DEMAND RESPONSE OPTIMIZATION FOR MIRCOGRID CLUSTERS WITH DEEP REINFORCEMENT LEARNING

Ayush Sinha*, Mrigyen Sawant†, Harsh Kochar‡,
Adarsh Abhijat§, Ritik Seth¶, Priyatam Reddy Somagattu‖ and O. P. Vyas**

Department of IT
Indian Institute of Information Technology, Allahabad
Prayagraj, India
*pro.ayush@iiita.ac.in
†iit2018033@iiita.ac.in
‡iit2018049@iiita.ac.in
§iit2018056@iiita.ac.in
¶iit2018032@iiita.ac.in
‖iit2018093@iiita.ac.in
**opvyas@iiita.ac.in

***Abstract*—As energy request keeps on expanding, Demand Response (DR) programs in the power distribution scenario are acquiring force and their reception is set to develop steadily over the course of the years ahead for Smart Grid (SG) infrastructure. DR plans try to boost buyers to utilize environmentally friendly power energy and lessen their power utilization during peak periods which helps support microgrid operator adjusting of DR and produce income by selling overflow of energy back to the SG. This paper proposes a novel approach by implementing a Deep Reinforcement Learning(RL) model for handling and optimizing DR with the aim of maximizing overall benefits for both demand and supply sides and minimizing energy losses and utilization by adjusting prices (tariff) which will be based on the consumer's usage pattern at different intervals of time. The work uses custom environment generated based on Deep Learning model i.e. Long Short Term Memory(LSTM) for the load forecasting thus facilitates the proposed RL agent with Supply, Demand and prices at each time step. In continuation to this, the work deployed Asynchronous Actor-Critic Agent (A3C) model that interacts with the simulated environment and predicts the future prices based on the reward that it gets. Finally, we juxtapose the results with a conclusion showing optimum balance among prosumers and microgrid operator.**

***Index Terms*—Reinforcement Learning, Demand Response, Smart Grid, Asynchronous Advantageous Actor Critic algorithm**

I. INTRODUCTION

With the increase in the use of smart grid-enabled technologies which enable communication between the producers and consumers we need to implement technology architecture that takes advantage of this ability, and one of the best architectures for this is the Demand Response architecture, as evidenced by the myriad of literature out there endorsing its effectiveness in improving the efficiency of use of electricity and reducing the variability of the demand and supply [3][9][4].

Here our primary objective is to maximize profits for both demand and supply sides and reduce the variation between demand and supply of electricity by implementing demand response management through reinforcement learning by altering prices of electricity - what we will call "tariff" - based on demand response [2]. This is done in order to reduce energy losses.

II. RELATED WORK

There has been significant work into the domain of demand response in smart grids. Some methodologies have focused on directly changing the demand at the consumer's end and others have focused on changing demand indirectly by modifying a factor or a signal like price. In this section we discuss about a few methods such as demand-side management using Monte Carlo tree search, multi-agent residential Demand Response and demand response using a Day Ahead pricing model.

A. Demand-side management using monte carlo tree search

Monte Carlo Tree search (MCTS) is a RL algorithm that does random sampling to find optimal decisions in the decision space and builds a tree based on partial results. Originally, the Monte Carlo Tree Search (MCTS) was implemented for games and is now used as a method for optimizing demand response in smart grid [8]. The demand is managed by controlling the demand at consumer's end where the devices take autonomous decisions with the use of Monte Carlo tree search [8]. The algorithm decides when the device should run or should not based on the current price and also considers the behaviour of other devices at the time when the decision is taken. The main idea is that the devices compete for resources but at the same time have an objective to minimize the cost of electricity.

The experiment in the paper [8] considered four devices that will compete by deciding the best time for them to use electricity which emphasized the shifting of demand. The experiment was run for nearly 100 days and each day was divided into 30 minute slots. Both approaches were tested with the four devices and at the end of testing it was observed that the approach was able to significantly reduce electricity costs. This showed us that Reinforcement Learning was a very viable tool to optimize the demand response on the demand end.

B. Multi-agent residential demand response

The paper talks about using multiple agents to manage residential demand response through load prediction. The idea is to use reinforcement learning agents that will decide whether a device should be used or not, based on the current price and price predicted based on predicted future load. Each device on the consumer end is controlled by an RL agent which learns how to meet the energy goal of the device by a given target time at a minimum possible price.

