation space: the set of values that the observation can take; it is the environment; it can be either a discrete, box or tuple space.

3. Step function: A detailed description of the environment's reaction to the action taken by the agent, after its execution, it returns four values:

   (a) observation: the state of the environment after an action is executed.

   (b) Reward: Reward given to the agent after executing the action.

   (c) Is Done: Boolean describing weather the current episode has ended.

   (d) Info: any additional info by the environment.

### 3.2.4 MicroGrid Enviroment

We used these ideas to create our environment; this environment consists of many parts that all are joined together to create our single microgrid. The environment we are working on consists of three microgrids, the main grid controlled by our agent and two other microgrids for trading. A single microgrid consists of loads, battery, and generation.

#### Battery

It is a storage unit with a max capacity, a discharge coefficient, and a charging rate. We control the battery using the supply and charge methods, which both take in an amount of electricity and charge the battery or supply the microgrid or any other microgrid with which we are trading.

#### Generation

The energy generated in a single microgrid has two sources; those are wind generation and solar generation. We will sum up both generations to get our total generation.

#### Load

The loads for a single microgrid are the village's basic building blocks; those are houses, schools, mosques, health centers, and water pumps. Each of our grids has a different configuration of these types of loads; we pass the load elements to our environment constructor. The loads are given at hourly intervals; each data point is that hour of the day percentage of that element's maximum load.

#### MicroGrid

All the parts mentioned above come together to create a single microgrid. When we initialize a microgrid, we create its loads, battery, and generation. We also define a method that returns its state, which is its total load, total generation, and remaining capacity in the battery at any given time.

#### Pricing and environment interaction

We set an optimal and worst limit for a microgrid to buy/sell energy. For a buyer, the maximum price that it will buy energy at is the network price, which is the price of electricity when provided by the national grid. The optimal price to buy electricity at is any price less than the microgrid's generation cost. This cost is the kWh price that will give us a return of investment at the time we set. Meanwhile, for a seller microgrid, the minimum price it will sell at is its generation cost while the optimal price is higher than the network price.

The environment observation is an array that contains the total load of the microgrid, the total generation, and the current capacity of the battery at the current date. It also contains the one-time price that a transaction occurred on. The action that the agent will take consists of four parts, the type of the action, whether its buy, sell or hold, the target microgrid, the amount we want to buy/sell, and the

price for the transaction. Given the type of action, we see that the action is not discrete as it can take any value in the range we specified. Therefore we will take a look at the continuous action space method in deep reinforcement learning.

### 3.2.5 Continuous Action Space

Modern problems in RL are of a different type than what we looked at; the continuous action is a more realistic type of action when looking at the problems that RL works in; robotics and non-bounded environments. These actions can be of different values in a specified range, that being a controller for a steering wheel, a robot joint angle, speed pedal, and in our case, a price and an amount to trade from a high and low value space.

As we will be working in the gym environment and we need to specify our action space, and as the action is continuous, we will use the *gym.spaces.Box* space and specify high and low values as well as a shape for the action. What we pass to the *step*() function is an array of the shape specified.

#### Advantage Actor Critic continuous

The first solution method we will look at is one that we have seen before, Advantage Actor-Critic $A2C$. We can edit this method to give us our desired output of an array of actions, the actor-network instead of giving a probability distribution of the actions to take, will give us the exact action to take. This determinism is a problem though, as this deterministic policy hinders exploration, we will solve it by substituting the actor-networks deterministic values by parameters of a Gaussian distribution, which are the distribution's mean and variance as we know this normal distribution is given by:

$$f(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \frac{(x-\mu)^2}{2\sigma^2} \tag{3.1}$$

We use it to get our policy as follows:

$$\log \pi_\theta(a|s) = -\frac{(x-\mu)^2}{2\sigma} - \log \sqrt{2\pi\sigma^2} \tag{3.2}$$

#### Deep Deterministic Policy gradients

This method is an off-policy algorithm, and a natural progression on A2C and the first difference is that the policy is deterministic, meaning that the action is given in itself, not as a distribution, the same observation will give the same action always. The actor and critic network are as follows

- Actor-Network: takes a state, gives an N-dimensional array representing the action,$\mu(s)$
- Critic Network: takes state and action and the Q value for those pair $Q(s, a)$ or $Q(s, \mu(s))$

Here $Q(s, \mu(s))$ depends on both $\theta_\mu$ and $\theta_Q$. We need to maximize the output of this network.

