Optimal Peak Shifting of a Microgrid Load Based on Deep Q-Learning Network

Mostafa Algabalawy (✉ mostafagabalawy@gmail.com)

Ain Shams University

Nesreen Hosny

Cairo University

Original article

Keywords: Microgrid Control, Peak Shifting, Optimal Peak Shifting, Reinforcement Learning, Deep Reinforcement Learning, Q-Learning Network, Deep Q-Learning Network

Posted Date: July 6th, 2020

DOI: <https://doi.org/10.21203/rs.3.rs-37008/v1>

License: © ⓘ This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Optimal Peak Shifting of a Microgrid Load Based on Deep Q-Learning Network

Mostafa Algabalay¹, Nesreen S. Hosny¹

Abstract

Background: Peak periods are a result of consumers generally using electricity at similar times and periods as each other, for example, turning lights on when returning home from work, or the widespread use of air conditioners during the summer. Without peak shifting, the grid's system operators are forced to use peaked plants to provide the additional energy, the operation of which is incredibly expensive and dangerous to the environment due to their high levels of carbon emissions.

Methods: Battery storage system (BSS) has been used to allow for the purchase the energy during off-peak periods for later use, with the primary objective of achieving peak shifting, is explored. In addition, reduction in energy consumption and the lowering of consumer's utility bills are also sought, making this a multi-objective optimization problem. Reinforcement learning methods are implemented to provide a solution to this problem by finding an optimal control policy which defines when is best to purchase and store energy with the objectives in mind.

Results: This achieves over a 20% reduction in energy consumption and consumers' energy bills, as well as achieving perfect peak shifting, thereby removing peaking plants from the equation entirely. This result was obtained using a simulator that the author has developed specifically for this task, which handles the model training, testing, and evaluation process. Secondly, the development of a novel technique, automatic penalty shaping, was also found to be crucial to the success of the learned model. This technique enabled the automatic shaping of the reward signal, forcing the agent to pay equal attention to multiple individual signals, a necessity when applying reinforcement learning to multi-objective optimization problems. The policy does, however, attempt to overcharge the battery about 7% of the time, and promising methods to address this has been proposed as a direction for future research.

Conclusion: The aim of this task was to verify that reinforcement learning is a suitable solution method to the peak demand problem. That is, can reinforcement learning be used in conjunction with a BSS to purchase energy at off-peak periods, in order to flatten the energy requirement

profile of consumers. Such an achievement would prevent the grid's system operators from needing to use peaked plants to provide additional energy during peak periods, lowering carbon emissions and energy prices for the consumer. This peak-shifting would allow the grid's system operators to be able to more easily predict electricity demand, thereby reducing their need to generate more energy than necessary, again lowering the tariffs for energy for the consumer. Secondary aims of directly reducing the energy consumption and utility bills were also sought, making this a multi-objective optimization problem. The used data, in conjunction with the created simulator which performs the full training and testing phases of the models, to find an optimal policy using the deep Q-network (DQN) and Proximal Policy Optimization (PPO) reinforcement learning algorithms. Finally, the proposed algorithm is able to achieve perfect peak shifting, a reduction in the monthly utility bill by 21% and also a reduction in energy consumption by 23%, achieving all of the aims of the task.

Keywords: Microgrid Control, Peak Shifting, Optimal Peak Shifting, Reinforcement Learning, Deep Reinforcement Learning, Q-Learning Network, Deep Q-Learning Network

1. Introduction

A significant problem in the generation and consumption of energy is that of *peak demand*. Although energy usage varies from household-to-household, season-to-season and many other factors such as location, there are still numerous trends in energy usage which can be predicted. For example, when people return home after work, they may turn on the television, or start cooking. Before they sleep, they may turn on the air conditioner if the temperature is far from comfortable. These trends result in times of significantly higher energy demand, known as peak periods, and can be incredibly difficult for the grid to manage. At times of high demand, *peaking power plants* generally need to be used to provide the additional required energy. This is especially the case when moving towards the intermittent decentralized network at times when the forecasted demand for energy is significantly lower than the actual demand. These peaked plants are harmful to the environment, generally generating energy through inefficient combustion of fossil fuels, and are also incredibly expensive to operate [7]. These higher costs are covered by the consumer, resulting in much higher than should-be-necessary utility bills. In addition to this, these peaked plants cannot be turned on instantly and generally take at least an

hour to become functional, adding yet another layer of uncertainty and potential for power outages at times of high demand.

The authors in [15] solve a similar problem to peak demand with a BESS by using a mixture of regression methods for load profile forecasting and dynamic programming (DP) for optimal *action* selection, where action refers to buying and storing energy. Although the results of this paper are impressive, there are many limitations which arise from their solution framework for this task. Firstly, in their simulation environment, each day is approached separately, by first predicting energy usage throughout the following day based on historical data, and without updating any of these beliefs until the end of the day. This is disadvantageous in that as we progress through the day, the actual load profile that we encounter provides additional information which can be bootstrapped from to update the belief over the load profile for the rest of the day. Secondly, their approach first splits each day into $n\psi$ time step separated by intervals of size Δt , and then finds the optimal actions at each of these time steps. In addition, the DP algorithm described within requires discretization of the battery capacity, thereby prohibiting a fully flexible, continuous energy storage system. Decreasing the time step interval and capacity discretization provides additional fine-grained control, but as DP grows polynomial in the problem size this quickly becomes computationally intractable [15], thereby restricting the flexibility of this solution methods. Finally, DP requires an explicit objective function for optimization, and as discussed previously, this is not available for our task as we are not interested in solely peak shifting, but also in energy consumption and utility bill reduction. Many other methods have been attempted to solve related problems, for example, the authors of [16] use an online modified greedy algorithm. This addresses the problem of computational complexity suffered by DP methods, but with the additional problem of requiring bounds on the pricing structure and not being able to constrain the storage size of the BSS [17]. These problems prevent their solution from being readily applicable in practice and yet again additional solution methods need to be researched into. Noticing the numerous issues with these methods, the authors of [17] tackle the problem of energy storage arbitrage using a reinforcement learning approach. Such methods can learn a policy independent of the price distribution and therefore can perform well even in an environment with a non-stationary price profile, which DP cannot [17]. The authors of this paper learn an optimal policy via a simple Q-learning algorithm verifying that reinforcement learning is a sufficient solution to their task. However, the task

researched into in their paper is just simple energy arbitrage, that is, the problem of buying energy at cheap prices and storing it to later sell back at higher prices for profit. It does not cover the more difficult task of peak shifting with non-stationary demand (energy limit) profiles, which also requires learning to forecast the consumer's energy consumption and generation profiles in the background. Although their problem definition is simple, their results far outperform those presented in non-reinforcement learning domains, and provide numerous advantages that make the application of reinforcement learning appear promising, based on its ability to learn a policy in a highly complex, dynamic, continuous, non-stationary environment without the need to explicitly define an objective function. As such, this report will investigate whether such reinforcement learning methods can be extended to provide a suitable solution to the more complicated task at hand.

This paper outlines a solution method to the peak demand problem utilizing a battery storage system (BSS), with the goal of learning an optimal policy of buying and storing energy in an incredibly dynamic and complex non-stationary environment via the use of reinforcement learning. A BSS has been chosen due to the fact it is becoming more and more feasible in recent times due to lowering costs, and also as it is easy to install, repair, replace and is being used by many companies to-date, verifying its effectiveness as an ESS [11]. More importantly, however, a simple battery is easy to model programmatically. Based on the data made available by *Informatics*, the end-user will be various households situated in Japan. This is an important stipulation, as the size of batteries used by households will be significantly different to that used at large facilities, and of course as different countries exhibit varying energy protocols, as discussed previously. It should be noted that the use of batteries introduces some additional complications. The first of these being that of *capacity fade*, a phenomenon whereby the maximum

capacity, or upper limit of stored energy, depletes over time. Secondly, *self-discharge* is another degrading phenomenon of batteries which refers to the stored energy being automatically discharged over time, similarly to a small leak in a storage tank. These two issues will both affect the optimal policy of buying and storing energy, but for now will be ignored due to their relatively small effect and in order to maintain an effort of keeping the approach as general as possible to various ESS. The goal is not to provide the exact optimal policy for a specific ESS for a specific household, but more to verify the effectiveness of reinforcement learning to the peak

shifting solution, where the choice of country, end-user and ESS simply are conveniences to aid in testing and evaluation.

The goal will be to achieve optimal peak shifting, thereby nullifying the need for use of peaked plants. With this achieved, the secondary aims will be to both lower utility bill costs and reduce overall energy consumption via leverage of on-site generated energy. Since there are multiple objectives, it is difficult to numerically assign a single metric to this task and in fact impossible to find a solution that simultaneously optimizes each objective independently. For example, consider two models which both exhibit positive traits. The first achieves near-perfect peak shifting but still requires the use of peaked plants once per year, but also reduces utility bills by 50%. The second model achieves perfect peak shifting but causes energy bills to skyrocket. Although the primary aim is peak shifting, employing the first model would grossly disadvantage the consumer and therefore would most likely be rejected. Without a clear balance of these aims, and additional subjective preference information, such a single metric function cannot be created and therefore each model will be evaluated on their merits. Previous research will be examined in order to understand the current state-of-the art in reinforcement learning-based approaches to peak shifting. After deriving the necessary reinforcement learning algorithms, the dataset will be introduced, followed by an overview of the solution framework. The results will then be evaluated, with a report on the effectiveness of such reinforcement learning methods in general to the problem of peak shifting, with a discussion on which algorithms specifically are best suited to this problem, if any. Following the evaluation of the models and algorithms, there will be a short discussion on areas which are owed additional thought and research to aid further development of this field.

