Documentation Of the Q-learning algorithm

Algorithm made by: Total-RD (github link: https://github.com/Total-RD/pymgrid)

Documentation made by: Vincenntius Patrick Tunas

0. Introduction to pymgrid

Pymgrid is a python library that can generate and simulate microgrids. [1]

Pymgrid has 3 main components:

1. **data folder:**

pymgrid make use of load datasets that comes from DOE OpenEI and is based on TMY3 weather data and pv production datasets based on TMY3. This dataset is made of a year long time series with a one hour time-step with a total of 8760 points. The load and pv data are based from five cities from different climate zone in the US

2. **microgrid generator class**: this class is used to generate the microgrids.
   When a microgrid is generated:
   1. maximum power load is generated by random
   2. select a random load file and scale it to the previous value that is generated.
   3. The architecrute of the microgrid is selected by random.
3. **microgrid simulator class**:

The microgrid simulator class named microgrid is a class that consist of the full implementation of a microgrid. Inside this class, there are time-series data and the specific sizing for one microgrid. The microgrid class has 3 main family functions, they are the control loop, the benchmark algorithms and the utility functions.

Run() function moves the time-step forward by one. It takes control of the argument dictionary and returns the updated state of the microgrid.

The Control dictionary centralizes all the power commands that needs to be passed down into the microgrid in order to operate each generator at each different time-steps.

Reset() function to reset microgrid back to its initial state, empty the trackin data structure and resetting the time-step.

[1]

1. Simulating microgrid Virtual Environment

We start by first installing Pymgrid using the following command:

```
pip install git+https://github.com/Total-RD/pymgrid/
```

import the packages that are necessary for this project, after that, we generate the microgrids.

We generate 2 microgrids

```
#generate microgrids
env = mg.MicrogridGenerator(nb_microgrid=2)
env.generate_microgrid(verbose= True)
mg0 = env.microgrids[0]
mg1 = env.microgrids[1]
```

	load	cost_loss_load	cost_overgeneration	cost_co2	PV_rated_power	battery_soc_0	battery_power_charge	battery_power_discharge	battery_capacity	battery_efficiency	battery_soc_min	battery_soc_max	battery_cost_cycle	grid_weak	gr
0	960	10	1	0.1	969.60	0.2	236	236	944	0.9	0.2	1	0.02	0	
1	60363	10	1	0.1	30785.13	0.2	14838	14838	59352	0.9	0.2	1	0.02	1	

Check on the architecture of the microgrid

```
#show the microgrids using for loops.
#identify the microgrid architecture
for i in range(env.nb_microgrids):
  print("Microgrids {} architecture: {}".format(int(i), str(env.microgrids[i].architecture)))
  #benchmark
  env.microgrids[i].benchmarks.run_benchmarks("rbc")
  env.microgrids[i].benchmarks.describe_benchmarks()
```

```
Microgrids 0 architecture: {'PV': 1, 'battery': 1, 'genset': 0, 'grid': 1}
In Progress 100%
Rules Based Calculation Finished
Cost of the last 8736 steps (100 percent of all steps) using rule-based control: 557883.88
Microgrids 1 architecture: {'PV': 1, 'battery': 1, 'genset': 1, 'grid': 1}
In Progress 100%
Rules Based Calculation Finished
Cost of the last 8736 steps (100 percent of all steps) using rule-based control: 19833161.66
```

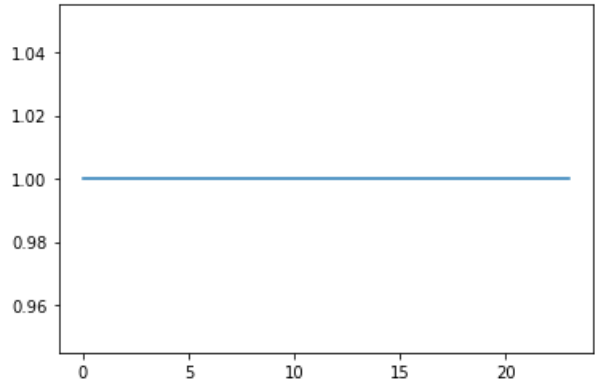
Both microgrid architecture consist of a PV source, a battery and a grid.