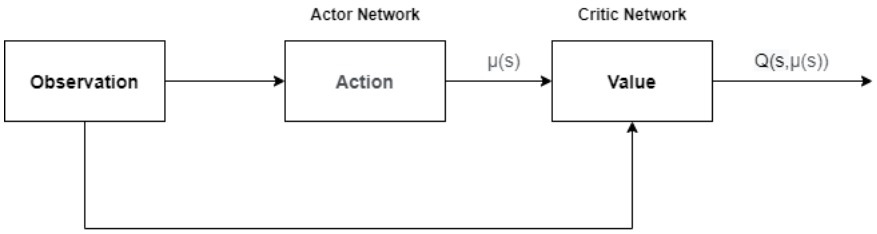


Figure 3.1: DDPG Actor and Critic networks

Silver et al.[16] proved that to improve this network, we only need to calculate the gradients of Q, which is given by

$$\nabla_Q Q(s,a) \nabla_{\theta_\mu}(s) \tag{3.3}$$

In $DDPG$, we can calculate this gradient of Q making the whole system differentiable, so we can optimize the whole system end-to-end using $SGD$ and using the Bellman equation to calculate the approximation of Q then we minimize the $MSE$.

What remains is the problem of exploration; we can solve it by adding noise to the policy's output before passing to the step function. The type of noise used in the original paper is Ornstein–Uhlenbeck process noise, a time-correlated noise, but later implementations suggest that an uncorrelated, mean-zero Gaussian noise works perfectly well and is much simpler to implement.

The algorithm is as follows:

1. Initialize policy parameters $\theta$, Q-function parameters $\phi$ empty replay buffer $D$.
2. Set target parameters equal to main parameters $\theta_{targ} \leftarrow \theta$, $\phi_{targ} \leftarrow \phi$.
3. repeat:
4. Observe the state $s$ and select action $a = clip(\mu_\theta(s) + \epsilon, a_low, a_High)$ Where $\epsilon \sim N$
5. Execute $a$ in the environment.
6. Observe next state $s'$, reward $r$ and $d$ done signal.
7. Store $(s, a, r, s', d)$ in replay buffer $D$.
8. **If** $s'$ is terminal, **reset** environment state
9. **if** it's time to update **then**
10. **for** however many updates **do**
11. Randomly sample a small batch of transitions $B = (s, a, r, s', d)$ form $D$
12. Compute target

$$y(r, s', d) = r + \gamma(1 - d) Q_{\phi_{targ}}(s', \mu_{\theta_{targ}}(s'))$$

13. Update the Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_\phi(s,a) - y(r,s',d))^2$$

14 Update policy by one step of gradient ascent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} Q_\phi(s, \mu_\theta(s)$$

15 Update the networks with

$$\phi_{targ} \leftarrow (p)\phi_{targ} + (1-p)\phi$$
$$\theta_{targ} \leftarrow (p)\theta_{targ} + (1-p)\theta$$

16 **end for**
17 **end if**
18 **Until** convergence

**Trust Region Policy Optimization**

The idea behind these on-policy family of methods is to improve the policy by taking large steps to improve the performance[17]; each step is constrained by the closeness between the new policy and the old one expressed as KL divergence which is explained as a distance between probability distributions.

With $\pi_\theta$ a policy with parameters $\theta$, $TRPOupdateis$ :

$$\theta_{K+1} = argmax_\theta \mathcal{L}(\theta_K, \theta) \quad s.t.\ \overline{D}_{K,L}(\theta||\theta_k) \leq \delta \tag{3.4}$$

$\mathcal{L}(\theta_k, \theta)$ is the surrogate advantage, a measure of how $\pi_\theta$ performs in resprct to $\pi_{\theta_K}$

$$\mathcal{L}(\theta_k, \theta) = E_{s,a\sim\pi_{\theta_K}}[\frac{\pi_\theta(a|s)}{\pi_{\theta_K}(a|s)} A^{\pi_{\theta_K}}(s,a)] \tag{3.5}$$

and $\overline{D}_{K,L}(\theta||\theta_k)$ is average KL divergence between policies across states visited by old policy

$$\overline{D}_{K,L}(\theta||\theta_k) = E_{s\sim\pi_{\theta_K}}[D_{K,L}(\pi_\theta(.|a)||\pi_{\theta_K}(.|s))] \tag{3.6}$$