2. System Description

Before exploring the various reinforcement learning methods, the data available for this task will first be examined. A comprehensive understanding of this data will allow for a more informed decision on which reinforcement learning algorithms are most applicable to this domain. The datasets for this task consist of what will be referred to as observational and household data, each consisting of minute-by-minute data points over a period of six months. The observational data comprise of three different datasets; the prices of electrical energy from the grid, weather data, and finally demand data, which is the technical term for the limit of available energy from the grid. As demand can also refer to how much energy is requested by the household, these terms

will be used interchangeably throughout, but the meaning will always be clear from the context. The price and demand datasets consist of multiple different plans, i.e. different pricing and demand profiles, but during the model learning phase only one plan for each will need to be chosen.

Figure 1 presents three days of pricing data using four different pricing structures; smart life plan, night-8, night-12 and random. Random is artificially created data; whereas the others are referring to actual pricing plans offered by Electric Power Company. Night-8 and night-12 are plans for customers who use more electricity at nighttime whereas the smart-life-plan is for customers moving towards smart homes who will in general be using electricity a lot throughout both day and night [25]. A successful algorithm should be able to train a model irrespective of which pricing plan is chosen, assuming there is some sort of structure to be learned. For this reason, random will not be used as it simply introduces unnecessary noise into the learning process. During the testing stage, one of the remaining plans will be chosen at random. Figure 2 gives four different meteorological datasets which can be taken advantage of to predict energy generation via photovoltaic. The model training process should be flexible enough to allow for any combination of these datasets to be used, noting that it may be possible to accurately forecast energy generated without the need of the rainfall data, for example.

Figure1 Pricing data for the first three days of Jan.

Figure 2 Normalized weather data giving the temperature, sunlight, rainfall and daylight for the first three days of Jan.

Note that although the data presented has been normalized, this does not affect the learning process as all data will be standardized before being used. Also note that this data does not necessarily represent the actual weather at the household, but rather at the closest meteorological site to the household. This difference in location will introduce some noise into the forecasting. Figure 3 presents four different demand plans over the first three days of January, here these data show that demand refers to the maximum amount of energy available to be purchased at any time it is necessary that the model be able to forecast the chosen plan well to aid in peak shifting. A model which successfully learns to peak shift will never try to purchase more energy than given by this graph. All data here has been artificially created by Informatics as public data on demand profiles is not yet available in Japan. It should be noted that the data here has been slightly modified from the raw values. Firstly, all values were first divided by 1000, to convert from W to kW, and then again by 60 to convert from kW to kWh, thereby giving the actual energy available for each single-minute time step. Due to a numerical error in the provided data, where the curator originally attempted to convert from W to Wh but incorrectly multiplied by 60 rather than dividing, this data has then been divided again by 60 to achieve the correct values.

Figure 3 Four different artificially created demand plans over the first three days of Jan.

In total, Informatics has prepared datasets for four different households in Japan. The datasets comprise of what is known as NILM, or Non-Intrusive Load Monitoring data. Usually when monitoring the energy consumption of a household, the data is simply a single number stating the overall energy consumption in kWh over all appliances and devices. However, NILM is an energy disaggregation method which attempts to provide consumption data of individual appliances to provide a clearer picture of exactly which devices are consuming how much energy at different times. This NILM data has been disaggregated using software being developed at Informatics and should provide an additional useful signal to the agent when forecasting data. For example, being able to exclusively monitor the energy used by the air conditioner can allow the agent to learn how a household responds to varying temperature levels. More clearly, one household may choose to use the air conditioner when the temperature, given by the weather data, is greater than 30°C, whereas another may never use their air conditioner. Without the disaggregated NILM data the agent would find it much more difficult to learn the energy profiles of these two different households. Each household has an associated ID key but for the sake of brevity will be referred to as A, B, C and D hereon. Table 1 provides a mapping between the original IDs and alphabet keys for future reference.

Table 1 Mapping of original household IDs to a simpler alphabet referencing structure.

Building Style	ID Key
A	0002_9100000042

B	0002_9100000112
C	0002_9100000113
D	0002_9100000114

Before using the data to train models, it is first important to get a better understanding of its structure. On first inspection it is apparent that over the six-month period there are many *NaN* values present for some of the households. This is shown in detail in Tab. 3.2. *NaN* values refer to missing appliances, in the case of *dishwasher* for household D, or times when the household temporarily removes the logging device from that device, preventing the NILM data from being measured. The data-processing protocol will be explained more in-depth during discussion of the simulator, but for now it is important to note that all *NaN* data have been replaced with the mean of that column for that specific household. This is just one choice out of many, such as filling with zeros, or interpolating values. Zero-filling was rejected as this would result in too much information loss. For the columns with large number of missing data, such as the 5882 counts for household B and the 99079 count for the TV column for household C, it was seen that these missing values were all situated in a single period of time, i.e. values before and after this period were not *NaN*. This would make interpolation incredibly difficult, especially considering the seasonal nature of this data and so the interpolation filling-method was also rejected in favor of mean-filling. It should still be noted that mean-filling ignores the variation in data due to seasonal effects but should result in minimal information loss compared to the other two considered methods. All NILM data have been divided by 60000 to convert from *W* to *kWh* and then again by 60 to correct for a similar mistake as in the demand data explained previously. The main entry refers to the total energy usage, that is the sum of all measured appliances (apart from photovoltaic) plus the consumption of unmetered devices.

Table 2 *NaN* counts for each NILM entry

Field/Household	A	B	C	D
data	0	0	0	0
Air conditioner	2	5882	1	2
Dish-washer	1	5882	1	260640
ecocute	2	5883	1	3
ih	2	5882	1	3
main	28	5882	1	125
microwave	1	10	1	1
photovoltaic	28	5882	31	125
refrigerator	10	10	1	2

Rice-cooker	1	9	1	34
TV	3	2	99079	8
washer	16	1	1	13
Total	94	35235	99119	260956

Photovoltaic then refers to the energy consumed by the solar panels, which is negative as it is a source of generation rather than consumption. Note that in some cases the photovoltaic value is positive, indicating energy consumption. This is since the controller attached to the solar panel cells can sometimes consume more energy than has been generated. Figures 4 through 7 show the averaged consumption and generation for each household per day over the six-month period. Through this data, by observing the peaks and general trend of the curves, it is immediately clear that the energy profiles of each household are significantly different. As the purpose of this task is to verify that a model for peak shifting is possible in this domain, rather than generating a perfect model for a specific set of data, it has been decided that due to this observation and the large number of missing data for households B, C, and D, that only household A will be used. Another reason for this decision is that, given more data it would be possible to cluster households into a few different energy profiles. Then, once it has been verified that reinforcement learning is a suitable solution method for peak shifting in this domain, it could then be applied separately to each cluster, creating a separate model for each of them. With the minimal data available currently, we will treat household A as the only available dataset of a single cluster, but all methods and code hereafter will be created with the cluster method in mind.

Figure 4 Daily averaged generation and consumption data for household A over the six-month period of available data.

Figure 5 Daily averaged generation and consumption data for household B over the six-month period of available data.

Figure 6 Daily averaged generation and consumption data for household Cover the six-month period of available data.

Figure 7 Daily averaged generation and consumption data for household D over the six-month period of available data

Figure 8 shows the disaggregated NILM data for the chosen household A over the first three days of January. Here it is possible to understand better the relative scales of each appliance's energy consumption and the generation owed to the solar panels, given by photovoltaic. Again, it is immediately clear from this graph that the generation of energy from the photovoltaic cells occurs during the middle of the day, seen by the larger negative values around noon for each of the three days presented.

Figure 8 NILM data for household A over a three-day period at the start of Jan.

3. Proposed Algorithm

Following the recent work in [17] on applying reinforcement learning to the problem of energy arbitrage, its application to the more difficult multi-objective optimization task at hand will be investigated. As mentioned previously, the aim will be to verify that reinforcement learning is able to learn a general policy which achieves perfect peak shifting whilst reducing energy consumption and lowering utility bills without the need to impose various constraints on the problem. More concretely, the method should work irrespective of what price and demand plans are chosen, or what the battery capacity is, or even where the end-user is located. Although the optimal model will vary depending on the values chosen for these variables, the solution method should remain general and unchanged.

