Home Energy Management System

A Machine Learning Approach

MSc Thesis

Karthikeyan Deivamani

Delft University of Technology

TU Delft

Home Energy Management System

A Machine Learning Approach

by

Karthikeyan Deivamani

Student Number : 5330645

Thesis committee:	Dr. Pavol Bauer, Chair
	Dr. Aditya Shekhar, Supervisor
	Dr. Jochen Cremer
Advisor:	Farshid Norouzi, Daily Supervisor
Project Duration:	November 2022 - October 2023
Faculty:	Faculty of Electrical Engineering, Mathematics and Computer Science
Research group:	DC Systems, Energy Conversion and Storage (DCES)

TU Delft

Acknowledgements

I started working on this graduation project in Fall 2022 and it took me almost a year to successfully come to the end. Although the road to graduation was challenging, I received constructive guidance from my supervisors Dr. Aditya and Farshid which made it a smoother and a rewarding journey. As I am at the end of my academic journey, words can't express how grateful I am to my parents for providing this point opportunity and all the others leading up to this along with their unwavering support. I am grateful to my friends for picking me up whenever my spirits was low. Ajay, Sappa, Kiran, Ravi to name a few people and my sister Rajeswari who were always available for a call. A special thanks to Lucy whose constant support has motivated me throughout the journey.

Karthikeyan Deivamani
Delft, October 2023

Abstract

The increasing adoption of renewable energy sources, particularly photovoltaic (PV) systems in residential sectors has raised important energy balancing challenges due to the intermittent nature of energy generation. To address these challenges and prioritize cost savings for residential consumers, this research investigates the integration of battery energy storage systems (BESS) and dynamic pricing strategies through an intelligent energy management system (EMS). Given the stochastic nature of PV generation, market prices, and load profile it is still challenging to achieve optimal control. Therefore Reinforcement Learning (RL)-based EMS is proposed in this research to make real-time optimal control decisions. RL is a machine learning approach where an agent learns to make decisions by interacting with an environment to maximize cumulative rewards. In this study, a deep deterministic policy gradient (DDPG) RL architecture is chosen due to its capability to handle continuous action spaces. In addition, deep learning-based models are employed to forecast uncontrollable load, PV generation and market prices for the integration into the EMS for which Bi-directional LSTM (Long Short Term Memory) was found to be the most accurate for all three uncertain variables. The DDPG algorithm is trained with data from a single household from the Lucerne region, Switzerland for 30 days and tested for a week. The results showed that compared to a deterministic rule-based approach the RL-based EMS increased cost savings for the end consumer by 14.2% but reduced the benefits for the grid operator to alleviate grid congestion quantified in terms of load factor, peak power consumption and ramping. Further work could be undertaken in testing the model on more extensive data and finding the best trade-off between customer and grid operator benefits.

Nomenclature

Abbreviations

ARIMA	Autoregressive Integrated Moving Average
BESS	Battery Energy Storage System
BiLSTM	Bi-directional Long Short Term Memory
CNN	Convolutional Neural Network
DDPG	Deep Deterministic Policy Gradient
DP	Dynamic Programming
DQN	Deep Q-Learning
EMS	Energy Management System
GRU	Gated Recurrent Units
HEMS	Home Energy Management System
LSTM	Long Short Term Memory
MADDPG	Multi-Agent Deep Deterministic Policy Gradient
MAE	Mean Absolute Error
MDP	Markov Decision Processes
MLP	Multilayer Perceptron
MPC	Model Predictive Control
OU	Ornstein–Uhlenbeck
PV	Photovoltaics
RL	Reinforcement Learning
RBC	Rule-Based Control
RNN	Recurrent Neural Network
RMSE	Root Mean Squared Error
RTP	Real-Time Price
SARIMA	Seasonal Autoregressive Integrated Moving Average
SOC	State of charge
STFL	Short-Term Load Forecasting
TD	Temporal Differencing
TOU	Time of Use

Symbols

y_t	Dependent variable
$P_v(t)$	Solar Generation
$P_d(t)$	Uncontrollable load
$P_{pr}(t)$	Real-time electricity price
$P_s(t)$	Total electricity sold
$P_p(t)$	Total electricity Purchased
$P_{b,in}(t)$	Battery charging capacity
$P_{b,out}(t)$	Battery discharging capacity
$P_{b,imp}(t)$	Total power imported to battery
$P_{b,exp}(t)$	Total power exported from battery
$P_{imp,pv}(t)$	Power imported from PV to battery
$P_{imp,grid}(t)$	Power imported from grid to battery
$P_{exp,grid}(t)$	Power exported from battery to grid
$P_{exp,load}(t)$	Power exported from battery to load
π	Policy
s_t	State at time t
a_t	Action at time t
R	Immediate reward
γ	Discount factor
α	Learning rate
E_b	Battery capacity

Contents

List of Figures

List of Tables

1 Introduction

According to Eurostat [1], the energy consumption in the residential sector accounted for 27% of the total energy consumption in the European Union in 2022. Currently, this demand has an electricity production mix with a renewable share of 21.1% and to reach the goal of reducing greenhouse emissions by 55% by 2023 the EU has set the goal to increase the share of renewables to 42% [2] .60% of this renewable energy growth is expected to come from locally generated rooftop photovoltaics (PV) and the rest from utility-scale solar farms [3].

The increasing prevalence of PV systems presents significant energy balancing challenges owing to the intermittent nature of energy generation. This intermittency poses obstacles to the widespread adoption of PV technology. However, implementing battery energy storage systems (BESS) has emerged as a promising solution to address these challenges [4]. Notably, the declining costs associated with BESS installations make them a financially sound decision, further incentivizing their adoption in the renewable energy landscape. Furthermore, in addition to BESS, dynamic pricing for electricity is seen as a strategic scheme to accelerate the integration of PV systems [5]. Dynamic pricing enables better resource utilization, reducing peak demand on the grid, cost savings, incentivizing BESS, and pushing the transition towards a decentralized energy system. This also opens the opportunity for prosumers to leverage BESS to benefit from financial arbitrage by exchanging with the grid [5].

An energy management system is a tool to monitor, control and optimize the energy flow in a system while ensuring energy balance is maintained in real-time, including optimal BESS operation [6]. EMS strategies range from elementary to more sophisticated ones. In recent times, an agent-based machine learning domain called RL which learns optimal control policy through trial and error has shown more than human-level performances in various sequential decision-making problems in the fields of autonomous robotics, health care, finance, gameplay and more [7]. By leveraging RL algorithms, EMS could gain the capacity to adapt and learn optimal energy control policies in complex and dynamic environments [8]. RL allows the control policy to be continuously updated which makes it robust to changes in variables over time and it autonomously takes optimal control decisions in real-time. These characteristics are aligned with the increasing complexity of the electric grid and the need to have autonomous control as decentralized energy grids are a promising pathway to a sus-

tainable grid and have shown that the RL-based models hold the potential to improve EMS performance.

To enhance the performance of the RL-based EMS in response to uncertainty in the variables over the future horizon, the model is incorporated with forecasted values of electricity load, PV generation, and dynamic electric price to inform its decision-making process [9]. For this part, various deep learning-based prediction models have been simulated to assess the best model for the EMS. Neural network-based forecasting algorithms were primarily examined due to their ability to fit non-linear data which results in higher accuracy [10]. The explored neural network architectures for this task were MLP, CNN, simple RNN, LSTM, BiLSTM and CNN-LSTM hybrid as well as evaluating its performance against naive and a statistical model (SARIMA).

This study focuses on forecasting the hourly load consumption, PV generation for a single household [11] and the real-time market price using deep-learning methods along with benchmarking the performances with a baseline model. The second part of the project dives into utilizing the predictions by feeding into the RL-based EMS to increase cost savings for the consumer. Here, the performance of RL-based EMS is evaluated against a baseline and benchmark algorithm.

1.1. Objective and research questions

This research aims to develop a reinforcement learning-based energy management system to minimize the net electricity cost for a home using the forecasted values of uncontrollable load, PV generation, and electricity price under a real-time pricing tariff. The system comprises a household consumer, an installed solar energy system, and a battery energy storage system.

The main research question is formulated as *"How can deep learning-based models and reinforcement learning algorithms be effectively employed to optimize energy management in a household and how do these models compare to traditional rule-based approaches in terms of cost savings for the consumer?"*

This primary research question is split into sub-questions, which assist in answering the main question.

1. What deep learning-based models could be employed to forecast household load, PV generation, and electricity prices?
2. What deep learning prediction algorithm achieves the highest accuracy when forecasting load, market price and PV generation for integrating into the HEMS?
3. What type of reinforcement learning algorithm is most suitable for solving energy management system problems
4. How to model an RL-based EMS with Load, BESS, and PV to increase cost savings for the consumer?
5. What is the performance of the RL-based EMS when compared to a rule-based baseline and benchmark model in terms of cost savings for the end consumer?

1.2. Thesis outline

The thesis report is organized as follows, chapter 1 outlines the relevance, motivation, objective and research questions of the project. Secondly, chapter 2 elaborates on the works in the literature on forecasting uncertain variables and energy management systems. Thirdly, chapter 3 contains the theory behind the models and algorithms used in this analysis. Next, chapter 4 details the implemented model, the hyper-parameters used, and architectures. An exploratory data analysis is carried out in Chapter 5 to gain insights into the data. Subsequently, chapter 6 presents the obtained results and discusses the performance and convergence of models. Finally, the last chapter summarizes the findings, gives the conclusion, and provides recommendations for future research.

2
Background

This chapter examines the existing literature, starting with the need for accurate forecasting and the strength of machine learning in prediction models. Next, the available forecasting methods for load, PV generation and electricity market prices are reviewed. The second part of this chapter delves into the applications of EMS, followed by an overview of available EMS strategies. Subsequently, RL and its characteristics are discussed along with existing RL-based EMS research in the current literature. Furthermore, this chapter aims to answer the first and third research sub-questions.

2.1. Overview of Energy Management Systems

Decentralized energy solutions have the potential to make the grid more reliable and sustainable by improving energy reliability, energy sharing and demand-side management aspects [12]. Home Energy management systems (HEMS) play a vital role in the realization of the distributed energy system. An EMS provides the necessary tools and functionalities to monitor, analyze and optimize energy usage within the distributed energy system. To make informed decisions regarding energy generation, distribution and storage, an EMS uses control strategies to maintain energy balance and optimize a specific objective function. Additionally, an EMS opens up opportunities for effective demand-side management by encouraging consumers to participate in energy conservation and load reduction actively [5]. Growing mass adoption of household solar installations has converted consumers into prosumers who consume, produce, and sell it back to the grid [4].

In addition to growing PV adoption, energy storage systems have become more affordable and are being installed along with PV systems. These additional components make the decision-making process more complex and crucial.

2.2. Forecasting Methods

Forecasting plays a significant role in optimizing energy usage and the overall performance of HEMS . By accurately predicting energy demand, electricity price and PV generation. In forecasting literature, the techniques range from statistical methods of the ARMA family to the utilization of neural networks for modeling non-linear relationships. A non-exhaustive list of time series forecasting methods is shown in Fig.2.1.

To leverage the full capabilities of EMS, accurate load forecasting becomes a critical task either from a consumer perspective to reduce consumption during peak price periods to increase cost savings or from a grid operator perspective to alleviate grid congestion or to use the BESS to its maximum potential. More granular and extensive data is being collected with the rise in advanced monitoring infrastructure. Deep learning forecasting methods have demonstrated significant potential in effectively managing more extensive and more complex datasets [10].

Short-term load forecasting (STFL) is the process of predicting the power demand of a power system over a short-term period, typically ranging from a few minutes to a few hours. Deep learning has demonstrated improved performance in modeling complex patterns for individual household load profiles, which tend to be more volatile due to their dependence on individual occupant behavior, as opposed to aggregate level modeling [13].

In literature, different types of models, both linear and non-linear, have been used for STFL. Family of Auto-regressive moving average (ARMA) models were pioneers in STFL [14], which was then evolved into SARIMA to account for seasonal variance [15].This set of statistical methods is limited because it assumes a linear system, whereas most often real-world cases exhibit non-linear properties. To solve this shortcoming, machine learning models like feed-forward neural networks have become attractive as they can model complex non-linear systems such as load forecasting [10]. Neural network techniques range from simple Multi layer-perceptron to convolutional neural networks (CNN) methods to recurrent neural networks (RNN) [16] along with their variants LSTM [17] and Gated recurrent units (GRU). Hybrid architectures have also been proposed in the literature between neural networks as well as between statistical and machine learning methods, such as CNN-LSTM hybrid presented in [18] and a hybrid LSTM-Exponential smoothening [19] respectively.

Globally, electricity is billed using either a fixed pricing structure or a time-of-use pricing model. With more renewable energy integration, dynamic pricing is a potential pricing structure to match real-time supply and demand. This structure also incentivizes consumers to adjust their electricity consumption during periods of high demand and encourages load shifting or demand response. Dynamic pricing gives consumers more control over their electricity costs, and they can make decisions to optimize their consumption patterns to minimize costs.

PV forecasting focuses explicitly on predicting the energy generated from solar panels, and accurate solar energy forecasting is crucial for planning available resources and reducing operational costs. It also assists in balancing supply-demand dynamics by predicting the amount of solar energy available at any given time. Solar predictions facilitate optimal energy storage management when working with energy storage systems. As it is a volatile energy source that depends on external environmental factors (Temperature, Pressure, Humidity) and has a diurnal pattern, there are three classifications of models for predicting solar generation [20]. The three categories are physical, statistical and machine learning-based models. Physical models try to capture the relationship between input features and output PV generation mathematically [21]. These equations incorporate meteorological data, design parameters and PV system characteristics. The model accuracy varies widely between models and system specifications and one such model is presented in [22]. F.M. Mulder

uses the longitude and longitude to capture diurnal variations in irradiance by utilizing each day's zenith angle and length. One of the significant drawbacks of this and other physical models is that its performance significantly reduces when accounting for local cloud cover and atmospheric variations, along with the complexity of assembling multiple models.

The statistical methods are similar to the concept under the umbrella of time series forecasting [23] where historical values are used for prediction and notable methods are ARMA models. The drawback is that it assumes an underlying linear relationship in the historical data [14]. Statistical methods are widely used PV forecasting techniques due to their simplicity and they do not require system specifications compared to physical models. In [24], an ARIMA model for a grid-connected 2.1 kW PV system is evaluated, and one of the limitations of ARIMA is not incorporating the other dependent variables, which is addressed by adding temperature, precipitation, and humidity as exogenous variables making it ARIMAX. Although external factors are included, this model still accounts for only one seasonality, which seasonal ARIMA models address called SARIMA [15].

The deregulation of electricity markets has shown the intrinsic complexity of the market prices. Predicting electricity prices has become critical for market participants to make purchasing and selling decisions. Predicting market prices is challenging due to their non-linear and non-stationary nature, making machine learning techniques a promising approach [25]. Vega et al conducted an extensive comparative analysis of various univariate machine-learning prediction models for market prices and showed that LSTM was well-suited for the task. Various factors, including the weather, gas prices, and consumption influence market prices. Zhang et al formulated this as a multivariate problem with features representing external factors, proposed a deep recurrent neural network architecture and results showed they improved the performance by 29.7% compared to support vector machines in terms of mean absolute percentage error. Kuo et al [26] have shown that the hybrid of CNN-LSTM performs better than the networks individually in forecasting day-ahead electricity market prices.

