Reinforcement Learning for Demand Response: A Review of Algorithms and Modeling Techniques

1st Harshita Vemula
Operations Research and Industrial Engineering
Univeristy of Texas at Austin
Austin, U.S.A
harshita.vemula@gmail.com

2nd Kapil Chandra Nidadavolu
Oden Institute for Computational Engineering and Sciences
University of Texas at Austin
Austin, U.S.A
kapilchandra81@gmail.com

***Abstract*—Demand response management is important as it can have an immediate effect on economic savings. As it is difficult to store electricity efficiently on utility scale, efforts have to be made to forecast demand and ramp up energy production during peak hours. If the peak demand is very high, more energy supplying facilities have to be built and more investments will have to be made in transmission and distribution assets to satisfy the peak demand. Demand response management was traditionally implemented manually and hence efficient energy management was limited to only hours of peak demand. Reinforcement Learning (RL) can help design an agent that can continuously help in energy management and also learn new strategies in a non-stationary environment where the energy consumption pattern changes from season to season.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

The goal of the project is to introduce autonomy in demand response management using reinforcement learning agents in order to facilitate economic savings. The project involves developing RL agents policy gradient and temporal difference methods. Experiments were conducted considering both continuous and discrete state and action spaces and using appropriate algorithms in the both cases. To simulate the demand responses and building energy coordination we use an open source OpenAI Gym environment called CityLearn. [16]

The environment has a heat pump and a cooling energy storing device. The heat pump will draw energy from the grid and cool down water at room temperature to some target temperature and supply it to the cooling water storage tank. Water is then drawn from this storage tank and circulated to the buildings to satisfy the cooling demand. The agent has to learn a policy where in the cooling energy storage is charged when the outside temperature is relatively less compared to the different temperatures experienced during the day and discharged when the outside temperature or the cooling demand is high. In this project, we have considered a simple case where there is one building and one RL agent and the agent is expected to learn the above mentioned policy.

Identify applicable funding agency here. If none, delete this.

To facilitate this, coming up with the right reward function was the challenge and different reward functions were tried to converge to the desired policy.

The focus of this project is to explore various algorithms in RL which can be applied to this domain and summarize the results while aiming to minimize the cost of energy consumed from the grid.

Please find the link to the video and GitHub repository below:

Repository:
https://github.com/nkc-137/Reinforcement-Learning-Project
Video Link:
https://www.youtube.com/watch?v=ujLPJWqUULA&list=LLqDVJBrDJ7mvnEV-_SeLnGQ&index=3&t=0s

II. Literature Review

Why is Demand Response Management important? Demand response management is important as it can have an immediate effect on economic savings. As it is difficult to store electricity efficiently on utility scale, efforts have to be made to forecast demand and ramp up energy production during peak hours. If the peak demand is very high, more energy supplying facilities have to be built and more investments will have to be made in transmission and distribution assets to satisfy the peak demand. [10]

There are two major Demand Response (DR) programs that are currently practiced.

- **Incentive based:**In incentive-based programs, consumers voluntarily participate in a scheme in which the system operator can directly turn off some appliances to reduce the energy consumption of consumers during the periods of peak demand.
- **Time based:**Time-based programs are generally based on dynamic pricing, and aim to flatten the curves of demand by offering the consumer electricity prices that vary in time.

These were implemented manually and hence efficient energy management was limited to only hours of peak demand. [12] Reinforcement learning can help design an agent that can

continuously help in energy management and also learn new strategies in a non-stationary environment where the energy consumption pattern changes from season to season. [4] [6]

RL for efficient energy management:
Reinforcement Learning for demand response management has been around for sometime. Different versions of Q-learning and SARSA with function approximation were used frequently to manage energy efficiently [7] [8] [11]. Policy gradient methods such as DDPG and A2C also seem to perform well. [8] Most of the algorithms used neural networks for function approximation. [9] [5]

III. RL Framework

The environment of building has a heat pump and a cooling energy storage device. The storage device allows the heat pump to store energy which can be used later to satisfy the energy demands of the building.