Using approximation we get

$$\mathcal{L}(\theta_k, \theta) \approx g^T(\theta - \theta_K) \tag{3.7}$$

$$\overline{D}_{K,L}(\theta||\theta_k) \approx 0.5(\theta - \theta_K)^T H(\theta - \theta_K) \leq \delta \tag{3.8}$$

This can be solved using the lagrangian duality, and using a backtracking coefficient $j$ that satisfies kl divergence as $\alpha \in (0,1)$

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{g^T H^{-1} g} H^{-1} g} \tag{3.9}$$

Using conjugate gradients, we solve $H^{-}1$ as

$$Hx = \nabla_\theta((\nabla_\theta \overline{D}_{K,L}(\theta, \theta_K))^T x) \tag{3.10}$$

As $TRPO$ is a stochastic policy, the agent explores by sampling the action based on the latest stochastic policy, training, and initial conditions that define randomness. As the training goes, the policy becomes less random and exploits the rewards it has already found.

The algorithm is as follows:

1. Initialize policy parameters $\theta_0$, and value function parameters $\phi_0$.
2. Hyper-parameters: KL-divergence limit $\delta$, backtracking coefficient $\alpha$, maximum number of backtracking steps $K$.
3. **for** $k = 0, 1, 2, ...$**do**
4. Collect set of trajectories $\mathcal{D}_k = \tau_i$ by running policy $\pi_k = \pi(\theta_k)$ in environment.
5. Compute rewards-to-go $\hat{R}_t$
6. Compute advantage estimates, $\hat{A}_k$ (Using any method of advantage estimation) based on the current value function $V_{\pi_k}$
7. Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau in \mathcal{D}_K} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t$$

8. Use conjugate gradient algorithm to compute

$$\hat{x}_k \approx \hat{H}_k^{-1} \hat{g}_k$$

Where $\hat{H}_k^{-1}$ is the Hessian of the sample average KL-divergence.

9. Update the policy by backtracking line search with

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{\hat{x}_k^t \hat{H}_k \hat{x}_k}}$$

Where $j \in 0, 1, 2, ..., K$ is the smallest value, which improves the sample loss and satisfies the sample KL-divergence constraint.

10. Fit value function by regression on mean-squared error

$$\phi_{k+1} = argmin_\phi \frac{1}{|\mathcal{D}_K T|} \sum_{\tau \in \mathcal{D}_K} \sum_{t=0}^{T} (V_\phi(s_t) - \hat{R}_t)^2$$

Typically via some gradient descent algorithm.

11. **end for**

### Proximal Policy Optimization

This method is motivated by the same idea behind $TRPO$; that is why we talked about it, although we are not using it. The idea of making the biggest possible improvement step on a policy without causing performance collapse. $PPO$ is a first-order method that uses simpler methods to achieve better performance[18] than

$TRPO$. We will be using $PPO-Clip$, a method that does not have KL-divergence; it relies on specialized clipping in the objective function to remove the incentive for a new policy to move away from the old policy.

The policy is updated using

$$\theta_{k+1} = argmax_{\theta} E_{s,a\sim\pi_{\theta_k}}[L(s,a,\theta_k,\theta)] \tag{3.11}$$

where

$$L(s,a,\theta_k,\theta) = min(\frac{\pi_\theta(s|a)}{\pi_{\theta_k}(s|a)} A^{\theta_k}(s,a), clip(\frac{\pi_\theta(s|a)}{\pi_{\theta_k}(s|a)}, 1-\epsilon, 1+\epsilon) A^{\theta_k}(s,a)) \tag{3.12}$$

$\epsilon$ us a small hyper-parameter denoting how far a new policy can move far away from the old one. Simplifying the objective expression we can arrive at

$$L(s,a,\theta_k,\theta) = min(\frac{\pi_\theta(s|a)}{\pi_{\theta_k}(s|a)} A^{\theta_k}(s,a), g(\epsilon, A^{\theta_k}(s,a))) \tag{3.13}$$

where

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0. \end{cases} \tag{3.14}$$

If the advantage is positive, meaning that the objective will increase if the policy increases, then the objective is bounded by a maximum value of $(1+\epsilon)A^{\theta_k}(s,a)$ limiting how far it can move from the previous policy. Meanwhile, if the advantage is opposing, meaning the objective will increase if the policy decreases, the objective is bounded by $(1-\epsilon)A^{\theta_k}(s,a)$ also limiting how far away the new policy can change from the old one. This intuition is behind using clipping as an excellent way to limit policy steps.