Markov Decision Processes, or MDPs, are a mathematical formalization commonly applied to the reinforcement learning problem in literature. Its use is based on its applicability to sequential decision making, in spaces where actions influence not only the present, but also future states, a common theme amongst reinforcement learning problems as previously discussed. Such a formulation allows for the problem of credit assignment and delayed rewards to be considered [2]. Figure 9 pictorially describes the interaction between the agent and environment when the model is framed by such an MDP. The agent is the learner, or decision maker, and the environment is what it interacts with. These two entities continuously interact with each other, with the agent selecting an action and the environment responding to the selection via the output of the updated state and feedback on the action selection via the reward. Formally, at each time step t , the agent receives a representation of the environment's internal state, $S_t \in \mathcal{S}$, based on which it then chooses an action $A_t \in \mathcal{A}(s)$. Upon executing this action, the environment responds by an internal change in state, observable by the agent as S_{t+1} and a reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$. This repeated interaction results in a sequence of states actions and rewards which define the MDP [2].

Figure 9 interaction between the agent and environment in an MDP.

The notion of a finite MDP is such that the sets of states, actions, and rewards all have a finite number of elements and so the associated random variables have well-defined probability distributions which, in a true sense, depend only on the previous state-action pair. Mathematically, the environment dynamics, p , of this model can be given as

$$p(s', r|s, a) \doteq \Pr \{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\} \quad (1)$$

$$\forall s', s \in \mathcal{S}, r \in \mathcal{R}, a \in \mathcal{A}(s)$$

Where the probability distribution is well-defined solely on the current and previous time steps only, satisfying the Markov property. This means that the state itself should carry with it all information of the past that may be necessary in positively influencing the future of the agent's interactions with the environment. In general, the full environment dynamics are not available and have to be learned. As Equation 1 is a probability distribution, it follows that:

$$\sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}(s)} p(s', r|s, a) = 1 \quad (2)$$

$$\forall s', s \in \mathcal{S}, a \in \mathcal{A}(s)$$

Returning to the concept of a value function, a metric which defines how good it is to be in a specific state, it is important to also define the notion of what *how good* means. This concept is given by the *return*, a discounted sum of all rewards that are achieved from the current time step onwards, as shown in Equation 3, where $0 \leq \gamma \leq 1$ is called the *discount factor*, a tunable hyper parameter. The purpose of the discount factor is to avoid infinite cumulative rewards in non-episodic, or never-ending, environments, but also introduces the useful notion of immediate rewards being more desirable than rewards in the distant future.

$$G_t \doteq \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (3)$$

For an agent to achieve the maximum return possible, it is the algorithm's job to incrementally modify the agent's policy, or its action selection process, such that its value function under the policy is maximized. Formally, if an agent follows a policy π , this is equivalent to saying it will act $A_t = a$ in state $S_t = s$ at time t with probability $\pi(a|s)$. In all reinforcement problems, there exists at least one optimal policy, π^* such that the corresponding optimal value function V^* performs better than all other non-optimal value functions in all states [2]. By the above definition, the value function of a state s which follows policy π is given by Equation 4, where the subscript π refers to the agent selecting actions based on policy π

$$V_\pi(s) \doteq \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s] \quad \forall s \in \mathcal{S} \quad (4)$$

A similar function, termed the *action-value function* can also be defined, which is an expectation of the return from starting in state s , selecting action a , and subsequently following policy π thereon. This is given by:

$$Q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a] \quad (5)$$

The optimal value function can then be defined as:

$$V^*(s) \doteq \max_{\pi} V_\pi(s) \quad \forall s \in \mathcal{S} \quad (6)$$

Similarly, for the optimal action-value function:

$$Q^*(s, a) = \max_{\pi} Q_\pi(s, a) \quad \forall s \in \mathcal{S} \text{ and } a \in \mathcal{A}(s) \quad (7)$$

A reinforcement learning algorithm should be able to efficiently learn the optimal policy, be it through the value function, action-value function or even a parameterized representation of the policy directly. However, in general, such an optimal policy can rarely be found due to the extreme computational cost required for problems of interest. For example, tabular methods work by maintaining a table of values over the states, or state-action pairs, and this is simply intractable for larger problems due to the curse of dimensionality [18]. A solution to this is to use a function to approximate these values, where the function consists of less parameter that needs to be learned than the original problem itself. As these functions are only approximations to the full problem, the learned policy will only approximate the optimal policy. *Dynamic programming* refers to a set of reinforcement learning methods which are able to compute the optimal policy given a perfect model, or full $p\psi$ dynamics matrix, of the problem. These methods

are clearly not desirable since the dynamics of most problems are not generally known. Another problem with these methods are their high computational complexity, owing to the fact that they are tabular methods. *Monte Carlo* methods offer an improvement upon dynamic programming in that they no longer assume any knowledge of the environment dynamics. Instead, these are estimated by sampling experience with the environment directly. However, these methods only work on problem domains which are episodic, i.e. have some notion of a terminal state. Once the terminal state is reached, the estimates for the value functions are updated based on the trajectory of *visited states* and obtained rewards from the first to final time step. Although it is an improvement upon dynamic programming in that knowledge of the environment's dynamics is no longer necessary, it is disadvantageous in that one must wait until the very end of an episode until a single update is made. This makes learning incredibly slow to learn and hence these methods are *sample inefficient*.

Temporal difference (TD) methods are like Monte Carlo in that they learn from direct experience with the environment, but differ in that they perform updates at intermediate steps, not just at the end of episodes, through a process known as bootstrapping. As updates do not wait until the end of an episode, these methods can also be used in continuous domains, where there is no terminal state. The more frequent updates also improve the speed of learning, but at the cost of introducing a significant amount of bias to the problem [2].

SARSA is an *on-policy* TD control algorithm which learns the action-value function through direct experience with the environment, and its update rule is given by Equation 8. The behavior policy chooses the action a_t in the current state based on a ϵ -greedy policy, and the target policy chooses an action a_{t+1} following the same ϵ -greedy policy. Such algorithms with equivalent target and behavior policies are known as on-policy methods. ϵ -greedy refers to a policy which chooses a random action with probability ϵ , and acts greedily otherwise, where acting greedily refers to selecting the action with the highest value in the current state. As long as all state-action pairs continue to be visited, SARSA converges to the optimal policy. If $\epsilon \rightarrow 0$ as $t \rightarrow \infty$, the behavior policy will converge to the optimal greedy policy [2].

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (8)$$

Q-learning is a very popular *off-policy* variant of SARSA. In this case the target policy, shown in the update rule in Equation 9, selects the maximum action-value in the current state. As such, the

action-value function Q directly approximates the optimal action-value function independent of the policy that it follows (updates push the policy towards optimal through the use of the maximization term). This algorithm is also known to converge if all state-action pairs are continued to be visited. As the target policy selects the action which maximizes Q , rather than selection via ϵ -greedy as in the case of SARSA, this algorithm converges to the optimal policy without having to worry about the exploration-exploitation tradeoff.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (9)$$

Although Q-learning has proven to be a very popular and promising algorithm, in order for it to be applied to complex problems with high-dimensional state spaces, it is essential to be able to use powerful non-linear function approximations such as deep neural networks. Unfortunately, due to a phenomenon known as the *deadly triad*, such non-linear function approximations cannot be used directly in the Q-learning method without causing the learning to diverge [2]. This phenomenon states that learning will be unstable when combining all of non-linear function approximation, bootstrapping, and off-policy training. Standard Q-learning uses the latter of these two techniques; bootstrapping via the TD update rule, and off-policy training via the maximization term in the target policy. Hence, introducing non-linear function approximations to model the Q^ψ function in Q-learning *activates* the deadly triad phenomenon.

The first method to successfully address this problem is known as *DQN*, or *Deep Q-Network*. The instability in the learning process is due to a variety of issues present in the problem set-up. The first of these is due to correlations in the sequence of observations, as each time step is fed into the network in order of observation, thereby making the data dependent on previous time steps. This is a problem as most reinforcement learning methods, including Q-learning, assume the data to be *iid* (independent and identically distributed). In experience replay, the agent stores experiences $\epsilon_t = (s_t, a_t, r_t, s_{t+1})$ at each time step in an overall replay memory $\mathcal{D}$. In the learning stage of the algorithm, updates to the Q-function are applied to mini-batches of experiences pooled at random from this memory, removing the correlations in the observations. The second problem refers to the fact that the distribution of data in reinforcement learning changes during the agent's learning process and its constantly updating policy. This is problematic as algorithms including Q-learning assume a fixed, non-stationary distribution. The

experience replay method also helps tackle this problem by smoothing the distribution of data and preventing erratic changes on each policy update.