A. State Space

The state space is continuous and the state variables consisted hour of the day (hr), state of charge of the storage device (soc) and temperature of the environment outside the building (t_out). Different encodings of the state space were used with different algorithms.

State representation 1:
The state vector is of size 3×1, where $s1$ represents hr, $s2$ represents T_{out} and $s3$ represents state of charge (SOC). $s1$ varies from 1 to 24 and $s3$ varies from 0 to 1.

State representation 2:
The state vector is of the size 24×1. The hour of the day is binary encoded and all other information is dropped.

B. Action Space

Algorithms were tried using a continuous and a discrete action spaces. The action space represents the action taken either to charge the storage device or discharge the storage device.

Action representation 1:
The action space is continuous and varies between -0.3 and 1.

Action representation 2:
The action space is discrete. The agent can choose to take any action from $\{-0.2, -0.1, 0, 0.1, 0.2\}$.

Action representation 3:
The action space is discrete. The agent can choose to take any action from $\{-0.2, 0, 0.2\}$

C. Reward Function

The reward obtained at any time is proportional to the negative of the energy consumed from the grid. The energy demand of the building is primarily satisfied by the energy stored in the storage device, any excess demand is satisfied by drawing energy directly from the grid. Less amount of energy will be required to charge the cooling storage device if the difference in temperature between water (which is approximated to be a constant) and the outside temperature is less.Hence under optimal conditions, energy should be drawn from the grid at night to charge the cooling energy storage device, which can be discharged during the day to satisfy the energy demands of the buildings.Different reward functions were tried out with different algorithms.

Reward Function 1:
The first reward function that has been tested was the obvious choice of choosing the negative of the electricity consumption by the building at the point in time, which is what we are trying to minimize. As the price of electricity at any instance is proportional to the demand, which is again proportional to the electricity consumed by multiple buildings (electricity price increases with increase in demand) , reward can be calculated to be equal to the electricity consumed by the building multiplied with the total electricity consumed. So the agent should learn to cool the water when the outside temperature is low, i.e, it is expected to take a positive action when outside temperature is low and soc is less than 1 and it is expected to take negative action when the outside temperature is high and soc is greater than 0.

Reward Function 2:
The agent will get a positive reward of 1 for charging from 2AM to 10 AM and discharging from 12:00PM to 8:00PM and remaining idle for the remaining time and a negative reward of -1 for any other action taken.

D. Transition Functions

The state transition functions are as shown in Figure 1 and Figure 2. For state representation 1 two components, soc and hour of the day are deterministic but the outside temperature is not. The transition probabilities for state representation 1 is positive. For state representation 2 where the hour changes and the binary encoded vector also changes, the transition function is deterministic.This means that the probability of getting to a particular state after taking a particular action from the current state is equal to 1.

The following figures provide a schematic of the transition functions:

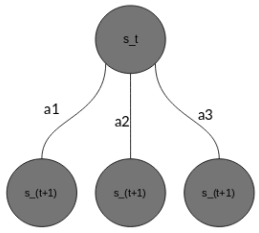


Fig. 1. State Representation 1

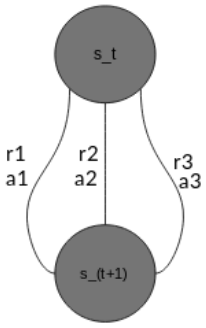


Fig. 2. State Representation 2

E. RBC controller and cost computations

We have used a manually optimized rule based controller as a baseline for the models we have created. The objective of the RL agents that we are going to train is to achieve the policy similar to that of the RBC controller. The data which we used to train the agent is taken for 4.8 months approximately spanning over the summer period and the agent is trained over and over again. The rule based controller basically releases the cooling energy during the morning and stores cooling energy in the night. The cost is defined as the sum of the square of electricity consumed at every instance of the episode.

The cost obtained by using RBC controller is around 156 and the cost obtained by not using a storage device device is around 186 units (baseline cost).