$PPO$ is an on-policy method; it explores by sampling the action based on the stochastic policy. Same randomness factors that affect $TRPO$ affect $PPO$.

The algorithm for $PPO$ is as follows:

1. Initialize policy parameters $\theta_0$ and value function parameters $\phi_0$.
2. **for** $k = 0, 1, 2...$ **do**
3. Collect trajectories $\mathcal{D}_k = \tau_i$ by running $\pi_k = \pi_0$.
4. Compute reward-to-go $\hat{R}_t$
5. Compute Advantage, $\hat{A}_t$ based on current value function $V_\phi$
6. Update the policy by maximizing $PPO-Clip$ objective

$$\theta_{k+1} = argmax_{\theta} \frac{1}{|\mathcal{D}_k|} \sum_{\tau in \mathcal{D}_K} \sum_{t=0}^{T} min(\frac{\pi_\theta(s|a)}{\pi_{\theta_k}(s|a)} A^{\theta_k}(s,a), g(\epsilon, A^{\theta_k}(s,a)))$$

using stochastic gradient ascent with Adam

7. Fit the value function by regression on mean-squared error

$$\phi_{k+1} = argmin_{\phi} \frac{1}{|\mathcal{D}_K T|} \sum_{\tau \in \mathcal{D}_K} \sum_{t=0}^{T} (V_\phi(s_t) - \hat{R}_t)^2$$

Typically via some gradient descent algorithm.

7. **end for**

### 3.2.6 Implementation of selected algorithms

In our work, we worked with an on-policy algorithm and an off-policy algorithm. We chose $DDPG$ and $PPO$ as our selection algorithms as they are both give excellent results in continuous action environments with $DDPG$ great results in stable environments than most other algorithms, and $PPO$ being a fast, stable algorithm with comparably better results than most on-policy algorithms[19].

We used the implementations created by Google's open AI labs at their RL section called *SpinningUp*[20]. The site provides implementations for most of the algorithms discussed in this thesis. We didn't perform much tweaking to the implementation, we changed the number of steps in each epoch and the maximum epoch number, and we transferred the networks to the $GPU$ using the methods mentioned above.

### 3.2.7 The Data

The effort to find data for this project was long and tiresome. We needed to find accurate solar irradiance data for specified locations in western Sudan and wind speed data for the same places. For the solar data, we used PVGIS[21], a tool created by the European Commission Joint Research Centre, which provides, among other things Full-time series of hourly values of both solar radiation and PV performance which we used to extract PV solar irradiance data to use in our MG simulation. For the wind speeds data, we used a tool developed by the Sudanese wind energy research center. The tool[22] provided different wind speed readings at different highest and for an extended period.

We also searched for load consumption data to create the load profile for our microgrids. The U.S. Department of Energy has a project called National Renewable Energy Laboratory (NERL)[23], where we could use the Rural African Load Profile Tool to get load profiles for our region and our loads.

# Appendix A

# Code

This contains codes for both the **MATLAB** microgrid simulation and the python simulation enviroment

