**Chapter Three**

**Reinforcement Learning Methodology**

Working on the basis of the Microgrid simulation and due to computation constraints and compatibility limitations with Matlab, therefore 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 easier 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 a number of 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 a number of functions that are optimized for 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 important, 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. Pythorch 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 it’s 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 importance 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 Corss Entropy Loss. It also has multiple optimizers which include Stochastic Gradient Descent SGD, RMSprop, Adagragd and the popular Adam optimizer.

Building a NN can be as smiple as the following code:

network = nn.Sequential(

nn.Linear(2,5),

nn.ReLU(),

nn.Linear(5, 20),

nn.ReLU(),

nn.Linear(20,10),

nn.Dropout(p=0.3),

s.Softmax(dim=1))

**OpenAI GYM:**

This library is the most important library for working in the field of RL, it provides a complete set of classes and functionalities for creating, testing and evaluating RL algorithms using it’s built in functions. It provides a set of working environments for learning and testing algorithms as well as the ability of creating custom environments that make use of GYM’s functionalities.

The concept of spaces is important 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 6 values then these values are zero through 5.
2. Box: a N-dimensional tensor of rational numbers, we set a lower value, a 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 sub spaces.

GYM provides us with building block for 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 value that the observation can take, it is the state of the environment, it can be either a discrete, box or tuple space.
3. Step function: A detailed discreption of the enviroment’s reaction to the action taken by the agent, after its execution it returns four values:
   1. Observation: the state of the environment after action is executed.
   2. Reward: reward given to the agent after executing the action.
   3. Is Done: a Boolean describing weather the current episode has ended.
   4. Info: any additional info by the environment.

**PTAN ? if used**

**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, a main grid that is controlled by our agent and two other microgrids to trade with. A single microgrid consists of loads, battery and generation.

**Battery:**

**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

**Load:**

The loads for a single microgrid are the basic building blocks of a village, 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 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.