The actions which the RBC takes during different hours of the day is shown in the Figure 3:

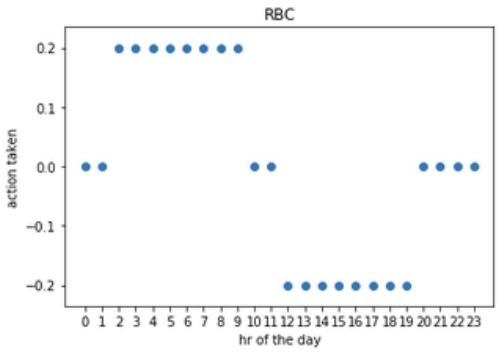


Fig. 3. Actions taken by the RBC

IV. Algorithms

Various algorithms were implemented and the cost obtained is compared with the cost obtained by the Rule Based Controller (RBC Controller) and Baseline cost (which the cost of drawing energy directly from the grid with out any storage). We had mentioned in the literature survey that we would be trying Soft Actor Critic as well but later we realized that the reward function itself is not good for making the agent learn hence instead of implementing a new algorithm which mostly woudn't work we spent time on researching possible options for reward function.

A. Policy Gradient Methods

1) Deep Deterministic Policy Gradient (DDPG): DDPG [1] learns Q function in an off-policy manner. It uses a deterministic policy. DDPG is suited for problems with continuous action space when it is not simply possible to take a max over Q to find the optimal action. This algorithm can basically be thought of as deep Q-learning for continuous action spaces. [1]

Also considering the fact that hyperparamter tuning for DDPG is difficult, we thought it was better to use other methods.The costs obtained had high variance and none of them even converged to a value less than the baseline cost.

2) Twin Delayed DDPG (TD3): DDPG requires a lot of hyperparameter tuning and it prone to policy breaking due to overestimation of Q values because of exploitation of errors in Q learning. Twin Delayed DDPG [15] employs three tricks to tackle this problem:

- **Clipped Double Q learning** - Two Q values are learnt instead of one
- **Delayed Policy Updates** - The updates in TD3 are made less frequently to prevent the overestimation of Q values. The policy updates are typically made once for every two Q function updates
- **Target policy smoothing** - Noise is added to the target action to make it more and more difficult for the policy to exploit Q function errors.

The costs obtained had high variance and none of them even converged to a value less than the baseline cost.

Both DDPG and TD3 were implemented by the PhD student for a single agent environment, hence we did not try doing it again and also we wanted to try out stochastic policy instead of deterministic policy.

3) Proximal Policy Optimization (PPO): Proximal Policy Approximation [2] belongs to family of algorithms which tries to keep new policies close to the old. We have used PPO clip for our problem. PPO is an on-policy algorithm. In this algorithm actions are sampled from a distribution and the policy decides the parameters of the distribution. Over the course of training the policy becomes less and less random and is encouraged to exploit the rewards already found. [2]

We have implemented PPO, part of the code was taken from Github [14].PPO was implemented with the states belonging to state representation 1 and actions belonging to action representation 1. Reward function 1 was used to

calculate the rewards obtained at every time step.

Model: Neural Networks were used to evaluate the state values and the distribution of the actions. The actions were sampled from a Gaussian distribution which mean capped between -0.1 and 0.1 and standard deviation belonging to $[-1, 1]$. 1 hidden layer was of size 32 units and mini batch size of 24 was used. The actions sampled were capped between $[-0.2, 0.2]$.