2.3. EMS Methodologies

Several methods have been presented in the literature, such as mathematical optimization, model predictive control and heuristic control for optimal scheduling of consumption and storage. Comparing existing methods in the literature can pose a challenge due to the wide range of model specifications, parameters, and objectives as mentioned in the work of Beaudin et al [6].

Some of the well-established EMS strategies are rule-based control methods (RBC), Dynamic programming (DP) and Model predictive control (MPC). The rule-based method develops a policy based on an arbitrary set of rules which govern the modes of operation. Although RBC requires simple implementation and less computational cost, it does not necessarily give the optimized solution for the control problem as it fails to model future uncertainty. On the other hand, DP is a model-based algorithm that uses a penalty function to give a globally optimized result for a control problem [28]. The computational intensity to find the global optimum every time step remains a main drawback of DP, but many HEMS have been implemented on the framework DP. MPC addresses this problem by reducing the global optimization function into a local cost

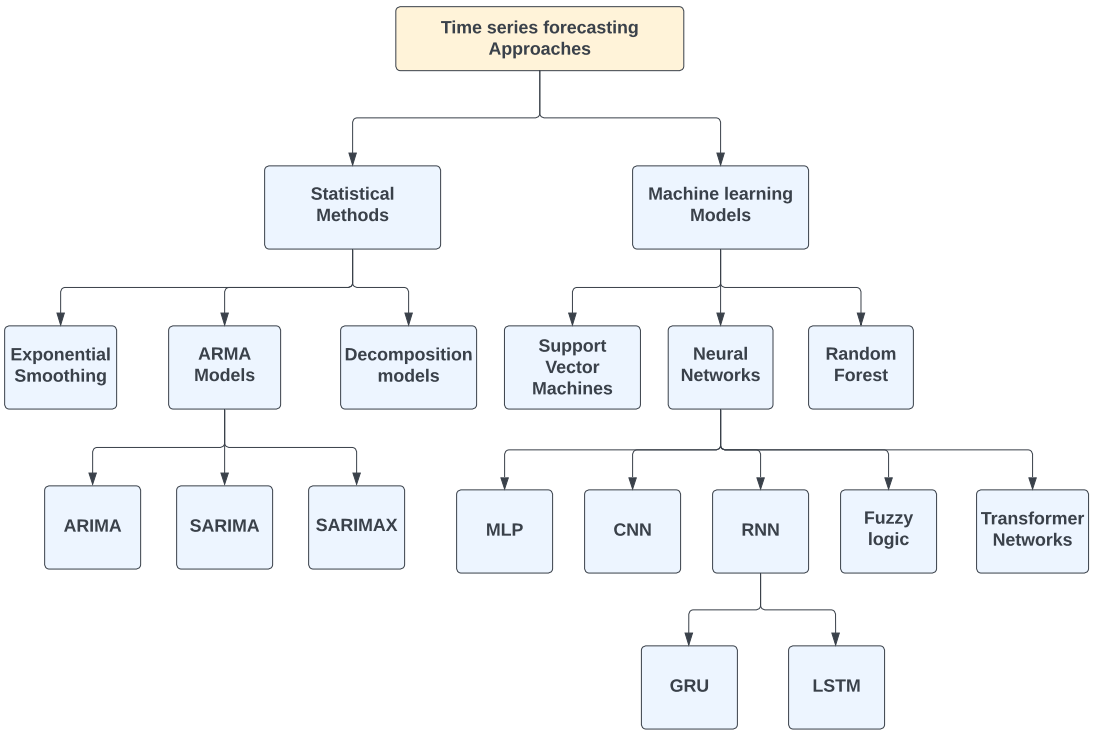


Figure 2.1: A non-exhaustive taxonomy of time series forecasting models [27]

function, which enables the system to provide real-time solutions to EMS. MPC also considers multi-step optimization to make decisions, which makes it robust in sequential decision-making. In most cases, the high computational cost of MPC and DP leads to modeling in reduced order, compromising the accuracy [29].

RL overcomes the challenges faced by the previously mentioned conventional methods. RL methods are formed as Markov decision processes (MDP) and similar to DP, Bellman's equations are used to continuously update the policy [7]. The core idea of RL is to train an agent that learns an optimal policy from experience by interacting with the environment, similar to human beings. The characteristics of RL that make it a promising control strategy for the development of home EMS as shown in the case of electric vehicle power-train EMS [30] are,

- Consumes less computational time and cost relative to DP or MPC, which makes sequential decision-making in real-time more efficient
- RL model-free approach allows the agent to learn non-linear relationships and handle larger state spaces more effectively than conventional control methods.
- DRL using function approximators enables it to approximate global optimal results.

A non-exhaustive taxonomy of EMS strategies is shown in Fig.2.2. The broad classification, as seen here is the rule-based and model-based optimization techniques. This analysis focuses on deterministic rule-based and reinforcement learning strategies.

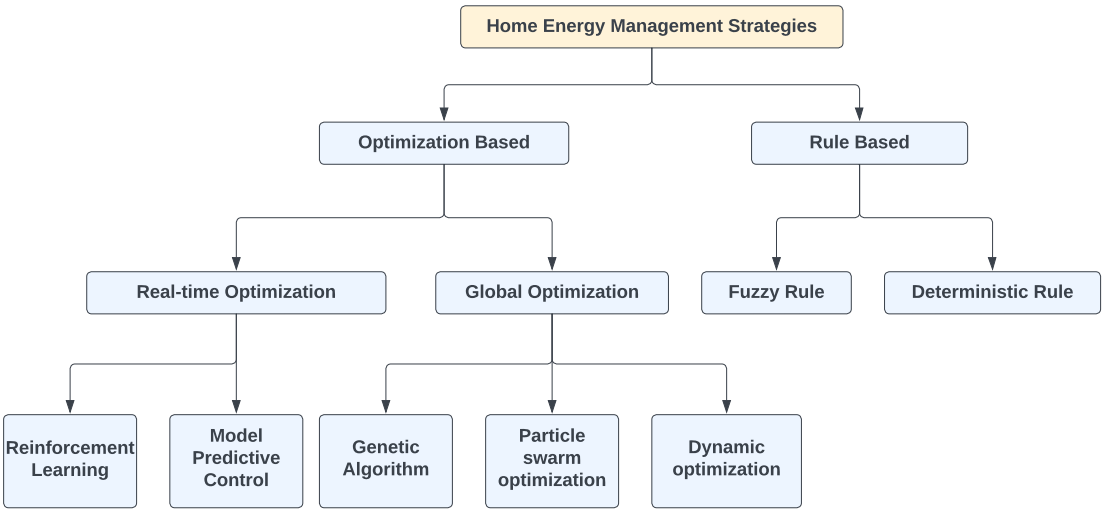


Figure 2.2: A non-exhaustive taxonomy of EMS strategies [31]

The rule-based EMS strategies in the literature are predominantly presented under the case of flat rate pricing or time of use pricing (TOU), as most retailers offer such tariffs to their customers. Kheri et al [4] presented an optimization strategy to find the optimal PV capacity and battery energy storage to minimize the net present cost of electricity for grid-connected households. This optimization utilizes a real-time rule-based HEMS that incorporates PV generation and load consumption under flat-rate pricing giving precedence to the locally available and produced energy. However, this becomes limited when operated under real-time pricing tariff (RTP) or TOU pricing and one of the ways to integrate this in the EMS is using threshold ranges as proposed in [5]. The mentioned work proposes the optimal trading strategy for a battery energy storage system. The strategy involves computing threshold values for each day that determine the limits of market price to charge or discharge the battery with the grid. This proposed EMS does not include PV which limits its application. In both of the discussed works, it was concluded after a cost analysis that the financial benefit for the end consumer could be increased by using an effective BESS dispatch strategy.

2.4. Reinforcement Learning

Reinforcement learning is a subcategory of machine learning with a mathematical framework where an agent learns an optimal policy through interaction with the environment [7].

The rise of deep learning has significantly improved many tasks such as natural language processing, object detection, speech recognition and more. This is mainly attributed to the ability of neural networks to extract low-dimensional feature representations of high-dimensional data. This property has also been leveraged in RL to address the curse of high dimensionality and integrating neural networks with RL is called deep reinforcement learning (DRL) [32]. Deep learning enables RL to scale to problems of high state and action spaces, which were previously not feasible. Among recent works that revolutionized DRL is the algorithm that learned to play a range of Atari video games at a superhuman level directly from image pixels [33]. Another

successful demonstration of a DRL agent that defeated a human world champion in AlphaGO [34]. Some of the most influential work done in RL is illustrated in Table.2.1

RL algorithms are broadly categorized into model-based and model-free [7]. In model-based RL, the agent creates and learns the model dynamics or representation and uses this learned model to make sequential decision-making. In comparison, the model-free algorithms try to directly learn the policy or value function to maximize the cumulative rewards. Although both have their advantages and fallacies, generally, model-based methods fail to model complex environments, whereas model-free methods could perform reasonably well in such environments [7]. The dichotomy in model-free methods is value-function based or policy-based. In a value-function based approach, the agent learns how valuable a state or a state-action pair which then partially dictates the policy on which action to take. The policy-based approach tries to learn the policy directly from the states, which maps states to action. A non-exhaustive classification of RL methods is shown in Fig.2.3, and other main categories within value-based RL methods are on-policy and off-policy. In on-policy, the agent learns and updates a single policy with experiences whereas the off-policy method learns from multiple policies experienced during the learning process. The property of off-policy, where the agent learns from different policies during exploration which enables better policy improvements has led to a broad adaptation of Q-learning [35].

Table 2.1: Key papers in Deep Reinforcement Learning

Number	Description	Algorithm
1	Playing Atari with Deep Reinforcement Learning [33]	DQN
2	Deep Reinforcement Learning with Double Q-learning[36]	Double DQN
3	Prioritized Experience Replay [37]	Prioritized Experience Replay (PER)
4	Deterministic Policy Gradient Algorithms [38]	DPG
5	Continuous Control With Deep Reinforcement Learning [39]	DDPG

In [40], Lu et al propose a DDPG based EMS to efficiently control HVAC systems and Energy storage scheduling to reduce energy costs while maintaining the temperature comfort of the consumers without the building dynamics. In the analysis, the DDPG agent was compared with two baselines and resulted in a more robust and better cost savings. Similarly, Lissa et al [9] proposes a smart home energy management system controlling indoor and domestic water temperature control using the DRL algorithm to reduce energy consumption by optimizing PV energy produced. The results showed that, on average DRL agents achieved 8% better energy savings than rule-based methods and efficiently predicted and delayed actions for PV self-consumption optimization. Likewise, Daniel et al [41] proposes an online DQN-based residential energy management algorithm to reduce energy costs and cut down peaks. Compared to conventional methods, this algorithm makes optimal decisions by making the agent price aware and anticipating the consumer's future behavior. This novel

strategy showed that by incorporating the price signal, the average energy price was reduced by 16% compared to one without the signal. The historic state transitions are used as this method explores online learning compared to offline learning. In online learning, the agent learns in a real-world setting that could get expensive and have exploration challenges.

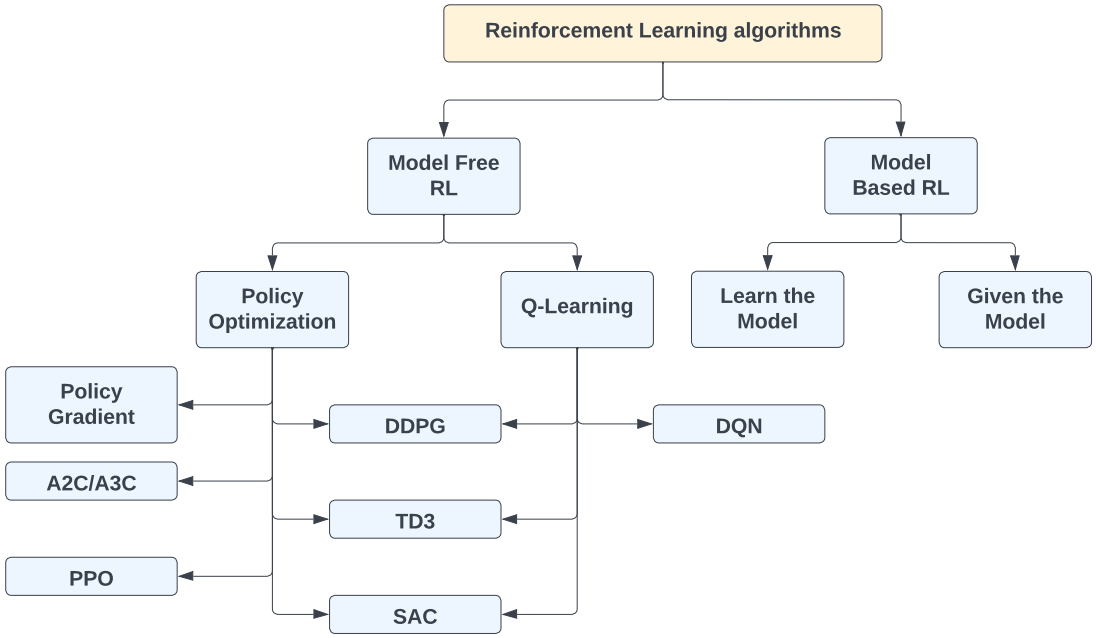


Figure 2.3: A non-exhaustive taxonomy of algorithms in modern RL [42]

Most RL-based EMS in literature discusses the combination of optimal ESS scheduling and temperature control or appliance scheduling, as model dynamics are complex and challenging to model. In [43], Han et al propose a combined RL and deep learning based arbitrage maximizing algorithm for an energy storage system operator. In this algorithm, to mitigate the uncertainties, the future load and market price are forecasted using a recurrent neural network fed into the DQN agent to learn the optimal policy. This algorithm implements a stimulus to the agent whenever ESS does a peak load shift. The results showed that the operating profit of ESS increased by 2.4 times and reduced the on-peak demand by 30%, benefitting both ESS and the grid operator.

Another domain where RL shows promise is scheduling controllable loads such as freezers, dishwashers, or even electric vehicles to achieve more cost savings. Lui [44] proposes a DQN agent and double deep q-network (DDQN) for a home energy management system for scheduling of controllable loads (air-conditioning, dishwasher, EV) to reduce the energy bill and validates the performance of the algorithm with particle swarm optimization (PSO). The study examines scenarios with and without PV generation as the effect of EV. It concludes that DDQN performed better and shows more advantage over DQN and PSO but still needs more examination as the algorithm is under partially observable conditions. Likewise, Moncanu et al [8] proposed an online building energy management system using DQN and compared the performance with a deep policy gradient. The analysis was conducted on both residential buildings and aggregate of buildings for which the Pecan database was used. The results

showed that the deep policy gradient method performed better than DQN for the online learning of building energy management, although both approaches succeeded in minimizing energy costs.

