Reinforcement Learning Based Battery Control for Optimal Peak Shifting

Informetis Co., Ltd

インフォメティス株式会社

Keir Simmons[1]

MSc Computational Statistics & Machine Learning

Academic supervisor: Dr Mark Herbster

Industry supervisor: Masato Ito (CTO)

Submission date: September 7, 2018

[1]**Disclaimer:** This report is submitted as part requirement for the MSc Computational Statistics & Machine Learning at University College London. It is substantially the result of my own work except where explicitly indicated in the text. The report will be distributed to the internal and external examiners, but thereafter may not be copied or distributed except with permission from the author.

Abstract

This project explores the application of peak-shifting in tackling the problem of large peaks in the demand of electrical energy from the grid. These 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 peaker 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. The use of a battery energy storage system to allow for the purchase and storage of 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 optimisation problem. Reinforcement learning methods are investigated 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.

Through the use of PPO, a popular policy gradient-based reinforcement learning method, a very strong policy has been found. 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 through the use of a simulator that the author has developed specifically for this task, which handles the model training, testing and evaluation process. Secondly, development of a novel technique, *automatic penalty shaping*, was also found to be crucial to the success of the learned model. This technique enabled 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 optimisation problems.

The policy does, however, attempt to overcharge the battery about 7% of the time, and promising methods to address this have been proposed as a direction for future research.

Contents

List of Tables

List of Figures

List of Snippets

Introduction

Project overview & personal contributions

This project explores the problem of large peaks in the demand of electrical energy due to trends in energy consumption habits of consumers. These large peaks force the electrical grid to activate peaker plants in order to provide the additional energy. However, these peaker plants increase carbon emissions and are expensive to operate, with the additional costs being footed by the consumer. The aim of the project is to incorporate a smart battery energy storage system at consumers' sites, providing the ability to purchase and store energy at non-peak times for later use, with the intention of reducing these peak demand periods and hence the reliance on peaker plants. Reinforcement learning methods are then explored to find an optimal control policy over the purchase of energy in order to achieve peak-shifting, that is, the reduction of these peak periods, and also lower energy consumption and utility bills.

The contributions made through this project are two-fold. The first is a fully contained, flexible, modular simulator that allows for training of these models using various reinforcement learning algorithms and battery environments. The second contribution is through the development of a method termed *automatic penalty shaping*, which is necessary to construct an appropriate reward signal from which the reinforcement learning agent can learn. This method has been incorporated into the simulator and it was found to be a crucial element to the success of reinforcement learning methods being applied to this domain.

Our reliance on electrical energy

Electricity is a critical necessity in everyday life, but, whilst reliant on this energy source, humankind takes its existence for granted, and is unaware of the extreme difficulties arising from its generation and consumption. For example, during a power outage, household temperatures can quickly become too hot or cold due to air conditioning units not working, which can result in severe health problems and even fatalities in

at-risk groups, such as growing children and the elderly. The treatment of water relies on electricity, and as such a power outage can result in the population consuming contaminated water, again likely resulting in hospitalisation and the death of many individuals. In addition, the many people who are already hospitalised may lose access to vital treatment and support such as life-support machines, if there is no energy available. As another example, the 2003 blackout in the USA and Canada was linked to almost 100 deaths of civilians due to various side-effects of a lack of power, such as accidents, carbon monoxide poisioning from generators and a variety of health issues [3]. Health reasons aside, a power outage causes major disruption to the economy as many people cannot continue their jobs due to health & safety reasons. These few examples only begin to scratch the surface of demonstrating just how important electricity is and our increasing reliance on it in our everyday lives.

Centralised energy network

Historically, and up until only recently, the electrical energy network has remained stagnant as a centralised system, known as the *electricity grid*, responsible for all of the phases of generation, distribution and storage. Centralised generation facilities are usually very large sites such as wind farms, hydroelectric dams, nuclear power plants and the more standard fossil fuel-based power plants. Once generated, this electrical energy needs to be transferred to households, businesses and other facilities such as factories via a process called transmission before then being *consumed* by the end-user.

This historical system has many issues which are becoming increasingly detrimental to our global, combined effort of moving towards renewables and reducing the reliance on depleting resources such as fossil fuels.

In 2011, generators at Japan's Fukushima Dai-Ichi nuclear power plant site suffered meltdowns and hydrogen-air explosions in 2011, thought to have resulted in over 1400 deaths. This is not the first accident to occur at a nuclear power plant site, but provides even more incentive for a shift towards renewable sources of energy as the demand for electricity continues to increase. Environmental concerns are not the only problem present in this centralised system. Many interruptions in the supply of power from the current grid system can be attributed to old equipment installed many years ago, which, whilst can be repaired and replaced, result in exceedingly high costs for maintenance and an overall reduction in the reliability of the entire grid. Due to the centralised nature of the grid, a single fault can affect multiple regions and leave millions of people without power. The centralised electricity system also has incredibly low efficiency due to having to transmit power over large distances. This low efficiency has a positive-feedback effect of having to generate increasingly more energy to meet the demand.

Shift to a distributed energy network

As civilians and companies alike become increasingly concerned about the future of electrical energy, there has been a tendency to shift towards on-site conversion of renewable sources of energy, such as in the use of solar panels. With this, and the various problems behind the centralised system in mind, we are quickly moving towards a new infrastructure, known as the distributed energy grid. This decentralised system refers to the use of numerous, smaller generation sites located closer to areas of high energy demand, reducing the need for complex and long transmission and distribution infrastuctures. By reducing the transmission losses inherent in the centralised systems, this change increases the overall efficiency of the system, reducing generation demand and also carbon emissions [4]. As well as increasing the efficiency, there is much more flexibility offered by such a decentralised system. For example, components of the distributed systems can be replaced and repaired more easily, and this can be done without affecting the entire system as a whole. This improvement increases the reliability of energy transmission and, in the now rarer case of a power outage, restricts the loss of power to specific regions only. A distributed energy network also allows us to take advantage of more renewables such as solar and pave the way for removing depleting fossil fuel resources from the equation altogether [5].

However, this shift to a decentralised energy network does not come without its own problems. To maintain a stable grid, it is necessary to be able to accurately forecast energy generation and consumption. In the centralised system this problem was comparatively simple, with the one major uncertainty in forecasting being unplanned maintance of power plants due to unforeseen complications [6]. Decentralised networks' main problem lies in the fact that much of the generated energy is intermittent. For example, although sunlight can be accurately predicted throughout the day via weather forecasting, cloud coverage introduces a lot of noise into these predictions and is difficult to account for [6]. In addition, system operators do not know that households are even generating their own energy, and instead see it as reduced consumption. This lack of knowledge introduces yet another layer of noise into the forecasting problem. These additional inaccuracies in forecasting present major inconviences in providing an efficient and reliable alternative energy network. Uncertainty in generation and consumption result in increased electricity prices which will negatively affect homeowners and companies alike [6].

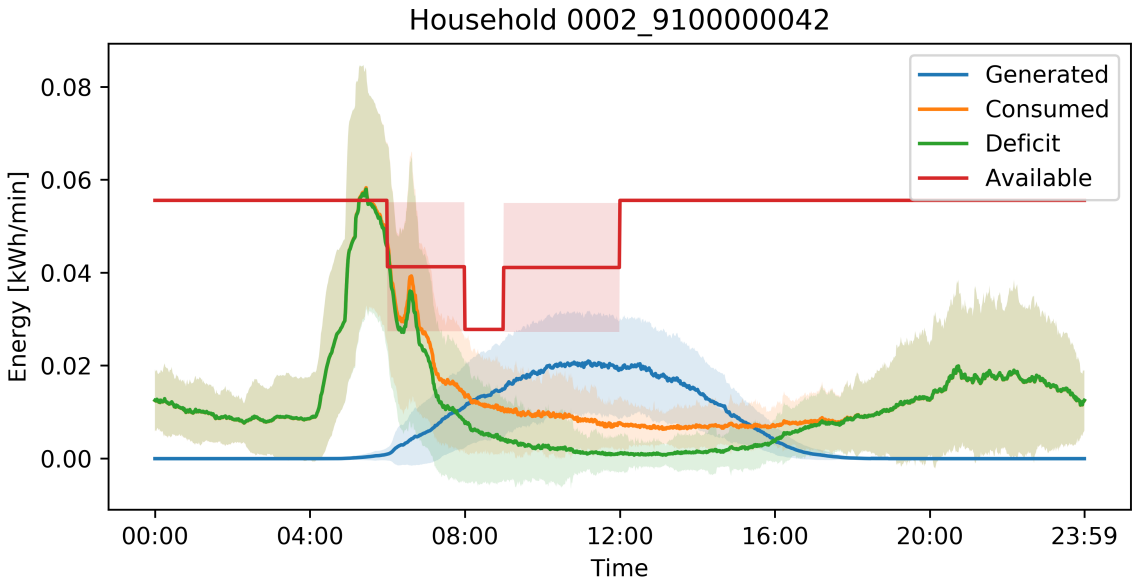


Figure 1.1: Averaged day-by-day energy consumption and generation by a random household in Japan over a period of six months.

The problem of peak demand

Another 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 decentralised network at times when the forecasted demand for energy is significantly lower than the actual demand. These peaker 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 peaker 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.

Fig. 1.1 clearly demonstrates this issue. In this figure, the energy consumption and

generation of a single random household in Japan is averaged daily over a period of 6 months and presented. The shadowed curves in this and all graphs that follow are a visualisation of 1 standard deviation either side of the mean, to more easily understand the variance in the presented data. The red curve, labeled *Available* shows the energy available for purchase from the grid. The green curve, *Deficit* shows the required energy at all times by the household after subtracting the amount generated from that being consumed. The generated energy curve is a representative example of energy generated via solar panels, with the maximum amount generated in the middle of the day when sunlight is generally strongest. In this graph, it is readily apparent that there are times, especially between 4AM - 8AM, where the required energy is greater than that available. It is times like this when expensive and inefficient peaker plants would need to be used.

The peak demand problem refers to finding solutions which mitigate this problem allowing the grid to provide sufficient energy without needing to rely on peaker plants. Such a solution would help reduce costs for the end-user, lower carbon emissions, and lead us away from the reliance on fossil fuels.

Load management as a solution

The peak demand problem is not new. It has existed since the start of the centralised energy network, and as such many solutions have been proposed and implemented to date. The most common form of solutions fall under the term *load management*. These are a collection of methods designed to manage the *load*, or *electricity demand*, during peak times in such a way that the demand is reduced to an amount below the limit thereby avoiding the use of peaker plants.

Ripple control

The most common form of load management methods is known as *ripple control*. In times of high demand, ripple control refers to superimposing a high-frequency signal, usually between 100 and 1600 Hz, onto the original $50-60$ Hz signal. There are simple devices which can be attached to non-essential appliances, which, upon receiving this high frequency signal, will suspend operation of the appliance [8]. This suspension results in a lower overall demand without jeopardising the operation of high-importance appliances. Although this at first appears as a disadvantage for the consumer, they are generally rewarded for taking part in such a programme with lower energy costs.

However, there remain numerous disadvantages to this method of load management. The most significant being that this is an opt-in solution and requires many

consumers to take part in order to have any significant effect on the overall demand for energy. It also increases the uncertainty in forecasting enegy consumption as there is now an additional variable to take into account; whether or not the household is using these ripple control load controllers and also to which devices they have been attached. Additionally, a problem potentially more significant for the end-user is that of failed signal interception. For example, once we transition back into an off-peak time, if the signal sent to the device is not intercepted it would result in that appliance remaining suspended and unable to be used by the consumer [8].

Frequency-based decentralised demand control

The next most common method, known as *frequency-based decentralised demand control*, works without the need to superimpose an additional signal, thereby alleviating the problem of missed signals. At times of greater loads, the rotors in the generators of a grid naturally rotate more slowly, resulting in a reduced frequency which is immediately noticeable. A cheap micro controller can listen for these changes and suspend appliances' energy consumption by using demand control relays [9]. For appliances which can operate at lower voltages, this method would also allow for some sort of interpolation of voltage rather than a binary on or off control system.

Although this solves ripple control's problem of missed signals, its major disadvantage is that the grid has much less control and authority on choosing what devices should be suspended in times of high load. This loss of control firstly means that the devices chosen by the household may not provide enough load management to significantly reduce the peak demand, but also increases the uncertainty in forecasting expected energy consumption, again leading to higher costs in energy prices.

Peak shifting as a solution

Peak-load shifting is the process of alleviating the grid from regions of high energy demand by delaying or advancing requests to off-peak times, when the grid is more capable of handling higher loads [10]. This regulation of load flow is a nice idea on the surface, but cannot always be put into practice. A more suitable solution is the utilisation of *energy-storage systems* (*ESS*). ESS allows consumers to purchase energy at off-peak times, and store the energy for later use during peak periods to alleviate the strain on the grid. Fig. 1.2 shows the result of peak-shifting on the overall energy demand profile, clearly demonstrating the lower maximum request for energy by the end-user.

This process gives rise to a multitude of advantages to both the consumer and

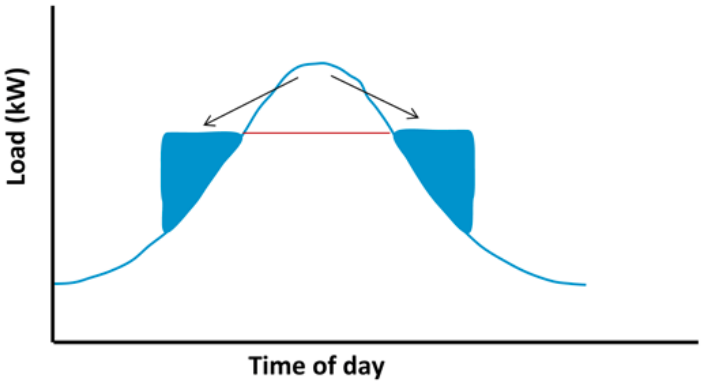


Figure 1.2: Visualisation of peak shifting, resulting in a more constant energy demand profile which is more manageable for the system operator. Modified from [1].

system operators alike. For the system operator, lower requests during what were previously peak periods means less, or no, reliance on peaker plants, and also the ability to provide electrical energy with greater reliability. Secondly, with the modified load profile becoming more constant and predictable, forecasting of consumption is made much easier allowing for more accurate planning and optimal generation of energy, in turn leading to a more reliable and stable energy output as well as a cheaper tariff. For the consumer, purchasing energy at off-peak times can result in overall cheaper utility bills as energy at these times is usually priced lower. When peak shaving is used primarily for the purpose of reducing cost, it is referred to as *peak shaving* [10].

Energy storage for peak shifting is also not a new technology, but in the past was not as feasible due to high costs for energy storage solutions. Between 1990 and 2005, the cost of lithium ion batteries has reduced by 90% and has continued to fall since, making ESS not only feasible for large businesses but also smaller households looking to take advantage of lower utility bill costs and renewables [11]. In addition, with the progressive shift to renewable energy generation, ESS solutions are becoming more and more attractive. In a traditional setup without ESS, any generated renewable energy, such as that via solar panels, needs to be used when it is generated. The generation profile from photovoltaics (solar panels), as seen in Fig. 1.1, in general yields high energy in the middle of the day and much less at other times when there is less sunlight. But this is only an optimal profile if the end-user requires more energy at these exact times, which is generally not the case. Using the generated energy as soon as it becomes available, without resorting to an ESS, is known as *self-consumption*. As in Fig. 1.1, this household requires more energy between 4AM - 8AM and so they are not optimally taking advantage of their energy generation profile. Utilising ESS allows for generation shifting, or the storage of generated energy to allow it to be used at a later time when it is needed. In this example, this would be similar to storing the energy generated by

the photovoltaic cells, and using it between 4AM - 8AM the next day.