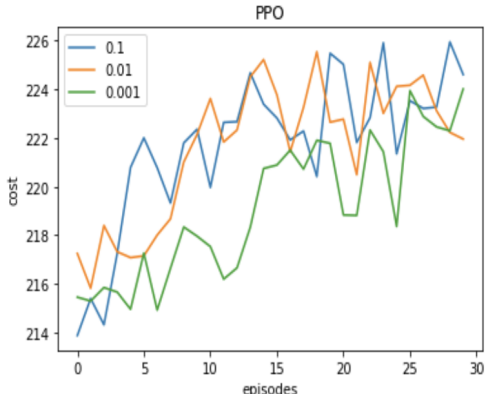


Fig. 4. Cost function in PPO

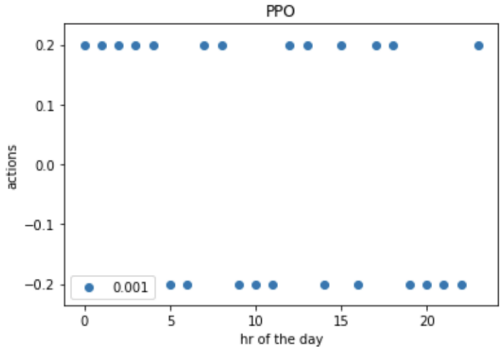


Fig. 5. Actions taken by PPO agent

From the plots we can infer that he algorithm was not robust and was not able to converge to an optimal policy.The final cost obtained had high variance.

4) Reinforce with baseline: Reinforce is the most basic monte-carlo policy gradient algorithm and we have implemented reinforce with baseline. Reinforce has good convergence properties but since it is a monte-carlo method it has high variance and thus is a little slow at learning. A baseline is some random variable which is independent of actions which is subtracted from the Q value in the gradient part of the update. The introduction of baseline does not introduce any bias to the model but reduces the variance of the model a lot.

Reinforce was implemented with tile coded state space i.e., state representation 2 along with continuous action space, similar to action representation 1 with actions capped between [-0.5,0.5] are used.Reward function 1 was used to generate rewards.

Model: The actions were obtained using the function $(sigmoid - 0.5)$, an α of 0.1 and γ of 0.9.

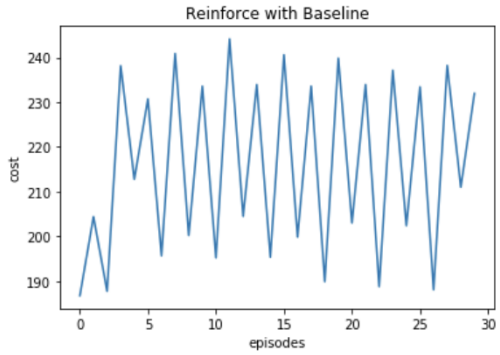


Fig. 6. Cost function in Reinforce with baseline

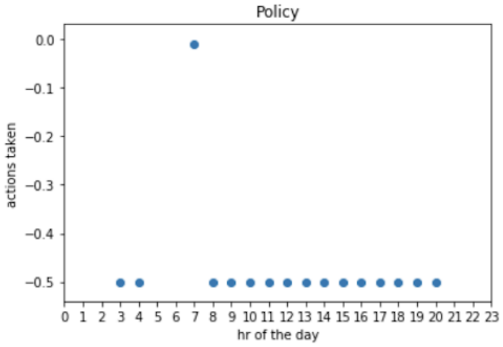


Fig. 7. Actions taken by Reinforce with baseline

Note: Some weights blew up exponentially, and hence we got nan values for some actions.

The algorithm was not robust and was not able to converge to an optimal policy. The final cost obtained had high variance.

5) One step Actor Critic: In actor critic methods there are two components, namely, actor and critic. The critic estimates the value of either V or Q, and the actor makes policy updates in the direction guided by the critic.

One step actor critic was implemented with the same space and reward function representations as that of reinforce with a linear function approximation for state values. Reward function 1 was used to calculate rewards.