Microgrid 0 isn't connected to a genset and Microgrid 1 is connected to a genset

check the grid status:

```
plt.plot(mg0.forecast_grid_status())
```

```
[<matplotlib.lines.Line2D at 0x7fa760cfa550>]
```



Based on the graph, we learn that the microgris is always connected to the grid. It is never disconnected from the grid.

Microgrid Parameters

The architecture of microgrid 0 consist of:

```
{'PV': 1, 'battery': 1, 'genset': 0, 'grid': 1}
```

The control dictionary {actions available in microgrid}:

```
['load', 'pv_consummed', 'pv_curtailed', 'pv', 'battery_charge', 'batte
ry_discharge', 'grid_import', 'grid_export']
```

Status:

```
['load', 'hour', 'pv', 'battery_soc', 'capa_to_charge',
'capa_to_discharge', 'grid_status', 'grid_co2', 'grid_price_import', 'g
rid_price_export']
```

The pv penetration is calculated by the load maximum power divided by pv maximum power. [1]

The penetration of microgrid = $(PV\_RatedPower\ /\ Load) * 100$

Based on the calculation for both microgrid:

```
Penetration of PV microgrid 0:  101.0 %
Penetration of PV microgrid 1:  51.0 %
```

Identify the state variable based on the current net load and the current battery capacity

$$NetLoad_{Microgrid} = Load_{Microgrid} - PV_{microgrid}$$

what are my state variables?

use current net_load (load-pv) and the current battery capacity for this case

$$NetLoad_(mg0) = Load_(mg0) - PV_(mg0))$$

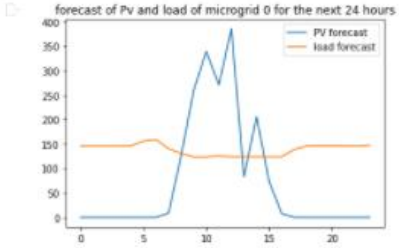
```
[10] #calculate net load of microgrid 0
     net_load = mg0.load-mg0.pv
     #print the current net load
     print("Current net load of microgrid 0: {:0.4} kWh".format(net_load) )
     print("Current battery capacity of microgrid 0:", mg0.battery.capacity)

     Current net load of microgrid 0: 145.6 kWh
     Current battery capacity of microgrid 0: 944
```

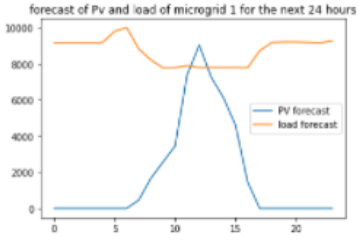
Forecast for PV and Load

Plot a Forecast for next 24 hours

```
[40] plt.title("forecast of Pv and load of microgrid 0 for the next 24 hours")
     plt.plot(mg0.forecast_pv(), label ="PV forecast")
     plt.plot(mg0.forecast_load(), label = "load forecast")
     plt.legend()
     plt.show()
```




```
[41] plt.title("forecast of Pv and load of microgrid 1 for the next 24 hours")
     plt.plot(mg1.forecast_pv(), label ="PV forecast")
     plt.plot(mg1.forecast_load(), label ="load forecast")
     plt.legend()
     plt.show()
```



Based on both plot for this 2 microgrid, it shows that in the next 24 hours,

Both microgrid will only receive solar power from 7am in the morning to 4pm in the evening.

Microgrid 0 is able to satisfy the consumer load during those peak time, however microgrid 1 is not able to meet the consumer load for almost all the hours of the day.

Microgrid 1 has a higher load forecast and is only able to cover the load from 11pm to 2pm.

2. Agent Design

Now that we have a microgrid environment set, this is where we start to design our agent.