The full DQN algorithm

```

Initialize replay memory  $\mathcal{D}$  to capacity  $N$ 
Initialize action-value function  $Q$  with random weights
for episode = 1,  $\psi$ M do
    Receive  $s_1$  from the environment
    for  $t = 1, \psi$ T do
        Select random action  $a_t$  with probability  $\epsilon$ 
         $\psi\psi\psi\psi\psi\psi\psi\psi\psi\psi\psi\psi$  otherwise select  $a_t = \max_a Q^*(s_t, a; \theta)$ 
        ----- Execute action  $a_t$  and observe  $r_t$  and  $s_{t+1}$ 
        Store transition  $(s_t, a_t, r_t, s_{t+1})$  in  $\mathcal{D}$ 
        Set  $s_t = s_{t+1}$ 
        ----- Sample random mini batch of transitions  $(s_j, a_j, r_j, s_{j+1})$  from  $\mathcal{D}$ 
        Set:
        
$$y_j = \begin{cases} r_j & \text{for terminal } s_{j+1} \\ r_j + \gamma \max_{a_{j+1}} Q(s_{j+1}, a_{j+1}; \theta) & \text{or non-terminal } s_{j+1} \end{cases}$$

        ----- Perform gradient descent on  $(y_j - Q(s_j, a_j; \theta))^2$ 

```

The gradient descent step is what adjusts the parameters θ of the neural network, or other function approximation, to improve the approximation of the Q-value function. In the original paper, the states are first pre-processed before being used in the above computations, but this has been omitted to maintain clarity and improve understanding of how the algorithm works without worrying about implementation details. The DQN algorithm was tested on a variety of Atari platform games and produced results achieving superhuman performance in many games, showing for the first time that deep learning can be applied successfully to the reinforcement learning domain. Although there have been many advancements to DQN in recent years, such as the advent of Double-Q learning which addresses problems pertaining to overestimation bias in the action values [20], DQN is still very frequently used as a first-attempt at solving a reinforcement learning-framed problem due to its ease-of-implementation and simplicity. Its simplicity is due to having only a very small number of tunable parameters, the discount factor γ and the learning rate α . For this reason, DQN will be a one of the methods attempted to achieve optimal peak-shifting. All the methods that have been discussed so far are known as action-value methods, which learn an action-value function $Q(s, a)$ -over state-action pairs and then use this for control by employing a greedy policy, for example. Another class of methods, known as *policy gradient* methods, instead learns a parameterization of the policy directly, avoiding the

need to perform maximization over actions to find the optimal policy. This provides a major advantage over action-value methods in that it is now possible to extend the application of reinforcement learning to problems with a very large and continuous action space. In the peak demand problem, the amount of energy that is bought at each time step is a continuous value and so this provides a very useful improvement over action-value methods which can be taken advantage of, removing the need to discretize the action space which would result in a significant loss of control. Although one might suggest using a fine-grained discretization to maintain sufficient control, the number of actions increases exponentially with the degrees of freedom in the system, and so even a relatively coarse discretization of the action space would result in an unwieldy number of actions [21].

An overview of policy gradient methods will be presented, with derivations leading into the development of *PPO*, or *Proximal Policy Optimization*, the current de-facto algorithm used in reinforcement learning problems. These methods parameterize the policy as $\pi(a|s, \theta) = \Pr \{A_t = a | S_t = s, \theta_t = \theta\}$, where $\theta \in \mathbb{R}^d$ with d referring to the number of degrees of freedom in the parameterization of the policy. The goal of these methods is then to learn the values of θ to shape the policy such that performance is maximized (i.e. finding the optimal policy). This is done via a measure of performance $J(\theta)$ and updating the parameters via gradient ascent as in Equation 10, where $\nabla_{\theta} \hat{J}(\theta)$ refers to a stochastic estimate with an expectation that approximates $\nabla_{\theta} J(\theta)$.

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta} \hat{J}(\theta) \quad (10)$$

Another advantage of using policy gradient methods is that the action selection changes smoothly with updates to the parameters, whereas in DQN with an ϵ -greedy policy, the action probabilities can suffer from erratic changes even after a small update to the action-value function, if such an update results in a different action having a maximal value. This property of policy gradient methods results in stronger convergence guarantees than available in DQN [2].

In the episodic case, the performance measure is usually defined as the value function from the initial starting state; $J(\theta) \doteq v_{\pi_{\theta}}(s_0)$, where $v_{\pi_{\theta}}(s_0)$ is the true value function for the policy π_{θ} . As $v_{\pi_{\theta}}(s_0)$ is generally not known, it is not possible to find its gradient. The policy gradient theorem handles this by rearranging it into a function whose gradient can be found. The result is shown below:

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s, a) \quad (11)$$

Thereby providing a computable term for the gradient of the performance metric, $J(\boldsymbol{\theta})$, which does not explicitly include the state distribution. The proportionality sign in Equation 11 is sufficient as the constant of proportionality will be absorbed by the α term, a tunable hyper parameter in the update. This can then be re-written as:

$$\begin{aligned} \nabla J(\boldsymbol{\theta}) &\propto \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s, a) \\ &= \mathbb{E}_\pi[\sum_a q_\pi(S_t, a) \nabla \pi(a|S_t, \boldsymbol{\theta})] \end{aligned} \quad (12)$$

By noticing that the summation is over the states and weighted by how often those states are visited under the policy π (this is what $\mu(s)$ represents). Therefore, if the policy π is followed, the expectation above holds. This could then be used directly in the stochastic gradient ascent update, as shown in Equation 13, where $\hat{q}$ refers to a learned approximation to q_π with $\boldsymbol{\omega}$ as the parameter vector.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \sum_a \hat{q}(S_t, a, \boldsymbol{\omega}) \nabla \pi(a|S_t, \boldsymbol{\theta}) \quad (13)$$

However, this update involves all the actions even though only a single action was taken, again limiting the domain to a discrete action space. Instead of directly using this, the *REINFORCE* algorithm modifies the update rule to consider only A_t , the action taken at time step t . This is done via the *gradient-trick* method, as used in the policy gradient theorem derivation, giving the result below.

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_\pi[G_t \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})] \quad (14)$$

This motivates the standard REINFORCE update rule, given by:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha G_t \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta}) \quad (15)$$

which now only depends on the actual action taken, A_t . This vector in the update rule points in the direction in the parameter space which will increase the probability of selecting the action A_t again, scaled by G_t , thereby forcing the policy to favor this action more if it yields a high return. A major change that can be made to this algorithm is the introduction of a baseline, $b(s)$, which does not affect the gradient as long as there is no dependence on the action [2]. A common baseline used here is $V(S_t)$. This would give the update:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha (G_t - V(S_t)) \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta}) \quad (16)$$

Noting that $G_t = Q(S_t, A_t)$, the parenthesized term is equivalent to what is known as the advantage function $\hat{A}_t$, giving:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \hat{A}_t \nabla \ln \pi(A_t | S_t, \boldsymbol{\theta}) \quad (17)$$

This is the result for the update in the REINFORCE algorithm. However, although REINFORCE provides many benefits over the previously discussed DQN algorithm, the updates take place at the end of the episode, making it a Monte-Carlo method and therefore suffers from high variance and slow learning [2]. Defining the objective function for REINFORCE as $J^{REINFORCE}(\boldsymbol{\theta})$ that is

$$J^{REINFORCE}(\boldsymbol{\theta}) = \mathbb{E}_{\pi}[\hat{A}_t \ln \pi(A_t | S_t, \boldsymbol{\theta})] \quad (18)$$

Since the discovery of this algorithm, many other performance measurements have been proposed, including that given by Equation 19 [22].

$$J^{TRPO}(\boldsymbol{\theta}) = \mathbb{E}_{\pi} \left[\hat{A}_t \frac{\pi_{\boldsymbol{\theta}}(a_t | s_t)}{\pi_{\boldsymbol{\theta}_{old}}(a_t | s_t)} \right] = \mathbb{E}_{\pi}[\hat{A}_t r_t(\boldsymbol{\theta})] \quad (19)$$

Where $\pi_{\boldsymbol{\theta}}$ refers to the updated policy and $\pi_{\boldsymbol{\theta}_{old}}$ the policy before the update. The term *TRPO* is used for this objective, as it is the metric used in the recently developed *Trust Region Policy Method* algorithm [23]. However, this algorithm also imposes a constraint on the problem; ensure that the KL-divergence between the new and old policies is lower than a small value δ in order to ensure that the policies do not differ by *too much*. This constraint restrains the updates and offers a theoretical guarantee of policy improvements on each update for a sufficient step size. However, this method is rarely used in practice as the computation of the KL divergence via the conjugate gradient method requires the expensive calculation of the Fisher information matrix [23]. To combat this, the more recent collection of algorithms, known as *PPO*, approaches the problem of constraining the updates using what is known as a *clipped surrogate objective*, acting as a first order approximation to $J^{TRPO}(\boldsymbol{\theta})$. The new objective is given by:

$$J^{PPO}(\boldsymbol{\theta}) = \mathbb{E}_{\pi}[\min(\hat{A}_t r_t(\boldsymbol{\theta}), \text{clip}(r_t(\boldsymbol{\theta}), 1 - \epsilon, 1 + \epsilon))] \quad (20)$$

Where, $\text{clip}(r_t(\boldsymbol{\theta}), 1 - \epsilon, 1 + \epsilon)$ refers to constraining the value of $r_t(\boldsymbol{\theta})$ to the interval $[r_t(\boldsymbol{\theta}) \wedge (1 - \epsilon), r_t(\boldsymbol{\theta}) \vee (1 + \epsilon)]$ as an alternative method of restricting the amount by which the policy updates at each step. The paper suggests using a value of $\epsilon \approx 0.2$. PPO is much easier to

implement as it no longer requires calculation of the KL divergence, and hence neither the Fisher information matrix. It also offers another improvement upon REINFORCE; it allows multiple updates to be made per epoch, which is not possible with REINFORCE as the large updates would cause it to be unstable [24]. This tackles the original problem of REINFORCE being too slow to learn, and many empirical results have shown PPO to learn faster than simple methods such as DQN based on this and the fact that its updates can be parallelized over multiple CPUs. PPO is the current de-facto standard policy-gradient algorithm and therefore will also be used to tackle the peak-shifting task, in addition to DQN.