Model parameters: $\alpha = 0.1$ and $\gamma = 0.99$

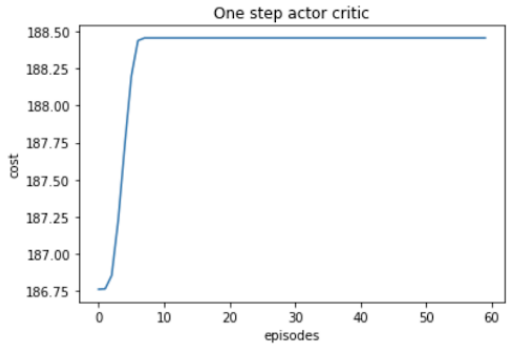


Fig. 8. Cost function in one step actor critic

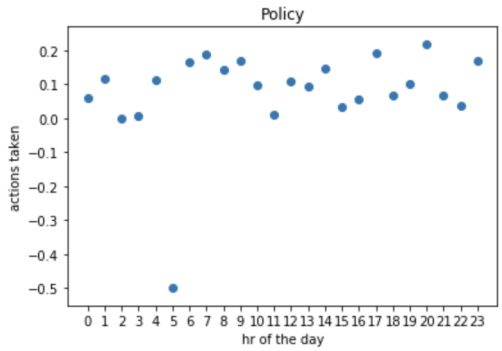


Fig. 9. Actions taken by one step actor critic

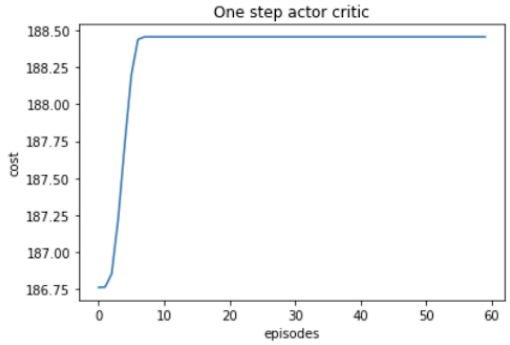


Fig. 10. Cost function in actor critic with eligibility traces

The algorithm was run multiple times and the cost obtained was slightly better better than those of previous algorithms.It seem to come down to around 186 and remain constant. It did not learn the optimal policy.

B. TD methods - SARSA

SARSA with state representation 2 and action representation 2 with reward function 1 were used to train the agent. However the agent did not learn the optimal policy and bring down the cost.

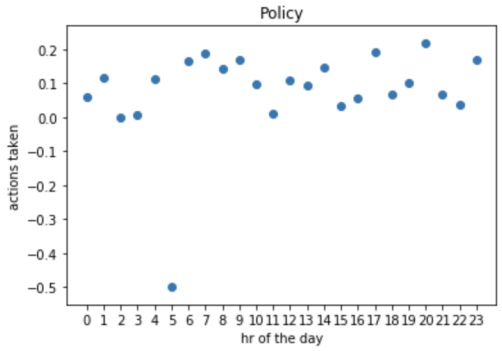


Fig. 11. Actions taken by actor critic with eligibility traces

Model parameters:alpha=0.1 and gamma =0.9.

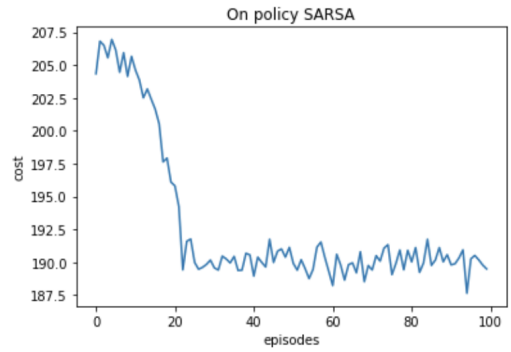


Fig. 12. Plot of cost function for on policy SARSA

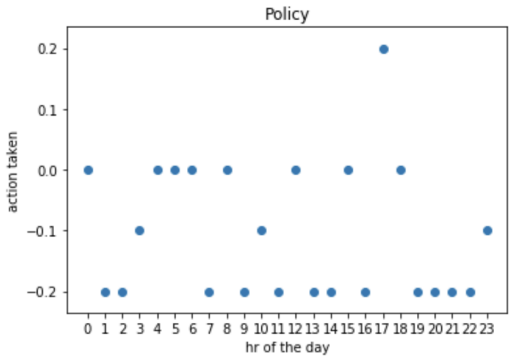


Fig. 13. Actions taken by SARSA

SARSA - greedy converged to a better cost than all the previous algorithms, but however it was not able to learn the optimal policy.

C. Q - Learning

Q-Learning used the representations and reward functions as SARSA.