In addition to these advantages, in the rare circumstances where the grid does not accurately forecast consumption and cannot provide all consumers with the required energy, consumers taking advantage of ESS can continue to operate as normal by using the energy stores until the system operator can return their service to normal. This is an attractive trait of ESS in critical fail-safe use cases, such as in life-support machines at hospitals where even back-up generators do not provide the essential peace of mind. Returning to the example of the 2011 nuclear meltdown in Fukushima, the tsunami not only caused the initial power outage, but also prevented the backup generators from turning on, causing the cooling pumps to fail, consequentially resulting in the disastrous accident which is well-known worldwide today. With this in mind, companies such as *Microsoft* and *Google* are now extensively using battery storages in their data-centres as a failsafe tool [12].

Optimal energy storage

On the surface, the use of ESS to aid peak shifting seems a suitable solution to the peak demand problem. However, without knowing when is the best time to purchase and store energy, this method provides no advantage to the initial balanced consumption & generation situation. This is not an easy problem to solve; in order to know when is the best time to purchase and store energy, we need to be able to accurately predict the available energy to purchase at any time, when the consumer will need energy, how much energy they will need and also how much energy they will generate. The latter of these has multiple layers of difficulties - for photovoltaic generation, this would require accurate weather forecasting taking into account the additional noise imposed by cloud coverage which affects the amount of sunlight which is actually absorbed by the solar panels. Secondly, if a reduction in cost of utility bills is also desired, it is also important to forecast the prices of electrical energy which can change multiple times a day and can be a function of demand, time, date and many other seasonal features.

Clearly, the solution to this multi-objective optimisation problem is not simple and is a function of many variables, some unique to each individual consumer, and some unique to different countries. For example, in the USA it is possible to buy and sell energy back to the grid to allow for profits through energy arbitrage, or the process of buying energy at low prices and selling it back when the price is higher [13]. However, in Japan, such arbitrage is not possible, where consumers are only permitted to sell energy that they themselves have generated at a price determined separately to the cost of energy from the grid [14]. This sell-price is also dynamic and changes every year whilst being a function of how much generated energy is being sold.

Problem specification

This report will outline a solution method to the peak demand problem utilising a battery energy storage system (BESS), 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 BESS 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 fairly easy to model programmatically.

Based on the data made available by *Informetis*, 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 ultimate goal is not to provide the exact optimal policy for a specific ESS for a specific household in a specific country, 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 ultimate goal will be to achieve optimal peak shifting, thereby nullifying the need for use of peaker-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. Due to the fact that 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 optimises 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 peaker plants once per year, but also reduces utility bills by 50%. The second model achieves absolutely 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 own 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.

Related Work

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 timesteps separated by intervals of size Δt, and then finds the optimal actions at each of these timesteps. In addition, the DP algorithm described within requires discretisation of the battery capacity, thereby prohibiting a fully flexible, continuous energy storage system. Decreasing the timestep interval and capacity discretisation provides additional fine-grained control, but as DP grows polynomially 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 optimisation, 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 BESS [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 the 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.

Datasets

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 datapoints over a period of six months. The observational data comprise of three different datasets; the *price* 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.

Price

Fig. 3.1 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 refer to actual pricing plans offered by *TEPCO*, or *Tokyo Electric Power Company*. *night_8* and *night_12* are plans for customers who use more electricity at night time 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.

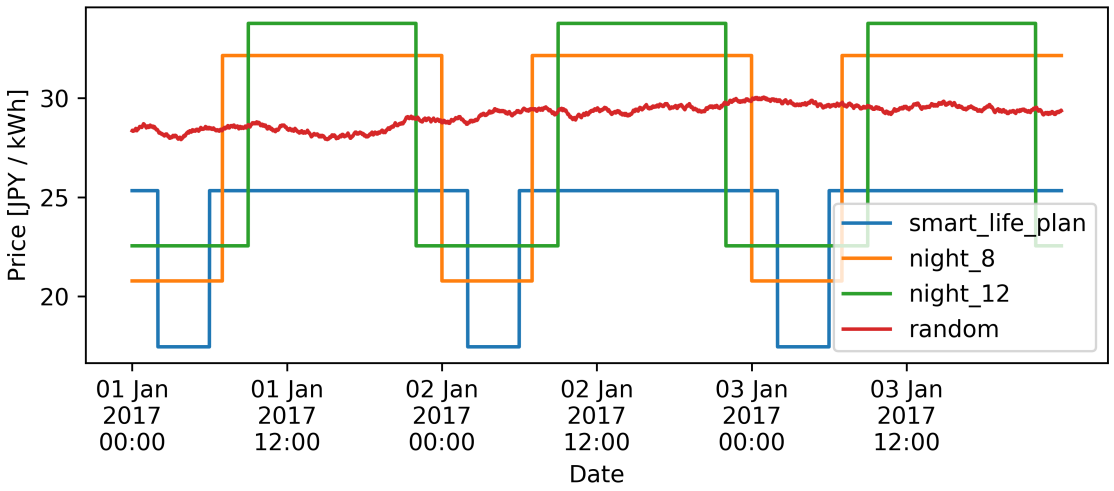


Figure 3.1: Pricing data for the first three days of 2017 following four different pricing bands.

Weather

Fig. 3.2 gives four different meteorological datasets which can be taken advantage of to predict energy generation via photovoltaics. 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. Note that although the data presented has been normalised, this does not affect the learning process as all data will be standardised 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.

Demand

Fig. 3.3 presents four different demand plans over the first three days of 2017. Recalling 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 *Informetis* 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

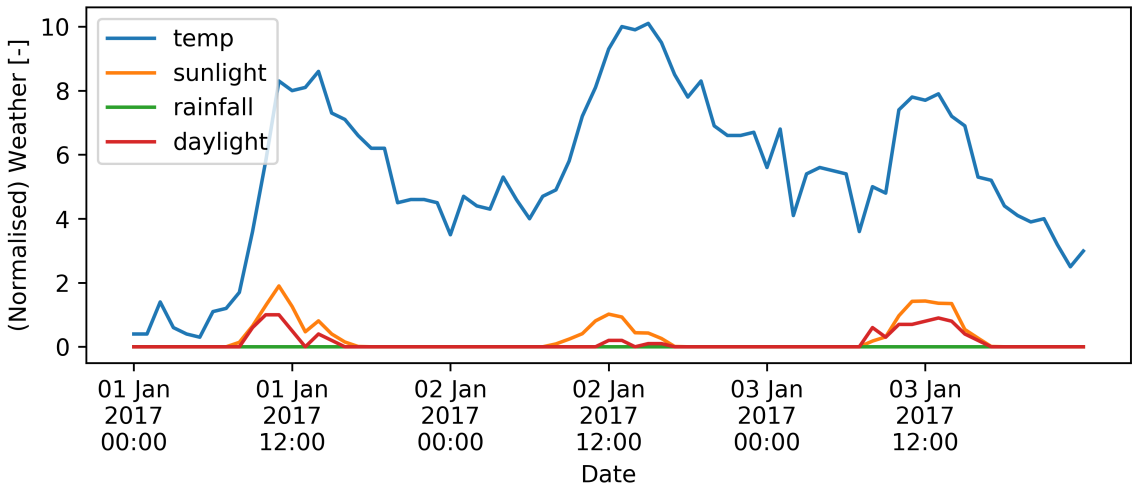


Figure 3.2: Normalised weather data giving the temperature, sunlight, rainfall and daylight for the first three days of 2017.

for each single-minute timestep. 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.

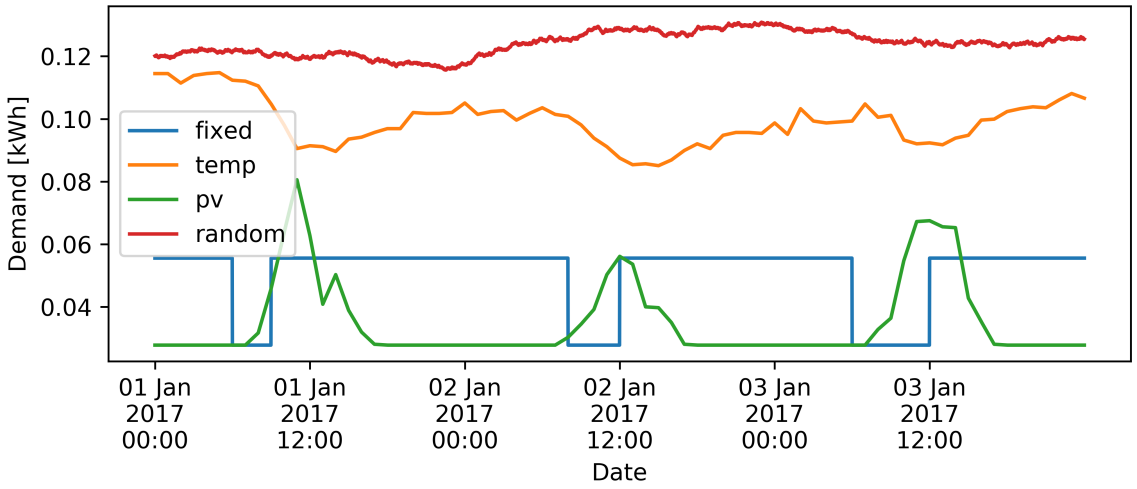


Figure 3.3: Four different artificially created demand plans over the first three days of 2017.

Households

In total, *Informetis* have 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 *Informetis* 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. Tab. 3.1 provides a mapping between the original IDs and alphabet keys for future reference.

Alphabet	ID Key
A	0002_9100000042
B	0002_9100000112
C	0002_9100000113
D	0002_9100000114

Table 3.1: Mapping of original household IDs to a simpler alphabet referencing structure.

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 *dish_washer* 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

Field \Household	A	B	C	D
date	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
photovoltaics	28	5882	31	125
refrigerator	10	10	1	2
ricecooker	1	9	1	34
tv	3	2	99079	8
washer	16	1	1	13
Total	94	35235	99119	260956

Table 3.2: NaN counts for each NILM entry in the household datasets.

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 taking into account the seasonal nature of this data and so the interpolation filling-method was also rejected in favour of mean-filling. It should still be noted that mean-filling ignores the variation in data due to seasonal effects, but should result in minimimal 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. *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 due to the fact that the controller attached to the solar panel cells can sometimes consume more energy than has been generated.

Figs. 3.4 through 3.7 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 is 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.

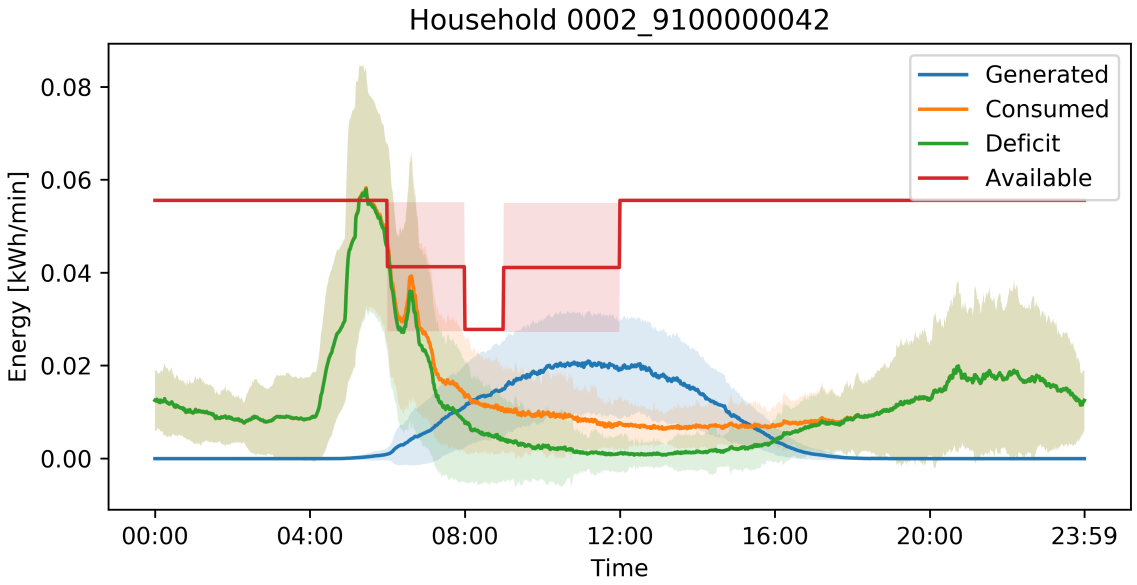


Figure 3.4: Daily averaged generation and consumption data for household A over the six month period of available data. Energy demand has also been given to provide a better understanding of the peaks.

Fig. 3.8 shows the disaggregated NILM data for the chosen household A over the first three days of 2017. Here it is possible to understand better the relative scales of each appliance's energy consumption and also 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.

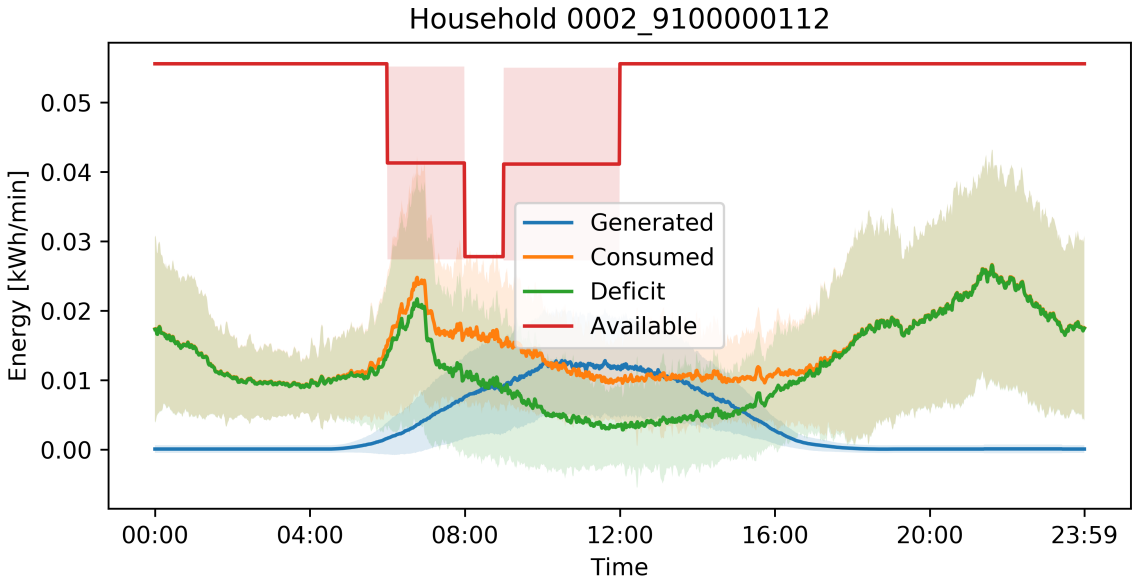


Figure 3.5: Daily averaged generation and consumption data for household B over the six month period of available data. Energy demand has also been given to provide a better understanding of the peaks.

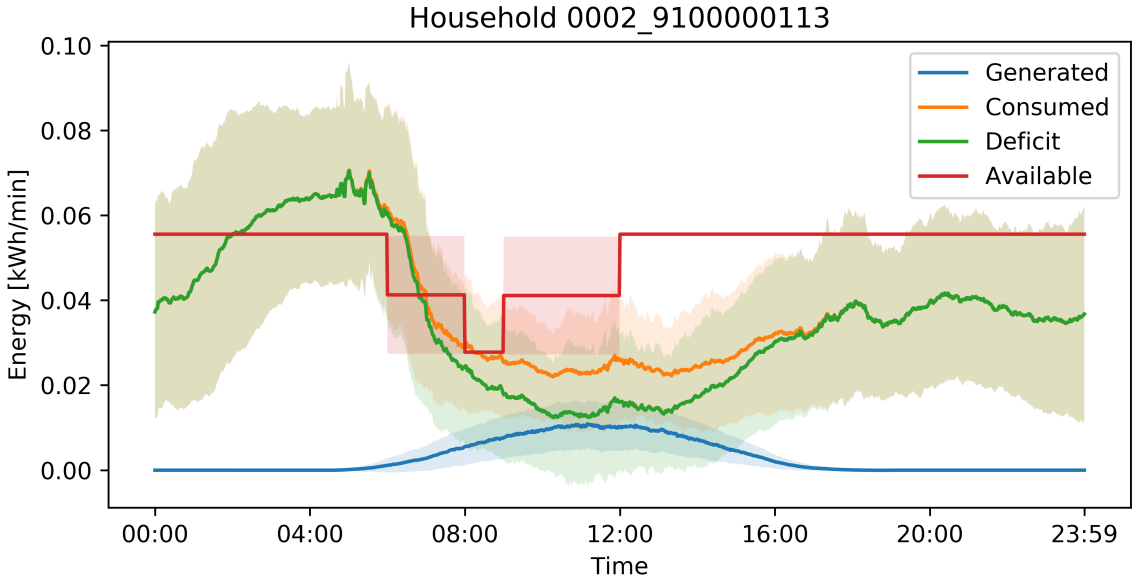


Figure 3.6: Daily averaged generation and consumption data for household C over the six month period of available data. Energy demand has also been given to provide a better understanding of the peaks.