The agents are implemented using the W learning algorithm, which is similar to Q learning but the agents have multiple Q learning policies that have different objectives and every action is associated with a W value [7]. In this multiple agent approach we have three types of agents: the load prediction agent that predicts future load, the price agent that predicts price for future load and the device agent that controls the devices.

The results [7] presented show that reinforcement learning is a suitable technique for residential demand response.

C. Demand response using a day ahead pricing model

This method unlike the other two methods mentioned doesn't directly affect the demand at the consumers end. The paper [6] proposes a pricing model that calculates day-ahead prices so that the consumer can plan when to meet their energy requirement. The authors [6] talk about real-time pricing being implemented in the real world such as a day-ahead real-time pricing (DA-RTP) tariff used by Illinois Power Company in the United States.

The results showed that the use of the Bender's Decomposition [5] optimization technique enhanced the reliability and efficiency of the grid. It was also seen that posting optimal day-ahead real-time prices encouraged the consumers to shift their demand to periods of low prices.

Our approach is focused on using price as the signal that will affect the change in demand, and therefore optimize the demand response. The difference is that this is done by the agent acting as broker between distribution companies and consumers, and works on the supply side - something that hasn't been done until now. Our model of consumers also takes into account consumers who are small-time producers of electricity, which means it also applies to scenarios where there are multiple suppliers of electricity.

III. Dataset

In order to train the simulated environment that we use to train the reinforcement learning agents, so that it mimics the real smart grid environment as close as possible, we use a dataset derived from the public data available at the Independent Electricity System Operator (IESO). The IESO contains various reports generated by the Ontario's power grid systems, and provides detailed reports indicating the demand, supply, tariffs, and other relevant parameters. The dataset we used had the data of the time ranging from 2010-01-01 to 2019-12-20, with each data point sampled at five minute intervals [1].

We reduced the number of columns from 47 to 13 columns by removing the columns which had a negligible correlation (whether positive or negative) with the demand and price. This was necessary since the greater the number of columns, especially the ones whose values had negligible effect on the core variables (demand and price), the harder it would be to train the LSTM to accurately simulate a smart grid in response to the agent's actions.

Since our dataset was of a time range for which we couldn't get the relevant supply data, we analyzed the latest supply and demand data from IESO and analysed its correlations and calculated a supply column whose correlations with the other variables were equal to that of the latest data. This was necessary to calculate the reward, especially since our model of the smart grid assumed that most consumers were also small-time producers of electricity, which meant their demand of electricty would fluctuate. A bad scenario for the producer running this agent would be when their supply of electricity is greater than the demand, which means the agent (and therefore the producer) would have to pay the consumers to consume their electricity. Without a supply value, there is just no way to model this.

IV. Environment

The environment uses the OpenAI Gym library as a standard API for the agent's use. This enforces a limitation: all the inputs the environment gets from the agent, and sends to the agent, are either simple integers or floating points, or numpy arrays.

The API consists of these main functions:

- $reset()$: Send the initial state to the agent.
- $step(action)$: The most important function; it involves taking the $action$ variable from the agent and compute the next state and the corresponding reward, which is then sent to the agent.

A. Actions

In reinforcement learning, there are two kinds of action spaces we can use: discrete and continuous. The bigger the action space is, the more complex our environment is for the agent to comprehend, which means that training the agent to give the right outputs and to process its inputs correctly takes even longer.

We have chosen to not use the more complex formulation to focus on correctness of the system and to ensure that training the system was not infeasible on a personal desktop computer. This means we use a fixed set of discrete actions, each of which increase or decrease the price by a fixed percentage, and an optional action that results in no change in price.

We have experimented with removing the ”no change” action, and that has resulted in a faster training of the agent.

B. Step Workflow

Here is a detailed workflow of the step function:

- Verify action is legal.
- Get the next state by sending the history of environment states (including the current environment state) to the LSTM.
- Calculate the new price by taking into account the effect of the action.
- Set the price of the next state to the new price that has just been calculated.
- Calculate the non-normalized reward based on the new price and the demand and supply values of the *next state*
- Add the next state to the historical record of environment states.
- Return the next state and the non-normalized reward to the agent.