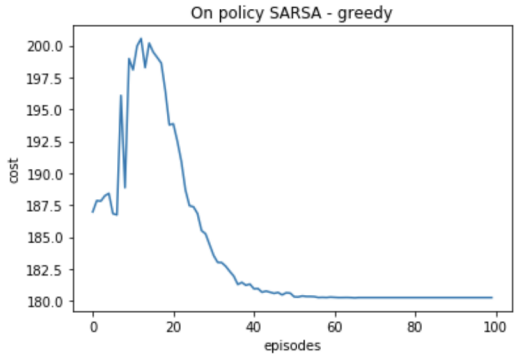


Fig. 14. Plot of cost function for on policy SARSA - greedy

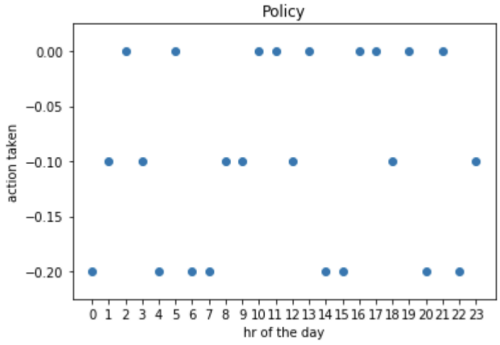


Fig. 15. Actions taken by on policy SARSA greedy

Model parameters:alpha=0.1 and gamma =0.9.

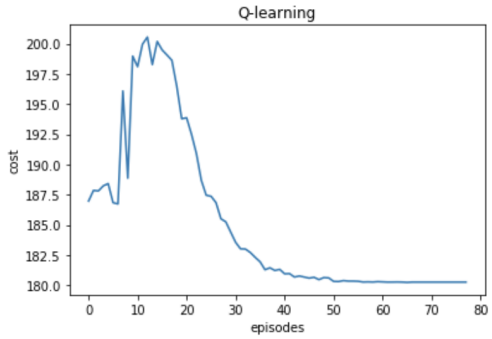


Fig. 16. Plot of cost function for Q learning

It performed similar to greedy SARSA.

V. Analysis

The reason for none of the algorithms to not work might be due to the way the reward function was defined. The reward function focuses on the amount of energy consumed to charge the storage device at night , which according to the definition of the actual reward function is proportional to the amount of energy drawn from the grid.

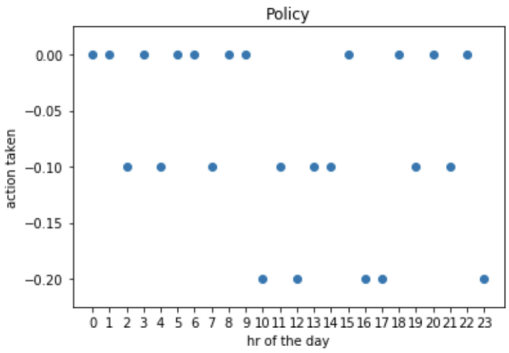


Fig. 17. Actions taken by Q learning

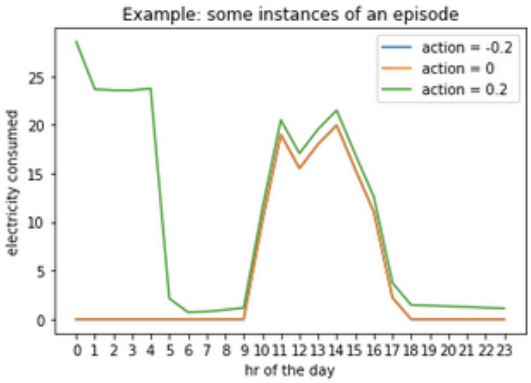


Fig. 18. Some instances of an episode

But we found that, for instance, at night even though lesser amount of energy will be required to cool the water and charge the storage device, the electricity consumption during night time increases (could be due to a myriad of reasons, such as heavy usage of air conditioning) and hence when soc is zero, if a positive action of charging the storage device is taken, energy will be drawn from the grid to charge the device and also to satisfy the electricity demand at that instance. Hence more energy will be consumed when a positive action is taken than when a negative action is taken, in which case electricity will be drawn from the grid only to satisfy the current energy demand.