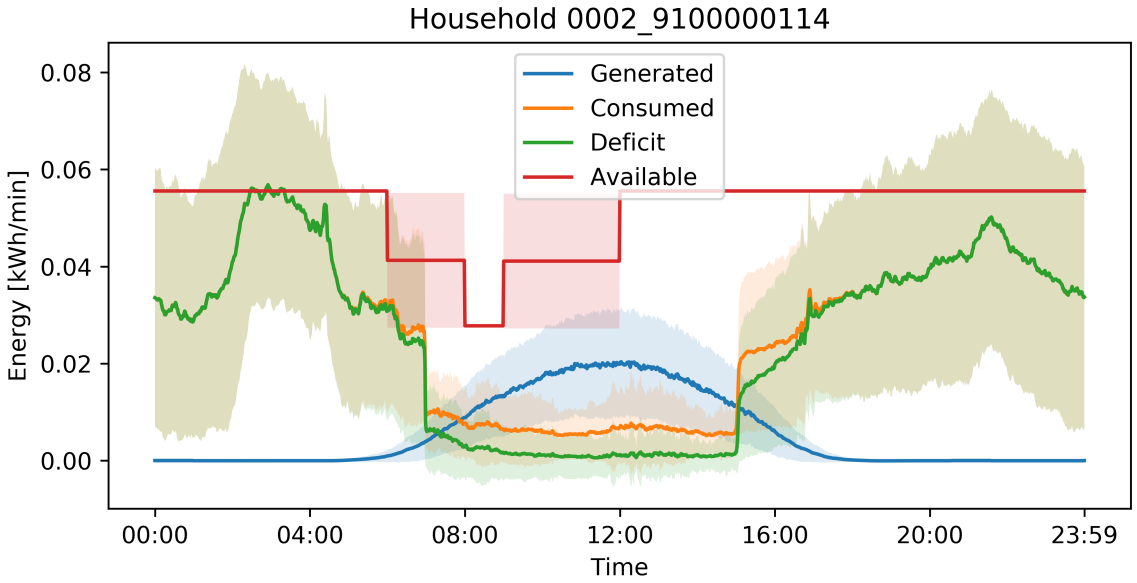


Figure 3.7: Daily averaged generation and consumption data for household D over the six month period of available data. Energy demand has also been given to provide a better understanding of the peaks.

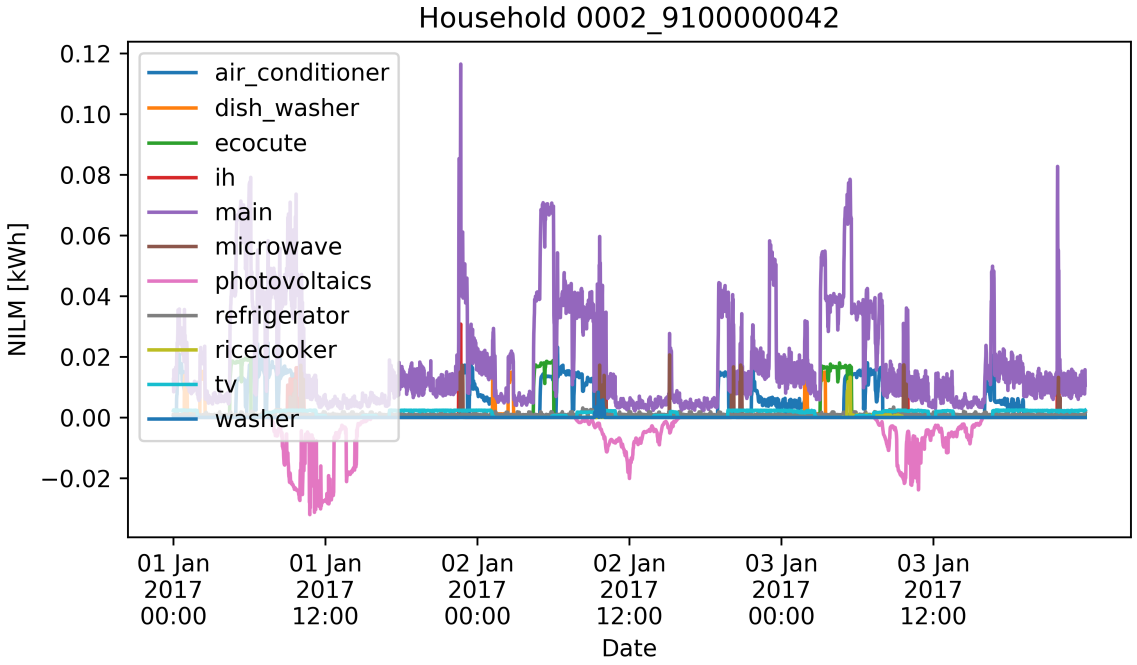


Figure 3.8: NILM data for household A over a three day period at the start of 2017.

Reinforcement Learning for Control

Following the recent work in [17] on applying reinforcement learning to the problem of energy arbitrage, its application to the more difficult multi-objective optimisation 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.

Overview of the reinforcement learning framework

Reinforcement learning refers to a collection of methods involving a learning agent which can interact with an environment, and based on these interactions can adjust its behaviour in order to achieve a specific goal by sensing how its interactions change its perception of the environment [2]. The agent is not told which actions are best to choose in any given situation, but rather needs to learn this by interacting with the environment and discovering which actions yield the most *reward* in different situations. Reward refers to a single numeric metric which aims to describe how good or bad it was to take an action in a given state. However, this is not as simple as trying each possible action in each state and monitoring the reward yielded by the environment, for a number of reasons. The first of these being that the environment may be stochastic, and as such the reward may not be a single, constant value. Another reason being that there may be far too many combinations of states and actions that it is simply not possible to *visit* them all. Another problem is the concept of delayed rewards. For example, in the game of draughts, a single tactical move may not remove any of the opponent's pieces from the board, but may be a vital move towards winning the game. Although that tactical move would not have received any immediate reward due to the

fact it did not produce any immediate effect on the state of the game, its delayed effect on the game needs to be taken into account; this is known as the *credit assignment* problem, or the concept of delayed rewards.

One way of tackling the problem of delayed rewards is by introducing the notion of the *value function*. Whereas the reward gives an immediate indication of how good it was to take a specific action in some state, the value function aims to quantify how good it is to be in that state long-term, by estimating the total reward, or return, that the agent will achieve from that state onwards. This is an important notion in the reinforcement learning framework, as a specific state may always yield low reward but be followed by a series of states with desirable high reward profiles. Therefore, in the learning process, we aim to guide the agent into selecting actions which maximise value, not immediate reward, as this will push it towards the overall goal of maximising the total reward signal. This is not an easy problem to solve; rewards are single values given directly by the environment upon interaction with it, whereas values are estimates which need to be continually updated based on long sequences of observations over the lifespan of the agent. If the value function over all states can be accurately determined, then the reinforcement learning problem has been solved and the agent is able to optimally select the most desirable actions in any given state.

It is clear that reinforcement learning agents learn by having their choices *evaluated*, whereas in the supervised learning setting learning takes place via *instruction*, where the learner is explicitly told whether or not their choice was correct. In reinforcement learning, this is not the case as its generally not even possible to define a notion of what a *correct* action even is. Another major difference between reinforcement learning and the supervised learning paradigm is what is known as the *exploration-exploitation* dilemma [2]. The aim of the agent is to maximise a reward signal, but in order to do so it must find which actions are the best to take in different situations. The agent should be able to prefer actions it has taken in the past which are known to yield high reward, selection of which is referred to as exploitation, but should also be able to try other actions to allow it to discover such promising actions in the first place, a process known as exploration. If an agent solely exploits an environment, and the first action it chooses provides a positive reward, it will always select that action again in the future even if one of the other actions would have yielded higher reward. Similarly, if the environment is stochastic, the agent should take actions multiple times to increase its certainty over the expected reward signal to avoid ignoring actions which, by chance, yield a low reward the first time they are selected. However, if an agent solely explores, it will never take advantage of its wide-breath of knowledge of which actions are better to take in different situations. As such, it is clear that there needs to be a trade-off between these two concepts. This exploration-exploitation dilemma is unique to the problem of

reinforcement learning and is of no concern in the supervised and unsupervised learning paradigms [2].

At a high level, reinforcement learning aims to guide an agent to learn how best to interact with its environment with the aim of achieving a predefined goal through the maximisation of a reward signal. It is important to make clear that the concept of a goal and reward signal are similar but not the same. For example, if one considers using the reinforcement learning framework to model the problem of a character stuck in a maze, the goal is simple - exit the maze. However, there are an infinite number of reward signals that could be designed to solve this task. Recalling that the agent modifies its behaviour to maximise its reward signal, one may shape the signal to yield $+1$ on escaping the maze, and 0 at all other times. Whilst an agent that successfully learns to maximise the reward signal would eventually learn to take actions that lead the character out of the maze, it should be clear that this is not the only such reward signal that would achieve this, and certainly not the best one. A much more efficient reward signal could yield $+1$ on escaping the maze, but -1 at all other times. In maximising the reward signal, the agent now would learn to take the minimal number of actions to escape the maze. Hence, although in both cases the goal of escaping the maze is the same, the reward signal is not. In general, there is no pre-defined reward signal and its design can be paramount to the efficient solution of the control problem.

Mathematical formulation

Markov Decision Processes, or *MDPs*, are a mathematical formalisation 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 taken into account [2].

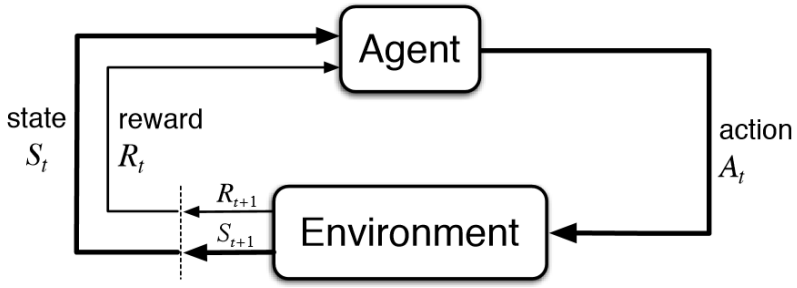


Figure 4.1: The interaction between the agent and environment in an MDP. Modified from [2].

Fig. 4.1 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 timestep 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].

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 Markovian sense, depend only on the previous state-action pair. Mathematically, the environment dynamics, p, of this model can be given as:

$$p(s', r|s, a) \doteq \Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\} \quad \forall s', s \in \mathcal{S}, r \in \mathcal{R}, a \in \mathcal{A}(s) \tag{4.1}$$

Where it can be seen that the probability distribution is well-defined solely on the current and previous timesteps 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 has to be learned.

As Eqn. 4.1 is a probability distribution, it follows that:

$$\sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}(s)} p(s', r|s, a) = 1 \quad \forall s', s \in \mathcal{S}, a \in \mathcal{A}(s) \tag{4.2}$$

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 timestep onwards, as shown in Eqn. 4.3, where $0 \leq \gamma \leq 1$ is called the *discount factor*, a tunable hyperparameter. 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.

$$G_t \doteq \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{4.3}$$

In order 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 maxmised. Formally, if an agent follows a policy π, this is equivalent to saying it will take action $A_t = a$ 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 Eqn. 4.4, where the subscript π refers to the agent selecting actions based on policy π.

$$V_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right] \forall s \in \mathcal{S} \tag{4.4}$$

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

$$Q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right] \tag{4.5}$$

The optimal value function can then be defined as:

$$V^*(s) \doteq \max_\pi V_\pi(s) \quad \forall s \in \mathcal{S} \tag{4.6}$$

Similarly for the optimal action-value function:

$$Q^*(s,a) \doteq \max_\pi Q_\pi(s,a) \quad \forall s \in \mathcal{S} \text{ and } a \in \mathcal{A}(s) \tag{4.7}$$

Solution methods

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 parametrised

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 parameters that need to be learned than the original problem itself. As these functions are only approximations to the full problem, the learned policy will only be an approximation to 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 dynamics matrix, of the problem. These methods are clearly not desirable due to the fact that 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 timestep. 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 similar to 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 Eqn. 4.8. The behaviour policy chooses the action a_t in the current state based on an 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 behaviour 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 $\epsilon \to 0$ as $t \to 0$, the behaviour policy will converge to the optimal greedy policy [2]. Algorithm 1 gives the full pseudocode for this algorithm.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{4.8}$$

Algorithm 1 SARSA on-policy control algorithm. Modified from [2].

Set step-size $\alpha \in (0, 1]$ and small $\epsilon > 0$
Initialise $Q(s, a)$ for all $s \in \mathcal{S}^+$ and $a \in \mathcal{A}(s)$ arbitrarily
$Q(\text{terminal}, \cdot) = 0$
for each episode **do**
    Initialise s
    Choose a from s using ϵ-greedy policy
    for each timestep in the episode **do**
        Take action a, observe r, s'
        Choose a' from s' using ϵ-greedy policy
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$
        $a \leftarrow a'$

Q-learning is a very popular *off-policy* variant of SARSA. In this case the target policy, shown in the update rule in Eqn. 4.9, 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 maximisation 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 maximises 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. Algorithm 2 gives the pseudocode for this algorithm.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{4.9}$$

Deep reinforcement learning

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 approximators such as deep neural networks. Unfortunately, due to a phenomenon known as the *deadly triad*, such

Algorithm 2 Q-learning off-policy control algorithm. Modified from [2].

Set step-size $\alpha \in (0, 1]$ and small $\epsilon > 0$
Initialise $Q(s, a)$ for all $s \in \mathcal{S}^+$ and $a \in \mathcal{A}(s)$ arbitrarily
$Q(\text{terminal}, \cdot) = 0$
for each episode **do**
    Initialise s
    Choose a from s using ϵ-greedy policy
    for each timestep in the episode **do**
        Take action a, observe r, s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_a Q(s', a) - Q(s, a)]$
        $s \leftarrow s'$

non-linear function approximators 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 maximisation term in the target policy. Hence, introducing non-linear function approximators to model the Q function in Q-learning *activates* the deadly triad phenomonen.

DQN

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 timestep is fed into the network in order of observation, thereby making the data dependent on previous timesteps. This is a problem as most reinforcement learning methods, including Q-learning, assume the data to be iid (independent and identically distributed). The DQN paper [19] addresses this through the use of *experience replay*. In experience replay, the agent stores experiences $\epsilon_t = (s_t, a_t, r_t, s_{t+1})$ at each timestep in an overall replay memory $\mathcal{D}$. In the learning stage of the algorithm, updates to the Q-function are applied to minibatches 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 is given by Algorithm 3.

The gradient descent step is what adjusts the parameters θ of the neural network,

Algorithm 3 Deep Q-learning with Experience Replay. Modified from [19].