## A.1 MATLAB code

## A.2 Python code

This following code is the compelete code for creating the enviroment listed in **chapter 3**

```
#Imports
import pandas as pd
import numpy as np
from matplotlib import pyplot
import gym
from gym import spaces
import random
import math

#CONSTANTS
#MAXIMUM LOADS
SCHOOL_MAX_LOAD = 6.012
HOUSE_MAX_LOAD =  5.678
MOSQUE_MAX_LOAD = 4.324
HEALTH_CENTER_MAX_LOAD = 5.8
WATER_PUMP_MAX_LOAD = 0.77
#MICROGRIDS PARAMETERS
UM_BADER_LOAD_PARAMETERS = [70, 1, 2, 1, 2]
UM_BADER_MAX_LOAD = 500
UM_BADER_BATTERY_PARAMETERS = [500, 0.02, 300, 0.3]
HAMZA_ELSHEIKH_LOAD_PARAMETERS = [50, 1, 1, 0, 1]
HAMZA_ELSHEIKH_MAX_LOAD =350
HAMZA_ELSHEIKH_BATTERY_PARAMETERS = [350, 0.02, 200, 0.3]
TANNAH_LOAD_PARAMETERS = [45, 0, 1, 0, 1]
TANNAH_MAX_LOAD = 300
TANNAH_BATTERY_PARAMETERS = [300, 0.02, 150, 0.3]

#DISTANCES AND PRICES
distances = {"Um_Bader_Tannah": 10, "Um_Bader_Hamza_Elsheikh": 50,
   "Tannah_Hamza_Elsheikh": 30, "Tannah_Um_Bader": 10, "
   Hamza_Elsheikh_Um_Bader": 50, "Hamza_Elsheikh_Tannah": 30}
```

```
NETWORK_PRICE = 19 #In cents

#Helper Classes

#defining a load, ie. schools, houses, health centers and water pumps
class Load:
  def __init__(self, name, max_load, num_of_units):
    self.name = name #name of the load item to get the usage trend
    self.max_load = max_load #maximum_load_needed_by_load_category
    self.usage_trends_df = pd.read_csv("data/usage_trends.csv")
    self.usage_trends_values = np.array(self.usage_trends_df[name])
    #trend_of_percentage_of_usage_during_a_day
    self.num_of_units = num_of_units #
   number_of_units_of_load_available_in_area

  def _current_single_Load(self, time):
    idx = self.usage_trends_df[self.usage_trends_df["Time"] == time
   ].index.values
    current_load = self.max_load * self.usage_trends_values[idx]
    return current_load

  def current_total_load(self, time):
    single_load = self._current_single_Load(time)
    return single_load * self.num_of_units

#battery Class, to define the storage of the system
class Battery:
  def __init__(self, max_capacity, discharge_cofficient,
   remaining_capacity, charge_rate):
    self.max_capacity = max_capacity #full_charge_capacity
    self.discharge_cofficient = discharge_cofficient #
   Discharge_coefficient
    self.remaining_capacity = remaining_capacity #
   Remaining_Capacity
    self.charge_rate = charge_rate #
   percentage_charged_from_inputed_amount

#charges the battery with given amount and returns leftover amount to be returned if amount + currnet capacity > max capacity
  def charge(self, amount):
    empty = self.max_capacity - self.remaining_capacity
    if empty <=0:
      return amount
    else:
      self.remaining_capacity+= amount
      leftover = self.remaining_capacity - self.max_capacity
      self.remaining_capacity = min(self.max_capacity, self.
   remaining_capacity)
      return max(leftover,0)

#takes energy from the battery providing the needed amount and returns amount provided form the battery

  def supply(self, amount):
    remaining = self.remaining_capacity
    self.remaining_capacity -= amount
    self.remaining_capacity = max(self.remaining_capacity,0)
```

```
77      return min(amount, remaining)
78
79  class Generation:
80    def __init__(self, name, maxCapacity = None):
81      self.solar_df = pd.read_csv("data/Solar/" + name + "
      _solar_generation.csv")
82      self.wind_df = pd.read_csv("data/wind/" + name + "
      _wind_generation.csv")
83      self.solar_generation = np.array(self.solar_df["value"], dtype
      = np.float32)
84      self.wind_generation = np.array(self.wind_df["value"], dtype =
      np.float32)
85      for i in range(len(self.wind_generation)):
86        if self.wind_generation[i] <0:
87          self.wind_generation[i] = 0
88      self.wind_generation = 0
89      self.generation = self.solar_generation + self.wind_generation
90      self.max_generation = max(self.generation)
91      #given current time, give the total generation of the solar and
      wind units
92    def current_generation(self, time):
93      idx = self.solar_df[self.solar_df["Time"] == time].index.values
94      return (self.generation[idx]/1000) #KW
95
96  class Microgrid:
97    def __init__(self, name, load_parameters, battery_parameters):
98      self.name = name #name of the microgrid used for data loading
99      self.load_parameters = load_parameters #np array of the
      parameters to create the load of the microgrid that is the
      number of schools, houses, mosques, health centers and water
      pumps
100     self.battery_parameters = battery_parameters #np array of the
      parameters to create the battery of the microgrid
101     self.battery = self._create_battery(battery_parameters)
102     self.houses, self.schools, self.mosques, self.health_centers,
      self.water_pumps= self._create_loads(load_parameters)