With the rising adoption of decentralized energy systems due to their potential to mitigate greenhouse emissions and build a sustainable electricity grid and RL strategies as a potential intelligent control agent, various defined RL environments have been developed, allowing more focus on algorithm implementation. One of them is pymgrid [45], a python package that is capable of generating and simulating 600 modifications of microgrids which are built based on domain expertise and this is built explicitly for RL algorithms as the environments are modeled as Markov decision processes. Another one is CityLearn [46], an open-source OpenAI gym environment for implementing agent-based RL models for building energy coordination on an individual or aggregate level and demand response. Citylearn enables easy comparison of various algorithms, setting benchmark performance and options for load shedding and energy storage scheduling.

3
Theory

This chapter provides the essential theoretical knowledge for this study. The chapter begins with exploring forecasting models like SARIMA and then different deep-learning architectures in detail. The second part of the chapter covers the fundamental principles and strategies of EMS, followed by the theory of the baseline and benchmark algorithm used in this analysis. Afterward, A detailed investigation is conducted into the foundational principles of reinforcement learning. Finally, the approaches utilized for optimizing hyperparameters are discussed.

3.1. Forecasting Models

Time series analysis is a crucial problem in many domains, offering the ability to predict future values based on its own or dependent historical values with some margin for error [47]. It has shown numerous successful applications in finance, economics and the energy sector. Firstly, a baseline model is presented which serves as a point of comparison for the performance of different models. The technique utilized for a baseline should be deterministic, easy to implement and not be problem-specific [48]. The chosen forecast baseline for this analysis is the persistence model as compared to the average forecast strategy due to the diurnal periodicity present in the data.

3.1.1. Persistence Model

A persistence model, a naive model, is a simple and essential technique for implementing reference baseline in time series forecasting. A naive forecast uses the previous observation directly as the predicted value without any change. This could be extended for seasonal data by using the observation simultaneously in the previous cycle. In this study, for seasonal data, a daily seasonal naive approach is employed as the baseline, which involves using the value from the same time in the previous cycle (yesterday's value) as the forecast.

3.1.2. SARIMA

Box and Jenkins developed a mathematical forecasting model where the future value of a variable is assumed to be a linear function of past observations and random errors. The family of such models is called an autoregressive integrated moving average

(ARIMA) model [15]. The ARIMA process intrinsically consists of two parts, an autoregressive part (AR) and a moving average (MA), where the AR part uses the previous values to model the prediction and the MA part models the error term or deviation from the mean. The model is represented using the notation ARIMA (*p,d,q*) where *p* specifies the autoregressive order on how many lags values are used in the model, *d* indicates the number of differencing applied to the time series and *q* specifies the moving average order on how many previous deviations are used in the prediction model.

The generalized ARIMA model could be expanded by adding seasonal AR, MA and differencing terms for a periodicity called seasonal ARIMA (SARIMA) to incorporate seasonality into the ARIMA model. The SARIMA model is represented as SARIMA (*p,d,q*)(*P, D, Q*)*s*, where *P, D, Q* are the respective order terms for the seasonal differencing *s*. The mathematical expression for the prediction is illustrated in Equation 3.1 where the c is a constant term, ϵ_t is the error term and ϕ_n,θ_n,Θ_n,Φ_n are the coefficients of lag terms. This removes the non-stationarity in the data, which might affect the performance as ARIMA models operate with the underlying assumption of stationary data. This also limits the inclusion of only one seasonality, whereas the data being analyzed in this study has multiple seasonality.

$$y_t = c + \sum_{n=1}^{p} \phi_n y_{t-n} + \sum_{n=1}^{q} \theta_n \epsilon_{t-n} + \sum_{n=1}^{P} \Phi_n y_{t-sn} + \sum_{n=1}^{Q} \Theta_n \epsilon_{t-sn} + \epsilon_t \tag{3.1}$$

3.1.3. Deep learning Methods

Neural Networks are part of a family of machine-learning techniques inspired by the functioning of the human brain which consists of interconnected nodes (neurons) compartmentalized into layers [49] . A typical neural network architecture consists of an input, hidden, and output layer, as seen in Fig 3.2. Network architectures having more than one hidden layer are called deep networks which increase the capabilities to extract information from the input data. Machine learning is broadly classified into supervised learning and unsupervised learning [7]. In supervised learning, the widely applied method learns a function to map the input to output by training on labeled data. Whereas unsupervised learning learns patterns or relationships without explicit labels or target data. In forecasting, given the historical data available the algorithms are widely trained using the supervised learning method.

In supervised learning, they are trained using labeled data to update their parameters and optimize for a loss function with the weights between neurons calculated and optimized using an optimization algorithm. Activation functions are applied to the weighted sum of synoptic weights and bias term introducing non-linearity to the model, this is depicted in Fig.3.1. There are different architectures of neural networks built on the foundational structure.

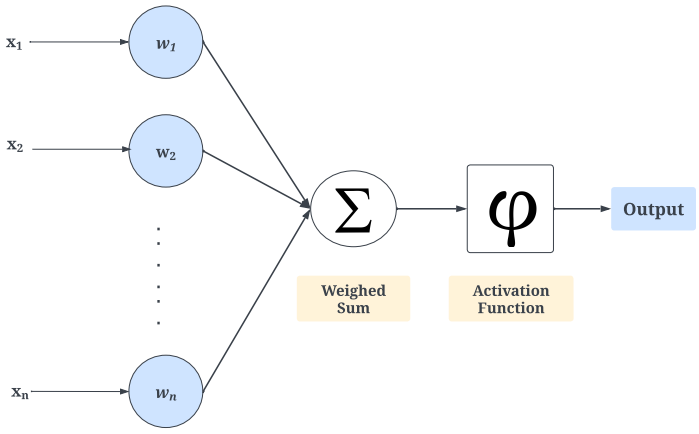


Figure 3.1: Artificial Neuron structure

In the context of machine learning models, they broadly fall under the batch learning category or online learning category [50]. Batch (offline) learning means training the model using the historical data all at once, whereas in online learning the model learns from the training instances incrementally. Online learning adapts to changes and sudden variations in the data more effectively than batch learning. However, the standard practice is to initially train the model offline and then continue with online learning. This analysis examines only batch learning as infrastructure for online learning needs to be in a real-world setting.

Multi-layer perceptron

Multi-layer perceptron is a type of feed-forward artificial neural network (ANN) where the information flows in only one direction from the input layer to the output layer without any loops or data feedback. MLP is introduced in the hidden region as Dense layers, and in the context of load forecasting, the model input takes each time step as a feature. This brings the limitation of not capturing the temporal dependencies regardless MLP has shown to have competitive performance in a few load forecasting use cases [51].

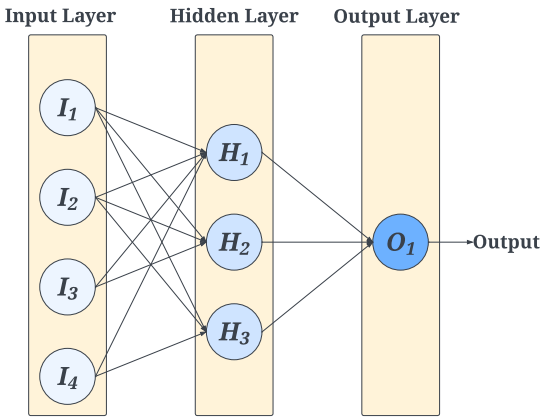


Figure 3.2: Deep Neural Network (DNN) Architecture

Convolutional Neural Network

Convolutional neural networks are a family of ANNs, that works with a grid-like structure and has been extensively used in image recognition and natural language processing [52]. The key property of CNNs to extract features has been leveraged for univariate time-series forecasting, where a filter/kernel is passed through the series to extract relevant features as visualized in Fig.3.3. Although CNN is effective in NLP and image recognition, they are not widely used in time series forecasting as CNN cannot model sequential data, which has been addressed recently by combining them with recurrent neural networks [13]. A vanilla CNN and CNN-LSTM hybrid architectures have been analyzed in this work.

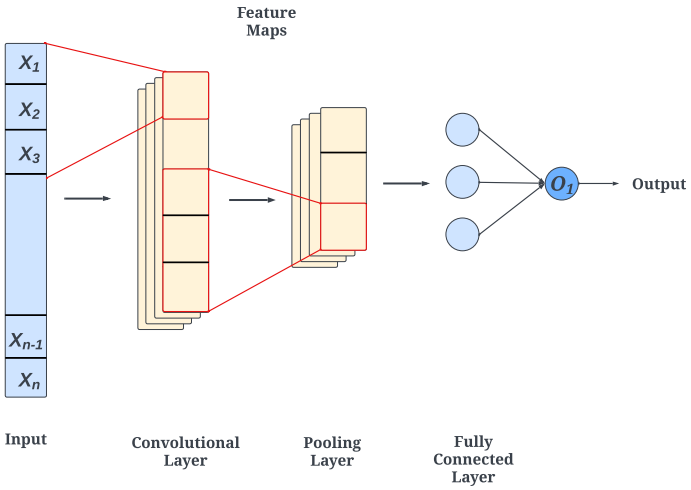


Figure 3.3: Convolutional Neural Network (CNN) Architecture

Recurrent Neural Network

Recurrent neural networks are an NN architecture that modifies feed-forward neural networks to handle sequential data and capture temporal patterns better, which makes it a powerful tool for time series forecasting [16]. Compared to feed-forward neural networks, RNNs maintain an internal state that allows them to remember information from previous inputs. Essentially, an RNN layer could be seen as multiple copies of the same network where the representation of each time step is learned in a loop, allowing it to learn the temporal dependencies, whereas in Fig. 3.2, the information makes only one pass through the layer. In Fig.3.4, an unfolded RNN is visually represented where A represents each neural network copy of a time step with x as the input and h as the hidden state which is stored and passed on to the next time step. RNNs experience vanishing/exploding gradient problems, and their inherent structure to update the hidden state using the current step and previous hidden step limits their capability to remember information over longer intervals.

Long Short Term Memory

LSTM is a variant of Recurrent Neural Networks, which addresses the problem of learning long-term patterns. Traditional RNN, as seen in Fig.3.4, can model temporal dependencies by having a hidden state which enables it to learn the relationship.

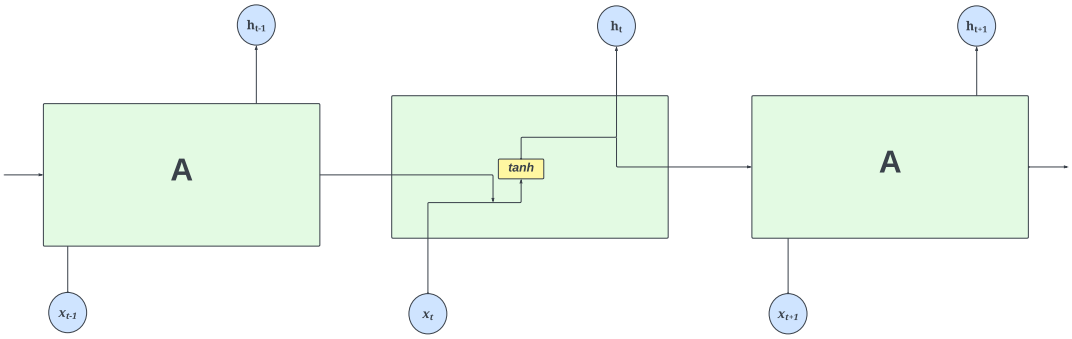


Figure 3.4: Recurrent Neural Network (RNN) Architecture

However, this architecture continuously updates its weights, leading to only capturing short-term dependencies. This is addressed in LSTM architecture by introducing an internal state that stores the relevant information and forgets irrelevant information using gating mechanisms named forget gate, input gate and output gate. This ability of LSTM to store short-term and long-term interpretations has made it a popular choice for time series forecasting.

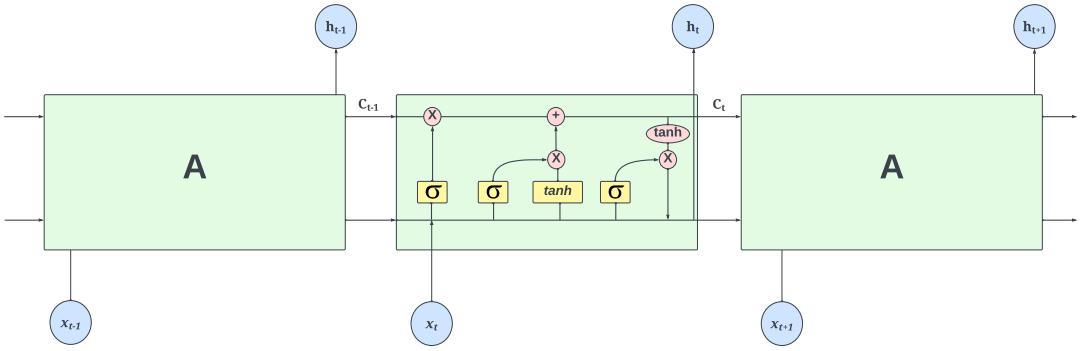


Figure 3.5: Long short-term memory (LSTM) Architecture

In RNN's architecture, each repeating module has a simple tanh layer (as seen in Fig.3.4), but LSTM has a different and complex structure with four layers that are interacting uniquely as shown in Fig.3.5. Here, the core component of LSTM is the internal cell state that runs through all the modules as represented in Fig.3.5 as the horizontal line running at the top. In brief, LSTM can add or remove information from the internal state through gates and this is how it stores long-term representation. Since it has more layers, more parameters make LSTM models take longer to train and prone to overfitting.

Bidirectional LSTM

Bidirectional LSTM is a variant of LSTM where the network is trained using the input data in forward and backward sequences. This is achieved by two connected layers where each processes the input data in a reversed direction and is combined using different merging methods. This architecture allows the network to learn from past and future contexts and avoids loss of information in LSTM where, in some instances,

the initial context fades away. Similar to LSTM, with two connected LSTM layers, the number of parameters is doubled which could lead to overfitting and taking longer to train the network.

CNN-LSTM Hybrid

The hybrid model combines the effectiveness of CNN to automatically extract and interpret features from a univariate time series along with utilizing the power of LSTMs to model temporal dependencies [53]. In this architecture, the input sequences are split into sub-sequences processed by the CNN model, and the interpretation of the sub-sequences is then fed into the LSTM model as input. Here, the number of sub-sequences is a hyperparameter and this expands into the number of time steps per sub-sequence based on the look-back period chosen. It should be noted that the output from CNN should be flattened to a single one-dimensional vector before it goes into the LSTM network.

3.2. Energy Management systems

3.2.1. EMS Baseline

When assessing the performance of a dynamic algorithm like RL for energy management, it is vital to establish a baseline for comparison. The baseline implemented in this analysis is a simple and deterministic approach called the rule-based EMS. This is a well-established control mechanism and this system relies on a set of predefined rules and logic to make decisions. Using a baseline provides a reference point for evaluating the RL-based model's effectiveness and the two approaches' comprehensibility.