Initialise replay memory $\mathcal{D}$ to capacity N
Initialise action-value function Q with random weights
for episode $= 1, \mathrm{M}$ **do**
  Receive s_1 from the environment
  for $t = 1, \mathrm{T}$ **do**
    Select random action a_t with probability ϵ
    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 minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from $\mathcal{D}$
    Set:

$$y_j = \begin{cases} r_j & \text{for terminal } s_{j+1} \\ r_j + \gamma \max_{a_{j+1}} Q(s_{j+1}, a_{j+1}; \theta) & \text{for non-terminal } s_{j+1} \end{cases}$$

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

or other function approximator, 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 overstimation 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.

Policy gradient methods

All of 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 learn a parametrisation of the policy directly, avoiding

the need to perform a maximisation 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 also continuous action space. In the peak demand problem, the amount of energy that is bought at each timestep 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 discretise the action space which would result in a significant loss of control. Although one might suggest to use a fine-grained discretisation to maintain sufficient control, the number of actions increases exponentially with the degrees of freedom in the system, and so even a relatively coarse discretisation of the action space would result in an unwieldly number of actions [21].

An overview of policy gradient methods will be presented, with derivations leading into the development of *PPO*, or *Proximal Policy Optimisation*, the current de-facto algorithm used in reinforcement learning problems.

These methods parametrise the policy as $\pi(a|s, \boldsymbol{\theta}) = \Pr\{A_t = a|S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}$, where $\boldsymbol{\theta} \in \mathbb{R}^d$ with d referring to the number of degrees of freedom in the parametrisation of the policy. The goal of these methods is then to learn the values of $\boldsymbol{\theta}$ to shape the policy such that performance is maximised (i.e. finding the optimal policy). This is done via a measure of performance $J(\boldsymbol{\theta})$ and updating the parameters via gradient ascent as in Eqn. 4.10, where $\widehat{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})}$ refers to a stochastic estimate with an expectation that approximates $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \widehat{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})} \tag{4.10}$$

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].

Policy gradient theorem

In the episodic case, the performance measure is usually defined as the value function from the initial starting state; $J(\boldsymbol{\theta}) \doteq v_{\pi_{\boldsymbol{\theta}}}(s_0)$, where $v_{\pi_{\boldsymbol{\theta}}}(s_0)$ is the true value function for the policy $\pi_{\boldsymbol{\theta}}$. As $v_{\pi_{\boldsymbol{\theta}}}(s_0)$ 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 derivation of this theorem is given in Appendix. B.1 (Eqns. B.1 and B.2). The final result is shown below:

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s,a) \tag{4.11}$$

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 Eqn. 4.11 is sufficient as the constant of proportionality will be absorbed by the α term, a tunable hyperparameter in the update.

REINFORCE

This can then be re-written as:

$$\begin{aligned} \nabla J(\boldsymbol{\theta}) &\propto \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s,a) \\ &= \mathbb{E}_\pi \Bigg[\sum_a q_\pi(S_t,a) \nabla \pi(a|S_t,\boldsymbol{\theta}) \Bigg] \end{aligned} \tag{4.12}$$

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 Eqn. 4.13, where $\hat{q}$ refers to a learned approximation to $q_p i$ with $\boldsymbol{w}$ as the parameter vector.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \sum_a \hat{q}(S_t,a,\boldsymbol{w}) \nabla \pi(a|S_t,\boldsymbol{\theta}) \tag{4.13}$$

However, this update involves all of 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 take into account only A_t, the action taken at timestep t. This is done via the *gradient-trick* method, as used in the policy gradient theorem derivation, giving the result below. The derivation for this is given in Appendix. B.2 (Eqn. B.3).

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_\pi \Bigg[G_t \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta}) \Bigg] \tag{4.14}$$

This motivates the standard REINFORCE update rule, given by:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha G_t \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta}) \tag{4.15}$$

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 favour 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:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(G_t - V(S_t))\nabla \ln \pi(A_t|S_t, \boldsymbol{\theta}) \tag{4.16}$$

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

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\hat{A}_t\nabla \ln \pi(A_t|S_t, \boldsymbol{\theta}) \tag{4.17}$$

This is the final 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].

The road to PPO

Defining the objective function for REINFORCE as $J^{REINFORCE}(\boldsymbol{\theta})$, we have:

$$J^{REINFORCE}(\boldsymbol{\theta}) = \mathbb{E}_\pi\left[\hat{A}_t \ln \pi(A_t|S_t, \boldsymbol{\theta})\right] \tag{4.18}$$

Since the discovery of this algorithm, many other performance measurements have been proposed, including that given by Eqn. 4.19 [22].

$$\begin{aligned} J^{TRPO}(\boldsymbol{\theta}) &= \mathbb{E}_\pi\left[\hat{A}_t \frac{\pi_{\boldsymbol{\theta}}(a_t|s_t)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(a_t|s_t)}\right] \\ &= \mathbb{E}_\pi\left[\hat{A}_t r_t(\boldsymbol{\theta})\right] \end{aligned} \tag{4.19}$$

Where $\pi_{\boldsymbol{\theta}}$ refers to the updated policy and $\pi_{\boldsymbol{\theta}_{\text{old}}}$ 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 also 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*, or *Proximal Policy Optimisation*, 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:

$$J^{PPO}(\boldsymbol{\theta}) = \mathbb{E}_{\pi}\Big[\min(\hat{A}_t r_t(\boldsymbol{\theta}), \text{clip}(r_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon))\Big] \tag{4.20}$$

where $\text{clip}(r_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon)$ refers to constraining the value of $r_t(\boldsymbol{\theta})$ to the interval $[r_t(\boldsymbol{\theta}) - \epsilon, r_t(\boldsymbol{\theta}) + \epsilon]$ 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. In addition, it offers the same advantages as TRPO, just without the additional computational complexity. 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 parallelised 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.

Definition of the problem setting

The terms used in reinforcement learning are very general and offer much flexibility. Therefore, it is up to the designer to explicitly define what is meant by the agent, environment, action space, state space, and even the reward signal. In literature, new algorithms are generally tested against common platforms such as Atari games to provide a common testbed for simple comparison against other algorithms and methods. As such, the definition of each of these terms need to be consistent throughout the literature in order to allow for such comparisons to be made without bias toward any single method. As an example, in the Atari setup, it is possible to define the state as the raw visible pixels on the game screen, or to apply post-processing methods to these pixels such as converting from the three-channel RGB space to a single greyscale channel. Such an augmentation of the state-space, whilst potentially useful for learning, prevents

comparison against previous research which does not use the same augmentation.

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 $a_t \geq 0 \ \forall \ t$ describing how much energy will be bought from the grid at timestep 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 different means. As the main problem in this domain is that of peak shifting, which requires explicit modeling of the amount of energy bought at every timestep, it was decided to keep this value as explicitly learnable by the agent, hence the choice of action space. It is then up to the agent to learn, based on its interaction with the environment, whether or not the amount of energy it buys is sufficient for providing the necessary energy for consumption at the current timestep, but also that it buys enough excess energy for stores to use at later timesteps where consumption may be abnormally high. This should make clear the considerable difficulty of the domain. The agent needs to learn to take risks in purchasing minimal energy to save costs and achieve peak shifting, but also purchase enough energy for times of high consumption.

In order to help make these decisions easier, the state, or observation made available to the agent by the environment, should be shaped to provide sufficient information to

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 timestep. 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 timestep is 1 minute, this is equivalent to using all past observational data to predict the values just 1 minute ahead. Based on the dissagregated 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 timestep, 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. In the Atari set-up, the reward is easy to define as the game score visible on the screen, but in this problem such a definition is not as simple to define. How does one numerically define *peak shifting*, for example?

Reward signal shaping

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 optimisation tasks are first defined. In the following, the term *penalty* is used for negative rewards and is solely a term of convenience. In maximising the reward, the agent is thereby minimising 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, non-reinforcement learning approach to opimisaition, we would mathematically formulate an objective function to maximise subject to constraints. In our problem, these constraints would be along the lines of:

1. Purchase enough energy for consumption
2. Do not buy more energy than available (grid demand)
3. 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 timestep after taking into account 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 timestep.

$$\text{current\_energy} = \text{current\_charge} + \text{generated\_energy} - \text{consumed\_energy} \quad (4.21)$$

$$\text{min\_required\_energy} = \max(0, -\text{current\_energy}) \quad (4.22)$$

To ensure that constraint 1 is satisfied, a penalty termed *underpurchase* is designed and given by Eqn. 4.23, 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 maximise it, or achieve the value of 0, corresponding to never purchasing too little energy.

$$r_{\text{underpurchase}} = -\max(0, \text{min\_required\_energy} - \text{to\_buy}) \tag{4.23}$$

Constraint 2 is then tackled using the penalty given by *abovelimit* in Eqn. 4.24, 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 maximising this term to result in 0 penalty.

$$r_{\text{abovelimit}} = -\max(0, \text{to\_buy} - \text{demand}) \tag{4.24}$$

The penalty for constraint 3, termed *abovecapacity* is given by Eqn. 4.25, 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.

$$r_{\text{abovecapacity}} = -\max(0, \text{current\_energy} + \text{to\_buy} - \text{max\_capacity}) \tag{4.25}$$

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 & 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 Eqn. 4.26, where *price* refers to the current price of energy, with units JPY / 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.

$$r_{\text{cost}} = -\text{price} \times \text{to\_buy} \tag{4.26}$$

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 timestep. To address this explicitly, the reward termed *belowlimit* has been designed, and is given in Eqn. 4.27. 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 favour purchasing similar amounts of energy at each timestep, but tries

to keep the chosen amount as close to 0 at each timestep. 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 look into defining overly intricate reward functions.

$$r_{\text{belowlimit}} = \max\left(0, 1 - \frac{\text{to\_buy}}{\text{demand}}\right) \tag{4.27}$$

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 Eqn. 4.28, to provide a single overall reward signal to guide the agent's learning process. The values of the α_i 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.

$$\begin{aligned} r = {} & \alpha_1 r_{\text{underpurchase}} \\ & + \alpha_2 r_{\text{abovelimit}} \\ & + \alpha_3 r_{\text{abovecapacity}} \\ & + \alpha_4 r_{\text{cost}} \\ & + \alpha_5 r_{\text{belowlimit}} \end{aligned} \tag{4.28}$$

Simulator & Development Platform

The aim of this task is to verify that reinforcement learning is a suitable solution method to the peak-demand problem. As the training of a single model through actual buying of energy from the grid in a real production environment is not only costly but also incredibly time-consuming, an appropriate simulation platform is sought. This should be a generic platform with the ability of training on any valid dataset using any valid reinforcement learning algorithm. The platform should not only train a model, but also provide results and metrics which can be used by the designer to evaluate the resultant model and make decisions on how to tweak the parameters for future models. This simulation platform needs to be able to take into account the entire problem, that is, the use of various reinforcement learning algorithms, different datasets, different environment definitions (such as how the battery works), and even different reward signals. It should be a completely generic platform to allow for any reasonable adjustment of parameters and datasets for future development and production.

Existing work

In an effort to tackle a very similar problem of using reinforcement learning in energy systems with BESS control, the authors of [26] provide a simple simulator architecture written in *Python* named *energy_py* to aid in the model learning process. Although it has been written in a flexible modular fashion, allowing for the inclusion of additional reinforcement learning agents, and even different battery system definitions, its reliance on training on one single dataset and producing one single model is a major shortfall for the task at hand, where being able to train on a cluster of multiple households and multiple models is desired. In addition, the author of *energy_py* has made the decision to use their own implementation of DQN and does not provide additional agents to be used at this time, although the infrastructure of the code is such that it is possible for new agents to be implemented in the future.

Another major shortcoming of the *energy_py* simulator is that it is intended for use with environments with a single reward signal. As discussed previously, a major

contribution to the problem of peak-shifting will include the linear combination of multiple different reward signals, and it will be seen later that such a combination requires additional layers of control in the model learning process that *energy_py* simply does not provide.

It is well known that deep reinforcement learning algorithms, including those as simple as DQN, can exhibit drastically varying levels of performance if there are even small mistakes in the implementation [27]. As many research papers do not give the full implementation details, but rather just the theory behind the algorithm itself, it can be difficult to find an implementation guaranteed to perform to the specifications set out by the author of the algorithm. In an effort to remove this additional layer of uncertainty in the learning process, OpenAI have set out to create *true-to-paper* implementations of common reinforcement learning agents through their Baselines project [28], and as such it would be more advantageous to use these implementations for the agents rather than those provided in *energy_py*.

New simulator

Due to the shortcomings faced in the *energy_py* simulator, it was decided that creation of a new simulation development platform from scratch would be the most beneficial path in tackling the peak-shifting problem. Although the simulator needs to be general enough to handle different agents, battery definitions and even reward signals, as previously discussed, creating a new simulator allows some areas to be specifically tailored to the task at hand. That is, the data handling can be tailored to the specific format of the dataset files, and can also be tailored to allow for multiple households to be used in training a single model. Creating the simulator from scratch will also allow for control over the agent interface, and allow for the OpenAI implementations of common reinforcement learning algorithms to be used in place of error-prone implementations.

The new simulator was also created in Python and follows a similar architecture to that provided by *energy_py*; it has been shipped as a fully extensible, modular, standalone Python package. It has been kept as general as possible in order to meet all of the expectations and requirements as discussed before. It should be noted that this simulator is one of the main contributions to this task and it will be shown later in the results section that through the use of this newly created simulator, the peak-shifting problem can be tackled successfully with full control over all autonomous parts in the system. That is, the reinforcement learning algorithm can be chosen with a single adjustable parameter, the definition of the battery can also be changed easily by creating new environment files, and the simulator automatically handles the process of *automatic reward signal shaping*, the second major contribution to this task which

will be discussed shortly. In addition, in an effort to maintain full autonomy over the model learning process and remove the requirement of any human-interaction with the simulator, it also automatically trains and tests an infinite number of models, saving these each to their own unique directories along with useful evaluation metrics such as numeric results and graphs, which aim to verify the model's learning process and even provide visualisations of the final learned policy on actual household data. These are all additional features not provided by the original *energy_py* system.

The following sections will give lower-level implementation details of how each component of the simulator has been created and should be read whilst viewing the accompanying code repository in parallel.

Modularity & directory structure

The simulator has been created in a class-based modular fashion, to allow for various components to be swapped in-and-out, such as for training using different agents, or using different battery environment definitions. Fig. 5.1 presents the top level view of the simulator module's directory structure. All of the model training and testing is performed within the *sim* sub-directory, with the evaluation metrics and resultant models being saved in *results*. *data* stores the household and observation data used in the training process. Finally, *torch_ppo* and *openai_baselines* are modified forks of two popular reinforcement learning algorithm libraries, which are then referenced within the *sim* directory.

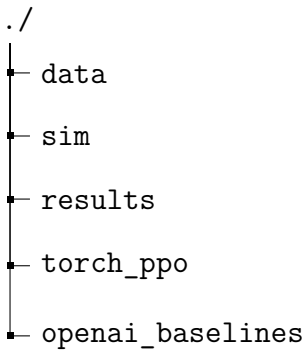


Figure 5.1: Top level view of the simulator module directory structure.

Fig. 5.2 provides a one-level deeper view of the root-level *sim* directory. The *sim/envs* directory is where the different environment files are stored, and *sim/agents* is the directory for the different reinforcement learning agents. This structure allows multiple environments and agents to be defined to provide greater control over the model training process, similar to how *energy_py* is set-up. The *sim/scripts* directory

contains essential modules used by the training & testing components for loading in data, and saving and storing results.

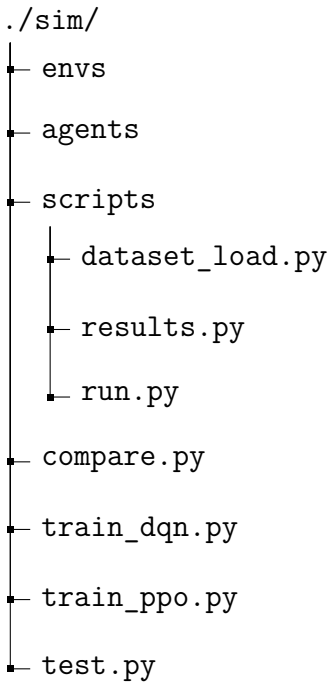


Figure 5.2: Internal view of the *sim* directory in the simulator module. Files not required for the simulator to function have been omitted.