Focus on the fourth point: ”Set the price of the next state to the new price that has just been calculated.” What we are doing is updating the price of the *next state*, not the current state. Similarly, when we are calculating the reward, we are using the new price - the price of the *next state* - and the demand and supply values of the next state. The reward is calculated based on the output of the next state, not based on the current state.
This is because we have decided to make any state returned to the agent as immutable. If we assume the current state as mutable, then whatever action the agent sends changes the price of the *current state*, making the price sent as part of the current state in the previous step as irrelevant information likely to confuse the agent’s calculations.
Next, our reward calculation function only takes one specific state into account when calculating the reward. It would make no sense, for example, to use the price of the previous state and the demand and supply variables of the current state. Since we intend to use the action’s effects as part of our reward calculation, we have to take into account the state that has been affected by the action, which is the next state - the state that will be returned to the agent once this function completes computation.

C. Reward function

The reward function is created keeping a few goals in mind:

- Ensure that the demand is greater than supply, so that the producer makes a profit, instead of having to pay consumers to consume electricity.
- Ensure that there is a buffer of demand, so that sudden changes in supply or demand do not reduce profitability of the producer by a huge amount.
- Ensure that the price of the electricity is not extremely high, which will cause long term reduction in demand.

To that end, we use the following formula to calculate the reward for some optimal value of x and y:

$$reward = |(demand - supply)^x| * |newPrice^y| * correction$$

Where correction is dependant on the below criteria:

- Whether $newPrice$ is within bounds.
- Whether $demand - supply$ is negative, or $newPrice$ is negative.

$newPrice$ is the non-normalized value of the price that the action has sent to the environment. We are also using non-normalized values of demand and supply for this function.

$correction$ is simply a factor used to ensure that the $newPrice$ is within the defined bounds, which ensures no exploding price or vanishing price problems. This occurs by modifying the reward to punish the agent for getting too far out of bounds.

D. Dataset normalization

We normalize the dataset to make training the neural networks easier.

By default, we use this normalized data for every operation of the environment and the reinforcement learning agent. This also means that the data generated by the LSTM is normalized, and the data that the policy network and the critic network expects is normalized. The only exception to this is the reward function. The input to the reward function is not normalized, and therefore the reward output is therefore based on the non normalized values. This reward is passed to the agent, and therefore even the agent, in an indirect sense, uses the non-normalized values.

V. REINFORCE

We implemented a vanilla policy gradient algorithm conventionally called a REINFORCE algorithm using PyTorch. This reinforcement learning agent interfaced with the simulated environment, which provided us with valuable feedback about the baseline performance of policy methods for our problem and helped us to refine the design and fine-tune the parameters of the environment.

This series of rigorous testing also exposed the limitations of REINFORCE: the greater the action space, the more computational resources it would take for the agent to learn a sensible policy. Using a continuous action space, as we found out, was entirely out of the question, even for better policy gradient algorithms. In the end, we had to balance the amount of actions by minimizing it, while also maximizing the impact the agent’s actions had on the environment.

VI. AGENT

A. General Architecture

1) Manager Agents and Worker Agents: In a standard A3C implementation, there are no distinctions between the different kinds of agents - while they have their own copy of the policy network, the critic network, and an instance of the environment, they all do the exact same thing, which is that they update the global model (the policy network and the critic network) asynchronously.

In contrast, in our implementation, we have different kinds of agents, described as manager agents and worker agents. The manager agents hold the sole policy and critic network and is responsible for updating these networks, whereas the worker agents are actually responsible for interacting with the environment, gathering the trajectories, and calculating the losses, and signalling the master agent to update the network.

The worker agents do not have local instances of the policy and critic networks. Instead, when they calculate the loss, they signal the manager agent to asynchronously update the sole policy and critic networks.

2) Offline Workflow: In the offline workflow we are training the sole policy and critic networks to have a headstart when they will operate in the online mode on the real environment. The environment we use is the simulated one that consists of the LSTM, and the worker agents act on that environment and attempts to improve the network.

3) Headstart Modification: In the online mode where the agent interacts with a real grid, if it starts from a random policy, it would take a very long time for the networks to achieve an acceptable level of performance.