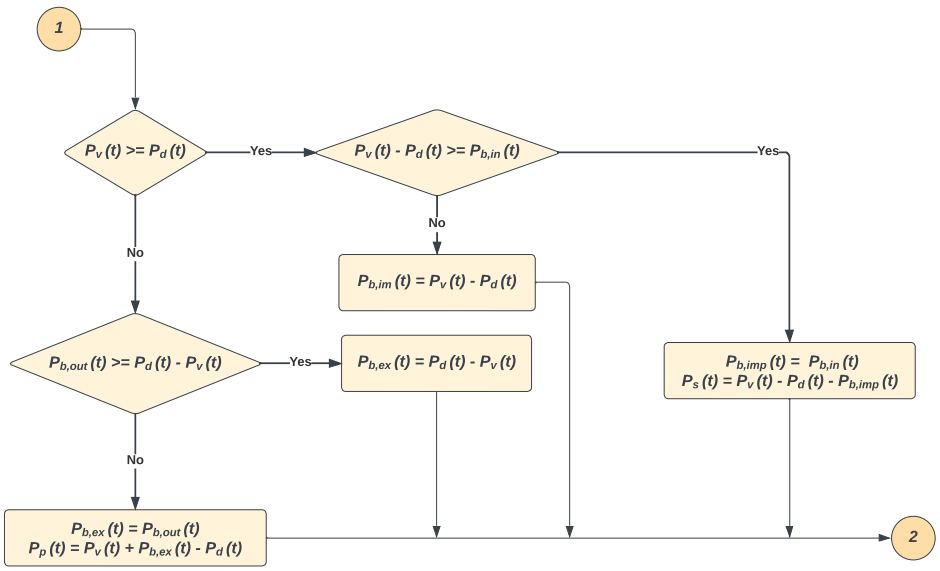


Figure 3.6: Flow diagram of Baseline EMS model

The baseline approach used here is adopted from optimal capacity optimization for households carried out by [4], where the control flow starts from checking if the load for that period is lower or higher than the PV produced. This allows the consumption

of power produced locally and is followed by the charging/discharging of the battery according to excess or deficit of energy depicted in Fig.3.6. The key purpose of this model is to provide a point of reference for comparing against other models. If more sophisticated models discussed in this study do not perform better than the baseline, it could understood that the complexity added in the other models does not necessarily translate into better performance. For simplicity, in this model, the BESS does not trade with the grid, and the price single is not considered.

3.2.2. EMS Benchmark

To conduct a more robust comparison, a model that takes optimal BESS decisions to trade with the grid along with dispatching to the load is developed. The approach is adopted from Dufo's work [5] of optimization of a grid-connected storage system under real-time electricity pricing.

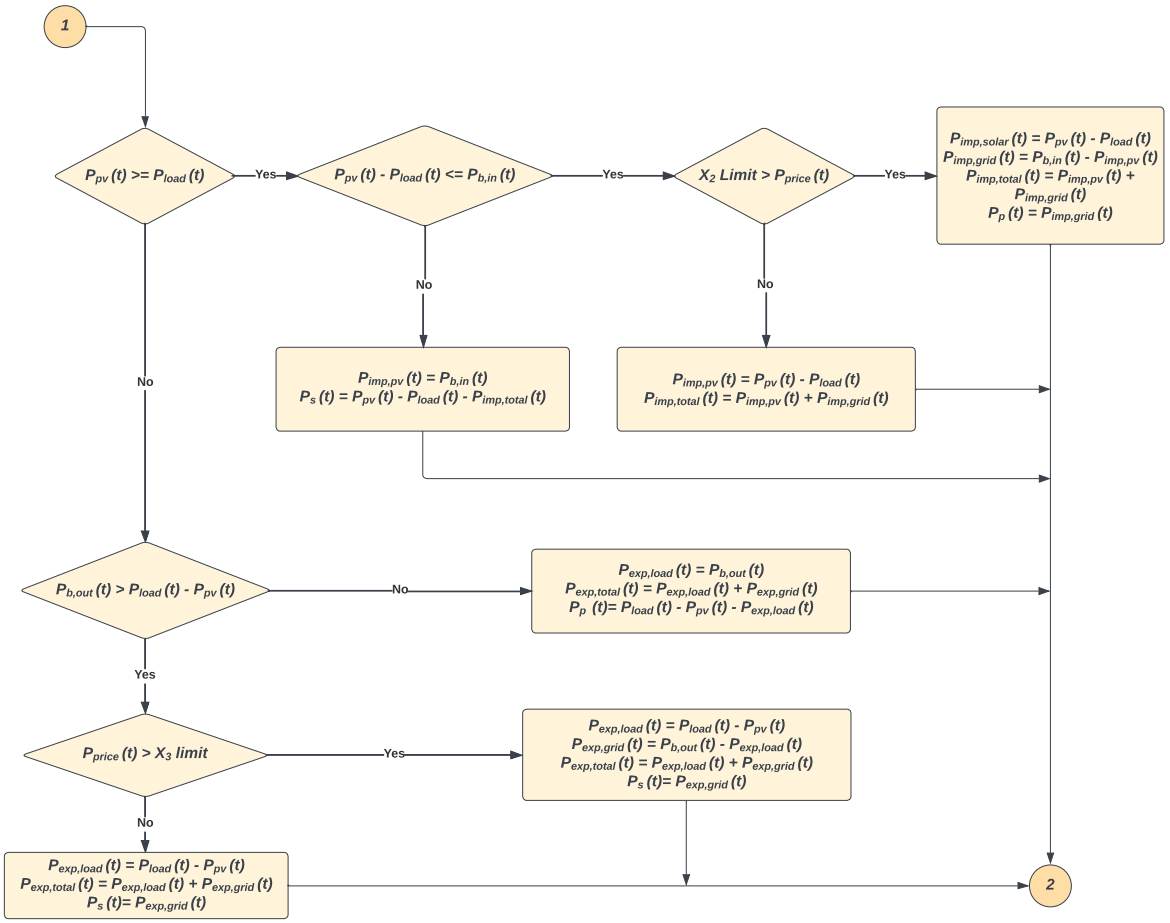


Figure 3.7: Flow diagram of Benchmark EMS model

This algorithm overcomes the limitation of the baseline model by incorporating the price signal to make optimal BESS decisions. The approach involves utilizing the price prediction of the subsequent day to make arbitrage decisions by computing two dynamic price thresholds daily. These threshold price values dictate when to buy electricity for battery charging and when to discharge it back into the grid. X_1 sets the

threshold for purchasing the electricity and X_2 is the cutoff value to sell back to the grid. The equation is shown in Equation 3.2 and 3.3. Here, X_1 and X_2 are hyperparameters that need to be determined. Intuitively this can be thought of as the percentage of the price difference.

$$X_{1,\text{daily}} = Pr_{\text{Min}} + Pr_{\text{diff}} \cdot X_1 \tag{3.2}$$

$$X_{2,\text{daily}} = Pr_{\text{Max}} - Pr_{\text{diff}} \cdot X_2 \tag{3.3}$$

This algorithm incorporates the price signal within the framework of the implemented baseline [4]. The power flow diagram is illustrated in Fig.3.7. Self-consumption is prioritized by giving precedence to discharging the battery to satisfy the load over importing from the grid and charging the battery if there is an excess of PV power. Although this approach takes optimal real-time decisions, it cannot integrate the future anticipated values of load, PV generation and electricity price.

3.2.3. Reinforcement Learning based EMS

Machine learning is broadly classified into supervised, unsupervised, and reinforcement learning (RL) [7]. In supervised learning, the widely applied method learns a function to map the input to output by training on labeled data. In comparison, unsupervised learning learns patterns or relationships without explicit labels or target data. Unlike other methods, RL involves an agent learning how to interact with its environment and make sequential decisions to maximize its cumulative reward. It is similar to the human nature of learning by trial and error, using feedback to optimize decisions.

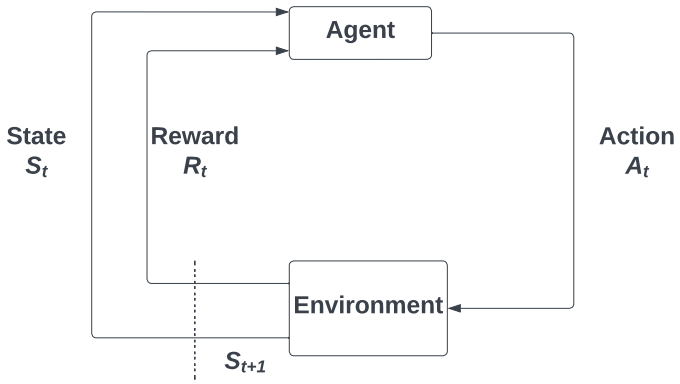


Figure 3.8: Schematic representation of agent environment interaction in RL

Problem setup

An RL agent learns through interaction with the environment through trial and error. At each time step upon the outcomes of its actions, it can learn to alter its behavior in response to rewards received. The key influence of RL has its roots in human behavioral psychology and the mathematical formalism of dynamic programming [7]. In the RL setup, an agent is an autonomous decision-making entity and at each time

step, the agent receives state s from the state space S and takes an action a from the action space A based on a policy (π) i.e mapping of states to actions and receiving a scalar reward r which also acts as a feedback. When interacting with the environment, the state s transitions to the next state $s+1$ based on the environment model or dynamics, and the transition probability $P(s_{t+1}|s_t, a_t)$. A diagram depicting the agent-environment interaction is shown in Fig.3.8.The assumption is that the provided state comprises all the necessary information for the agent to make the best decision. The agent's goal is to learn a policy (π) that maximizes the expected return (cumulative discounted reward) and this policy which gives the maximum reward is called optimal policy. In essence, RL tries to solve the problem similar to optimal control but the difference is that RL does not have the state transition dynamics it learns through trial and error.

In an episodic problem i.e. if the agent is trained in an environment that has a terminal state or an end state, then the agent reaches the end of the episode and restarts. The cumulative reward is computed when the terminal state is reached and the return is discounted to give varying importance to delayed rewards using a discount factor. The agent aims to maximize the expected rewards from each state.

Markov Decision Processes

Markov decision process (MDP) is a mathematical framework used for modeling sequential decision-making problems where the outcome is partly random and partly controllable. In essence, the MDP states that the future state is only dependent on the current state and independent of the past. This means that a decision made at s_t is only dependent on s_{t+1} and independent of $s_0, s_1, ..., s_{t-1}$. A Markov decision process is typically defined as a tuple $(S, A, P_{sa}, \gamma, R)$ where,

- S is a set of all possible states
- A is a set of all possible actions
- P_{sa} are the state transition probabilities
- γ is the discount factor
- R is the immediate/instantaneous reward

Value Function

In almost all RL algorithms, value functions are crucial in the learning process. Value functions estimate how good it is for the agent to be in a specific state (state-value function) or take a specific action from a specific state (state-action value function) when following a policy π [7]. In an episodic learning problem "how good" refers to the sum of immediate and discounted rewards over the time steps till the terminal state is reached. This quantifies the value associated with the agent being a state s for a state function or being in a state s taking an action a for a state-action function.

For an agent following policy π starting from state s the state-value function $v_\pi(s)$ could be expressed as in Equation 3.4. Here, E_π denotes the expected value of the reward. It should be noted that the state-value function intuitively gives a value based on the average of total reward taken from all possible actions from state s. Similarly, the value when an agent takes an action a in state s under a policy π could be expressed as Equation 3.5

$$v_\pi(s) = E_\pi[R_t|s_t = s] = E_\pi\left[\sum_{k=0} \gamma^k R_{t+k+1}|S_t = s\right] \tag{3.4}$$

$$q_\pi(s,a) = E_\pi[R_t|s_t = s, a_t = a] = E_\pi\left[\sum_{k=0} \gamma^k R_{t+k+1}|S_t = s, a_t = a\right] \tag{3.5}$$

For the value-function v^π, the Bellman equation could be expressed as Equation 3.6. Here, the Bellman equation is a fundamental property that holds a recursive relationship between a state and the values of the successive states under a policy π. This property is used widely throughout RL and dynamic programming. This property also lays the foundation to compute, learn, and approximate the value function. For a stochastic process, $\pi(a|s)$ defines the action probabilities and $p(s', r|s, a)$ defines the transition probabilities and the sum over all the possibilities gives the expected value. This could be extended to include the state-action pair q value function as expressed in Equation 3.7.

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{3.6}$$

$$q_\pi(s,a) = \sum_{s',r} p(s', r|s, a)[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a')] \tag{3.7}$$

The goal of RL is to find a policy that maximizes the cumulative reward that the agent receives and this policy is defined as the optimal policy (π^*). Finding optimal policy partially is defined by the function functions and part by the stochastic system dynamics. A policy is said to be optimal ($\pi^* >= \pi$) only when the value function of that policy is yielding the maximum cumulative reward that is $v_\pi(s) > v_{\pi'}(s)$ for all states. The optimal state-value function that follows policy π^* is expressed in Equation 3.8 and the Bellman equation for v^* or the Bellman optimality equation is defined in Equation 3.9. The Bellman optimality equation instinctively expresses that the value of a state under optimal policy must give the expected return for the best action taken from that state s

$$v_*(s) = max_\pi v_\pi(s) \tag{3.8}$$

$$v_*(s) = max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')] \tag{3.9}$$

The optimal action-value function with a similar definition as the optimal value function is denoted as q_* and is expressed as Equation 3.10. This function gives the cumulative reward starting from state s taking action a when the agent follows the optimal policy π^*. The Bellman optimality equation for optimal action-value is shown in Equation 3.11.

$$q_*(s,a) = max_\pi q_\pi(s,a) \tag{3.10}$$

$$q_*(s,a) = \sum_{s',r} p(s', r|s, a)[r + \gamma max_{a'} q_\pi(s', a')] \tag{3.11}$$

Q-learning

The Q-learning method is an off-policy model-free reinforcement learning approach built on the mathematical framework of temporal differencing learning (TD) [33] . In TD learning, the agent learns from the experiences in time. This approach aims to update the value of some estimate V for the value of policy π for all states. The most fundamental method is TD(0), where the value V is updated for a policy using the immediate reward the agent receives and the value of the state it transitions to, this is shown in Equation 3.12. In this equation, $r + \gamma V(s') - V(s)$ is called the TD error and γ is the discounting rate for future rewards. Here, α is the learning rate or constant step size parameter, influencing how quickly the TD algorithm learns or the value is updated. The learning rate also influences how quickly the model converges [7].

	A_1	A_2	...	A_m
S_1	$Q(S_1,A_1)$	$Q(S_1,A_2)$	..	$Q(S_1,A_m)$
S_2	$Q(S_2,A_1)$	$Q(S_2,A_2)$	..	$Q(S_2,A_m)$
...	...	..	..	...
S_n	$Q(S_n,A_1)$	$Q(S_n,A_2)$	..	$Q(S_n,A_m)$

Figure 3.9: Visual representation of a Q-table

In Q-learning, when the agent interacts with the environment, the agent forms state-action pairs and assigns a Q-value for each pair. This could be seen as an extension of TD(0) learning where it is only for the state value but here it is for state-action pairs. This Q value could be interpreted as how beneficial it is for the agent to take action a and transition to state s' from state s. This is stored in a table shown in Fig.3.9 and is called a Q-table. The values in the Q-table are iteratively updated at every step based on the Bellman equation, which is shown in Equation 3.13

$$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)] \tag{3.12}$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)] \tag{3.13}$$

Deep Q-learning

Deep Q-Network is a combination of Q-learning and deep neural networks. In DQN, the traditional Q-table shown in Fig.3.9 is replaced by a deep neural network architecture as seen in Fig.3.10. Here, instead of computing the Q-value for each state-action pair, the neural networks act as a function approximator to estimate the Q-value, and similar to the foundation RL approach, the action with the highest Q-value could be used to make optimal decisions.