Hence we defined a new reward function which depends only on the outside temperature and current soc and not on the total electricity consumed.

New Formulation: Different algorithms were tried on the environment and Q learning and greedy SARSA converged to the least cost amongst all algorithms tested in this environment but none of the algorithms converged to the optimal policy or a policy similar to that of the rule based controller.

We then hard coded a reward function which gives the agent a positive reward if it takes the desired action i.e. charge or discharge the storage device during certain hours of the day and a negative reward if it doesn't. This worked well, but information about the optimal policy was given away in the reward function and also this reward function cannot help decide upon the magnitude of the action that has to be taken.

We then tried defining a new reward function and a simplified environment which had the following spaces.

The state space consisted information about the hour of the day and the current state of charge (soc was binned into bins of size 0.1) of the storage device. In any state any of the actions belonging to the actions space $\{-0.2, -0.2, 0, 0.1, 0.2\}$ can be taken.

The reward function was defined as follows: Let T_{out} be the outside temperature.Our aim is the cool the water in the storage device to 8c (target temperature) with as much less power as possible. Minimum power will be required when the temperature difference between the atmosphere and the temperature of the water in the water tank is minimum.

Algorithm 1: New reward function formulation

temp = T_{out} (List of temperatures);
$\mu(meantemperature) = 12°C$;
$p = sign(\mu - temp$;
$a = actiontaken$;
$b = (sign(doc + p \times a - \epsilon - 1) + 1) \times p$;
if $action < 0$ **then**
  $c = sign(a) \times p \times action \times \epsilon \times sing(soc + a)$;
else
  $c = sign(a) \times p \times action \times \epsilon$;
end
$reward = (p + 1) \times a - b + c$;
$\mu = mean(or\ median)\ of\ T_{out}$

This reward function is better than the hard coded reward function 2 as using this reward function, the agent can learn, based on the outside temperature and soc, what action to take.

The aim of this reward function is the following:It is ideal to cool the water temperature to the target temperature when the outside temperature is less. μ kind of works like a threshold to decide whether to charge or discharge the storage device.

A. Testing the reward function

The reward function we formulated seemed to work well i.e maximum reward is obtained by taking the optimal action. The figure below shows the action taken for different values of soc.

Algorithm 2: CASE 1

$T_{out} < \mu$ (ideal time to cool water to supply it to storage tank);
if $soc = 0$ **then**
  A positive reward will be obtained if a positive action is taken, and value of the reward on the value of the positive action taken;
end
if $soc + positive\ action > 1$ **then**
  Positive reward will be obtained only for taking $action = 1 - soc$.
end
if $soc = 1$ **then**
  Positive reward will be obtained only for taking $action = 0$ i.e. not supplying any water to the storage tank.
end

Algorithm 3: CASE 2

$T_{out} > \mu$ (Ideal time to discharge the storage device to supply cooling energy to the building);
A positive reward will be obtained if a negative action is taken, and value of the reward depends on the value of the negative action taken, as long as $(soc + action) > 0$ i.e. as long as there if enough energy in the storage device to supply the building with the energy proportional to the negative action taken.

The reward function was tested was two cases,
Case 1: $\mu > T_{out}$ (ideal action that is to be taken is to either charge or remain idle when $soc = 1$.)
Case 2: $\mu < T_{out}$ (ideal action that is to be taken is to either discharge or remain idle when $soc = 0$.)

B. Testing the reward function for CityLearn environment

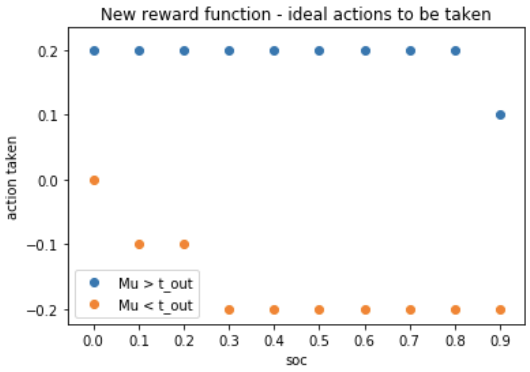
The following figure summarized the performance of the reward function for CityLearn environment.