4. Relation to the peak shifting domain

In the peak shifting task, there is again much flexibility in how we define these terms, where the chosen definitions will greatly affect how the model learns and also what the final policy will describe. It should be noted that the choice of definitions given below is not unique, and many other choices could have been made. The environment will refer to a system encompassing the battery itself, and its interaction with the outside world. That is, the weather, the electricity grid, the grid's system operator, the households, battery, solar panels, etc, are all encompassed by the notion of the environment. The agent will then refer to a non-physical CPU, or the *brain* of the battery which makes decisions on how much energy to buy at any single time from the grid. Following from this, the action space will be a single continuous value $a_t \geq 0$ describing how much energy will be bought from the grid at time step t . At all points in this discussion, any reference to energy will assume units of kWh. Upon choosing this action, the environment responds by internally calculating how much energy is required from the battery to provide the necessary energy for consumption, and therefore automatically performs the necessary charging and discharging of the battery without the agent needing to know how this computation works. Another obvious choice of action space here would have been to allow the agent to choose how much energy to charge or discharge from the battery. If this action space were to be used, the internal computation by the environment would instead need to calculate how much additional energy is required to provide enough energy for the consumption, and this would then be purchased from the grid. Therefore, both methods achieve the same result but via the agent to help direct its future decisions. Firstly, the agent needs to know how much energy is currently stored in the battery to make a well-informed decision on how much more energy may

be required, and so the current battery charge should be included in the state definition. In addition, it is important to know how much energy is being generated and consumed by the household at the next time step. Although this may seem like cheating, that the agent knows explicitly how much energy the household will require, it should be noted that in production these values can be estimated through simple forecasting methods. As a single time step is 1 minute, this is equivalent to using all past observational data to predict the values just 1 minute ahead. Based on the disaggregated data provided by the NILM technology, this forecasting should be both simple and of very high accuracy. As such, the problem of forecasting of this *1 minute ahead* data is ignored and the learning of how to optimally buy energy to approach the peak demand problem with reinforcement learning is given full attention. However, the agent still needs to learn how to effectively buy energy taking into account the seasonal nature of the data. Even though the agent has full knowledge of the next time step, it should be able to leverage information pertaining to the weather, which, if it can learn to predict, can help in its forecasting of energy generation due to photovoltaic solar cells. For this reason, all weather data (sunlight, rainfall, etc) will also be included in the state. Finally, the current price and demand (available energy from the grid) will also be added to the state, as this will allow the agent to include these data in its decision-making process. Finally, the total consumption of energy is then also included in the state representation. Note that this single value was chosen over the disaggregated data, but either choice would be suitable. In the effort to keep the method as general as possible, the simulator, which will be discussed in detail later, will allow for user defined state-space representations, such as removing some of these features such as weather data, or even adding in more features that could be useful for learning. Finally, the reward function needs to be defined. At this point it is important to reiterate exactly what the aim of the task is; achieving peak shifting whilst reducing energy consumption and lowering utility bills, with peak shifting being of primary importance. The process of defining a suitable reward signal for the task is known as *reward signal shaping*. It should be noted again, that there is no specific *correct* reward signal, and that the choice is up to the designer. For this task, individual rewards related to the various optimization tasks are first defined. In the following, the term *penalty* is used for negative rewards and is solely a term of convenience. In maximizing the reward, the agent is thereby minimizing these penalties. Before looking into the overall goals of peak shifting and cost & energy reduction, it is first important to look at the constraints of the system. In a

standard, no reinforcement learning approach to optimization, we would mathematically formulate an objective function to maximize subject to constraints. In our problem, these constraints would be along the lines of; Purchase enough energy for consumption, do not buy more energy than available (grid demand), and do not store more energy in the battery than permitted by its maximum capacity. In the reinforcement learning setting, we cannot explicitly enforce these constraints, but can instead shape the reward signal in such a way that we guide the agent towards a policy which satisfies each of these. It should be noted that it is not guaranteed such a policy exists, but even if one does not, given a suitably defined reward signal, the optimal policy will be such that the constraints are violated as infrequently as possible. Before defining the various individual reward signals, a few useful terms will first be given that are internally calculated by the environment. *current_energy* refers to how much energy remains in the current time step after considering the charge of the battery, how much energy has been consumed and generated, but before purchasing energy from the grid. Hence, if this value is negative, the amount by which it is below 0-refers to the minimum amount of energy that the agent needs to purchase to provide enough energy for the household to continue using its appliances, with any excess being used to charge the battery. The term *to_buy* will be used for the agent's choice of how much energy it will purchase at the current time step.

$$current_energy = current_charge + generated_energy - consumed_energy \quad (21)$$

$$min_required_energy$$

$$= \max(0; -current_energy) \text{-----} (22)$$

To ensure that constraint 1 is satisfied, a penalty termed *under_purchase* is designed and given by Equation 23, where the minus sign signifies that this is a penalty and should be avoided by the agent. If the agent were given this reward signal alone, it should learn to maximize it, or achieve the value of 0, corresponding to never purchasing too little energy

$$r_{underpurchase} = -\max(0; min_required_energy - to_buy)$$

(23)

Constraint 2 is then tackled using the penalty given by *abovelimit* in Equation 24, where demand refers to the current limit of available energy by the grid. If this were the only reward signal, the agent would learn to always buy less or equal to the available energy by maximizing this term to result in 0-penalties.

$$r_{abovelimit} = -\max(0; to_buy - demand)$$

(24)

The penalty for constraint 3, termed *abovecapacity* is given by Equation 25, where *max_capacity* refers to the maximum capacity of the installed battery. Similarly, to before, using this reward signal by itself would guide the agent into never purchasing an amount of energy that would force the battery to store more charge than is physically possible.

$$r_{abovecapacity} = -\max(0; current_energy + to_buy - max_capacity)$$

(25)

The above penalties individually tackle the constraints on the system but do no effort in explicitly working towards the intended goal of peak shifting and cost and energy consumption reduction. In order to guide the agent into buying energy at cheap prices, a penalty termed *cost* has been designed and is given in Equation 26, where *price* refers to the current price of energy, with units/kWh. This penalty should also cover the goal of reducing energy consumption as reducing the amount of energy purchased will also lower this penalty. As such, this single penalty goes some way in addressing two of the objectives posed in the peak shifting problem.

$$r_{cost} = -price \times to_buy$$

(26)

The main problem of peak-shifting is not as easy to describe in terms of penalties and rewards. The previously designed penalty *abovelimit* goes some way in addressing this by making sure the agent does not purchase more energy than available, but it does not explicitly tell the agent to smooth its load profile, that is buy similar amounts of energy at each time step. To address this explicitly, the reward termed *belowlimit* has been designed and is given in Equation 27. This positive reward has a maximum value of 1-when the agent chooses to not buy any energy, and this decreases linearly towards 0-as the amount of energy purchased approaches the demand limit. This does not explicitly tell the agent to favor purchasing similar amounts of energy at each time step but tries to keep the chosen amount as close to 0 at each time step. As discussed previously, there are an infinite number of rewards that could be designed to tackle the same or similar problems. The purpose of this task is not to find the best reward function to address the problem, but to verify that reinforcement learning is a suitable solution method to this task, and as such, it is not necessary to investigate defining overly intricate reward functions.

$$r_{\text{belowlimit}} = \max\left(0, 1 - \frac{\text{to_buy}}{\text{demand}}\right) \quad (27)$$

It should be noted that these rewards may not be mutually exclusive. For example, *abovelimit* and *belowlimit* both address very similar problems to do with influencing the amount of energy purchased with respect to how much is available. This observation means that there may be some interaction between the different terms upon combining the individual rewards and penalties. These five individual signals are then linearly combined, as shown in Equation 28, to provide a single overall reward signal to guide the agent's learning process. The values of the α_{ip} coefficients will be discussed at a later stage. It will be seen later in the results & evaluation section that the appropriate selection of these values is paramount to the success of the agent learning an optimal policy, and is therefore one of the main contributions presented in this work.

$$r = \alpha_1 r_{\text{underpurchase}} + \alpha_2 r_{\text{abovelimit}} + \alpha_3 r_{\text{abovcapacity}} + \alpha_4 r_{\text{cost}} + \alpha_5 r_{\text{belowlimit}} \quad (28)$$