The headstart modification alleviates this problem by pre-training the policy and critic networks on a simulated environment in the offline mode so that in the online mode , The agent starts from a trained policy network rather than a random one which results in better convergence rate.

4) Online Workflow: In the online workflow, the agent interacts with the actual environment, that is the smart grid, and in contrast to the offline workflow, the agent waits for some number of timesteps to gather a trajectory, then uses this to update the network, flushes this trajectory and repeates this indefinitely.

In this mode the agent keeps on learning and adapting to the environment and tries to take the most optimal actions possible.

This online workflow can also be deployed on multiple smart grids simultaneously with each worker agent controlling the price of one grid and utilizing the same actor and critic networks with asynchronous updates.

B. The problem with Sliding Window model updates

We had devised two approaches to implement the online real-time training of the agent. One of them is the sliding window model update workflow, which has some theoretical shortcomings.
In this method we create a deque data structure which contains the SAR (State,action,reward) tuples of a trajectory. At every call of env.step() , the agent records a new SAR tuple, removes the oldest entry from the deque and appends the new tuple, then uses this window to calculate $V_{Predicted}, V_{Target}$ and $Advantage$ values to update the networks.
Suppose the window is of size n at which the model updates the policy π_θ to π'_θ then at $n+1$ step, model makes use of updated policy π'_θ to generate a new SAR tuple, then again uses this window, which is now a mixture of SAR tuple of multiple policies to update the networks.(Since last action is generated by π'_θ whereas the first $n-1$ are made by of π_θ).

For the $(n+1)_{th}$ state, however, the trajectory is not an accurate indicator

Due to the use of multiple policies to improve the networks and calculation of V_{Target}, the algorithm will not be able to correctly improve the policy and critic networks for the current policy. Thereby making it inaccurate.

For the $(2n)_{th}$ step, all the SAR tuples in the queue are ones generated by different policies and are used to calculate V_{target}, making it an extremely inaccurate measure, which is being used to optimize both the policy function and the loss function.Therefore we conclude that the sliding window policy is incorrect.

C. Episodic model updates

The episodic model updates is an alternative to the sliding window model update workflow that doesn't have the issues it does and therefore is usable with an on-policy reinforcement learning algorithm like A3C.

Here we create a trajectory using the same policy, and in an episodic manner, update the networks using the trajectory, flush the trajectory, and repeat. This ensures that the same policy is used for generating a particular trajectory, making the algorithm on-policy as it should be.

VII. Experiment Setup

A. Environment

In order for us to test the agents, we need to set up a proper simulated environment which mimics the real environment as close as possible. To achieve this, we trained our custom LSTM model on the Ontario dataset and tracked its performance, making sure that it performs as optimally as possible.

We considered using an environment that directly iterated over the dataset itself, however that very idea is flawed since the aim is to model the effects of the agent's actions on the environment, and one cannot do that if by simply iterating over the original dataset.

B. Action Space

Our environment supports a discrete action space in which the agent increases or decreases the current price by a fraction of the current price. For example, if the action space is [-5, -3, 0, 3, 5], then the agent can increase or decrease the price by 3 or 5 percent of the current price, or it can keep it the same by choosing the third action, which returns the value of 0 to the environment.

This method of defining the action space has allowed us to significantly reduce the training time as opposed to using a continuous action space, which allows the agent extreme flexibility and choice when modifying the price. Choosing a discrete action space allows us to trade extreme flexibility in modifying the price for a reduced training time.

For the tests specifically, we use the following action space: [-20, -15, -10, -5, 5, 10, 15, 20]. By removing the action of keeping the same current price, we force the agent to learn faster since it cannot spam the action without getting useful feedback.

C. Training

The REINFORCE algorithm is a purely policy gradient algorithm, which uses the reward function directly as a reinforcing signal. The policy network has the following structure:
Layer 1 (Size=13) $\rightarrow$ TanH Activation $\rightarrow$ Layer 2 (Linear) (Size=150) $\rightarrow$ Output Layer (Size = 8)

We trained this policy network with the Adam optimizer, with a learning rate of 0.009. The discount factor is 0.99 and an experience buffer with a maximum length of 800.

For training of the A3C agent, we have deployed it on our custom simulated LSTM enabled environment.