Using the function approximators in Q-learning overcomes the curse of dimensionality and can handle large or continuous state spaces. Another advantage of using Deep Q-networks is that they generalize the learning process and equip the agent to approximate decisions even to states it has not trained during the learning process.

The learning process is also stabilized using techniques such as experience replay, target network and greedy epsilon strategy exploration-exploitation. Here, the experience replay stores the experiences, and the network is trained on a randomly chosen set of transitions, which helps break sequential correlation and leads to a more stable learning process.

Although DQN solves problems with high-dimensional observation spaces, it can only handle discrete action spaces. However, most real-world/physical problems have high-dimensional or continuous action spaces. Here, DQN cannot be applied directly as it depends on finding the action that maximizes the Q-value. A continuous action space case requires optimization in every single step. One of the straightforward approaches to deal with this issue is discretizing the action space. However, this has a few limitations and the most prominent one is the curse of dimensionality where the number of actions increases exponentially with degrees of freedom which leads to an explosion of the action space. Such large action spaces are difficult to effectively explore which could lead to sub-optimal performance or sometimes even infeasibility. Additionally, the discretization of the action space might lead to loss of information about the structure of the action domain which might be essential to solving many problems.

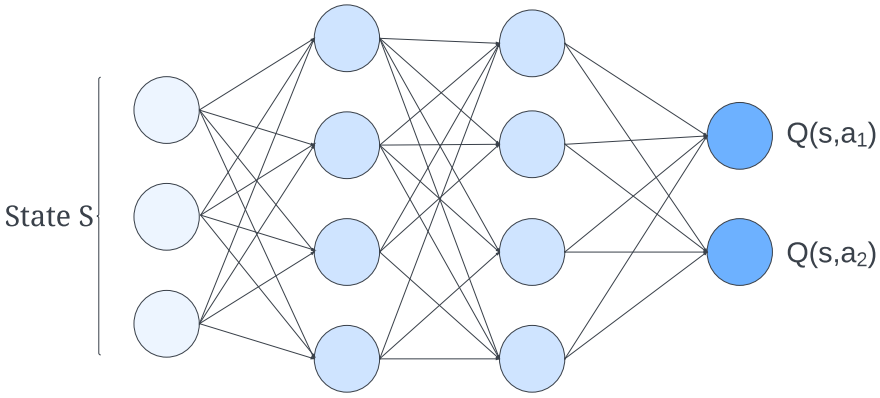


Figure 3.10: Deep Q-Learning (DQN) Architecture

Deep Deterministic Policy Gradient

DDPG is a model-free, off-policy actor-critic algorithm that uses neural networks as function approximators to learn policies that are capable of handling high-dimensional, continuous state and action spaces [39]. This architecture can be seen as an expansion of DQN to continuous action spaces. DDPG has an actor-critic framework where the critic network is analogous to the Q- function in DQN and the actor is a policy function that functions simultaneously. The critic network computes Q-values using a TD error update accordingly. On the other hand, the actor network directly generates actions and is updated using a policy gradient that rely on the Q-values computed by the critic network [54].

The policy gradient method used in the original implementation is built on top of the deterministic policy gradient method proposed by [38]. The learning process of DDPG is visually illustrated in Fig.3.12. The actor learns a deterministic policy function $\mu(s)$

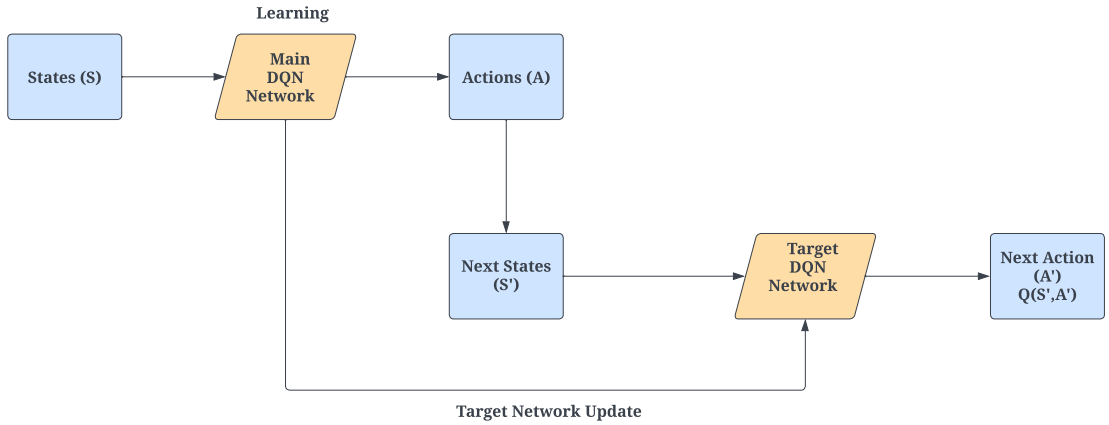


Figure 3.11: Illustration of target network structure in DQN

and the parameters are denoted by θ^{μ}. The action taken by the actor network can be represented by $a(t) = \mu(s_t|\theta_t^{\mu})$. The problem of exploration vs. exploitation in DDPG is addressed by adding noise to the action values generated by a random process represented by $a(t) = \pi(s_t|\theta_t^{\mu}) + N_t$ where N is the random process. In the original implementation, the Ornstein–Uhlenbeck process was utilized to generate exploration noise due to its time-correlated nature which works efficiently in conjunction with MDP processes. This algorithm uses target networks as implemented in DQN to tackle the problem of divergence so in total, this algorithm uses four networks in total.

The critic network's state-action value function is learned using Bellman's equation, similar to DQN. In Equation.3.11, the state s_{t+1}, r_i are obtained as result of the actor network's output $a(t) = \mu(s_t|\theta_t^{\mu})$ following policy μ. The critic network learns a policy π to compute the q-value using the state and the action taken by the actor network. In Equation 3.14, y_i is the computed Q-value which is used to compute the mean squared error loss function. The loss as expressed in Equation 3.14 is minimized using gradient descent propagated through the critic network.

$$y_i = r_i + \gamma\pi(s_{t+1}, \mu(s_{t+1})) \tag{3.14}$$

$$L = \frac{1}{N}\sum_i (y_i - \pi(s_i, a_i|\theta^Q))^2 \tag{3.15}$$

For the policy function, the objective is to maximize the cumulative reward which is the Q-value output of the critic network Equation 3.16. The derivative of the objective function ($\pi(s_{t+1}, \mu(s_{t+1}))$) with respect to the policy parameter (θ^{μ}) is used to compute the policy loss (J). Policy μ is updated using the chain rule to the loss function and since this is a batch-wise learning the mean over the sum of gradients is calculated as shown in Equation 3.17

$$J(\theta) = E[Q(s,a)|_{s=s_t, a_t=\mu(s_t)}] \tag{3.16}$$

$$\nabla_{\theta^{\mu}} J(\theta) \approx \frac{1}{N}\sum_t \nabla_a Q(s,a)|_{s=s_t, a_t=\mu(s_t)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s=s_t} \tag{3.17}$$

Finally, the target networks are updated in a similar way to DQN but instead of directly copying this implementation follows an exponential smoothing approach to update the target networks gradually. The hyperparameter for the soft update is τ «1, the lower the value, the slower the target network gets updated.

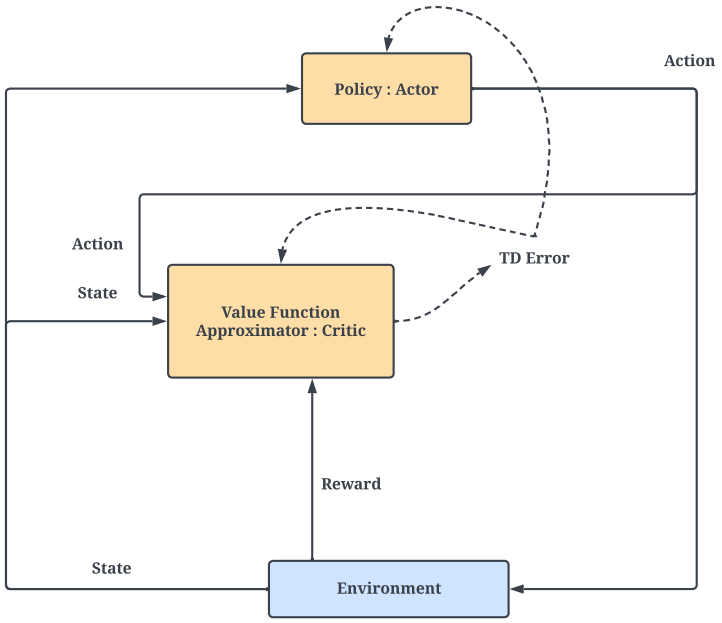


Figure 3.12: Actor-Critic Architecture [7]

Although DDPG has proven to be very effective in problems with continuous action spaces, the algorithm is very sensitive to hyperparameters which can make the learning process very unstable and depends on correct hyperparameters for a specific problem. Another problem faced by DDPG is the overestimation bias where the target critic network overestimates the value of the Q-value. This can lead to sub-optimal performances [55].

3.3. Hyperparameter Tuning

Hyperparameter tuning is one of the crucial parts of deep-learning model implementation, where hyperparameters are the predefined parameters set before the model starts the learning process and tuning is finding the optimal hyperparameters to achieve the best performance or minimize the loss function. The standard methods widely used in hyperparameter tuning are manual or heuristic approach, grid search, random search and optimization-based searches. The heuristic approach is based on manually implemented based on the domain expertise of experts. Grid search is a brute force method where the entire search space is explored to find the best fit, whereas random search uses sampling techniques to split the search space for exploration. The optimization-based search involves different optimization methods to generate and adaptively adjust the parameters to optimize the model's performance over a loss function[56].

For the statistical method SARIMA, the hyperparameter encompasses the order of auto-regressive, moving average and differencing terms including the seasonal terms.

These are primarily determined by visually inspecting the correlogram and by doing an exhaustive grid search. For the auto-regressive terms, partial auto-correlation plot is inspected and for moving average terms auto-correlation plot is examined. In both plots, statistically significant orders which show high correlation are arbitrarily chosen. The other approach is to define a search space for the tuning process and do an exhaustive search; this is more resource-intensive [15].

As deep learning networks involve several hyperparameters, grid and random search become computationally resource-intensive and sometimes even infeasible due to the large parameter search space. This is where adaptive optimization-based tuning techniques have shown potential, making them widely used [57]. In contrast to grid search or random search, Bayesian optimization is a probabilistic model that takes previous validation scores in selecting the next set of hyperparameters for evaluation. As it makes well-informed decisions on what hyperparameters to test, it actively avoids regions that do not improve the performance of the objective function leading to more quicker search as it needs fewer evaluations to find the optimal parameters [56]. Bayesian search uses Bayes theorem to set out to find the global optimum of the objective function systematically.

The core of the optimization process consists of three components namely, the objective function, surrogate function and acquisition function. The objective function is the function that needs to be maximized or minimized and in this approach, the optimization process does not depend on the underlying factors influencing the objective function's value. In the context of forecasting models, this can be defined as one of the error metrics that needs to be minimized. The surrogate function is a critical part of the Bayesian process as it provides a probabilistic approximation of the objective function, guiding efficient future sampling. This is often modeled as a Gaussian Process (GP) due to their property of providing mean prediction and quantifying their variance. The GP model constructs a joint probability function over the variables which is the hyperparameter assuming a multivariate Gaussian distribution [58]. The acquisition function uses the mean and variance from the GP process to determine the next set of parameters to evaluate using the objection function. The commonly used acquisition functions are expected improvement (EI), probability of improvement (PI), and upper confidence bound (UCB).

Bayesian optimization starts with randomly sampling the search space and evaluating the objective function. With this sampling, a GP distribution is constructed as part of the surrogate function. The acquisition function uses the mean and variance derived from this function to choose the next set of parameters for evaluation. Following that, the output of the objective function is added to the GP process and this is iteratively executed to find the optimal parameters.

4

Methodology

This chapter elaborates on implementing the algorithms discussed in the previous chapter in the context of this particular study. The chapter discusses the data collection, pre-processing steps, evaluation metrics used, and results of the hyperparameter tuning. It then provides a detailed examination of how the environment and agent learning were configured for the reinforcement learning approach in this study.

4.1. Resources and Tools

In this research project, various data analyses and algorithm simulations were developed in Sypder IDE in Python 3.10.9. Several additional libraries were used on top of the built-in libraries for computation and visualization tasks such as pandas,numpy, and matplotlib. The machine-learning parts of the simulation were implemented using the TensorFlow framework, especially using the Keras backend. The RL models were set using object-oriented programming, and the gym library was utilized for the custom environment.

4.2. Data Collection

The dataset used in this project is data from a single household in the greater Lucerne region in Switzerland, which was retrieved from the IHomeLab RAPT dataset [11]. The data spans from 1st December 2016 to 31st July 2019, which is around two and a half years with a sampling frequency of 5 minutes. The dataset contains the local total load, consumption of individual appliances and the weather data from the local weather station. The electricity price data used in this analysis was taken from the ENTSOE transparency platform, a pan-European repository of energy-related data [59]. As the assumed pricing structure for the consumer in this study is dynamic pricing, to make the wholesale electricity prices from ENTSOE take transmission and distribution costs into account, the prices were multiplied by a proportional constant. Here, the proportional constant was obtained by comparing the wholesale retail price of the dynamic pricing structure in Switzerland. The schematic of the household connections is shown in Fig 4.1. This corresponds to house E in the group of house data available in the RAPT dataset [11]. The available weather data include indoor and outdoor humidity, indoor and outdoor temperature, and outside pressure. This would

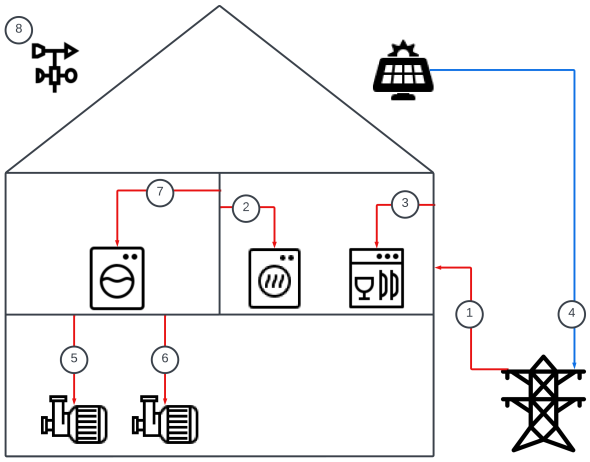


Figure 4.1: Schematic overview of the sub-metered appliances in the household data [11]

be more helpful when optimizing for HVAC systems, as maintaining thermal comfort is crucial. The description of the data is shown in Table 4.1, all the data is resampled to one hour from 5-minute resolution for ease of calculation and the unit for all the consumption and PV generation is in Kilowatts (kW).