103     self.generation = Generation(name)
104     self.unit_price = 10
105
106
107 #creats a battery given its battery parameters ie max cap, dis
      coeff, initial rem cap and it's charge rate
108   def _create_battery(self, battery_parameters):
109     max_capacity = battery_parameters[0]
110     discharge_cofficient  = battery_parameters[1]
111     remaining_capacity = battery_parameters[2]
112     charge_rate  = battery_parameters[3]
113     battery = Battery(max_capacity, discharge_cofficient,
      remaining_capacity, charge_rate)
114     return battery
115
116 #creates the loads of the MG, using the number of units for each
      load and its name for data reasons
117   def _create_loads(self, load_parameters):
118     num_houses, num_schools, num_mosques, num_health_centers,
      num_water_pumps = load_parameters
119     houses_load = Load("House", HOUSE_MAX_LOAD, num_houses)
120     schools_load = Load("School", SCHOOL_MAX_LOAD, num_schools)
```

```
121     mosques_load = Load("Mosque", MOSQUE_MAX_LOAD, num_mosques)
122     health_centers_load = Load("Health_center",
    HEALTH_CENTER_MAX_LOAD, num_health_centers)
123     water_pumps_load = Load("Water_pump", WATER_PUMP_MAX_LOAD,
    num_water_pumps)
124     return houses_load, schools_load, mosques_load,
    health_centers_load, water_pumps_load
125
126 #returns the total load by all MG load units
127   def total_load(self, time):
128     houses_load = self.houses.current_total_load(time)
129     schools_load = self.schools.current_total_load(time)
130     mosques_load = self.mosques.current_total_load(time)
131     health_centers_load = self.health_centers.current_total_load(
    time)
132     water_pumps_load = self.water_pumps.current_total_load(time)
133     total_load = houses_load + schools_load + mosques_load +
    health_centers_load + water_pumps_load
134     return total_load
135
136 #current status of the MG containing it's battery's remaining
    capacity, it's current power generation and its current total
    load
137   def state (self, time):
138     total_generation = self.generation.current_generation(time)
139     total_load = self.total_load(time)
140     battery_status = self.battery.remaining_capacity
141     return total_load, total_generation, battery_status
142
143   def to_trade(self, time):
144     load, generation, battery = self.state(time)
145     to_trade = load - (generation + battery)
146     return to_trade
147
148   def supply(self, load, time):
149     if load >= self.generation.current_generation(time):
150       load -= self.generation.current_generation(time)
151       if load <= self.battery.remaining_capacity:
152         self.battery.remaining_capacity -= load
153         load = 0
154       else:
155         load -= self.battery.remaining_capacity
156         self.battery.remaining_capacity = 0
157     else:
158       load = 0
159     return load
160
161 class MicrogridEnv (gym.Env):
162   def __init__(self):
163     self.main_mG = Microgrid("Hamza_Elsheikh",
    HAMZA_ELSHEIKH_LOAD_PARAMETERS,
    HAMZA_ELSHEIKH_BATTERY_PARAMETERS)
164     self.first_mg = Microgrid("Um_Bader", UM_BADER_LOAD_PARAMETERS,
    UM_BADER_BATTERY_PARAMETERS)
165     self.second_mg= Microgrid("Tannah", TANNAH_LOAD_PARAMETERS,
    TANNAH_BATTERY_PARAMETERS)
166     self.time_step = 0
```

```
167     self.dates = np.array(pd.read_csv("data/Solar/" + self.main_mG.
        name + "_solar_generation.csv")["Time"])
168     self.start_date = self.dates[self.time_step]
169     self.current_price = NETWORK_PRICE
170     self.action_space = spaces.Box(low=np.array([0,0,0,self.main_mG
        .unit_price]), high=np.array([3, 2, self.main_mG.battery.
        max_capacity, NETWORK_PRICE]), dtype = np.float32)
171     self.observation_space =  spaces.Box(low =np.array([0.0, 0.0,
        0.0, 0.0]), high =np.array([1, HAMZA_ELSHEIKH_MAX_LOAD, self.
        main_mG.generation.max_generation, NETWORK_PRICE]), dtype = np.
        float32)
172
173   def _status(self):
174     if self.time_step + self.start_date_idx >= len(self.dates):
175       self.time_step = 0
176       self.start_date_idx = 0
177     self.current_date = self.dates[self.time_step + self.
        start_date_idx]
178     current_load, current_generation, remaining_capacity = self.
        main_mG.state(self.current_date)
179     time_s = self.time_step
180     previous_price = self.current_price
181     a = np.array([1,2,3])
182     if type(remaining_capacity) == type(a):
183       state = [remaining_capacity[0], current_load[0],
        current_generation[0], previous_price]
184     else:
185       state = [remaining_capacity, current_load[0],
        current_generation[0], previous_price]
186     return state
187
188   def to_trade_m (self, mg):
189     cl, cg, rc = mg.state(self.current_date)