5. Results

All models have been trained using 70% of the data for household A, and the results that follow have been generated using the remaining 30% of the data for the same household, corresponding to approximately 2 months. This train/test split is used to ensure that the models do not over fit to the data used for training but instead generalizes to unseen data and therefore can be used in production where future data is of course not readily available. All of the graphs and results in this section have been generated automatically using the simulator, at the same time that the model has been trained. In order to be able to verify whether or not the reinforcement learning agents have learned a useful policy, it is necessary to first construct a simple baseline with which results can be compared against. This baseline should follow as good a policy as is possible without using reinforcement learning to ensure that the comparisons are meaningful. The baseline has been chosen to follow a policy which requests exactly the amount of energy it needs at every time step, and therefore does not require use of a battery. To this end, the baseline has an

instant advantage over all reinforcement learning ends in that it does not need to learn to infer how much energy is needed at any time. The calculation is done internally, taking into account energy consumption and generation, given by the Equation 29, taking into account that there is no battery and hence charge is removed from the equation:

$$\text{required_energy} = \text{consumed_energy} - \text{generated_energy} \quad (29)$$

As the baseline simply purchases what it needs at the time there will be times when it tries to purchase more energy than is currently available. This is the heart of the problem and is exactly why the reinforcement learning agents will need to learn how to peak-shift. Although there is no training necessary for the baseline model, in order to allow comparison with the results from the reinforcement learning agents, the following graphs and results are generated using just the 30% testing data. Figure 10 gives the requested energy profile for the baseline agent. This, along with all other graphs that follow, presents the averaged data over a 24 hour period. The shadows behind each curve show a single standard deviation either side of the mean in order to visualize the variance in the data. As the actions that the agent takes are the decision of how much energy to buy, this graph is a direct visualization of the agent's policy.

Figure 10 Daily-averaged requested energy profile for the baseline model on household A.

In this graph, the peak is not as apparent as that seen in the original household data, which included all the training data. Hence, this baseline agent does not attempt to purchase more energy than available too frequently, but the purpose of the task is also to achieve peak shifting in general, and to flatten the requested energy profile as much as possible, whilst taking into account the other objectives of lowering utility bills and

reducing overall energy consumption. It should be noted again that the baseline does not use any control; it purchases exactly what the amount of energy it needs at every time step, and hence does not take into account the price or demand (limit) of energy at all. Figure 11 shows the same requested energy profile but now graphed against the price of energy and normalized. This allows the designer to verify if the agent has attempted to buy more energy when the prices are lower. Of course, in the baseline model, there is no relationship between the two curves as the agent does not take the pricing data into account. However, by coincidence, the cheaper band of energy lines up with the peak in the requested energy profile and for this reason the baseline agent luckily has purchased a lot of its energy at the cheapest possible price.

Figure 11 Daily-averaged normalized pricing data for the baseline model on household A.

It should be noted that although the baseline may seem simple at first, the fact that it does not need to learn how much energy is required at any time, and also that it purchases a lot of its energy at the cheapest price possible makes it a very strong baseline to outperform. Table 3 gives the results for the baseline agent on the test data. The *abovcapacity* error refers to the number of times the agent attempted to charge the battery above its maximum capacity. As the baseline agent does not use a battery, this error is never encountered. *underpurchase* refers to the agent buying less energy than is required at that time step. As the definition of the baseline policy is such that it will always buy exactly as much as it needs, again this is never encountered. Finally, the *abovelimit* error refers to the number of

times that the agent attempted to purchase more energy than was available by the grid. As a single time step corresponds to 1 minute, the baseline agent would have left this household without the required electrical energy for 2 hours, corresponding to 0.15% of the test data period. For other households this number is much higher, and so the reinforcement learning agent needs to outperform an agent which itself is already quite strong in the case of household A.

Table 3 Numerical results for the baseline agent.

Parameter	Results
Error <i>abovcapacity</i>	0 (0.00%)
Error <i>underpurchase</i>	0 (0.00%)
Error <i>abovelimit</i>	122 (0.15%)
Monthly bill	3517.79 Unit price
Total energy purchased	292.51 kW

The first reinforcement learning algorithm that was used was DQN based on its simplicity and ease of training. This allowed for many models to be trained and is how the initial choice of individual reward signals was chosen. This was based on first trying one reward signal, and then subsequently adding the rest whilst observing the resultant model's policy. At this stage it became very apparent that due to the large difference in magnitudes of the individual reward signals, without the automatic penalty shaping method none of the models were able to learn a useful policy. To this end, as discussed in the Simulator section, once a model was trained the coefficients in the linear combination of the individual signals were automatically scaled in order to improve the next model's policy. The discretization of the action-space was varied in the initial models, and $[0, \psi 0.005, \psi 0.01, \psi 0.015, \psi 0.02, \psi 0.03, \psi 0.04]$ was chosen to be used in all future models. This choice was based on a number of factors as well as from the initial results that proved this to be a promising set of actions. Firstly, keeping the action space constant throughout the models allows comparisons to be made much more easily. Secondly, on observation of the requested energy profile for the baseline agent in Figure 10, it was seen that the maximum required energy at a single time step hovers around the 0.03-0.04-range, and so this was chosen as the highest values in the discretization. Also, as most time steps require much less energy, it was chosen to use more actions between 0.00-and 0.02. All of these values have units

of kWh. Increasing the number of actions made the training much slower and so this upper limit of 7 actions was kept final. In addition to the varying reward signal coefficient values, the episode length and neural network architecture were also varied on each model. For each model, an episode length was chosen at random from 1440 and 10080, corresponding to 1 day and 1 week, respectively. The model architecture was also chosen at random between $[64]$, $[64, \psi 64]$ and $[64, \psi 128, \psi 64]$, where each value represents the number of nodes in each layer. It should be noted here that the variation in these two parameters will most likely also affect the optimal values for the reward signal coefficients, and this is one downside to the simple scaling method used. In future work a more intricate penalty shaping method could be used which takes into account all past results and the model architectures. The best model was selected after training approximately 300 models using the created simulator. As previously discussed, there is no explicit definition of what *best* here means, and so the author's judgment has been used, but ensuring that peak shifting maintains the number one priority with reduction in energy consumption and lowering of utility bills also being important. Table 4 gives the results for this model on the test data, and is compared side-by-side with the baseline results from the previous section. Now that there is a battery, it is possible to encounter the *abovcapacity* error. The DQN model attempts to overcharge the battery approximately 13% of the time. This is a significantly high amount, but it should also be noted that most models encountered this error at least 25% of the time. Although high, in production it is possible to explicitly constrain the amount of energy bought so as not to overcharge the battery, and so the error is not of too much concern. However, the fact that the model has relied on overcharging the battery at some time steps implies that the policy it has learned is not optimal. One way of tackling this in the future would be to explicitly give the battery's maximum capacity as an additional feature in the environment's state representation. The DQN agent has also encountered the *underpurchase* error at 512 time steps, corresponding to not buying enough energy for the required consumption. However, as this is only 0.65% of the time, this implies the agent has almost perfectly learned how to infer the required energy, making a mistake very infrequently. The times at which it *underpurchased* could be attributed to the coarse discretization of the action space. For example, consider the case where the policy in a specific state which requires 0.025-kWh of energy assigns 49% chance of purchasing 0.03-kWh and a 51% chance of

purchasing 0.02-kWh of energy. In this case the agent would *underpurchase*, but if there were a less coarse discretization, with some actions in between 0.02 and 0.03, then the agent would be likely to select a more appropriate action. This alone is a motivation for using a continuous-action method such as PPO. However, the *abovelimit* error has been reduced by 76% compared to the baseline, which is evidence that the agent has learned to peak shift considerably. It is still purchasing more than the limit at some time steps, but there has been considerable improvement in this domain. Finally, the monthly electricity bill has had a small reduction by 6% and energy consumption has also reduced by 12%. These results prove that the DQN agent has outperformed the baseline and that it has successfully achieved all three of the goals in the multi-objective optimization problem.

Table 4 Results for the best DQN model compared against the baseline.

Parameter	Baseline Agent	DQN Agent	% Difference
Error <i>abovcapacity</i>	-	10374 (13.10%)	-
Error <i>underpurchase</i>	0 (0.00%)	512 (0.65%)	$+\infty\%$
Error <i>abovelimit</i>	122 (0.15%)	29 (0.04%)	-76%
Monthly bill	3517.79 UC	3296.43 UC	-6%
Total energy purchased	292.51 kWh	257.94 kWh	-12%

Figure 12 visualizes the policy with control (DQN agent) and compares it to that of the baseline, providing verification that peak-shifting has successfully occurred. There is quite a bit of fluctuation in the DQN model's requested energy profile, but this is due to the discrete nature of the action space. As the PPO agent is able to work in a continuous action space, its resultant profile should appear to be a lot smoother. There is a slight peak in the purchase of energy between 2am and 6am. Observing Figure 13, which shows the daily-averaged normalized values of the requested energy profile against the price of energy, it can be seen that this peak corresponds to a time period where energy is cheaper. This is therefore evidence that the agent has learned to buy energy when it is cheaper in an effort to reduce the electricity bill.

Figure 12 Daily-averaged requested energy profile for the best DQN model on household A.

Figure 13 Daily-averaged normalized pricing data for the DQN model on household A.