Table 4.1: description of the dataset utilized

Number	Description
1	Total Power consumption
2	pc of dishwasher
3	pc of dryer
5	ps of heating gas pump
6	pc of water pump
7	pc of washing machine
8	Environmental data logged by weather station

4.3. Data Pre-processsing

Preparing the data before feeding it into a machine-learning model is crucial to achieving optimal knowledge discovery [60]. If there is redundant or irrelevant information present in the raw data, it could lead to sub-optimal performance so the output of data pre-processing is the final training set. Pre-processing includes missing data, cleaning, normalization, and transformation.

4.3.1. Missing Data

The raw data set is prone to missing data or invalid values, which could lead to spurious output. Therefore, it should be at the outset. As with time series, removing missing values gives unreliable results as the data has a temporal relationship, hence imputation has to be done. The missing values could be imputed using linear interpolation, back-fill, front-fill, seasonal fill, or regression [61]. In this analysis, the data used from

[11] has already dealt with the missing values using linear interpolation. However, the electricity price dataset contained empty values which were imputed using.

Linear interpolation is a method to fill missing data in a time series assuming a linear relationship between known data points and it also assumes the step size to be uniform. One of the significant drawbacks of using linear interpolation is that it needs to include underlying seasonalities and patterns and the assumption of linearity, which could lead to distortion of the true nature of the time series. In this analysis, linear interpolation was deployed to deal with missing data.

4.3.2. Stationarity

A time series comprises three components: trend, seasonality and noise. Trend represents the long-term movement or direction of the data. This indicates whether the data is increasing or decreasing or stable over time. Next, seasonality is the regular repeating pattern at regular intervals and in the context of load, it could be daily, weekly and yearly seasonality. Finally, noise is the random fluctuations or irregularities in the data that cannot be captured by trend or seasonality. The data is said to be stationary if the properties do not change over time. There is no trend or seasonality as they will affect the value of the time series at different times. In practice, most of the data is non-stationary and even after transforming it into stationary data it could be observed that there is some degree of degree of trend or seasonality which is acceptable.

Most statistical forecasting models, especially in this context SARIMA, have an underlying assumption that the data is stationary. Although neural network methods can learn the abstraction in non-stationary data, it is generally recommended to make it stationary for it as well. However, during the analysis it was observed that making the data stationary before feeding the machine-learning model did not improve accuracy; instead reduced it. Possible reasons for this observation could be that differencing may lead to loss of information. Consequently, linearizing the data limits the potential of neural networks to learn non-linear information in the data.

The most common practice to evaluate the stationarity of the data is by visually inspecting auto-correlation plot (ACF), partial auto-correlation plot (PACF) or using unit root tests such as Augmented Dicky-Fuller (ADF) test or Kwiatkowski-Philips-Schmidt-Shin (KPSS) test. The unit tests (ADF/KPSS) are statistical significance tests which are hypothesis tests that will give results in null and alternative hypotheses. With the p-value obtained from the tests, we will need to infer whether the time series is stationary. In this analysis, the ADF test was carried out using the Adfuller method in the statsmodels package. The results were also visually checked by observing the ACF and PACF plots.

The most common method to achieve stationarity is by differencing the data, the trend could be stabilized by differencing with the previous value and seasonal differencing to remove seasonalities in the data. This could also be achieved using decomposition methods such as STL and MSTL where the time series is decomposed into trends, seasonalities and noise. Log transformation could be used to stabilize variance in the data which also distorts the stationarity of the data[61].

In this analysis, the data was made stationary only when modeling SARIMA as they cannot handle non-linearity. The trend and daily seasonality were removed using differencing and the stationarity was checked using the ADF test.

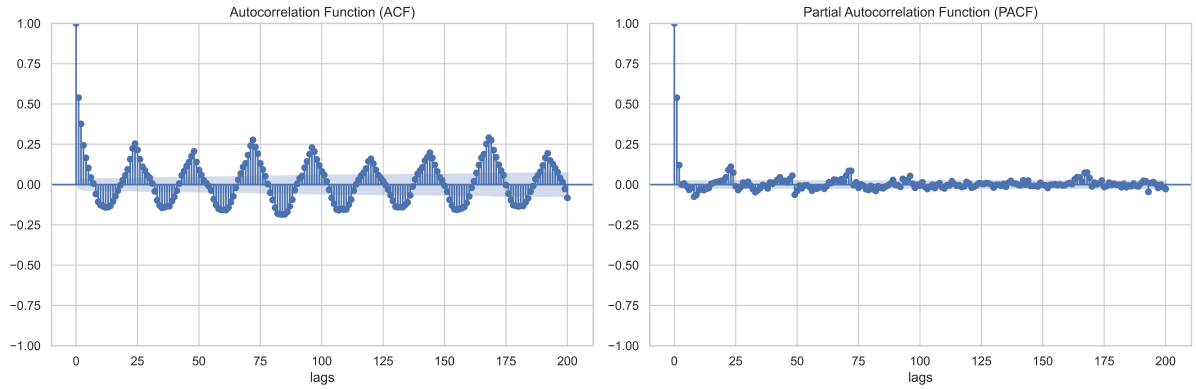


Figure 4.2: Load Correlogram

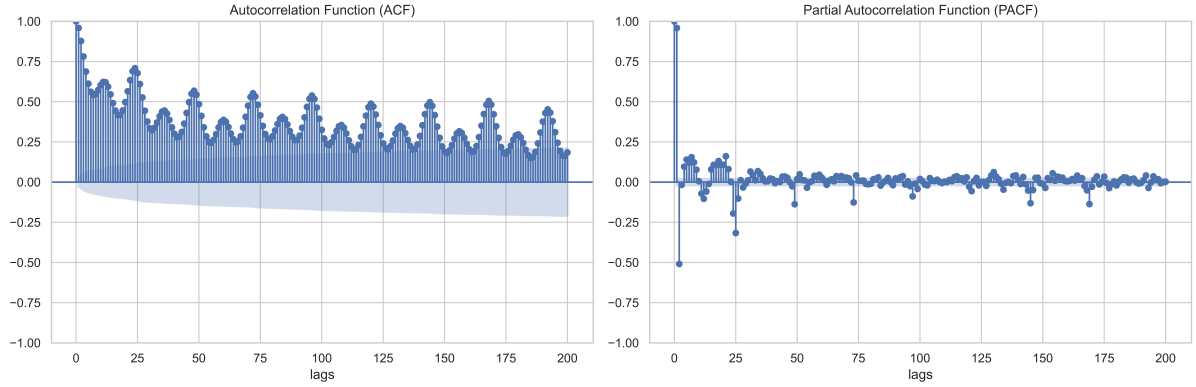


Figure 4.3: Electricity Price Correlogram

4.3.3. Feature selection

The selection of relevant features is one of the central concepts in machine learning and hugely impacts the performance of the prediction models. Feature/Attribute selection is a problem of identifying the related features that contribute to the target variable to achieve better accuracy for the model. Fewer features reduce the complexity of the models and increase the interpretability.

This study uses the Pearson correlation to extract relevant features under Filter methods. Filter methods offer a high-level understanding of feature relevance, making them an initial step in feature engineering. Pearson correlation is a statistical term that measures the linear relationship between variables. It is denoted as r and ranges from -1 to 1, with -1 indicating the most negative/inverse relationship between the variables, whereas 1 indicates a positive relationship. Since the input data from the iHome dataset has the consumption of individual appliances' indoor and outdoor meteorological data. Therefore, it becomes crucial to choose the relevant features to avoid overfitting. The heat map shown in Figure.4.5 illustrates the correlation between the target and predictor variables, where the correlation intensity could be inferred from the color bar.

Calendar attributes are also used in all three target variables as they have daily, weekly and monthly periodicity and the lag values which are previous values. Here, the number of lag values or look back is a hyperparameter and due to the strong

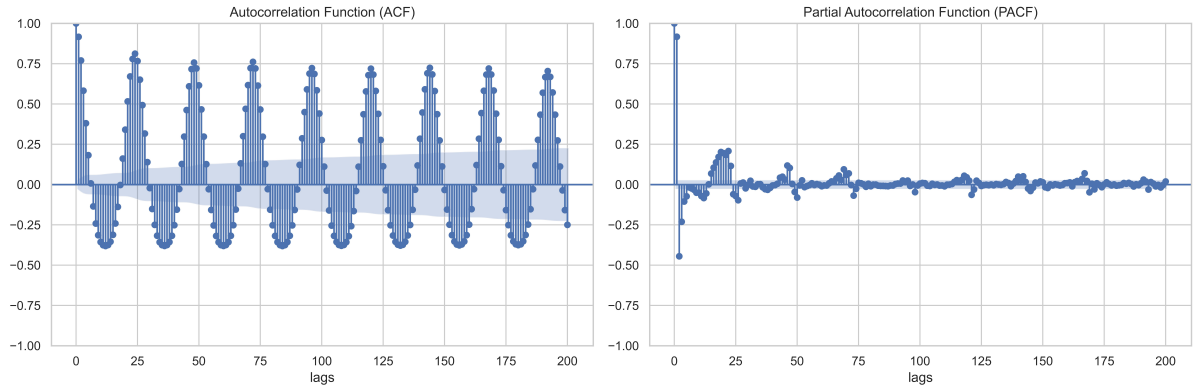


Figure 4.4: PV Generation Correlogram

Table 4.2: Input features for machine learning forecasting models

Load	PV Generation	Electricity Price
Lags (previous values)	Lags (previous values)	Lags (previous values)
Hour of the day	Hour of the day	hour of the day
Day of week	Day of week	Day of week
Month	Month	Month
Dryer Load	Outside Humidity	
Outside Temperature	Outside Temperature	
Outside Humidity	Outside Pressure	

diurnal pattern in the target variables, this was assumed to be 24 so it uses the past 24 hours value to predict the next one. Therefore, the highly correlated values from the heat map Figure.4.5, calendar attributes and the lag values make the forecasting task features described in Table.4.2. The correlation measured here is based on linear assumptions, whereas the data has non-linear relationships, but this is only used as an indicator.

4.3.4. Feature scaling

Feature scaling is a critical pre-processing step in machine learning models that aims to normalize or standardize the input features before feeding them into deep learning models. Most supervised and unsupervised learning models make decisions according to the distance between data points, particularly Euclidean distance. When different features are in different scales, it might take the algorithm longer to converge as the model has to learn the abstraction or affect model performance. The most common and effective ways are normalization or standardization to bring all the input features to a uniform scale to make the learning efficient [61].

In statistics, normalization is a method of re-scaling the data between 0 and 1. Here, the minimum value of any feature gets converted into 0 and the largest value into 1. Some drawbacks are not being robust to outliers, which may distort the pre-

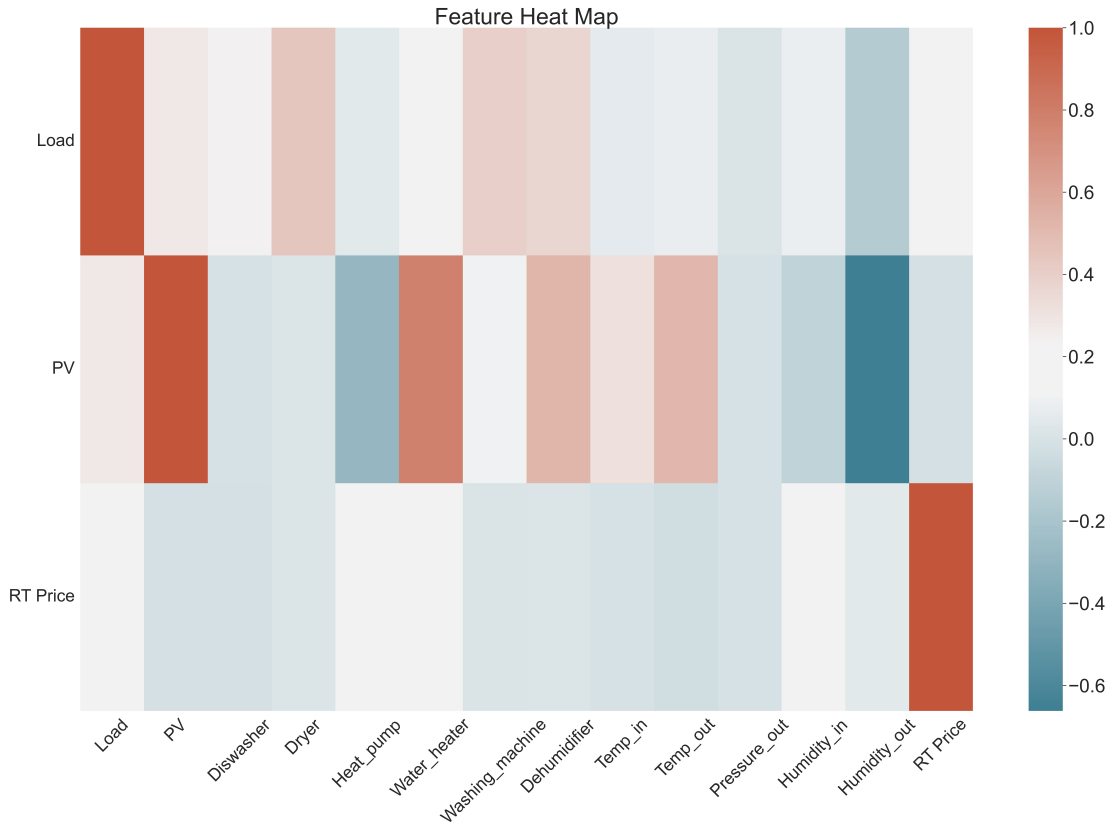


Figure 4.5: Heatmap displaying the correlation matrix for all the features

dictions and loss of information when scaling and inverse scaling. Another approach is standardization, where the data is transformed to have a mean of zero and a standard deviation of one. This centers the data around zero which makes the model learn efficiently. Similar to normalization, this does not preserve the initial distribution of data but is more robust to outliers. The mathematical equations for normalization and standardization are Equation 4.1 and 4.2, respectively.

$$X_{\text{norm}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}} \tag{4.1}$$

$$X_{\text{stand}} = \frac{X - \text{Mean}}{\text{Standard deviation}} \tag{4.2}$$

In this analysis, forecasting was simulated without any scaling as neural networks can learn complex patterns after normalization and standardizing. It was observed that the model performed better after normalizing the data.

4.3.5. Data Preparation

The choice of neural network architecture heavily influences the required input shape. Keras, the deep learning framework, offers diverse architectures for time series forecasting and requires distinct input shapes. After the raw data has been pre-processed as discussed in the earlier steps and relevant features have been selected, the data is ready for reshaping to accommodate various neural network architectures.

In supervised learning, a sample consists of input features (Independent variable) and a target variable (Dependent variable). The input shape of features is discussed here and the target is a scalar value or a single-dimensional vector, depending on the forecasting horizon. Since this analysis focuses on batch learning, the network is updated depending on the batch size which is the concatenation of samples. An overview of the input shapes for different NN architectures is described in Table 4.3.