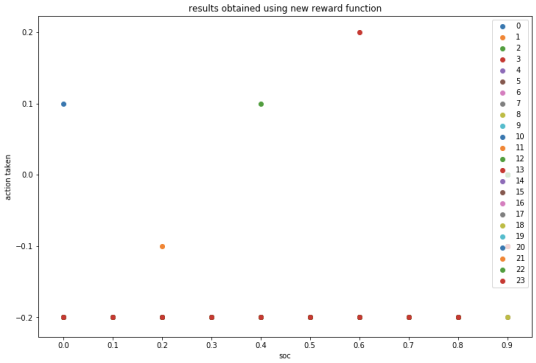
But when this reward function was tested using on policy salsa and Q-Learning in the city learn environment, it did not work and we were not able to figure out why.

VI. Conclusion

Different algorithms were tried out with continuous and discrete actions spaces, but performance of none of them was comparable to RBC. So we shifted the focus to coming up with a better reward function which the agent can learn and also which is not hard coded.

The environment and reward function were analyzed again and the reward function was modified to better suit the goal we were trying to achieve. The new formulation was tested using SARSA and Q-Learning, but satisfactory results were not obtained.





Also, it is observed from the data that electricity consumption from the grid might not be low at night as considered while designing the rule based controller, due to various reasons like heavy usage of air conditioning or heaters etc..

For designing an agent to facilitate such optimization it is imperative to first come up a proper reward function which can actually help the agent learn and also to define our state-action space and correct function approximation for the same.

REFERENCES

[1] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra, "Continuous control with deep reinforcement learning", 2015.

[2] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, "Proximal Policy Optimization Algorithms", 2017.

[3] Tianshu Wei, Yanzhi Wang and Qi Zhu, "Deep Reinforcement Learning for Building HVAC Control." IEEE, 2017.

[4] Ivars Namatēvs., "Deep Reinforcement Learning on HVAC Control". Information Technology and Management Science, 2018

[5] Ruoxi Jia, Ming Jin, Kaiyu Sun, Tianzhen Hong and Costas Spanos, "Advanced Building Control via Deep Reinforcement Learning". ICAE 2018

[6] Zhiang Zhang and et al., "A Deep Reinforcement Learning Approach to Using Whole Building Energy Model for HVAC Optimal Control," Ashrae, 2018

[7] Guanyu Gao, Jie Li and Yonggang Wen., "Energy-Efficient Thermal Comfort Control in Smart Buildings via Deep Reinforcement Learning" ArXiv, 2019.

[8] Vázquez-Canteli, J.R., and Nagy, Z., "Reinforcement Learning for Demand Response: A Review of algorithms and modeling techniques", Applied Energy 235, 1072-1089, 2019.

[9] Vázquez-Canteli, J.R., Ulyanin, S., Kämpf J., and Nagy, Z., "Fusing TensorFlow with building energy simulation for intelligent energy management in smart cities", Sustainable Cities and Society, 2018.

[10] Vázquez-Canteli J.R., Kämpf J., and Nagy, Z., "Balancing comfort and energy consumption of a heat pump using batch reinforcement learning with fitted Q-iteration", CISBAT, Lausanne, 2017

[11] Karl Mason and et.al. "A review of reinforcement learning for autonomous building energy management." ArXiv, 2019.

[12] Sunyong Kim and Hyuk Lim. "Reinforcement Learning Based Energy Management Algorithm for Smart Energy Buildings". MDPI, 2018.

[13] https://gym.openai.com/

[14] https://github.com/higgsfield/RL-Adventure-2/blob/master/3.ppo.ipynb

[15] Scott Fujimoto, Herke van Hoof, David Meger. "Addressing Function Approximation Error in Actor-Critic Methods". ICML 2018.

[16] https://github.com/intelligent-environments-lab/CityLearn