Finally the *train_ppo.py* and *train_dqn.py* files are the executable scripts which invoke the infinite train-test cycle for the corresponding reinforcement learning algorithms. *test.py* is an additional script which allows the testing of an already trained model, and provides an interface of loading in trained models and using them for a production environment. These scripts can be run as follows:

```
python -m sim.train_ppo
```

Snippet 5.1: Example of how to train and test a PPO model with the provided scripts.

```
python -m sim.test --result_dir PPO
```

Snippet 5.2: Example of how to test an already trained model which resides in sim/results/PPO/.

compare.py allows multiple trained models to be tested and evaluated against each other. This produces plots which superimpose the performance of each chosen model on top of each other, along with the baseline model, which will be discussed later. This script will be used to compare the performance of the DQN and PPO agents in the results & evaluation section. This script can be invoked as follows:

```
python -m sim.compare --result_dir DQN PPO
```

Snippet 5.3: Example of how to compare two models named DQN and PPO both of which are saved in the results directory.

Data handling

Fig. 5.3 gives a closeup view of the top-level *data* directory and the structure that the simulator requires to be used. It shows a clear distinction between household data and observatory data. The structure for the household data is such that it does not impose any limit on the amount of data that can be used in training. Placing files here does not automatically make them part of the training process, and the designer has control over which files will be imported during training.

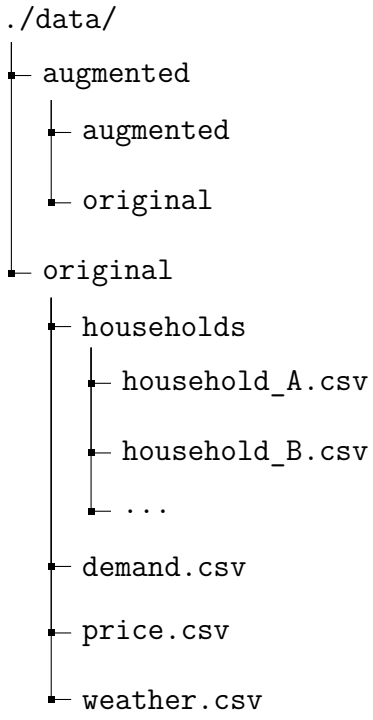


Figure 5.3: Internal view of the *data* directory in the simulator module. Files not required for the simulator to function have been omitted.

The *sim/scripts/dataset_load.py* script handles all of the data loading and augmentation, and is heavily tailored to the format of the data files provided by *Informetis*. This script defines a new class, *DataLoad*. When instantiating this class, the designer is able to pass a list of parameters which provide full control of how the data is handled and augmented.

The designer is able to select explicitly which demand & price plan is used, as discussed previously. The default values for these are *fixed* for the demand plan, and

smart_life_plan for the price plan, and are what will be used henceforth. It is also possible to pass a list of household IDs, for example *household_A*, to refer to the files found in *data/original/households*, and these are the households that will be used in both the training and testing phases. In our case, household A will be the only consumer data used for training and testing, as previously discussed. As the household and observational data are the main features (apart from battery charge) that will be used in the learning phase, an additional parameter has been added to allow full control over which fields are kept and removed. This allows, for example, removal of some of the weather data if it is deemed to be unimportant to learning an optimal policy. It also gives control over which appliance's data is kept in the household data. Within the script itself, additional seasonal features are created and these can also be turned on and off. For example, an additional feature called *weekday* is added, which is given a value of 0 for any datapoint on a weekend, and 1 otherwise. The mindset behind this is to allow the agent to distinguish between energy patterns during weekdays and the weekend, as they are likely to be significantly different. It is very easy to create additional seasonal features, such as distinguishing between each day of the week, or the month, or even a feature to mark the different seasons, with the mindset behind this being to help in forecasting of the weather data. However, due to the dataset size being relatively small and only covering 6 months, it was decided to leave these features out of the training process.

Finally the designer is also able to choose the train/test split at this stage. The *DataLoad* class will automatically separate these two sub-datasets and provide them as necessary to the agent during each phase. As a final level of separation, the class will also take the data and split it up into episodes, in order to provide the environment with only the data available for the current episode of training/testing and restricting access to all other data. The episode length is another parameter which can be modified, with the default being 1440 minutes, corresponding to 1 day. For this task, a train/test split of 70%/30% has been chosen.

When instantiating the *DataLoad* class, it first loads in all of the observational data and selected household datasets. After creating the additional features, it then removes all fields which are not required for the training purpose as chosen by the designer, and also creates two categeories for generated energy and consumed energy. In this light, the photovoltaic data in the household files are added to the generated energy with the rest of the data being moved to consumption. All values are modified as appropriate to keep the units consistent. The data at this point is then saved under *data/augmented/original*. All values are then standardised, by first subtracting the mean and then dividing by that field's standard deviation. This essentially makes each feature have 0 mean and unit variance, a necessary step to aid in the learning process when the

features are of varying scales. This data is then saved to *data/augmented/augmented* in order to prevent future models needing to go through this process again. When the designer selects different features to be used, the *DataLoad* class notices this change and re-augments the data before continuing.

This class also handles the transition between different episodes and timesteps, and is what serves the environment the necessary data which is then further augmented by the environment by adding in additional features such as current battery charge before being passed on again to the agent as the representation of the environment's state. As such, this is arguably the most crucial component of the simulator.

Battery environment

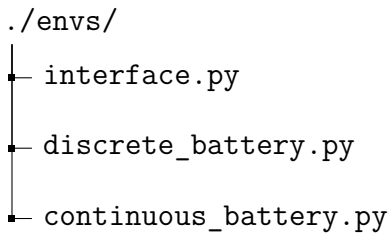


Figure 5.4: Internal view of the *envs* directory in the simulator module. Files not required for the simulator to function have been omitted.

Fig. 5.4 gives an overview of the *sim/envs* directory. The *interface.py* file describes the methods and parameters which are required to be implemented by an environment in the simulator, allowing for multiple environments to be created to serve different purposes. At a high level this can allow for batteries with different properties to be tested, for example modelling the effects of capacity fade or self-discharge on the final learned policy. The two environments that have been created and provided are very simple implementations of a battery without either of these phenomena, with the ability to handle discrete and continuous action spaces with the foresight of being used by the DQN and PPO agents. Again, the designer has full control over the specification of these batteries, which in this simple case, is the ability to adjust the battery's maximum capacity.

The environment class is responsible for augmenting the data given to it by the *DataLoad* component to provide a representation of the current state to the agent. To this end, both of these classes provide the received data, untouched, in addition to the current battery charge. This is an essential addition, as otherwise the agent would be unable to infer how much energy is currently available for use. Another useful addition would be the maximum capacity of the battery, but it was thought that this value should be relatively easily to learn via the penalties imposed by over-charging

the battery and so was omitted. It is however very simple to add in such additional features and this is a perfect example of when a new environment file would be created for testing.

As well as augmenting the state-space representation, the environment class is also responsible for the internal computations related to the battery itself. That is, on each time-step, it needs to update the battery charge based on the amount of energy that was purchased by the agent, and then also provide feedback, through a reward signal, to the agent to aid its learning process. On each timestep, the calculation that takes place is:

$$\begin{aligned} \text{new\_charge} \; = + \; & \text{old\_charge} \; + \; \text{generated\_energy} \\ & - \; \text{consumed\_energy} \; + \; \text{purchased\_energy} \end{aligned} \tag{5.1}$$

It is up to the agent to learn this internal representation as it is not explicitly given this knowledge. To aid this learning process, as discussed before, the environment then provides feedback to the agent based on how much energy it decided to purchase and its subsequent effect on the environment. To this end, the designer is also able to provide the α_i coefficients of the individual reward signals to be used in the overall reward signal. It will be seen during the results stage that this feature, linked with the infinite model-learning ability of the simulator is paramount in training a model which can successfully achieve peak-shifting.

Reinforcement learning agent

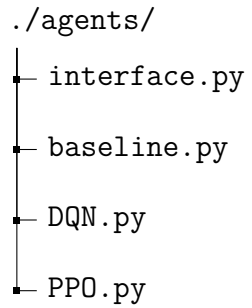


Figure 5.5: Internal view of the *agents* directory in the simulator module. Files not required for the simulator to function have been omitted.

Fig. 5.5 shows the directory structure of *sim/agents*, which is where the different reinforcement learning agents are kept. Similarly to the environment set-up, the *interface.py* file prescribes the required methods and parameters of any new agents that

are to be introduced to the simulator. This is essentially ensuring that each agent has a fully-defined train and test method. Three agents have been created and provided for the simulator, but the structure is such that new agents can be added easily. This means that when new reinforcement learning algorithms are created in the future, once they have been implemented, they can be added in just a few short lines of code by following the interface pattern prescribed within. Again, this is due to the requirement of the simulator being as flexible and general as possible.

The current simulator has been setup to be able to use both DQN and PPO. The DQN implementation is provided by Open AI Baselines [28], stored in the root directory *openai_baselines*. Modifications were made to this implementation to allow for multiple models to be trained and tested in one run, which is not possible with OpenAI's original implementation. The PPO implementation offered by OpenAI is configured specifically to be used with the Atari and MuJoCo environments and was not able to be applied to a more generic environment such as the battery environment being used here. For this reason, *PyTorchRL* was used instead, a popular alternative to OpenAI Baselines for actor-critic based methods, offering well-tested implementations of PPO, ACKTR and AES [30]. The *sim/agents/PPO.py* and *sim/agents/DQN.py* files abstract away the specific implementations of these algorithms, and allows for simple setting of the relevant agent parameters and linking with the rest of the simulator. To this end, it is of no concern exactly how these algorithms are implemented.

Training & testing

The *sim/scripts/run.py* provides the *Run* class which is what brings the different components of the simulator together. This allows the designer to select the environment, environment parameters, reinforcement learning agent and also its parameters, before then training and testing a model. Snippet 5.4 shows the full code required to invoke the infinite model train-test loop, using PPO as the chosen agent, and demonstrates at a high level the flexibility of the created simulator and how simple it is to control.

Before beginning training, the *Run* class instantiates the *Results* class from *sim/scripts/results.py* which provides the results and evaluation metric handling. It creates a new directory in *results/* where the results for the current model will be stored. It updates the current model status to *2*, referring to *training started, but not complete*. Training is then handled by the chosen agent, in this case PPO. Once this has completed, the model status is updated to *3*, or *training complete, testing incomplete*. At this point the model is also saved, as well as multiple graphs. The first of these plots the episode reward against episode number, and is an analogy to the loss-curves usually seen in supervised learning. If the episode reward generally increases throughout, then

```
from sim.scripts.run import Run
from sim.agents.PPO import PPO
from sim.envs.continuous_battery import ContinuousBattery

while True:

    runner = Run(
        env=ContinuousBattery,
        agent=PPO,
        env_params={
            households : [
                "household_A"
            ]
        }, # change on each iteration
        agent_params={
            learning_rate : 1e-3
        } # change on each iteration
    )

    runner.full_cycle() # trains a model then tests it
```

Snippet 5.4: Minimum working example on how to train and test an infinite number of models using the newly created simulator with an example of how to pass various parameters.

the model is assumed to have learned *successfully*, although this does not necessarily mean it has found an optimal policy. A graph is also generated showing the individual decomposed reward/penalty signals against each episode. This provides greater insight into exactly what the agent has learned, and also provides a visual description of the relative magnitudes of each reward.

After this, the testing phase is entered. This loads in the saved model, and the environment is now passed only data in the test set, unseen during training, with the episodes coming in sequential order. This allows the designer to see exactly how the agent performs in a subset of the real time-series data. With the default settings of a 70%/30% train/test split, this equates to the model being tested on about 2 months of data. This model is tested individually for each household, and generates a text-based results file as well as a multitude of graphs. The results file logs the amount of energy bought over the test data, and also provides data for the average monthly bill, which allows the designer to easily assess whether or not the model has learned to reduce energy consumption and utility bills. In addition, it also logs how many times the constraints were ignored. That is, how many times did the agent buy less energy than was required, or how many times did it buy more energy than was available, and finally how many times did it try to store more energy than possible by the battery. These constraints cannot be enforced directly like in classic optimisation methods, and instead need to be learned by the agent via the reward signal. The first graph that is

generated shows the average daily requested energy profile throughout the test set data, and it is this where we can see if peak shifting has actually occurred. If it has occurred, one would expect the requested energy to be a relatively flat line in comparison to the original data, and far below the available demand line which is also shown on the graph for comparison. The next graph plots this same data against the price of energy, to see if the agent has learned to buy energy at times when it is cheaper or not. The final graph shows the charge of the battery throughout the day, which is indirectly a visualisation of the learned policy. The graphs and results files are superimposed with the results from a baseline, which will be introduced later. This comparison allows the designer to verify that the reinforcement learning agent has actually learned a useful, non-naive policy.

Once this process has been completed, the model status is set to *4*, or *training and testing completed*. Every time a new model is trained, all previous model's statuses are checked, and anything below *3* (fully trained) is deleted to free-up available memory for more models to be trained.

Automatic penalty shaping

After training a few models with the simulator, it became apparent that the individual rewards were of drastically varying magnitudes and this prevented any learning from taking place. Instead, the agent only focused on maximising the reward of highest magnitude as the other reward signals did not provide much benefit to the overal return. This is a major problem to take into account when attempting to combine various reward signals. Without an appropriate combination of each individual signal, the model may not try to learn to peak-shift, or even worse, some of the essential constraints on the problem may be ignored. For this reason, the coefficients of each individual reward signal was exposed as an environment parameter, and hence a parameter that can be set by the designer.

In problems where the reward signal profile is known in advance, the standard method of ensuring that the individual reward signals are of similar magnitude is by first standardising each before adding them together. This essentially results in all signals having an equivalent mean of 0 and variance of 1. As such knowledge of the reward signal profile is not known in this environment, the mean and standard deviation of the individual penalties are also not known in advance, and would have to be updated on each timestep as new values are yielded by the environment. This method was attempted, but the estimates of mean and standard deviation varied too much over time and did not help the problem at all.

A novel technique, termed *automatic penalty shaping* has been proposed to address

this problem. In the *sim/train_ppo.py* and *sim/train_dqn.py* code, after each model is trained, the history of each individual reward signal is observed. Based on these observations, the signal coefficients are adjusted with the goal of making the magnitudes of each signal as similar to each other as possible, forcing the agent to treat each reward as equally as each other. The simple method used in these scripts to scale the coefficients first finds the average value of each penalty over all episodes during training. It then takes the coefficients of each signal used for that model, and scales these such that an equivalent scaling of these averages coincide with each other. Essentially, it scales the coefficients in such a way to direct the resultant individual reward signals to be as close to each other as possible. It then scales these coefficients down such that the maximum coefficient is equal to 1. This is a very simple scaling method, and ignores the profile of each signal throughout the training process, instead using an average value which could be heavily affected by large penalties near the start of training. However, it was found that this simple scaling method was more than sufficient in obtaining strong results. The downside to this method is that it requires multiple models to be trained, with each model improving on the last, and hence increases overall training times. This is, however, the main reason behind the choice of allowing an infinite number of models to be trained automatically in the simulator.

The full simulator, minus existing work from the two forked repositories, consists of approximately 5,000 lines of code, and is one of the main contributions made during this project [1].

[1]Please contact *Informetis, Tokyo* directly if you wish to request access to the code repository.

Results & evaluation

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 overfit to the data used for training but instead generalises 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.

Baseline agent

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 timestep, 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 Eqn. 6.1, taking into account that there is no battery and hence charge is removed from the equation:

$$\text{required\_energy} = \text{consumed\_energy} - \text{generated\_energy} \tag{6.1}$$

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.

Fig. 6.1 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 visualise the variance in the data. As the actions that the agent takes is the decision of how much energy to buy, this graph is a direct visualisation of the agent's policy.