The A3C model Policy network has following structure:
Layer 1 (Size=13) $\rightarrow$ HardTanH Activation $\rightarrow$ Layer 2 (Linear) (Size=20) $\rightarrow$ SeLU Activation $\rightarrow$ Layer 2 (Linear) (Size=30) $\rightarrow$ SoftMax Activation $\rightarrow$ Output Layer (Size = action space)

The A3C Critic network has following structure:
Layer 1 (Size=13) $\rightarrow$ HardTanH Activation $\rightarrow$ Layer 2 (Linear) (Size=15) $\rightarrow$ SeLU Activation $\rightarrow$ Layer 2 (Linear) (Size=10) $\rightarrow$ HardTanH Activation $\rightarrow$ Output Layer (Size = 1)

We trained this policy and critic network with the Adam optimizer too, with a learning rate of $1e^{-3}$, the discount factor is 0.99, and the trajectory length of 50.

VIII. Results

A. REINFORCE performance

The performance of the REINFORCE algorithm serves as a baseline with which we compare the A3C algorithm with. Figure 1 shows the average model price set by the trained REINFORCE agent, versus the exchange (that is the $demand - supply$). As is evident from the plot, the agent does not create a buffer of exchange.

Figure 2 shows the profits accumulated by the REINFORCE agent. The plot makes it clear that the profits generated by the agent are better than the profits that would be generated if the agent hadn't been used. However, the performance is far from adequate, since even the maximum profits generated by the model is half that of the maximum profits generated if the agent wasn't even used. Since the order of magnitude of the profit is $1e6$, this is an immense amount of profit being left at the table, so to speak.

B. A3C performance

Figure 3 shows the results of training the agent. This has multiple implications, all positive:

- The agent consistently keeps the demand greater than the supply. This ensures that the producer is profitable.

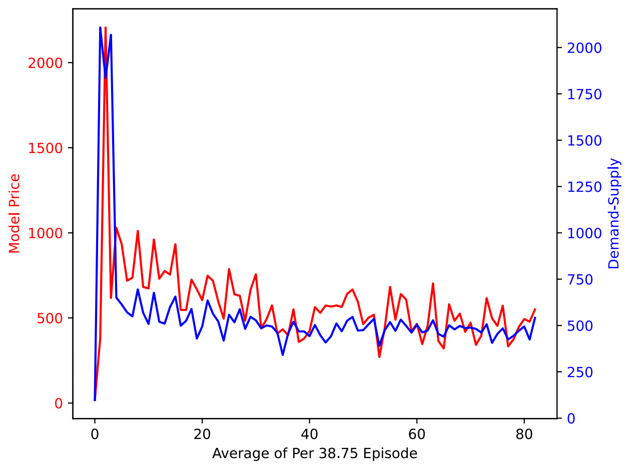

Fig. 1. Average model price versus exchange (demand - supply) of the trained REINFORCE model

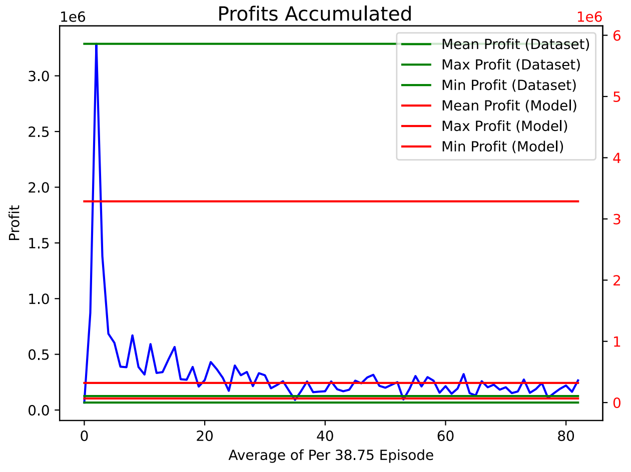

Fig. 2. Profits accumulated by a trained REINFORCE agent compared with that made by a producer that did not use an agent at all

- The agent consistently keeps a buffer between the demand and supply values. This ensures that in the occasion of sudden increases in supply, there is still demand for electricity to absorb the increase in supply, ensuring that the producer is still profitable.
- The agent endeavours to decrease the price in such a manner as to keep profits high. This benefits both the producer and and consumers.