For MLP, the input shape is (number of samples, number of features). The number of features includes the lag values as each feature and other independent variables as mentioned in Table 4.2. In the context of the other architectures, there is an inherent structure to model temporal dependencies which translates into time steps referred to in Table 4.3 which is a hyperparameter that considers the chosen period of lag values. For CNN-LSTM hybrid, the capability of CNN to extract features from data is leveraged which is then fed into the LSTM layer so the input sample is broken down into sub-sequences to extract feature information. For instance, if the look back is 24 time steps this could be broken into three sub-sequences of 8 time steps.

Table 4.3: Input shapes for Neural Networks

Neural Network architecture	**Input shape**
Multilayer perceptron (MLP)	(samples,features)
Convolutional Neural Network (CNN)	(samples, time steps, features)
Recurrent Neural Network (RNN)	(samples, time steps, features)
Long short-term memory (LSTM)	(samples, time steps, features)
Bi-directional LSTM	(samples, time steps, features)
CNN-LSTM Hybrid	(samples, sub-sequences, time steps,features)

4.4. Energy management systems

In this analysis, the agent was trained on data for one month (May 2019) and tested the following week from 1st June to 7th June 2019. The limited training and testing horizons are because of the computational resource-intensive learning process of the RL agent. Training the agent for a year would require at least 150,000 training episodes to attain reasonable performance. This also limits fine-tuning opportunities as it takes longer for iterations.

4.4.1. Pricing structure

The cost of electricity used in this analysis is based on dynamic pricing, the Switzerland day-ahead market price is adjusted for the retail market by using a scaling factor of 4. As the market currently does not provide dynamic pricing for the retail market, the scaling factor was assumed based on the observation that the wholesale market price is on average 4-5 times lower than what end consumers pay. Renewable energy exchanges with the grid occur under a net-metering system where the consumer

receives credit for any excess electricity they generate and feeds back into the grid. Under dynamic pricing, the electricity sent back into the grid is paid the same amount as the corresponding price at that hour. Net metering incentivizes consumers to adopt energy storage technology to store surplus energy, sell it back to the grid during peak times, and promote consumption at source.

4.4.2. Battery system sizing

The optimal sizing of a battery system widely varies depending on the system specifications and the chosen cost function. However, This study does not primarily focus on determining the optimal battery energy storage system (BESS) size. Instead, the main objective is to develop a control algorithm that maximizes financial benefits for end consumers. Therefore, for this study, it is reasonably assumed that the BESS sizing has been predetermined. The assumed battery capacity is based on optimizing a similar load profile with comparable average daily consumption. The specification of the chosen battery is illustrated in Table.4.4

Table 4.4: Battery sizing specifications

Parameter	**Value**
Battery Capacity	6.6 kWh
Maximum charging power	2.8 kW
Maximum discharging power	2.8 kW
Charging efficiency	90%
Discharging efficiency	90%
Initial SOC	0.6

4.5. Reinforcement learning

The main goal of RL problem setup is to enable stable and efficient learning of HEMS agents. This is achieved by creating an easily interpretable environment and reward setting for the agent. The schematic representation Fig.4.6 shows an overview of the learning process and the state spaces employed. Here, the problem is formulated as a partially observable markov decision process (POMDP) as the agent does not have the complete knowledge of the environment. POMDP is an extension of MDP where the agent is fed with a set of observations and conditional observation probabilities.

4.5.1. Environment

The custom environment for the agent to learn uses an Open AI gym environment, providing a standardized interface, easy reproducibility and sharing. The action and state spaces are defined to be continuous with their respective lower and higher bounds. For each episode, a day is randomly chosen in May 2019, where the agent iterates through each time step to maximize the cumulative reward for the day. At each state transition from state S_t to the next state S_{t+1}, the state of charge is updated using the battery model as shown in Equation 4.5. The run-time observations also include the

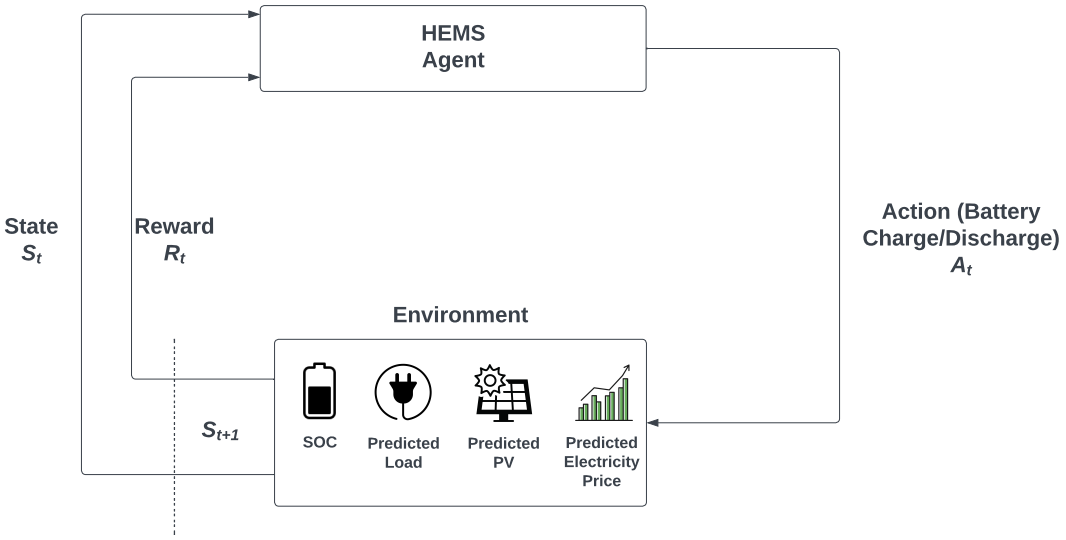


Figure 4.6: Schematic Representation of the RL problem setup

maximum battery capacity that could be charged/discharged at time t, which is shown in represented in Equation 4.3 and Equation 4.4

$$pb_{in,t} = min\left(pb_{min}, E_b * (soc_{max} - soc_t)\right) \tag{4.3}$$

$$pb_{out,t} = min\left(pb_{max}, E_b * (soc_t - soc_{min})\right) \tag{4.4}$$

$$soc_{t+1} = soc_t + (Bat_{\text{action},t}/E_b) \tag{4.5}$$

During the offline learning phase, the state transition step returns individual reward components and other power flow elements for further analysis. The custom environment is designed to terminate each episode after 24 steps and reassign the next random day. While the environment is resetting after each episode, the state of charge is randomly initialized within its limits, and other historical observations are set to their respective values from the previous day. The HEMS agent is tested chronologically on the data from 1st June to 7th June 2019.

4.5.2. Observation space

A fundamental element in agent-environment interaction is the observation space, which determines how the agent perceives the environment and takes action accordingly. Understanding and choosing the state space is crucial to the RL problem. These were chosen heuristically for this analysis based on their influence on the actions and the summary of all the observations illustrated in Table.4.5. It should be noted that the forecasted values from the forecasting part of the study are used here.

In Table 4.5, it could be seen that observations for this problem could be classified as index attributes that reset after every episode, forecasted values that are retrieved from a CVS file and run time calculation which follows an internal model to compute the next observation.

Table 4.5: description of the observations used in the RL environment

Category	Name	Source	Unit
Calendar	Hour	Index attribute	0 to 23 hours
Total Consumption	Load predicted 1h	Forecasted values csv	KW
Total Consumption	Load predicted 6h	Forecasted values csv	KW
Total Consumption	Load predicted 12h	Forecasted values csv	KW
Total Consumption	Load predicted 24h	Forecasted values csv	KW
Electricity price	Price predicted 1h	Forecasted values csv	Euro/Kwh
Electricity price	Price predicted 6h	Forecasted values csv	Euro/Kwh
Electricity price	Price predicted 12h	Forecasted values csv	Euro/Kwh
Electricity price	Price predicted 24h	Forecasted values csv	Euro/Kwh
PV Generation	Price PV Generation 1h	Forecasted values csv	KW
PV Generation	Price PV Generation 3h	Forecasted values csv	KW
PV Generation	Price PV Generation 6h	Forecasted values csv	KW
State of Charge	SOC	Runtime calculation	Between 0 to 1

4.5.3. Reward setting

Reward setting is a critical part of RL and guides the agent in learning the best policy, so it is essential to set the rewards so the agent can learn effectively. The main objective of this strategy is that the more rewards the agents earn, the more the overall benefit for the agent. This could also be seen as a multi-objective optimization with the primary reward broken down into components that reflect each goal. In this analysis, the cost function is to minimize the operating cost of energy while operating the battery under its physical limits and to avoid simultaneous charge and discharge when two intelligent agents interact. The reward function is split into five components and each is multiplied with a coefficient to set the importance of each reward on the policy.

Having the reward wholly positive or negative is good practice as it would facilitate more stable learning. Here, all the rewards are negative as the agent tries to maximize its reward, which translates to minimizing the energy cost. The weight for each reward component is ξ. To prolong battery life and reduce degradation, the state of charge of the batteries is operated under certain limits. Since a model-free RL algorithm is implemented here, the agent has to learn the battery dynamics which is signaled through rewards (r_1, r_2) as seen in Equation 4.6 and 4.7. Suppose the state of charge limits are breached. In that case, the agents receive a penalty corresponding to the deviation according to weight, which makes the agent make decisions under the constraint over the learning process.

$$r_1 = \begin{cases} -\xi_1 * abs(soc_{min} - soc) & if soc < soc_{min} \\ 0 & if soc < soc_{min} \end{cases} \tag{4.6}$$

$$r_1 = \begin{cases} -\xi_2 * abs(soc - soc_{max}) & if soc > soc_{max} \\ 0 & if soc > soc_{max} \end{cases} \tag{4.7}$$

The primary reward component to minimize the energy cost is r_3, where the net electricity cost for that period (hourly) is calculated by multiplying the energy arbitrage and real-time electricity price as seen in Equation 4.8.

$$r_3 = -\xi_3 * (P_p - P_s) * Pr_t \tag{4.8}$$

As the problem is formulated as a multi-agent RL problem, when agents 1 and 2 are making charging/discharging battery decisions as shown in Fig.4.7, it is prone to violate battery constraints and operate in a charging and discharging state simultaneously. To avoid the battery charging and discharging simultaneously, r_4 is introduced where agent 2 is penalized if it charges as in the problem definition. It is allowed to only discharge to the load. Here, r_5 keeps the battery's maximum charge and discharge limits at check and penalizes proportionally if it breaches the constraints. It should noted that all the rewards here are modeled to be instantaneous and proportional instead of sparse rewards which is effective in some cases but needs more training episodes to learn.

$$r_4 = \begin{cases} \xi_4 * a_2 & if a_2 < 0 \\ 0 & if a_2 > 0 \end{cases} \tag{4.9}$$

$$r_5 = \begin{cases} -\xi_5 * (abs(a_1 + a_2) - pb_{max}) & if abs(a_1 + a_2) < 0 \\ 0 & if a_2 > 0 \end{cases} \tag{4.10}$$

At every state transition and step the agent takes, all the five reward components are computed separately and the cumulative value is sent back as a feedback signal to the agent.

$$r_t = r_1 + r_2 + r_3 + r_4 + r_5 \tag{4.11}$$

4.5.4. Action Exploration Strategy

The agent must explore the continuous action space effectively to learn the optimal policy. One of the advantages of using off-policy methods such as Q-learning and DDPG is that the exploration process is independent of the RL algorithm. For DQN, most commonly, epsilon greedy exploration strategy is used where it sets the balance between exploration and exploitation but for continuous action this could not be implemented; hence, a noise is added to the action. The exploration policy could be defined as,

$$a_t = \mu(s_t|\theta^\mu) + N \tag{4.12}$$

Here, the noise process N chosen was the Ornstein-Uhlenbeck process [62] as used in the original implementation [39], OU process generates temporally correlated noise which enables smoother learning and follows a realistic exploration strategy. Gaussian noise was implemented and the performance observed was relatively lower

than OU noise. For both noise processes, the standard deviation was progressively reduced as the learning process advanced to facilitate narrowing down on exploration and fixing an optimal policy.

4.5.5. Multi-agent DDPG

The RL problem in this study is formulated with two independent, intelligent agents making coordinated decisions to maximize the cumulative reward. The main reason behind choosing a multi-agent framework for this problem is to avoid simultaneous discharging and discharging from the battery. In a single-agent scenario, when the battery is discharged, it becomes ambiguous if it has been sent to satisfy the load or exchanged back to the grid. A centralized approach is taken here as sharing global state space information allows for optimal coordination. Another benefit is that it guarantees coordination and has a more straightforward implementation.

The outline of agent interaction in this analysis is shown in Fig 4.7 where a_1 and a_2 are the agents. Here, a_1 has an action space of $[-2.8, +2.8]$ where it deals with grid exchange (charging and discharging) whereas a_2 only discharges to the load. The rest of the power flow elements are calculated based on these actions and it is also important to note that the flow from PV is set by deterministic rules which give precedence to the load then to charging the battery and if there is still some excess after the battery is complete, then it is sent back to the grid.

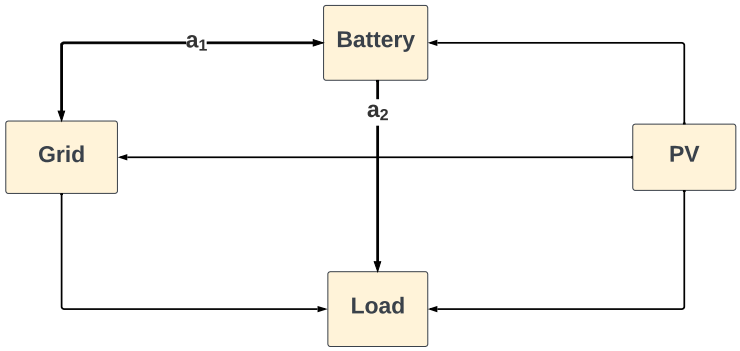


Figure 4.7: Representation of Multi-agent interaction in DDPG algorithm

The DDPG agent in this analysis is implemented closely similar to the original architecture, the pseudo-code [39]. The agent learning starts with initializing the main actor and critic network of each agent a_1 and a_2, this is done by the default initializer provided in Keras which is Xavier Uniform initialization. Subsequently, the weights of the respective target networks are initialized by duplicating the weights from the main network. The replay buffer is then initialized separately for the two agents, the buffer stores the state transitions (state, action, cumulative reward, next state). Here, both agent's policy is shaped using the same cumulative reward which coordinates the actions of both agents to maximize the cumulative reward. The Ornstein-Uhlenbeck exploration noise is initialized for each agent with different initial standard deviations.

The main training loop runs for M episodes, and the initial state is reset at the start of each episode. The initial state is reset from the predetermined forecasted values and with a random Soc value in the operational range. The episode length is 24 so at

each episode the inner loops for the length of one episode before reaching a terminal and resetting the initial state. In the inner loop, both agents take action based on their current policy, which is added to the OU noise for exploration. To ensure that the BESS does not charge and discharge simultaneously if action a_{t1} is positive, BESS charges from the grid so the discharge to the load is restricted by assigning $a_{t2} = 0$. The state transitions are stored in the replay buffer individually for each action with the same cumulative reward.