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 timestep, and hence does not take into account the price or demand (limit) of energy at all.

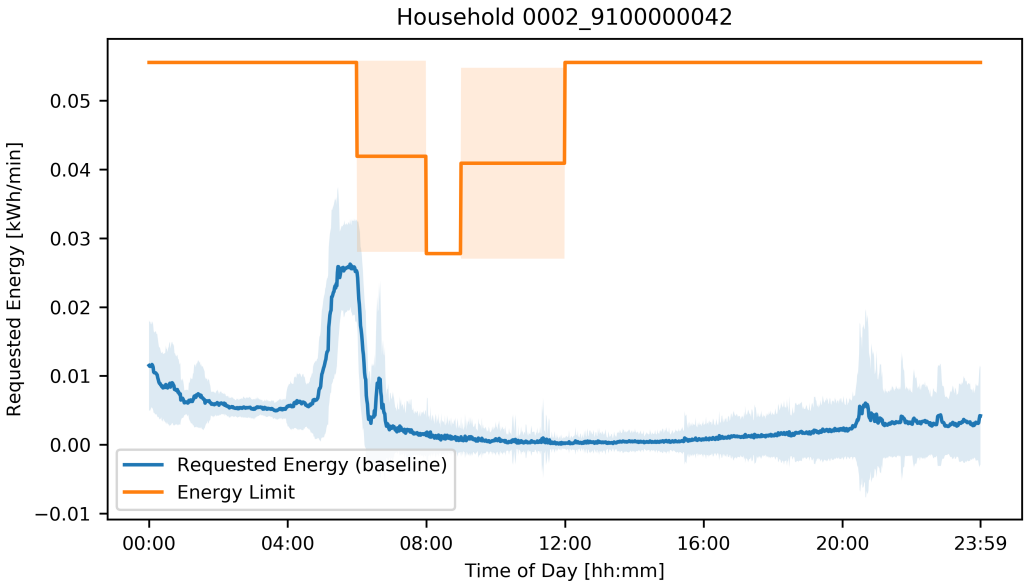


Figure 6.1: Daily-averaged requested energy profile for the baseline model on household A.

Fig. 6.2 shows the same requested energy profile but now graphed against the price of energy and normalised. 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.

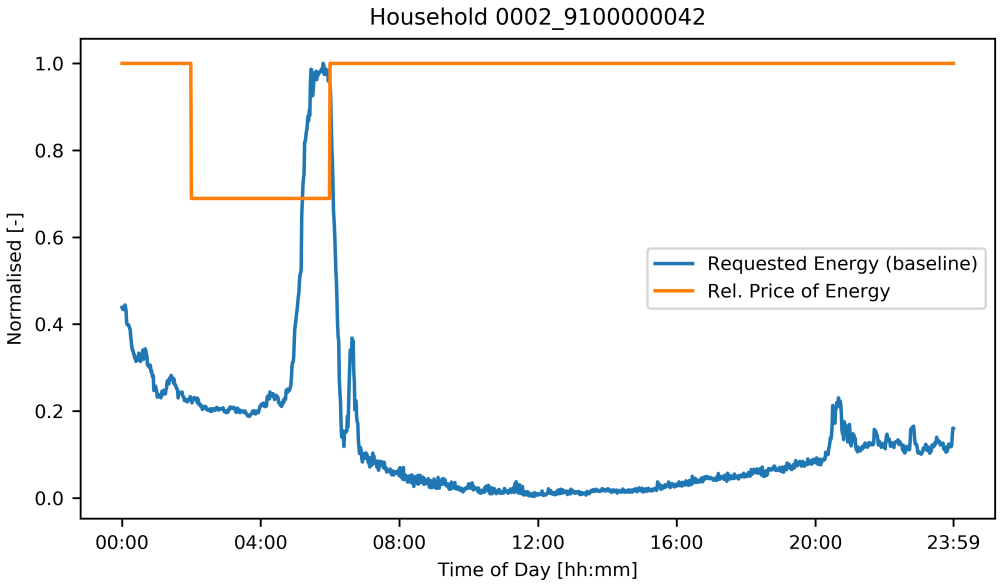


Figure 6.2: Daily-averaged normalised pricing data for the baseline model on household A.

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.

Tab. 6.1 gives the results for the baseline agent on the test data. The *abovecapacity* 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 timestep. 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 timestep 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.

Parameter	Result
Error *abovecapacity*	0 (0.00%)
Error *underpurchase*	0 (0.00%)
Error *abovelimit*	122 (0.15%)
Monthly bill	3517.79 JPY
Total energy purchased	292.51 kW

Table 6.1: Numerical results for the baseline agent.

DQN agent

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 were 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.

During the initial training stages, it was found that the model learned well using the parameters given in Tab. A.1 and so these were kept constant throughout all models. The discretisation of the action-space was varied in the initial models, and $[0, 0.005, 0.01, 0.015, 0.02, 0.03, 0.04]$ 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 Fig. 6.1, it was seen that the maximum required energy at a single timestep hovers around the 0.03-0.04 range, and so this was chosen as the highest values in the discretisation. Also, as most timesteps 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 $[64]$, $[64, 64]$ and $[64, 128, 64]$, 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 judgement 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.

Tab. 6.2 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 *abovecapacity* 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 timesteps 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 timesteps, 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 discretisation 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 discretisation, with some actions inbetween 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 timesteps, 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 optimisation problem.

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

Table 6.2: Results for the best DQN model compared against the baseline.

Tab. A.2 gives the parameters of this model, including the coefficients for each of the individual reward signals as defined previously.

Fig. 6.3 visualises 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.

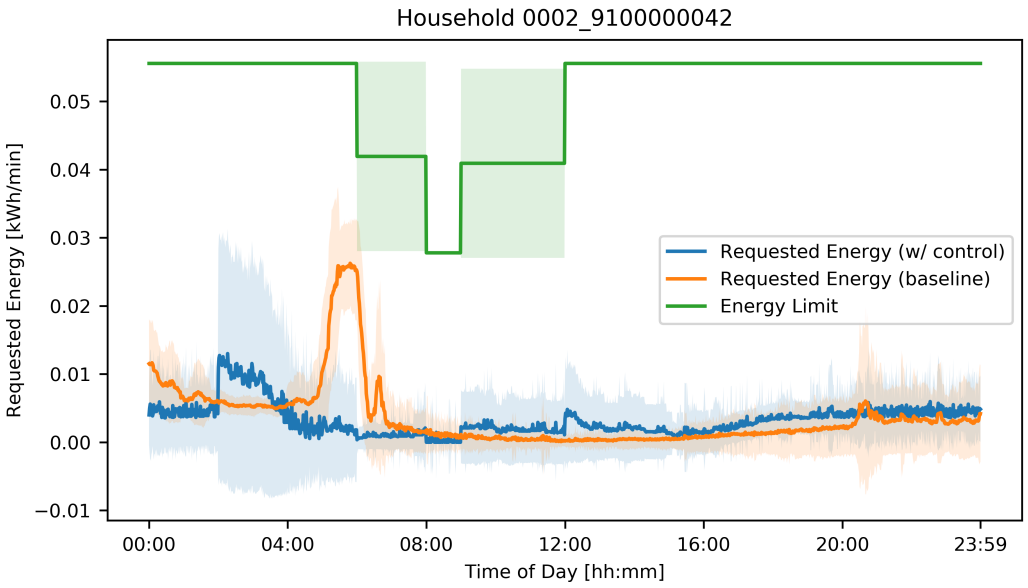


Figure 6.3: Daily-averaged requested energy profile for the best DQN model on household A.

There is a slight peak in the purchase of energy between 2am and 6am. Observing Fig. 6.4, which shows the daily-averaged normalised 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.

Fig. 6.5 shows the daily-averaged charging profile of the battery over the test data

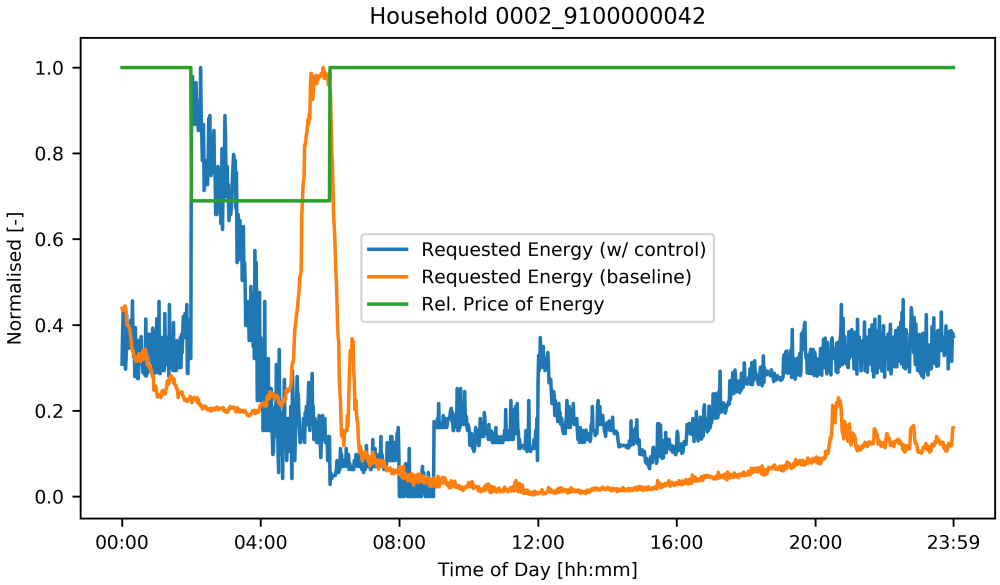


Figure 6.4: Daily-averaged normalised pricing data for the DQN model on household A.

when using this model, an indirect method of visualising 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 realised this, and is purchasing the energy well in advance. This is corrobated 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 topup the energy in advance of the high consumption period.

Fig. 6.6 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 Fig. 3.4. 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

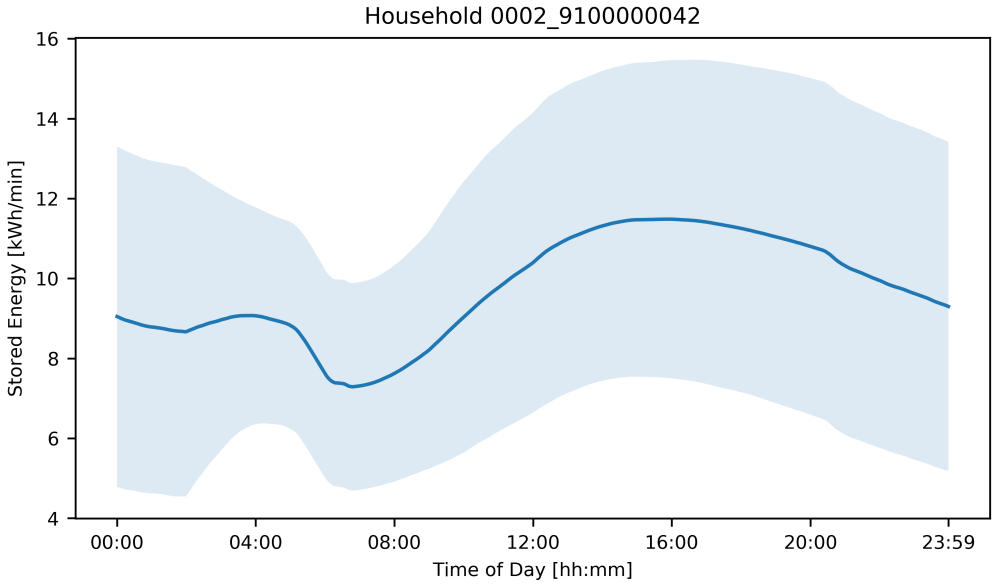


Figure 6.5: Daily-averaged normalised charging profile for the DQN model on household A.

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 overfitting that may have occurred between that and the end episode. This is all handled automatically by the simulator.

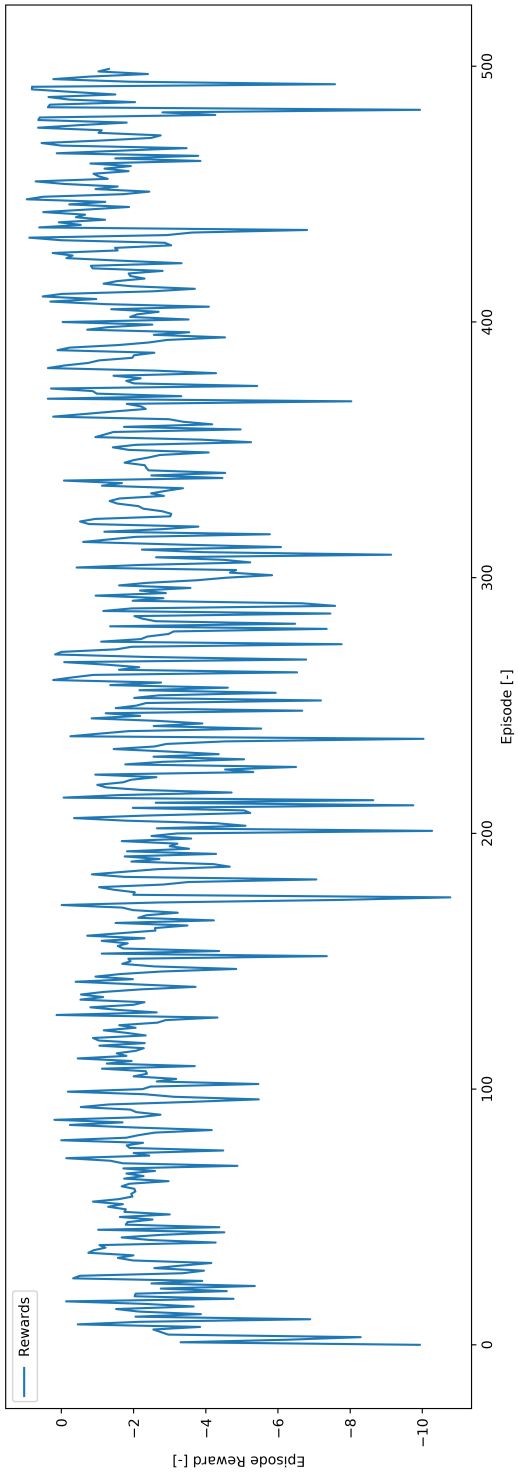
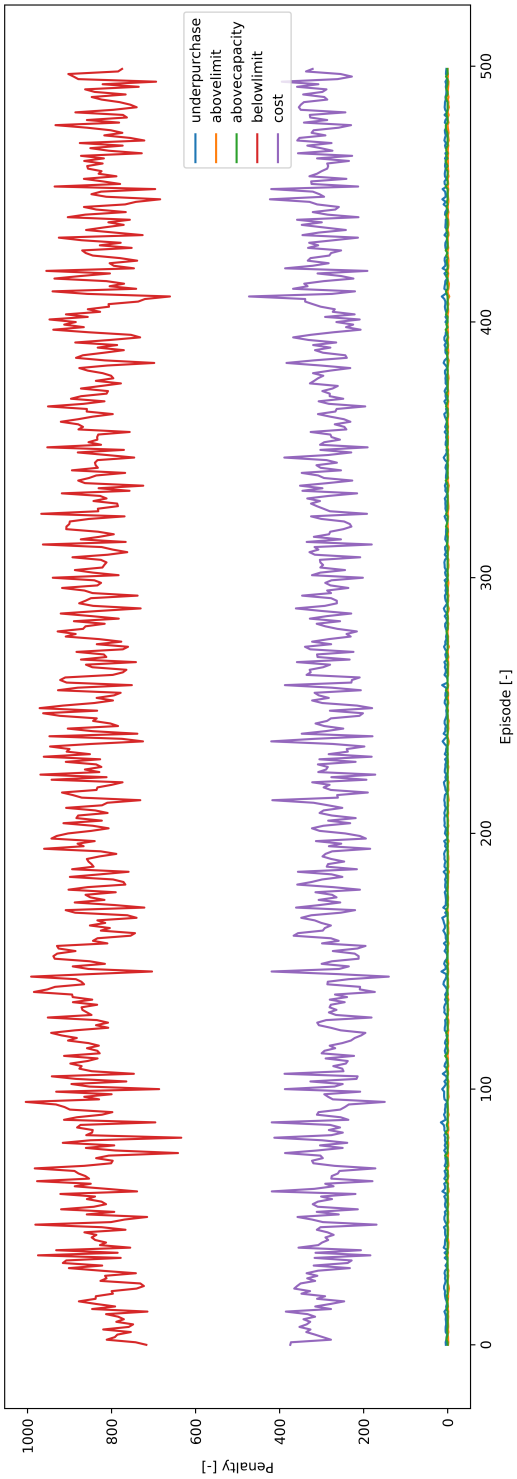