Figure 14 shows the daily-averaged charging profile of the battery over the test data when using this model, an indirect method of visualizing the policy as the battery charge is related to the amount of energy that is purchased. As the battery that has been tested has a maximum capacity of 14.4-kWh, it can be seen that the relatively high charge state of the battery between noon and 8pm most likely corresponds to the numerous *abovecapacity* errors encountered. The peak between 2am and 6am corresponds to the agent purchasing more energy at this time due to the lower price, with the excess being stored in the battery. The sharp loss of charge just after this time then corresponds to the necessary use of these stores to tackle the large consumption period

as given by the initial peak before implementing the battery. The relatively high amount of energy being stored thereafter corresponds to the agent purchasing energy even though consumption is relatively low, and so the majority of this purchased energy is being stored in the battery. It could also be due to it requiring more energy the next day quite early on and so the agent has realized this, and is purchasing the energy well in advance. This is corroborated by the fact that it then begins to use this energy continuously from about 4pm until 2am the next day when it again decides to top up the energy in advance of the high consumption period.

Figure 14 Daily-averaged normalized charging profile for the DQN model on household A.

Figure 15 shows the total reward achieved per episode. The fluctuation can be attributed to the fact that each episode corresponds to a single day of data, and the day-to-day data varies significantly as seen by the large error shadow in the household data as previously seen in Figure 4. However, the model can be seen to learn very quickly by the initial upwards trend, and learning slows by around episode 75. All 1-day episode models were trained for 500 episodes to allow for sufficient exploration of the domain before stopping. At the end of the training process, the model from around episode 75 would have been used to avoid any over fitting that may have occurred between that and the end episode.

Figure 15 Total rewards per episode during the training period for the best DQN model. It should be noted that the ability of the agent to successfully achieve the goals of the task was highly reliant upon it learning from an optimized reward signal, which was shaped using the automatic penalty shaping feature discussed previously. Figure 16-a shows the individual reward signals per episode during training for the first DQN model that was trained. This model does not use the automatic penalty shaping feature as there is no previous model from which to learn. It can be seen that the *belowlimit* reward, the reward given for being below the available energy limit, is much higher than the rest. Such a reward signal would bias the agent into placing more emphasis on learning to maximize this reward whilst ignoring the rest. Figure 16-b then shows the same graph, but for the best DQN model that has been evaluated in this section. It can be seen that, through the automatic penalty shaping feature, the magnitudes of each individual signal are much closer to each other. The only anomaly now is the erratic fluctuation in the *abovcapacity* penalty. This fluctuation corresponds to the large *abovcapacity* error achieved by this agent. This implies that if there were a way to smoothen this specific reward signal, then the model agent would be more likely to learn a policy which avoids such a large number of errors related to the battery capacity.

In summary, the DQN results are very promising, having achieved all three intended results of peak-shifting, lower energy bills, and reduced energy consumption. There is, however, still room for improvement in regards to the aforementioned errors. To this end, it is worth mentioning again that although it has achieved considerable peak shifting, it is not perfect and still purchases more energy than is available sometimes, albeit very infrequently.

a. Initial model (not optimal)

b. Final model (best)

Figure 16 Disaggregated reward signals for two different DQN models.

The main functional difference between DQN and PPO is that PPO can handle a continuous action space and so does not suffer the same problems as DQN does by discretization. PPO learns to modify the mean and variance of a Gaussian which is then used in the action selection. As Gaussians have an infinite domain, this is then clipped to $[\gamma, 1/\psi]$ and then scaled appropriately to $[0, \psi \cdot 0.03]$, as suggested in literature.

This modified range is chosen for the same reasons as with the DQN considerations based on the observed required energy from Figure 10. Similarly to when training the DQN models, the parameters of the model were adjusted over the first few initial models until a parameter set which performed well was found.

All models were trained using episodes of 1 day (1440 time steps). The agents did not seem to learn efficiently when using an episode length of one week with the chosen parameters, and it

was decided that finding parameters for one week episode model would be unnecessary additional extra work given the 1 day episode models were training easily and more quickly. More than 75 models using a PPO agent were trained, with each successive model learning from the last via the automatic penalty shaping feature. The results for this model have been compared with the baseline in Table 5. As before, the model tends to try and charge the battery above its maximum capacity from time-to-time, but much less so than the DQN model did, reducing this to 7.48% of the time compared to 13.1% as a major improvement. In addition to this, the model never *underpurchases* or goes above the limit, i.e. it always has at least enough energy than is required for the consumption and never goes above the amount of energy available by the grid. Hence, it has achieved perfect peak-shifting, the main requirement for this task. Finally, it has also managed to reduce the total energy bill and energy consumption by over 20% in both cases, thereby making this model production-ready and a perfect validation that reinforcement learning is an adequate solution to this task.

Table 5 Results for the best PPO model compared against the baseline.

Parameter	Baseline Agent	DQN Agent	% Difference
Error <i>abovcapacity</i>	-	5928 (7.48%)	-
Error <i>underpurchase</i>	0 (0.00%)	0 (0.00%)	+0%
Error <i>abovelimit</i>	122 (0.15%)	0 (0.00%)	-100%
Monthly bill	3517.79 UC	2777.75 UC	-21%
Total energy purchased	292.51 kWh	226.49 kWh	-23%

Figure 17 shows the requested energy profile of this model against the baseline. It follows a similar trend to that of DQN, but with a much smoother profile due to the ability to choose actions from a continuous space. Again, it is immediately obvious from this graph that peak-shifting has been achieved; the requested energy profile for the PPO model maintains a relatively flat profile without any significant peaks. It has also learned to react significantly to changes in the demand (available energy) profile as can be seen just after 8am. At this time, the energy limit drops very slightly, and the model reacts to this by purchasing no energy at all, due to its prediction that the limit will rise again shortly.

Figure 17 Daily-averaged requested energy profile for the best PPO model on household A.

Figure 18 shows this data against the price of energy after being normalized. Similarly to the DQN model, the PPO agent decides to purchase more energy between 2am and 6am when the price of energy is at its lowest. It also seems to predict the end of this low pricing period and decides to purchase a large amount of energy again just before the price goes up, in order to minimize the overall energy bill.

Figure 18 Daily-averaged normalized pricing data for the PPO model on household

A.

Figure 19 shows the charge profile for this model, again showing a similar trend to that of the DQN model and reacting to the households' energy consumption habits almost identically.

Figure 19 Daily-averaged normalized charging profile for the PPO model on household A.

Figure 20 shows the total reward achieved per episode. There is still fluctuation in this learning process due to the variation in consumption data from day-to-day. The model was only trained for 75 episodes based on the fact that it was trained over 8 different CPUs and so this equates to much more clock-training time than in the DQN model, although due to the different learning processes these cannot be directly compared anyway. The model appears to continue learning until the end of the 75 episodes, but due to memory reasons it became a lot more difficult to train any further than this although it may have proven beneficial if possible.

Figure 20 Total reward per episode during the training period for the best PPO model.

Figures 21-a and 21-b, similarly to the DQN case, shows how the individual reward signals have been shaped from the initial model to the final, best model chosen. Again

it is very clear that in the first model, before any smart shaping has taken place, the *belowlimit* reward overpowers the rest and so the model ignores the remaining reward signals. In the final, improved model, the reward signals are much closer to each other, although there is still some fluctuation in the *abovecapacity* reward, albeit much less than in the DQN case. However, this fluctuation begins to die off near to the end of the training period, resulting in a much lower *abovecapacity* error of approximately 7% compared to 13% with DQN.

In summary, PPO gives perfect results minus its reliance on going above the battery's capacity approximately 7% of the time. It has achieved perfect peak shifting, never purchases less energy than is required, and has reduced energy consumption and lowered the utility bill by a significant amount. Although it is obvious that the PPO agent has managed to improve upon the DQN agent on all fronts, it serves useful to compare the policies on both fronts to see exactly what was learned. Figure 22 shows the requested energy profile, the direct policy, of both the best PPO and DQN agents against the baseline.

a. Initial model (not optimal)

b. Final model (best)

Figure 21 Disaggregated reward signals for two different PPO models

Figure 23 does the same but for the charge profile which is an indirect visualization of the policy. From both of these graphs, it is apparent that the learned policies for both agents are incredibly similar. Also, it can be seen that the variance in action selection is much lower for the PPO agent, due to the continuous action space, and this results in much smoother profiles and is also likely the reason why PPO never purchases more energy than is available. The results of PPO fare better than the DQN model, and this is again undoubtedly due to its use of a continuous action space and the fact that it is a more robust reinforcement learning algorithm in general, being the first choice for many problems to date. Although it is impossible to tell what the actual optimal policy in this domain is, from these results and the fact that both agents' policies are very similar, it is very likely that the policy learned by PPO is exceedingly close to the optimal policy, if not the optimal policy itself. It is hypothesized that the variation in results between DQN and PPO would disappear if the discretization of the action space in the DQN model were made infinite, approaching that of the continuous domain. Of course, this is not possible to test due to computational reasons, but the sheer similarity in the outlined policies for both agents in these graphs backup this statement.

Figure 22 Daily-averaged requested energy profile for the best PPO and DQN model on household A.

