**Chapter Three**

**Reinforcement Learning Methodology**

Working based on the Microgrid simulation, and due to computation constraints and compatibility limitations with Matlab, we are creating 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.

**Python and basic libraries:**

$Python$ is the leading programming language in applications of machine learning; it’s high flexibility, speed and support make it a perfect language for research in this field. $Python$ has several libraries designed to the purpose of 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 is a representation of 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.

**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.

$Pytorch$ 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 the optimization of NNs. $Pythorch$ will calculate the network gradients, backpropagate the values and apply the new parameters to the network automatically. $Pytorch$ 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 which include Stochastic Gradient Descent $SGD$, $RMSprop$, $Adagragd$ and the famous $Adam$ optimizer.

**OpenAI GYM:**

This library is the most central library for RL researchers and practitioners; it provides a complete set of classes and functionalities for creating, testing and evaluating RL algorithms using its built-in functions. It provides a set of working environments for learning and testing algorithms as well as 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:

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

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

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

Observation: the state of the environment after an action is executed.

Reward: reward given to the agent after executing the action.

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

Info: any additional info by the environment.

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

**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 any price higher than the network price.

The environment observation is an array which 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 past price that a transaction occurred on. The action that the agent will take consists of also 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 method in deep reinforcement learning.

**Continous 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.

A2C 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 is a problem though, as exploration is hindered by this deterministic policy, we will solve it by substituting the deterministic values of the actor-network by parameters of a gaussian distribution, which are the mean and variance of the distribution as we know this normal distribution is given by:

%% normal dist eq

We use it to get our policy as follows

%% A2C cont eq

Deep Deterministic Policy gradients:

This method is 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, (s)
* Critic Network: takes state and action and the Q value for those pair Q(s, a) or Q(s,(s))

Here Q(s,(s)) depends on both p and. We need to maximize the output of this network.

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

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 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 output of the policy before passing to the step function. The type of noise used in the original paper is Ornstein–Uhlenbeck process noise, which is a time-correlated noise, but later implementations of the algorithm suggest that an uncorrelated, mean-zero Gaussian noise works perfectly well and is much simpler to implement.

# 

# The algorithm is as follows:

# 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, each step is constrained by the closness between the new policy and the old one expressed as KL divergence which is explained as a distance between probability distributions.

# Insert equations 1, 2, 3

# Approximation we get

# 4,5,6

# Using lagrangian duality, and using a backtracking coefficient j that satisfies kl divergence

# 8

# Using conjugate gradients we solve $H^-1$ as

# 9

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

# The algorithm is as follows

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 taking 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 than TRPO. We will be using PPO-Clip, a method that doesn’t have KL-divergence, it relies on specialized clipping in the objective function removing incentive for new policy to move away from old policy.

Ppo eqn 1,2

A less complex expression to represent L is

3