2.1 Action space

In Q-learning, an agent has to select an action in order to maximize reward in an environment.[2] An action space is where we define all the actions that are available for the agent to make.

```
mg0.get_control_dict() #show the list of action that can be take

['load',
 'pv_consummed',
 'pv_curtailed',
 'pv',
 'battery_charge',
 'battery_discharge',
 'grid_import',
 'grid_export']
```

This is the control dictionary of the microgrid 0. Inside the control dictionary it shows the list of actions that can be taken. However in this training, we set the agent to only be able to take up to 5 actions defined at the full rate of the net load.

The actions that the agents can take are:

- Action 0: battery_charge
- Action 1: battery_discharge
- Action 2: grid_import
- Action 3: grid_export

We use selection to change the control dictionary based on the different action the agent take.

- If action 0 is selected, battery is charged. We set the battery charge value in control dictionary into p_charge
- If action 1 is selected, battery is discharged. We set the battery_discharge value in control dictionary into p_discharge
- If action 2 is selected, we import the current net load into the grid. We set the grid_import value in control dictionary with net load
- If action 3 is selected, we import the current net load into the grid. We set the grid_import value in control dictionary with net load

The function then returns the control dictionary for the run function.

2.1.1 Q-table

We set a function to initialize Q-table function. In this function we initialize the state and Q. round the state off in order to reduce the state space.

```
psuedocode

init_Q_Table(microgrid, action_chosed):

    net_load -> microgrid.forecast_load - microgrid.forecast_pv

    state - > empty

    Q -> empty

    iterate through range from minimum net load to maximum net load

        for all battery state of charge level from min to max:

            round off

            append to state

    for all the states:

        initialize to 0

        for all actions:

            set the state to 0

    return Q
```

2.2 Exploration strategy: greedy-decreasing strategy

In an exploration/exploitation strategy, an agent doesn't know in the beginning what is the best action to be taken for each state. Therefore, the algorithm needs to explore for a certain number of rounds. [3]

The method that we are using for exploration/exploitation is the epsilon-greedy action selection. An epsilon-greedy is used to balance the exploration and exploitation by choosing randomly between exploring and exploiting. The epsilon value refers to the probability of choosing to explore or to exploit.[4]

In this algorithm however, the epsilon value decreases as the exploration goes further. This leads the agent to have a very explorative behaviour in the beginning and becomes more highly exploitative as the training progresses. This is known as the greedy decreasing strategy.

2.2.1 define functions

1. epsilon_decreasing greedy: set a random number, if the random number value is above 1-epsilon return the random action taken.
2. max_dict: find the maximum value in the dictionary, the function returns the max kaey and max value after the entire dictionary has been searched through.
3. Update_epsilon: this function is used to decrease and update the epsilon. As the exploration continues, the epsilon decreases.

2.3 training Q learning

This function is used to start training our agent in Q learning.

We first initialize all the variables to be used

- Number of actions → 4
- Q →Q-table taking the action and microgrid name as a parameter
- Number of states → the total length of Q
- Number of episodes → 100
- Alpha → alpha is the learning rate of the algorithm, it is generally set between 0 and 1. If alpha is set to 0, it means that the Q value is never updated and therefore the agent learn nothing. The higher the value set to alpha, the faster the learning can occur. [5]
- Gamma → gamma is the discount factor. It is also generally set between 0 and 1. Discount factor indicates the notion that the current reward is more valuable then it is now in the future. If the value is near 0, then the future reward will not be taken into consideration but if it is near 1, it shows that the future reward will be great. [1]

Iterate through all the number of episodes and Train every episodes and then print the value for the end of each episodes after all the episodes are finish.

Inside each episode, run the training for the total number of horizon. Horizon is the total number of hours we want our agent to go through.

In every hour (horizon),

1. an agent takes an action.
2. The action taken in the microgrid environment changes the status of the environment
3. The reward is subtracted by the cost of the action. The cost value is added to the total episode cost.
4. The state that the agent arrive in is based on the current net load and the state of charge of the agent
5. We then choose the action that leads to the best state in the table using max_dict function
6. Calculate the temporal difference error (td error). Temporal difference error indicates the difference between the agent's current estimate and target value.
7. Enter the new Q-table state.