C. Online performance of our A3C model

In the online real-time training of the agent, Figure 4, 5, and 6 show the results of the online training of the agent with the episodic method of model updates (that is, we do not use the sliding window approach here) and use a pre-trained network that uses the head-start approach described earlier.

The implications of the graphs are:

- From Figure 4 we can observe that the agent constantly keeps the demand greater than supply, keeping a buffer

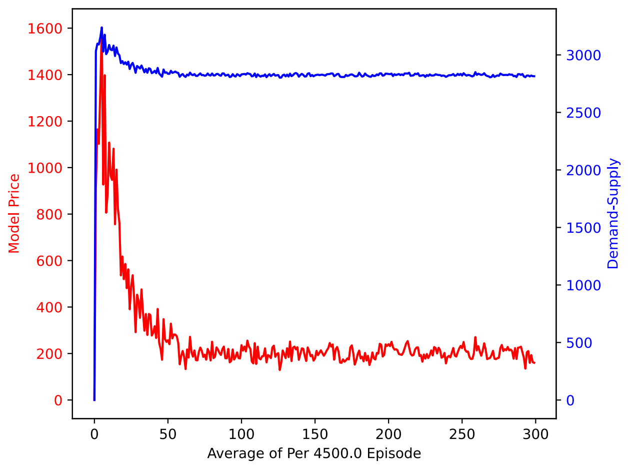

Fig. 3. Demand-supply vs Model price curve predicted by our RL model keeping their product to be maximized for maximum profit based on our assumptions

between the demand and supply, ensuring the profitability of the producer.

- While maintaining the buffer between Demand and supply, The agent tries to minimize the Price , ensuring the profitability of consumers.

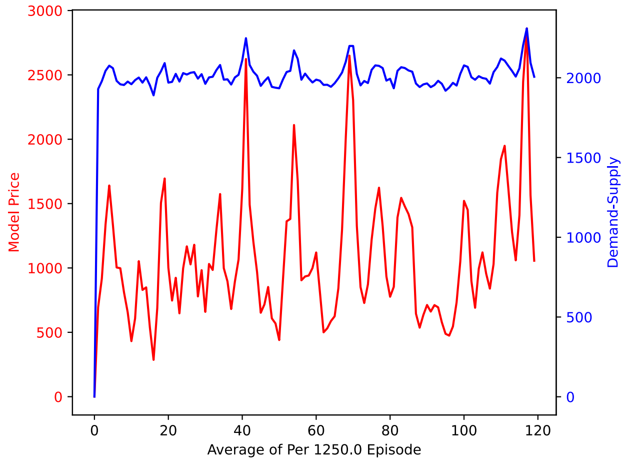

Fig. 4. Average Model Price vs Exchange(Demand-Supply) curve predicted by our RL model keeping their product to be maximized for maximum profit based on our assumptions

- Figure 5 shows that the mean profit accumulated by the agent is much higher than the dataset profit, implying that the agent learns to maximize the profits while keeping the price within acceptable limits.

IX. Conclusion

In this paper, we presented a deep reinforcement learning approach for optimizing demand for smart grids. Our approach began with creating a custom simulated environment using LSTMs and OpenAI to train and evalute the RL algorithms we implement. For this, we used the Ontario dataset to

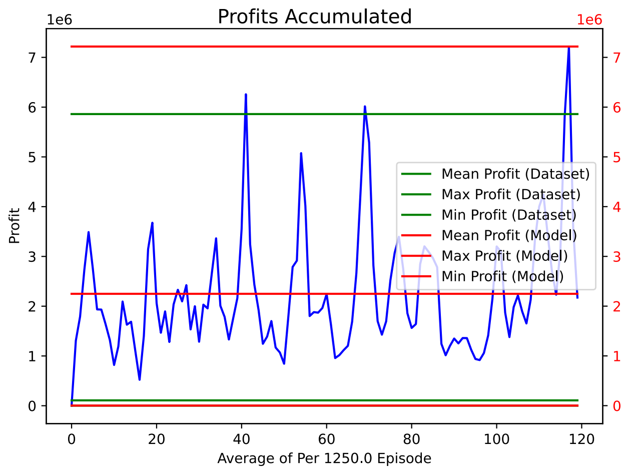

Fig. 5. Profit curve along with the model's and original dataset's mean,max and min profits