Figure 23 Daily-averaged requested energy profile for the best PPO and DQN model on household A.

Ethics approval and consent to participate Not applicable

Not applicable

Consent for publication

Not applicable

Availability of supporting data

We can submit all available data

Competing interests

Not applicable

Funding

Applicable

Authors' contributions

The authors read too much relevant articles in order to determine the contribution points, collect all applicable data, and do the programming.

Acknowledgements

Many thanks for the Pyramids Higher institute for Engineering and technology, where it introduces the full support to finalize this work.

References

- [1] Richardson S. Load shifting [Text]; 2011. Available from: <https://www.eex.gov.au/opportunity/load-shifting>.
- [2] Sutton R, Barto A. Reinforcement Learning: An Introduction. 2nd ed. MIT Press; 2017. Available from: https://drive.google.com/file/d/1xeUDVGWGUUv1ccUMAZHJLej2C7aAFWY/view?usp=embed_facebook.
- [3] Spike in deaths blamed on 2003 New York blackout. Reuters. 2012 Jan; Available from: <https://www.reuters.com/article/us-blackout-newyork/spike-in-deaths-blamed-on-2003-new-york-blackout-idUSTRE80Q07G20120127>.
- [4] What is decentralised energy? | Decentralised energy experts - E.ON.; Available from: <https://www.eonenergy.com/for-your-business/large-energy-users/manage-energy/energy-efficiency/decentralized-energy-experts/What-is-decentralized-energy>.
- [5] Why a Distributed Energy Grid is a Better Energy Grid; 2018. Available from: <https://www.swellenergy.com/blog/2016/05/20/why-a-distributed-energy-grid-is-a-better-energy-grid>.
- [6] Green A. Machine learning in energy - part two; 2017. Available from: <http://adagefficiency.com/machine-learning-in-energy-part-two/>.
- [7] Gruen A. 'Peakers' plants provide electricity when it's hot, but at the highest price; 2010. Available from: http://www.nj.com/business/index.ssf/2010/07/peakers_plants_provide_electri.html.
- [8] NZ HWC. What Is Ripple Control? | Hot Water Cylinders LTD New Zealand; 2018. Available from: <https://www.hotwatercylinders.nz/blog/what-is-ripple-control/>.
- [9] Kalsi K. Frequency Responsive Demand. US Department of Energy. 2014 Sep;p. 52.
- [10] Corson R. Implementing energy storage for peak-load shifting; 2014. Available from: <https://www.csemag.com/single-article/implementing-energy-storage-for-peak-load-shifting/95b3d2a5db6725428142c5a605ac6d89.html>.
- [11] Naam AR. Why Energy Storage is About to Get Big –and Cheap; 2015. Available from: <http://rameznaam.com/2015/04/14/energy-storage-about-to-get-big-and-cheap/>.
- [12] Guo Y, Fang Y. Electricity Cost Saving Strategy in Data Centers by Using Energy Storage. IEEE Trans Parallel Distrib Syst. 2013 Jun;24(6):1149–1160. Available from: <http://dx.doi.org/10.1109/TPDS.2012.201>.
- [13] Rhodes J. Energy Storage Is Coming, But Big Price Declines Still Needed; 2018. Available from: <https://www.forbes.com/sites/joshuarhodes/2018/02/18/energy-storage-coming-but-big-price-declines-still-needed/#592db0dc5e1d>.
- [14] Come on! Renewable energy fixed price purchase system; 2018. Available from: http://www.enecho.meti.go.jp/category/saving_and_new/saiene/kaitori/fit_kakaku.html.
- [15] Bao G, Lu C, Yuan Z, Lu Z. Battery energy storage system load shifting control based on real time load forecast and dynamic programming. In: 2012 IEEE International Conference on Automation Science and Engineering (CASE); 2012. p. 815–820.
- [16] Qin J, Chow Y, Yang J, Rajagopal R. Online Modified Greedy Algorithm for Storage Control under Uncertainty. IEEE Transactions on Power Systems. 2016 May;31(3):1729–1743. ArXiv: 1405.7789. Available from: <http://arxiv.org/abs/1405.7789>.
- [17] Wang H, Zhang B. Energy Storage Arbitrage in Real-Time Markets via Reinforcement Learning. arXiv:171103127 [cs, math]. 2017 Nov;ArXiv: 1711.03127. Available from: <http://arxiv.org/abs/1711.03127>.

- [18] Arulkumaran K, Deisenroth MP, Brundage M, Bharath AA. A Brief Survey of Deep Reinforcement Learning. IEEE Signal Processing Magazine. 2017 Nov;34(6):26–38. ArXiv: 1708.05866. Available from: <http://arxiv.org/abs/1708.05866>.
- [19] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, et al. Playing Atari with Deep Reinforcement Learning. arXiv:1312.5602 [cs]. 2013 Dec;ArXiv: 1312.5602. Available from: <http://arxiv.org/abs/1312.5602>.
- [20] van Hasselt H, Guez A, Silver D. Deep Reinforcement Learning with Double Q-learning. arXiv:1509.06461 [cs]. 2015 Sep;ArXiv: 1509.06461. Available from: <http://arxiv.org/abs/1509.06461>.
- [21] Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, et al. Continuous control with deep reinforcement learning. arXiv:1509.02971 [cs, stat]. 2015 Sep;ArXiv: 1509.02971. Available from: <http://arxiv.org/abs/1509.02971>.
- [22] Kakade S, Langford J. Approximately Optimal Approximate Reinforcement Learning. In: In Proc. 19th International Conference on Machine Learning; 2002. p. 267–274.
- [23] Schulman J, Levine S, Moritz P, Jordan MI, Abbeel P. Trust Region Policy Optimization. arXiv:1502.05477 [cs]. 2015 Feb;ArXiv: 1502.05477. Available from: <http://arxiv.org/abs/1502.05477>.
- [24] Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal Policy Optimization Algorithms. arXiv:1707.06347 [cs]. 2017 Jul;ArXiv: 1707.06347. Available from: <http://arxiv.org/abs/1707.06347>.
- [25] TEPCO. List of price plans (home customers) Begin! Electric power deregulation: Tokyo Electric Power Energy Partner; 2018. Available from: <http://www.tepco.co.jp/jiyuuka/service/plan/index-j.html>.
- [26] Green A. Reinforcement learning agents and environments for energy systems; 2018. Original-date: 2017-04-03T10:14:01Z. Available from: https://github.com/ADGEfficiency/energy_py.
- [27] Irpan A. Deep Reinforcement Learning Doesn't Work Yet; 2018. Available from: <http://www.alexirpan.com/2018/02/14/rl-hard.html>.
- [28] Dhariwal P, Hesse C, Klimov O, Nichol A, Plappert M, Radford A, et al.. OpenAI Baselines. GitHub; 2017. <https://github.com/openai/baselines>.
- [29] Schaul T, Quan J, Antonoglou I, Silver D. Prioritized Experience Replay. arXiv:1511.05952 [cs]. 2015 Nov;ArXiv: 1511.05952. Available from: <http://arxiv.org/abs/1511.05952>.
- [30] Kostrikov I. PyTorch Implementations of Reinforcement Learning Algorithms. GitHub; 2018. <https://github.com/ikostrikov/pytorch-a2c-ppo-acktr>.

Figures

Figure 1
Pricing data for the first three days of Jan.

Figure 2
Normalized weather data giving the temperature, sunlight, rainfall and daylight for the first three days of Jan.

Figure 3

Four different artificially created demand plans over the first three days of Jan.

Figure 4

Daily averaged generation and consumption data for household A over the six-month period of available data.

Figure 5

Daily averaged generation and consumption data for household B over the six-month period of available data.

Figure 6

Daily averaged generation and consumption data for household Cover the six-month period of available data.

Figure 7

Daily averaged generation and consumption data for household D over the six-month period of available data.

Figure 8

NILM data for household A over a three-day period at the start of Jan.

Figure 9

interaction between the agent and environment in an MDP.

Figure 10

Daily-averaged requested energy profile for the baseline model on household A.

Figure 11

Daily-averaged normalized pricing data for the baseline model on household A.

Figure 12

Daily-averaged requested energy profile for the best DQN model on household A.

Figure 13

Daily-averaged normalized pricing data for the DQN model on household A.

Figure 14

Daily-averaged normalized charging profile for the DQN model on household A.

Figure 15

Total rewards per episode during the training period for the best DQN model

a. Initial model (not optimal)

b. Final model (best)

Figure 16

Disaggregated reward signals for two different DQN models.

Figure 17

Daily-averaged requested energy profile for the best PPO model on household A.

Figure 18

Daily-averaged normalized pricing data for the PPO model on household A.

Figure 19

Daily-averaged normalized charging profile for the PPO model on household A.

Figure 20

Total reward per episode during the training period for the best PPO model.

a. Initial model (not optimal)

b. Final model (best)

Figure 21

Disaggregated reward signals for two different PPO models

Figure 22

Daily-averaged requested energy profile for the best PPO and DQN model on household A.

Figure 23

Daily-averaged requested energy profile for the best PPO and DQN model on household A.