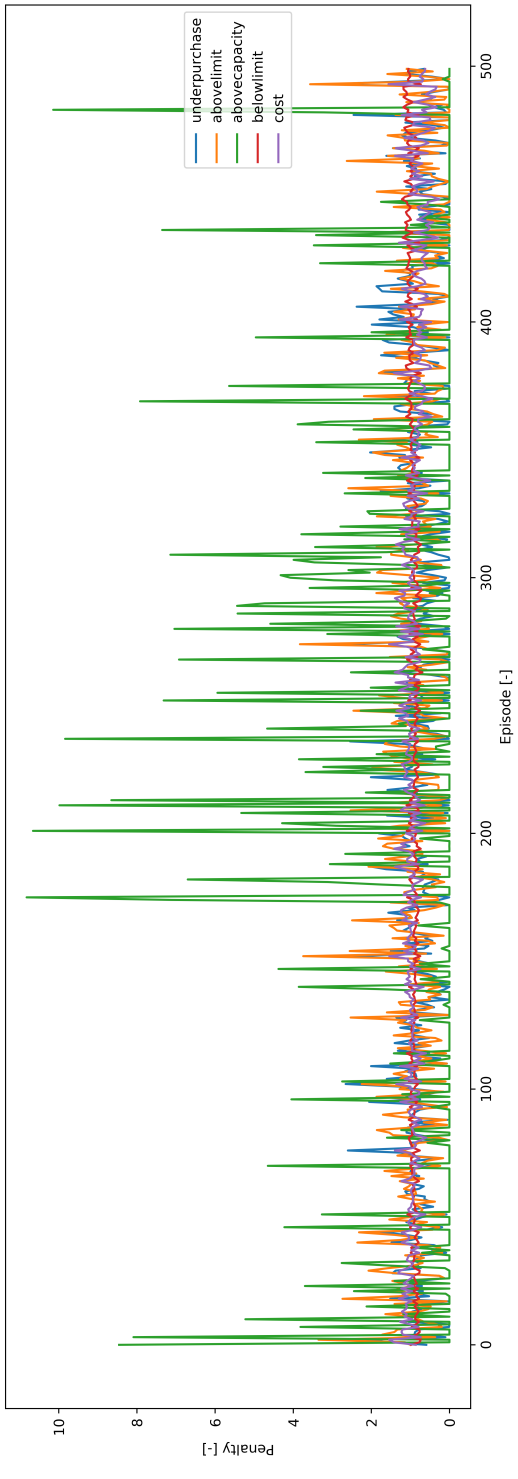
Figure 6.6: Total reward per episode during the training period for the best DQN model.

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 optimised reward signal, which was shaped using the automatic penalty shaping feature discussed previously. Fig. 6.7a 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 much higher than the rest. Such a reward signal would bias the agent into placing more emphasis on learning to maximise this reward whislt ignoring the rest.

Fig. 6.7b 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 *abovecapacity* penalty. This fluctuation corresponds to the large *abovecapacity* 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.



(a) Initial model (not optimal).



(b) Final model (best).

Figure 6.7: Disaggregated reward signals for two different DQN models.

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.

PPO agent

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 discretisation. 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 $[-1, 1]$ and then scaled appropriately to $[0, 0.03]$, 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 Fig. 6.1.

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. These are given in Tab. A.3 and are the parameters which are used throughout all of the saved PPO models. All models were trained using episodes of 1 day (1440 timesteps). The agents did not seem to learn efficiently when using an episode length of 1 week with the chosen parameters, and it was decided that finding parameters for a 1 week episode model would be unnecessary additional extra work given the 1 day episode models were training easily and more quickly.

75 models using a PPO agent were trained, with each successive model learning from the last via the automatic penalty shaping feature. Tab. A.4 gives the coefficients of the individual reward signals as optimised by the automatic penalty shaping feature for the best model found. These were the only parameters that were changed on each run, and again, this was handled automatically by the simulator.

The results for this model have been compared with the baseline in Tab. 6.3. 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 underpurchaches 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.

Parameter	Baseline Agent	PPO Agent	% Difference
Error *abovecapacity*	-	5928 (7.48%)	-
Error *underpurchase*	0 (0.00%)	0 (0.00%)	+0%
Error *abovelimit*	122 (0.15%)	0 (0.00%)	-100%
Monthly bill	3517.79 JPY	2777.75 JPY	-21%
Total energy purchased	292.51 kWh	226.49 kWh	-23%

Table 6.3: Results for the best PPO model compared against the baseline.

Fig. 6.8 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.

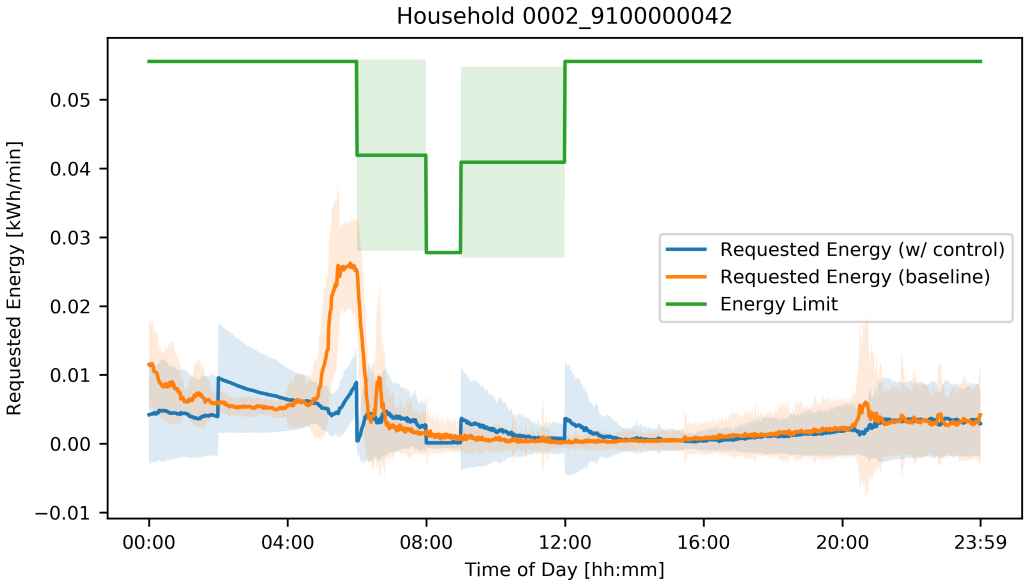


Figure 6.8: Daily-averaged requested energy profile for the best PPO model on household A.

Fig. 6.9 shows this data against the price of energy after being normalised. 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 minimise the overall energy bill.

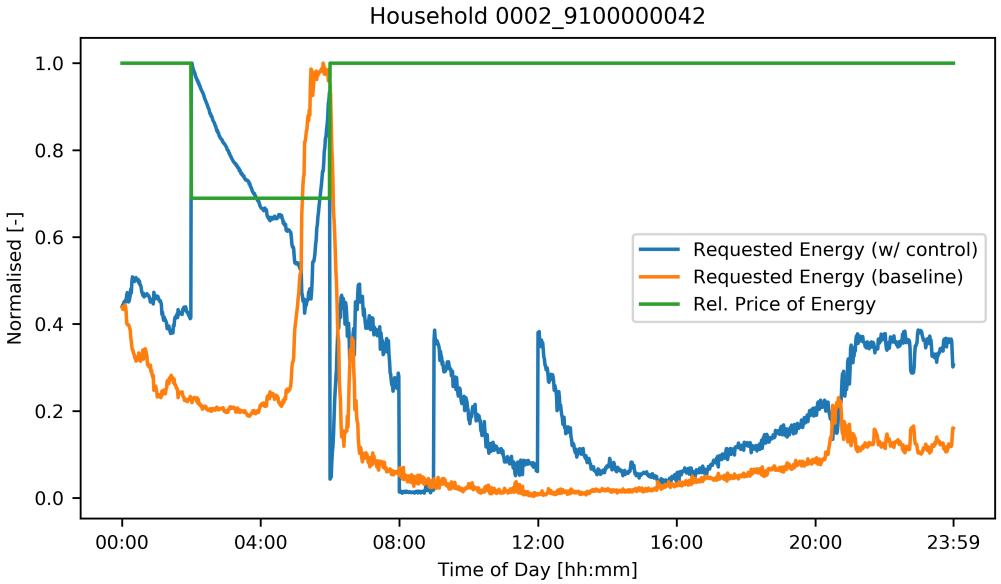


Figure 6.9: Daily-averaged normalised pricing data for the PPO model on household A.

Fig. 6.10 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.

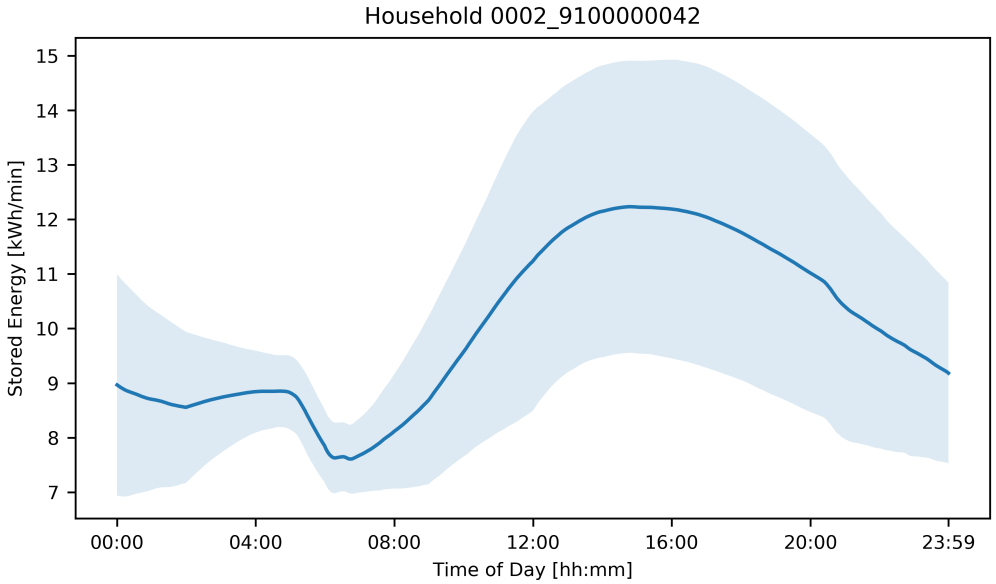


Figure 6.10: Daily-averaged normalised charging profile for the PPO model on household A.

Fig. 6.11 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.

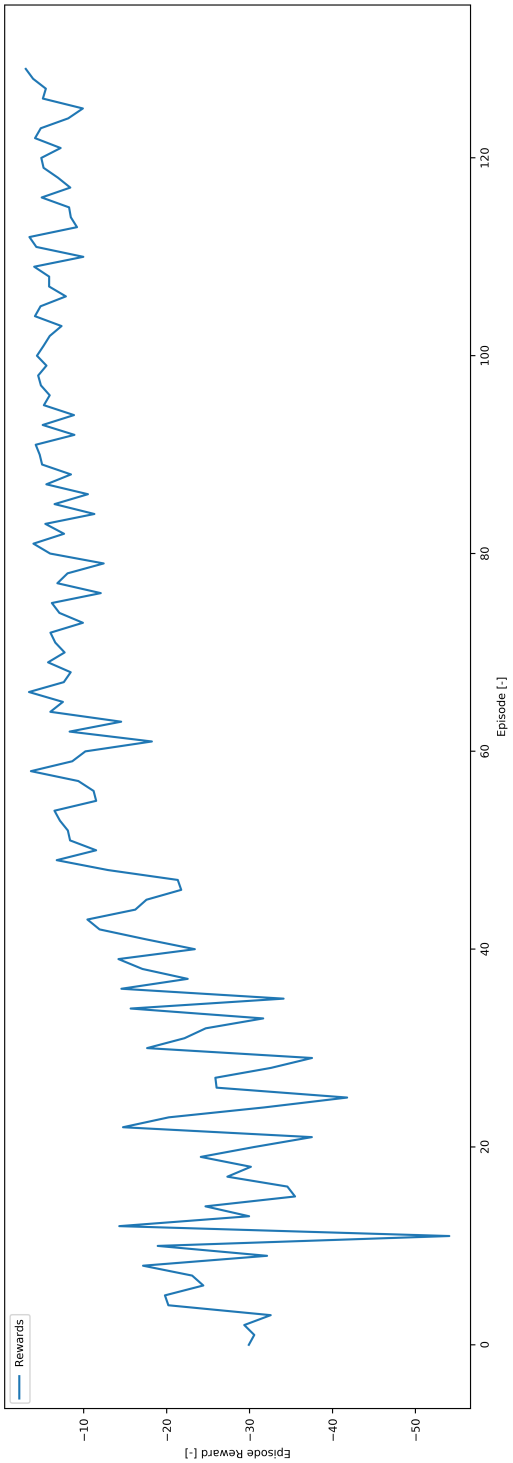
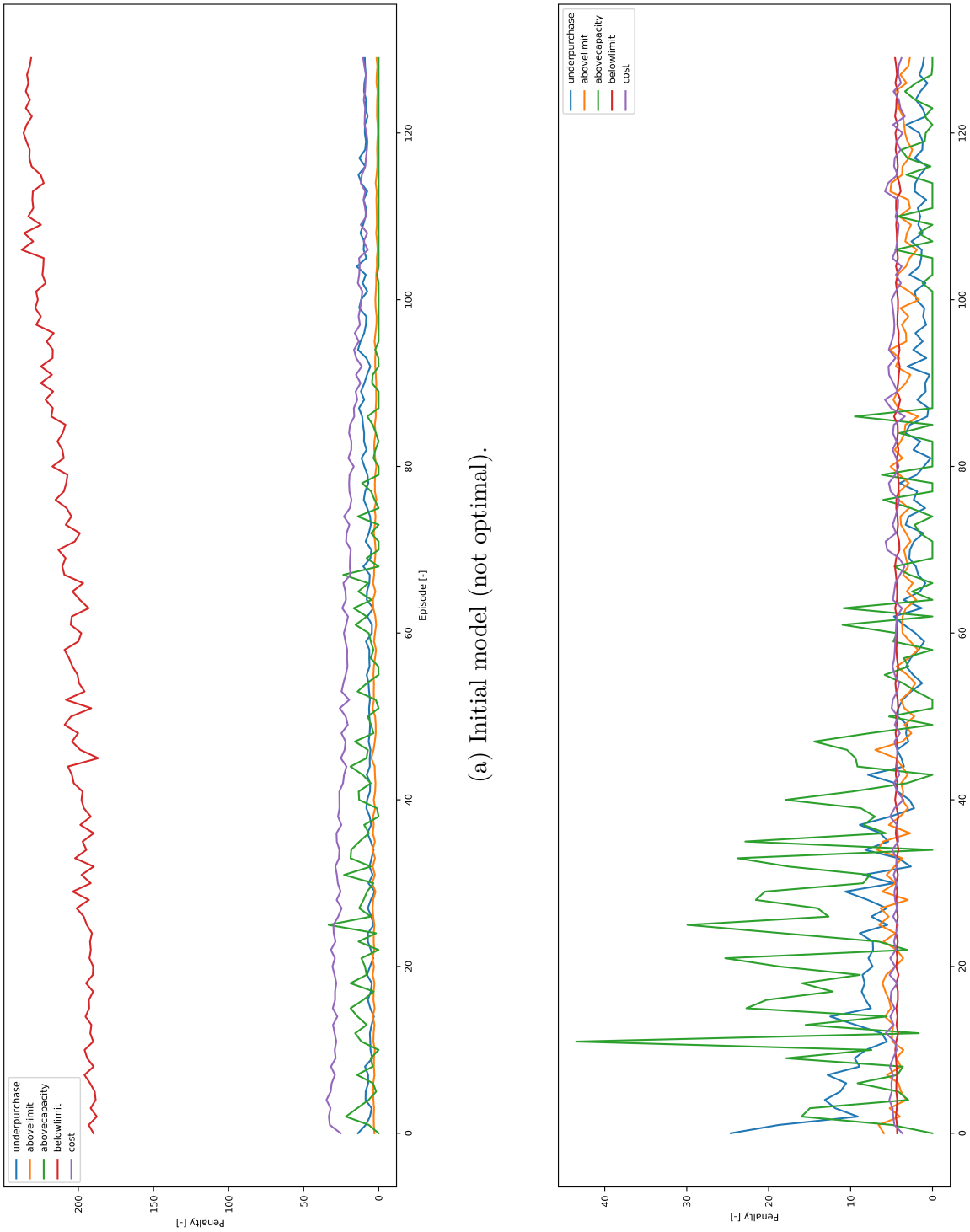


Figure 6.11: Total reward per episode during the training period for the best PPO model.