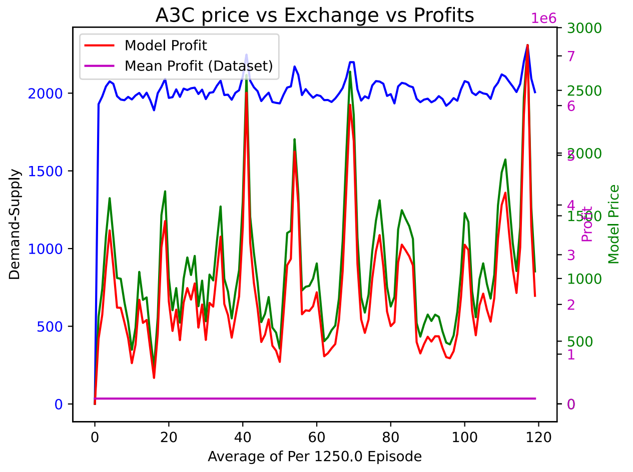

Fig. 6. Average Model Price vs Exchange vs Profit curve predicted by our RL model keeping the product of price and exchange to be as maximized and thereafter ensuring optimum profit

train the LSTM model and simulate a smart grid. We then implemented the REINFORCE RL algorithm as a baseline for the performance of reinforcement learning algorithms. With the performance of the REINFORCE as a baseline, we next implemented the A3C algorithm which makes use of deep neural networks - the policy and the critic network - and uses the policy network to adjust the price in the smart grid, and uses the critic network to calculate the advantage, which acts as a reinforcing signal for the policy network. In order to make the online training of the algorithm more efficient, we have used a headstart modification which makes the agent use a pre-trained set of networks instead of completely random ones, thereby improving the initial performance and rate at which it achieves acceptable performance. The results showed that our approach outperforms that of REINFORCE by a considerable order of magnitude; it not only optimizes the profit, but also maximizes the difference between the demand and the supply, while keeping the price within the stated bounds. In future work, we intend to modify our A3C implementation to use Generalized Advantage Estimation, and implement the deep reinforcement algorithms PPO and DPPG for optimizing the demand response to gain better performance in comparison to the A3C implementation.

X. References

[1] Independent Electricity System Operator (IESO). *Power Data*. URL: https://www.ieso.ca/power-data.

[2] Sashank Mishra Agam Dwivedi Ruchin Agrawal. *Autonomous Decision Making in Smart Grids*.

[3] Ashfaq Ahmad et al. "Demand Response: From Classification to Optimization Techniques in Smart Grid". In: *2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops*. 2015, pp. 229–235. DOI: 10.1109/WAINA.2015.128.

[4] Mohammed Albadi and Ehab El-Saadany. "A summary of demand response in electricity markets". In: *Electric Power Systems Research* 78 (Nov. 2008), pp. 1989–1996. DOI: 10.1016/j.epsr.2008.04.002.

[5] J.F. Benders. "Partitioning procedures for solving mixed-variables programming problems". English. In: *Numerische Mathematik* 4 (1962), pp. 238–252. ISSN: 0029-599X. DOI: 10.1007/BF01386316.

[6] Meysam Doostizadeh and Hassan Ghasemi. "A Day-Ahead Electricity Pricing Model Based on Smart Metering and Demand-Side Management". In: *Energy* 46 (Oct. 2012). DOI: 10.1016/j.energy.2012.08.029.

[7] Ivana Dusparic et al. "Multi-agent residential demand response based on load forecasting". In: *2013 1st IEEE Conference on Technologies for Sustainability (SusTech)*. 2013, pp. 90–96. DOI: 10.1109/SusTech.2013.6617303.

[8] Edgar Galván-López et al. "Autonomous Demand-Side Management system based on Monte Carlo Tree Search". In: *2014 IEEE International Energy Conference (ENERGYCON)*. 2014, pp. 1263–1270. DOI: 10 . 1109 / ENERGYCON.2014.6850585.

[9] K. Kuroda, T. Ichimura, and R. Yokoyama. "An effective evaluation approach of demand response programs for residential side". In: *9th IET International Conference on Advances in Power System Control, Operation and Management (APSCOM 2012)*. 2012, pp. 1–6. DOI: 10.1049/cp.2012.2175.