190     a = np.array([1,2,3])
191     if type(rc) == type(a):
192       state = rc[0] + cg[0]
193       state =  cl[0] - state
194     else:
195       state = rc + cg[0]
196       state -=  cl[0]
197     return state
198
199   def reset(self):
200     self.start_date_idx = random.randint(0,len(self.dates))
201     self.start_date = self.dates[self.start_date_idx]
202     self.current_date = self.start_date
203     self.main_mG.battery = self.main_mG._create_battery(
        HAMZA_ELSHEIKH_BATTERY_PARAMETERS)
204     self.current_price = NETWORK_PRICE
205     self.energy_bought = []
206     self.energy_sold = []
207     self.prices = []
208     self.tot = []
209     state = self._status()
210     return state
211
212   def _travel_loss(self, target_mg, amount):
213
```

```
    src_name = self.main_mG.name
    dist_name = target_mg.name
    final_name = src_name +"_"+ dist_name
    distance = distances[final_name]
    base_res = 1.1 #25mm aluminium
    voltage = 33000#use sub_transmission?
    loss = ((amount**2) * (base_res*distance))/(voltage **2)
    return loss

  def step(self, action):
    action_type = action[0]
    target_mg_idx = action[1]
    amount = action[2]
    price = action[3]
    reward = 0
    is_done = False
    main_mg = self.main_mG
      target_mg = self.first_mg
    else:
      target_mg = self.second_mg

    needed_main_mg = self.to_trade_m(main_mg)
    if amount > abs(needed_main_mg):
      reward -= 10
    amount += self._travel_loss(target_mg, amount)
    offer = abs(target_mg.to_trade(self.current_date))
    if action_type <1:#buy from target MG
      if main_mg.to_trade(self.current_date) < 0:
        reward -= 10
      if price >= target_mg.unit_price:
        if offer != 0:
          if amount == 0:
            reward -=10
          elif offer >= amount:
            target_mg.battery.supply(amount)
            main_mg.battery.charge(amount)
            rem_amount = 0
            reward -= rem_amount / amount
            reward += (NETWORK_PRICE - price)/main_mg.unit_price
            self.energy_bought.append(amount - rem_amount)
            self.energy_sold.append(0)

          else:
            target_mg.battery.supply(offer)
            main_mg.battery.charge(offer)
            rem_amount = amount - offer
            reward -= rem_amount / amount
            reward += (NETWORK_PRICE - price)/main_mg.unit_price
            reward = reward[0]
            self.energy_bought.append(amount - rem_amount)
            self.energy_sold.append(0)
        else:
          reward -= 1
      else:
        reward -= 1
      self.prices.append(price)
      self.energy_bought.append(0)
      self.energy_sold.append(0)
```

```
272       main_mg.supply(main_mg.total_load(self.current_date), self.
    current_date)
273
274
275     elif action_type < 2:#Sell to target MG
276       if main_mg.to_trade(self.current_date) > 0:
277         reward -= 10
278       if price >= main_mg.unit_price and price <= NETWORK_PRICE:
279         if offer != 0:
280           if amount == 0:
281             reward -=10
282           elif offer >= amount:
283             main_mg.battery.supply(amount)
284             target_mg.battery.charge(amount)
285             rem_amount = 0
286             reward -= rem_amount / amount
287             reward += (price - main_mg.unit_price)/main_mg.
    unit_price
288             self.energy_sold.append(amount - rem_amount)
289             self.energy_bought.append(0)
290           else:
291             main_mg.battery.supply(offer)
292             target_mg.battery.charge(offer)
293             rem_amount = amount - offer
294             reward -= rem_amount / amount
295             reward += (price - main_mg.unit_price)/main_mg.
    unit_price
296             reward = reward[0]
297             self.energy_sold.append(amount - rem_amount)
298             self.energy_bought.append(0)
299         else:
300           reward -= 1
301       else:
302         reward -= 1
303       self.prices.append(price)
304       self.energy_bought.append(0)
305       self.energy_sold.append(0)
306       main_mg.supply(main_mg.total_load(self.current_date), self.
    current_date)
307
308     else:
309       self.prices.append(price)
310       self.energy_bought.append(0)
311       self.energy_sold.append(0)
312       main_mg.supply(main_mg.total_load(self.current_date), self.
    current_date)
313
314     tot = self.to_trade_m(main_mg)
315     if tot >0:
316       reward += 100
317     self.time_step +=1
318     self.tot.append(tot)
319     state = self._status()
320     tgt_total_load, tgt_total_generation, tgt_battery_status =
    target_mg.state(self.current_date)
321     target_mg.battery.charge(tgt_total_load - tgt_total_load)
322     is_done = False
323     return state, reward, is_done, {}
```

```
324
325   def render(self):
326     pass
```