Fig. 6.12a and 6.12b, 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.



(a) Initial model (not optimal).

(b) Final model (best).

Figure 6.12: Disaggregated reward signals for two different PPO models.

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.

Reinforcement learning agent comparison

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.

Fig. 6.13 shows the requested energy profile, the direct policy, of both the best PPO and DQN agents against the baseline. Fig. 6.14 does the same but for the charge profile which is an indirect visualisation 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 the majority of 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 hypothesised that the variation in results between DQN and PPO would disappear if the discretisation 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.

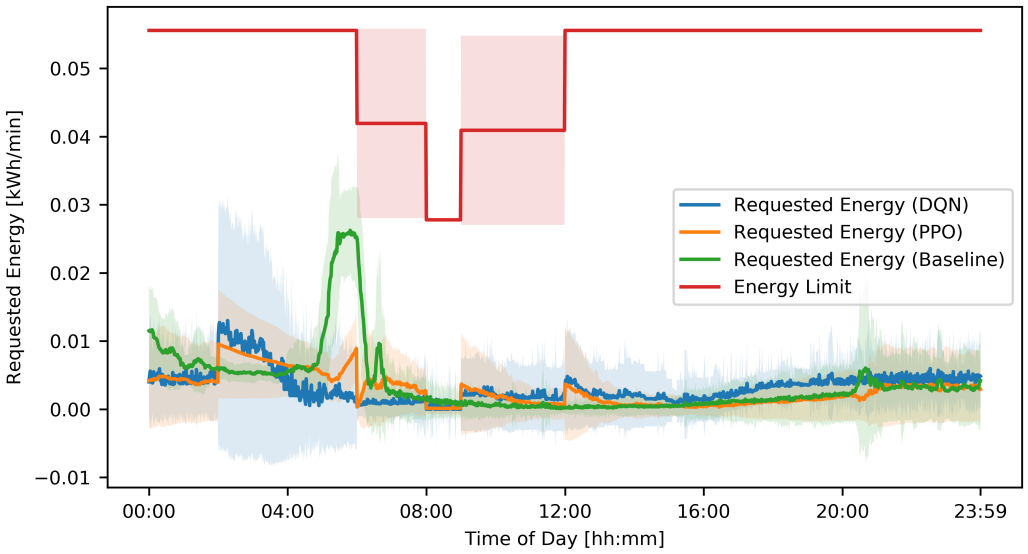


Figure 6.13: Daily-averaged requested energy profile for the best PPO and DQN model on household A.

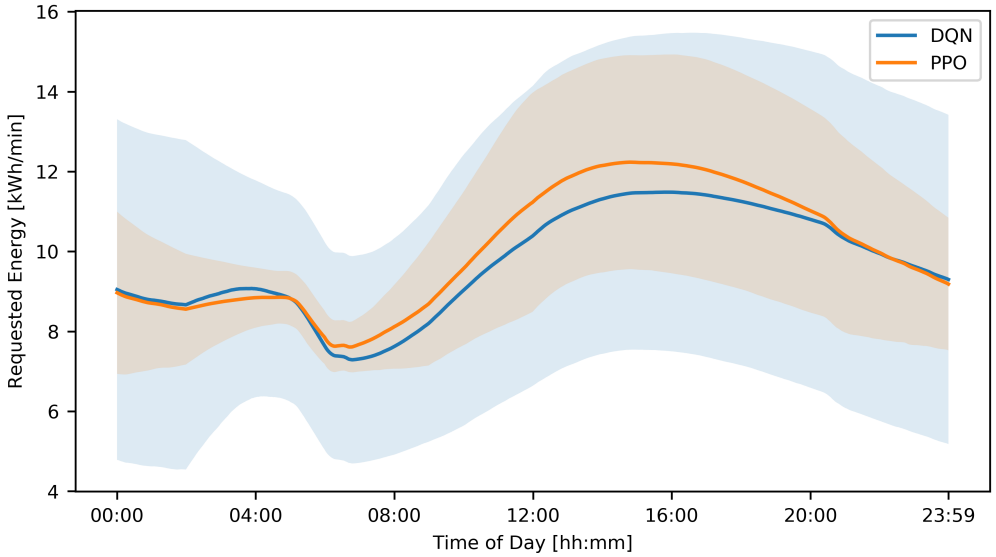


Figure 6.14: Daily-averaged requested energy profile for the best PPO and DQN model on household A.

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 BESS 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 peaker plants to provide additional energy during peak periods, lowering carbon emissions and also energy prices for the consumer. This requires accurate forecasting of the energy market, grid demand, and consumer energy consumption & generation. 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 tarriffs for energy for the consumer. Secondary aims of directly reducing the energy consumption and utility bills were also sought, making this a multi-objective optimisation problem.

In order to generate these models, *Informetis* provided observatory and household consumption & generation data over the period of the first 6 months of 2017. This data was used, in conjunction with the created simulator which performs the full training and testing phases of the models, to find an optimal policy using the DQN and PPO reinforcement learning algorithms. To provide a comparison for the results and enable a meaningful evaluation of the models, a strong baseline agent which has full knowledge of its energy requirement profile was created. Both the strongest DQN and PPO agents significantly outperformed this baseline, with the PPO agent exhibiting near-perfect results. The final PPO model was 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. The only downside to this model is that it occassionaly attempted to charge the battery above its maximum capacity, inferring that it may not have accurately learned the battery's maximum capacity from the data available to it.

Significant contributions

There were two significant contributions made, without which, these results would not have been possible.

Simulator

The first of these is the creation of an fully enclosed end-to-end simulator. This simulator is a completely modular, adaptable, standalone Python module that handles the data, environment definitions, reinforcement learning agents, results, evaluation and also saving of all results for future analysis. This includes running an infinite number of models sequentially, with each one improving upon the previous. This simulator has been created with future development in mind, allowing for new battery definitions and even new reinforcement learning agents to easily be added and swapped in. To this end, two additional continuous action-space reinforcement learning algorithms, *A3C* and *ACKTR* are already implemented and ready to be used if necessary. The simulator offers a wealth of flexibility to the designer, allowing them to hand-pick all parameters of the environment and agent, as well as making further changes to the reward signal as this was found to be a crucial element to achieving success.

Finally, although not important for the model training purpose, additional methods within the simulator were created to provide the designer with the ability to compare multiple models to see how their policies differ, and also to re-test models. The comparison feature was used in the previous section to compare the learned policies of the DQN and PPO models, although there is no limit as to how many models can be compared using this method. Many other non-critical methods have been provided to allow visualisation of the data, and also to provide statistics on household data, which can be used to assess whether the data is of sufficient quality for training, and it was through using this that the decision to ignore households B-D from the training process was made.

Automatic penalty shaping

The final significant contribution made during this project was that of *automatic penalty shaping*. Due to the multiple objectives that were necessary to achieve, such as ensuring physical constraints to the system were adhered to, as well as achieving peak-shifting and reducing costs and energy consumption, it proved difficult to construct a single reward signal from which the agent was to learn.

In order for the agent to give each individual objective equal attention, they were combined in a linear relationship with the coefficients of each signal being modifiable by

both the designer and the simulator itself. The automatic penalty shaping feature, after training each model, would automatically adjust the coefficients of each signal based on the previous results in an effort to allow the next model to more appropriately attend to reward signals it may have ignored. This automatic adjustment and re-shaping of the reward signal, as shown in the previous section, proved to be fundamental to the success of the learned models.

Future work

Although the results presented are almost perfect and provide the necessary verification that reinforcement learning is more than suitable for this task, there are still some areas which could be given additional attention.

Firstly, the maximum capacity of the battery could be exposed as an additional feature in the state space, which would mean the agent does not need to learn this value by itself. This change is suggested based on the relatively large number of times the final model attempts to overcharge the battery, indicating it has not learned the battery's capacity accurately. This is an easy change to be made in the simulator as it provides the necessary framework for additional features to be augmented in the state-space.

Next, given more household data, each household could be clustered into a few different energy profiles. These profiles would indicate households which use energy significantly differently to each other, such as some that use energy during the night as opposed to in the morning, or households with a stay-at-home parent indicating high electricity usage throughout the day. A model could then be trained for each cluster of households, providing a model which generalises well to new households once they have been assigned to a specific cluster. The simulator is already setup for handling multiple households during training and testing, and this was only omitted in this study based on lack of available data.

Also, if more than six months of data were made available, it could prove beneficial to add in additional seasonal-based features to the state-space. This would provide the agent with more information in forecasting the weather, indirectly improving its forecasting of the generated energy from photovoltaic solar panels.

More complicated battery environments could be incorporated, taking into account the self-discharge and capacity fade phenomena that would surely modify the final optimal policy.

A final change that would be suggested, is to include the ability to sell back energy to the grid to further reduce the monthly utility bill. This was omitted from this model based on the fact that different countries operate different guidelines. For example, in

the USA it is possible to sell back all energy to the grid, whereas in Japan it is only possible to sell energy which has been generated on-site.

Bibliography

[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/1xeUDVGWGUUv1-ccUMAZHJLej2C7aAFWY/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/decentralised-energy-experts/What-is-decentralised-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://adgefficiency.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] なっとく！再生可能エネルギー　固定価格買取制度; 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:13125602 [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:150906461 [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:150902971 [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:150205477 [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:170706347 [cs]. 2017 Jul;ArXiv: 1707.06347. Available from: http://arxiv.org/abs/1707.06347.

[25] TEPCO. 料金プラン一覧（ご家庭のお客さま） はじまる！電力自由化　東京電力エナジーパートナー; 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:151105952 [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.

Model parameters

DQN

Parameter	Value
Learning rate	1e-3
Buffer size	50000
Exploration time	15%
Final exploration	1%
Batch size	32
Discount factor γ	1.0
Target network update frequency	500
Prioritised replay α	0.6
Prioritised replay β_0	0.4
Prioritised replay ϵ	1e-6

Table A.1: Fixed parameters for the DQN agent over all models. The parameters which have not been introduced are specific to this implementation of DQN and are explained in [28].

Prioritised replay refers to an improvement to the original DQN algorithm, which samples *important* episodes more frequently in the learning process, rather than at random, improving the learning efficiency [29].

Parameter	Value
Penalty underpurchase coefficient	1.59e-1
Penalty abovelimit coefficient	-5.38e-1
Penalty abovecapacity coefficient	1.00
Penalty belowlimit coefficient	9.32e-4
Penalty cost coefficient	1.73e-3
Episodes	500
Episode length	1440 (1 day)
Network architecture	[64, 128, 64]

Table A.2: Parameters for the resultant best DQN model.

PPO

Parameter	Value
Learning rate	7e-5
Number of processes	8
Number of steps	128
Clip parameter	0.1
PPO epoch	4
Number of mini batches	4
Value loss coefficient	1
Entropy coefficient	0.01
Maximum gradient norm	0.5
GAE	Yes
Discount factor γ	1.0
ϵ	1e-5
τ	0.5

Table A.3: Fixed parameters for the PPO agent over all models. The parameters which have not been introduced are specific to this implementation of PPO, and are described in [30].

Parameter	Value
Penalty underpurchase coefficient	0.220
Penalty abovelimit coefficient	0.887
Penalty abovecapacity coefficient	1.00
Penalty belowlimit coefficient	4.24e-3
Penalty cost coefficient	9.24e-3

Table A.4: Coefficients for each of the individual reward signals for the resultant best PPO model.

Mathematical derivations

Policy gradient

The $\boldsymbol{\theta}$ term in $\nabla_{\boldsymbol{\theta}}$ and $\pi_{\boldsymbol{\theta}}$ have been omitted for brevity.

$$
\begin{aligned}
\nabla v_\pi(s) &= \nabla \left[\sum_a \pi(a|s) q_\pi(s,a) \right] \quad \forall s \in \mathcal{S} \\
&= \sum_a \left[q_\pi(s,a) \nabla \pi(a|s) + \pi(a|s) \nabla q_\pi(s,a) \right] \text{ (product rule)} \\
&= \sum_a \left[q_\pi(s,a) \nabla \pi(a|s) + \pi(a|s) \nabla \sum_{s',r} p(s',r|s,a)(r + v_\pi(s')) \right] \\
&= \sum_a \left[q_\pi(s,a) \nabla \pi(a|s) + \pi(a|s) \sum_{s'} p(s'|s,a) \nabla v_\pi(s') \right] \\
&= \sum_a \left[q_\pi(s,a) \nabla \pi(a|s) + \pi(a|s) \sum_{s'} p(s'|s,a) \cdot \right. \\
&\quad \left. \sum_{a'} \left[q_\pi(s',a') \nabla \pi(a'|s') + \pi(a'|s') \sum_{s''} p(s''|s',a') \nabla v_\pi(s'') \right] \right] \text{ (unrolling)} \\
&= \sum_{x \in \mathcal{S}} \sum_{k=0}^{\infty} \Pr(s \to x, k, \pi) \sum_a \nabla \pi(a|x) q_\pi(x,a) \qquad \text{(B.1)}
\end{aligned}
$$

The final step is achieved after unrolling to infinity. $\Pr(s \to x, k, \pi)$ is the probability of transitioning from state s to state x in k steps under the policy π [2]. Following from this derivation, the policy gradient theorem then states:

$$
\begin{aligned}
\nabla J(\boldsymbol{\theta}) &= \nabla v_\pi(s_0) \\
&= \sum_s \left(\sum_{k=0}^{\infty} \Pr(s_0 \to x, k, \pi) \right) \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&= \sum_s \eta(s) \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&= \left(\frac{\sum_{s'} \eta(s')}{\sum_{s'} \eta(s')} \right) \sum_s \eta(s) \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&= \sum_{s'} \eta(s') \sum_s \frac{\eta(s)}{\sum_{s'} \eta(s')} \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&= \sum_{s'} \eta(s') \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&\propto \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s,a) \text{ QED.}
\end{aligned} \tag{B.2}
$$

REINFORCE

$$
\begin{aligned}
\nabla J(\boldsymbol{\theta}) &= \mathbb{E}_\pi \left[\sum_a \pi(a|S_t, \boldsymbol{\theta}) q_\pi(S_t, a) \frac{\nabla \pi(a|S_t, \boldsymbol{\theta})}{\pi(a|S_t, \boldsymbol{\theta})} \right] \\
&= \mathbb{E}_\pi \left[q_\pi(S_t, A_t) \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})} \right] \\
&= \mathbb{E}_\pi \left[G_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})} \right] \\
&= \mathbb{E}_\pi \left[G_t \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta}) \right]
\end{aligned} \tag{B.3}
$$