After the horizon is finish and the episode is complete, decrease the epsilon and move forward to the next episode. This process Is repeated until it has finished through all the defined number of episodes. Return the q table in the end of the process.

2.3 Testing Q learning

After the agent is being trained for the specified number of episodes. Its Q-table has been filled with good Q-values. The agent is then tested in the environment. Through this function, it also displays to us in each time-step, what action did the agent take, what is the state that the agent arrived in and what is the cost of the agent's action.

```
testing_Q_Learning(mg0,Q1, 48) #start q learning for 2 days
```

```
t -     STATE  -  ACTION - COST
================================
0  - (299, 0.2) discharge 73.0 €
1  - (300, 0.2) discharge 147.2 €
2  - (264, 0.2) discharge 212.4 €
3  - (264, 0.2) discharge 277.4 €
4  - (281, 0.2) discharge 346.7 €
5  - (311, 0.2) discharge 423.0 €
6  - (518, 0.2) import 550.8 €
7  - (639, 0.2) discharge 708.9 €
8  - (597, 0.2) import 898.0 €
9  - (377, 0.2) discharge 1017.4 €
10 - (203, 0.2) discharge 1081.9 €
11 - (262, 0.2) import 1164.8 €
12 - (158, 0.2) import 1262.4 €
13 - (421, 0.2) discharge 1521.9 €
14 - (293, 0.2) discharge 1703.0 €
15 - (404, 0.2) import 1952.1 €
16 - (498, 0.2) discharge 2257.7 €
17 - (548, 0.2) discharge 2590.5 €
18 - (689, 0.2) import 2800.9 €
19 - (720, 0.2) import 3020.2 €
20 - (775, 0.2) import 3256.4 €
21 - (718, 0.2) import 3425.1 €
22 - (600, 0.2) discharge 3566.5 €
23 - (434, 0.2) discharge 3670.0 €
24 - (272, 0.2) discharge 3736.7 €
25 - (253, 0.2) discharge 3799.6 €
26 - (235, 0.2) import 3858.0 €
27 - (233, 0.2) discharge 3915.9 €
28 - (249, 0.2) discharge 3977.8 €
29 - (318, 0.2) discharge 4057.2 €
30 - (565, 0.2) import 4198.3 €
31 - (721, 0.2) import 4378.8 €
32 - (463, 0.2) discharge 4527.2 €
33 - (365, 0.2) discharge 4643.8 €
34 - (162, 0.2) discharge 4695.6 €
35 - (156, 0.2) import 4745.5 €
36 - (226, 0.2) discharge 4885.4 €
37 - (299, 0.2) discharge 5070.6 €
38 - (416, 0.2) discharge 5327.9 €
39 - (384, 0.2) discharge 5564.8 €
40 - (485, 0.2) discharge 5863.1 €
```

References

[1] L. R, R S Milton, and S. S, 'Reinforcement Learning for optimal energy management of a solar microgrid', *2014 IEEE Glob. Humanit. Technol. Conf.*, Sep. 2014.

[2] P. Kofinas, G. Vouros, and A. I.Dounis, 'Energy Management in solar microgrid via reinforcement learning using fuzzy reward', *17 April 2017*.

[3] Gonzague Henri_, Tanguy Levent, Avishai Halev, Reda Alami, and Philippe Cordier, 'pymgrid: An Open-Source Python Microgrid Simulator for Applied Artificial Intelligence Research', Nov. 2020.

[4] GeeksforGeeks. 2021. *Epsilon-Greedy Algorithm in Reinforcement Learning - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/ [Accessed 22 July 2021].

[5] Vilches, V., 2021. *A comprehensive approach to Reinforcement Learning*. [online] Deep Robotics. Available at: https://vmayoral.github.io/robots,/ai,/deep/learning,/rl,/reinforcement/learning/2016/07/10/rl-tutorial/ [Accessed 22 July 2021].