Listing A.1: Microgrid python enviroment

# Bibliography


[1] Mushtaq N Ahmed et al. "An overview on microgrid control strategies". In: *International Journal of Engineering and Advanced Technology (IJEAT)* 4.5 (2015), pp. 93–98.

[2] Sina Parhizi et al. "State of the art in research on microgrids: A review". In: *Ieee Access* 3 (2015), pp. 890–925.

[3] Daniel E Olivares et al. "Trends in microgrid control". In: *IEEE Transactions on smart grid* 5.4 (2014), pp. 1905–1919.

[4] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

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

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

[7] Matthias Pilz and Luluwah Al-Fagih. "Recent advances in local energy trading in the smart grid based on game-theoretic approaches". In: *IEEE Transactions on Smart Grid* 10.2 (2017), pp. 1363–1371.

[8] Sheikh Muhammad Ali. "Electricity trading among microgrids". In: *Department of Mechanical Engineering, University of Strathclyde* (2009).

[9] Sivapriya Mothilal Bhagavathy and Gobind Pillai. "PV Microgrid Design for Rural Electrification". In: *Designs* 2.3 (2018), p. 33.

[10] Jean Pierre Murenzi and Taha Selim Ustun. "The case for microgrids in electrifying Sub-Saharan Africa". In: *IREC2015 The Sixth International Renewable Energy Congress*. IEEE. 2015, pp. 1–6.

[11] Sunyong Kim and Hyuk Lim. "Reinforcement learning based energy management algorithm for smart energy buildings". In: *Energies* 11.8 (2018), p. 2010.

[12] Elena Mocanu et al. "Unsupervised energy prediction in a Smart Grid context using reinforcement cross-building transfer learning". In: *Energy and Buildings* 116 (2016), pp. 646–655.

[13] Leo Raju et al. "Reinforcement learning in adaptive control of power system generation". In: *Procedia Computer Science* 46 (2015), pp. 202–209.

[14] Fabrice Lauri et al. "Managing power flows in microgrids using multi-agent reinforcement learning". In: *Agent Technologies in Energy Systems (ATES)* (2013).

[15] Liang Xiao et al. "Reinforcement learning-based energy trading for microgrids". In: *arXiv preprint arXiv:1801.06285* (2018).

[16] David Silver et al. “Deterministic policy gradient algorithms”. In: 2014.

[17] John Schulman et al. “Trust region policy optimization”. In: *International conference on machine learning*. 2015, pp. 1889–1897.

[18] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).

[19] Peter Henderson et al. “Deep reinforcement learning that matters”. In: *arXiv preprint arXiv:1709.06560* (2017).

[20] Josh Achiam. *Open AI Spinning Up*. 2020. URL: `https://spinningup.openai.com` (visited on 08/28/2020).

[21] *Photovoltaic Geographical Information System (PVGIS)*. 2020. URL: `https://ec.europa.eu/jrc/en/pvgis` (visited on 08/28/2020).

[22] *Sudan Wind Prospecting*. 2020. URL: `http://sudan.windprospecting.com` (visited on 08/28/2020).

[23] *National Renewable Energy Laboratory*. 2020. URL: `https://data.nrel.gov` (visited on 08/28/2020).