Table 4.6: Pseudo algorithm for the proposed DDPG agent training

DDPG algorithm

Randomly initialize critic networks $Q_1(s, a|\theta_\mu)$, $Q_2(s, a|\theta_\mu)$ and actors $\mu_1(s|\theta_\mu)$, $\mu_2(s|\theta_\mu)$ with weights $\theta_{Q_1}, \theta_{Q_2}\, and\, \theta_{\tau_1}, \theta_{\tau_2}$
Initialize target network Q'_1, Q'_2 and μ'_1 , $\mu'_2$with weights $\theta_{Q'_1} \leftarrow \theta_{Q_1}, \theta_{\tau'_1} \leftarrow \theta_{1_{\tau_1}}\, and\, \theta_{Q'_2} \leftarrow \theta_{Q_2}, \theta_{\tau'_2} \leftarrow \theta_{2_{\tau_2}}$
Initialize replay buffer R_1 and R_2
Initialize a random process N_1, N_2 for action exploration
for episode = 1, M **do**
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_{t1} = \mu(s_t|\theta_\mu) + N_{t_1}$, $a_{t2} = \mu(s_t|\theta_\mu) + N_{t_2}$ according to the current policy and exploration noise
    if a_{t1} > 0, a_{t2}=0
    Store transition $(s_t, a_{t1}, r_t, s_{t+1})$ in R_1
    Store transition $(s_t, a_{t2}, r_t, s_{t+1})$ in R_2
    Sample a random mini-batch of N_1 transitions $(s_i, a_{i,1}, r_i, s_{i+1})$ from R_1
    Sample a random mini-batch of N_2 transitions $(s_i, a_{i,2}, r_i, s_{i+1})$ from R_2
    Set $y_{i,1} = r_i + \gamma Q'_1((s_{i+1}), \mu'_1(s_{i+1}|\theta_{\mu'_1})|\theta_{Q'_1})$
    Set $y_{i,2} = r_i + \gamma Q'_2((s_{i+1}), \mu'_2(s_{i+1}|\theta_{\mu'_2})|\theta_{Q'_2})$
    Update critic networks by minimizing the loss $L_1 = \frac{1}{n_1}\sum_i (y_{i,1} - Q_1(s_i, a_{i,1}|\theta_{Q_1}))^2$
    $L_2 = \frac{1}{n_2}\sum_i (y_{i,2} - Q_2(s_i, a_{i,2}|\theta_{Q_2}))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta_{\mu_1}} J_1 \approx \frac{1}{n_1}\sum_i \nabla_a Q_1(s, a|\theta_{Q_1})|_{s=s_i, a=\mu_1(s_i)} \nabla_{\theta_\mu} \mu_1(s, \theta_{\mu_1})|_{s_i}$$

$$\nabla_{\theta_{\mu_2}} J_2 \approx \frac{1}{n_2}\sum_i \nabla_a Q_2(s, a|\theta_{Q_2})|_{s=s_i, a=\mu_2(s_i)} \nabla_{\theta_\mu} \mu_2(s, \theta_{\mu_2})|_{s_i}$$

    Update the target networks:

$$\theta_{Q'_1} \leftarrow \tau\theta_{Q_1} + (1-\tau)\theta_{Q'_1},\ \theta_{Q'_2} \leftarrow \tau\theta_{Q_2} + (1-\tau)\theta_{Q'_2}$$
$$\theta_{\mu'_1} \leftarrow \tau\theta_{\mu_1} + (1-\tau)\theta_{\mu'_1},\ \theta_{\mu'_2} \leftarrow \tau\theta_{\mu_2} + (1-\tau)\theta_{\mu'_2}$$

  end for
end for

Since a batch learning approach is taken, all four neural networks involved in this implementation are updated on mini-batches randomly sampled from the replay buffer. Firstly, the main critic networks estimate the cumulative reward based on their current

policy, which includes the immediate reward of r. The temporal differencing error is calculated and the neural networks are optimized by minimizing the mean squared error (MSE). TD error is propagated to the actor network, where it is updated by multiplying with the gradient of the actor. Finally, the actor and critic target networks are soft-updated for stable learning proportionally to τ.

4.6. Hyperparameter tuning

4.6.1. Forecasting

For SARIMA, the auto-regressive, differencing and moving average terms and their seasonal equivalents were first analyzed using a correlogram individually for each forecasting task as seen in Fig 4.2,4.3 and 4.4. The auto-regressive terms were inferred from a partial auto-correlation plot, whereas the moving average terms were inferred from an auto-correlation plot and the differencing terms were estimated by observing the seasonality in the data. After the initial search parameter trial, search space was extended and a grid search was performed using different combinations of parameters while minimizing the RMSE. The resulting terms for each forecasting task are shown in Table 4.7.

Table 4.7: SARIMA Parameters

Forecast	p	d	q	P	D	Q
Load	2	1	1	3	0	2
PV Generation	3	2	2	3	0	1
Price	2	2	1	3	1	1

Choosing the optimal hyperparameters becomes more critical and problem-specific for neural networks, given the higher number of hyperparameters to tune. As mentioned in the section, a simple Bayesian optimization was performed and due to the computational constraints the search space was limited to as shown in Table.4.8.

Table 4.8: Hyperparameter search ranges for Deep Learning Models

Hyperparameter	Range
Number of hidden layers	[1,2,3]
Number of neurons in hidden layers	(8,120)
Learning rate	(0.00001,0.001)
Dropout ration	(0,0.5)
Number of epochs	(40,200)
Batch size	32
Look-back period	24

Table 4.9: Neural Network configurations for the comparative assessment

Model	Load	PV Generation	Electricity Price
Multilayer Perceptron (MLP)	Dense Layer (60) Dropout Layer (0.18) Dense Layer (40) Dropout Layer(0.18) Epochs = 80	Dense Layer (120) Dense Layer (80) Epochs = 40	Dense Layer (90) Dropout (0.1779) Dense Layer (20) Dropout (0.1779) Epochs = 80
Convolutional Neural Network (CNN)	Filters = 7 Kernel Size =3 Dense Layer (10) Epochs = 100	Filters = 16 Kernel size = 4 Dense Layer = 10 Epochs = 100	Filters = 16 Kernel Size= 4 Dense Layer (10) Epochs = 100
Recurrent Neural Network (RNN)	SimpleRNN (64) Dropout Layer (0.4) Epochs = 80	SimpleRNN (64) Dense Layer = 8 Epochs = 70	SimpleRNN (64) Dense Layer (8) Epochs = 70
Long Short Term Network (LSTM)	LSTM Layer (64) Dropout (0.1) Epochs = 100	LSTM Layer (64) Dense (8) Epochs = 50	LSTM Layer (64) Dropout (0.3) Epochs = 50
CNN-LSTM Hybrid	Filters = 7 Kernel Size = 4 Dropout Layer (0.2) LSTM Layer (24) Dense Layer (6) Epochs = 150	Filters = 12 Kernel Size = 3 LSTM Layer (24) Dense Layer (6) Epochs = 150	Lr = 0.00001 Filters = 12 Kernel Size = 3 LSTM Layer (24) Dense Layer (6) Epochs = 150
Bidirectional LSTM	BiLSTM Layer (64) Dense Layer (24) Dropout (0.1) Epochs = 50	BiLSTM Layer (64) Dense Layer (32) Dropout (0.1) Epochs = 50	BiLSTM Layer (32) Dropout (0.1) Epochs = 50

The search space and some hyperparameters were not tuned, such as the look-back period chosen as 24 because of the previous day's influence on the future value. After the Bayesian search, further tuning was done by analyzing the training vs. validation loss curve to achieve better convergence, yielding better results. It should also be noted that different problems were approached with different initial numbers of layers and then tuned individually.

The ReLU activation function was chosen for the hidden layers as it mitigates the vanishing gradient problem experienced when using sigmoid or tanh activation functions. They are also computationally efficient with implementing non-linearity, which is essential in modeling complex relationships in data. They also come with limitations such as dying ReLU problems when they constantly output zero due to gradient di-

rection and unbounded activation. For the output layer, the required output is a real number given the regression problem; hence, a linear activation function is used. The loss function chosen to minimize the gradient is root mean squared error (RMSE), as it penalizes significant errors and is also the primary metric for evaluating among other neural network architectures. The activation functions are pre-defined in the keras package, whereas the RMSE loss function was custom-implemented.

Adam optimizer is used in this analysis to minimize the loss function, a popular optimization technique used in neural network training. Adam optimizer is an extension of the stochastic gradient descent (SGD) optimization method with an adaptive learning rate, which means that the learning rate is adjusted according to the gradient information throughout the learning process. Since this is adaptive, it increases efficiency, reduces the need for manually tuning learning rates, and is robust to initialization. However, this optimizer is prone to overfitting which was addressed using dropout layers.

The final tuning process for individual tasks yielded the configurations presented in Table.4.9. It was determined that all the architectures had better performances when the learning rate was 0.0001,

4.6.2. Reinforcement learning

The core of RL lies in using neural networks as function approximators and how they interact with each other. As with neural networks used in forecasting tasks, each network in this analysis, namely actor and critic, has its main and target network, resulting in four networks. Using hyperparameter tuning methods for RL is limited by computational and time constraints as several moving parts exist. In this analysis, initial hyperparameters are set heuristically based on the complexity of the state space and then fine-tuned heuristically after inferring from the networks' loss curves and the episode rewards over the learning duration. It should also be noted that the learning rate of the critic network is lower than the learning rate of the actor-network to help stabilize learning.

Table 4.10: System parameters for DDPG model

Parameter	**Value**
Length of each episode	24
Number of episodes	10000
Discount rate	0.99
Update rate of target network (τ)	0.05
Buffer size	50000

Apart from the function approximators, the discount rate influences the agent's performance sequentially. Hence, it was taken as 0.99 as all the time steps contribute to the episodic reward. Here, the target networks are soft-updated, incorporating τ (update rate of target network) for both actor and critic networks. Here, it was set as 0.05, which would update the target network gradually improving convergence in learning

and maintaining stability in training. The number of episodes is also a determining factor in how good the learned policy is.The algorithm uses checkpoints to save the best-performing models throughout the learning episodes, preventing overfitting of the policy and ensuring better generalization. The overview of the RL system parameters is shown in Table.4.10

The length of each episode was chosen to be 24 as it allows the agent to learn optimal policies to minimize the cost per day, which could be extrapolated to several days when implemented. Similar network architectures were utilized for both actor and critic networks which is illustrated in Fig.4.11

Table 4.11: Actor and Critic Network configuration

Parameter	Value
Number of hidden layers	3
Neurons in each layer	(64,124,32)
Batch size	32
Actor-network learning rate	0.00005
Critic-network learning rate	0.00001

4.6.3. Benchmark Model

As discussed in the sub-section 3.2.2, it is essential to fine-tune hyperparameters X_1 and X_2 in order to achieve optimal performance. The hyperparameters were determined by minimizing the base electricity cost which is the chosen loss function for one month (May 2019). Given the small hyperparameter search and to execute an exhaustive search over all possible combinations, a grid search was conducted. The optimization is visually shown in Fig.4.8

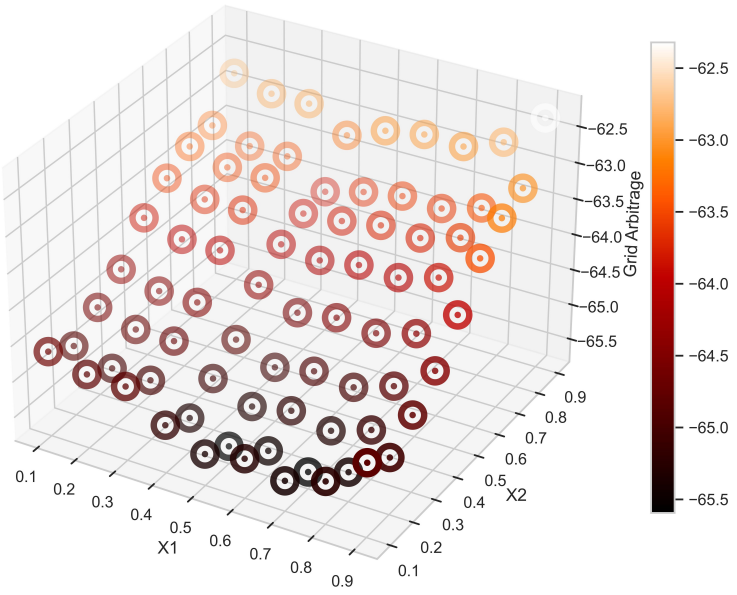


Figure 4.8: Parameter Tuning visualization for Benchmark EMS model

From the results, the base electricity cost reaches a minimal value with increasing X_1 and a decreasing X_2. In Fig.4.8, the base electricity cost is negative due to more PV generation than electricity consumption. Here it could also be instinctively observed that charging the battery from the grid is desirable even at relatively higher price ranges. However, the low X_2 value indicates that selling it back to the grid is profitable only at higher price levels.

4.7. Performance Metrics

4.7.1. Forecasting

The accuracy of forecasting models is typically assessed using scale-dependent error metrics like root mean squared error (RMSE) and mean average error (MAE) [63]. RMSE is particularly effective as it places greater weight on large errors and can handle values close to zero. Since they are scale-dependent, they cannot be used to compare different datasets. R-squared (R^2) is a statistical measure representing the proportion of the variance in predicted values explained by the true values. Equation 4.13,4.14 and 4.15 shows the mathematical representation

Mean Absolute Error (MAE)

$$MAE = \frac{1}{n}\sum |\hat{\mathbf{y}}_\mathbf{t} - \mathbf{y}_\mathbf{t}| \tag{4.13}$$

Root Mean Squared Error (RMSE)

$$RMSE = \sqrt{\frac{1}{n}\sum |\hat{\mathbf{y}}_\mathbf{t} - \mathbf{y}_\mathbf{t}|^\mathbf{2}} \tag{4.14}$$

R-Squared

$$R^2 = 1 - \frac{\sum(\mathbf{y}_\mathbf{i} - \hat{\mathbf{y}})^2}{\sum(\mathbf{y}_\mathbf{i} - \bar{\mathbf{y}})^2} \tag{4.15}$$

In the above equations, $\hat{\mathbf{y}}_\mathbf{t}$ is the predicted value and y is the actual value. A combination of metrics is needed to assess model accuracy and inspect different forecast aspects. In this study, RMSE is chosen as the metric over which models are optimized and for evaluation.

A holdout validation was undertaken for evaluation where the dataset was split into 70% for training, 20% for validation and 10% for testing without shuffling of data points to preserve the sequential aspect of the dataset.

4.7.2. Energy Management Systems

Several statistics have been considered to analyze the performance of the RL-based EMS against the baseline and benchmark. The metrics calculated are used to analyze the benefits gained by the grid operator and the customer where it examines grid congestion, battery life, and cost analysis. The chosen metrics are discussed below.

Base electricity cost

The primary financial aspect defined here is the base electricity cost which is the energy bill that the customer faces. This cost does not consider the additional costs of the BESS and PV. With the net metering option, this becomes the sum of the net cost of electricity per hour.

$$Base\ Electricity\ Cost = \sum_{t=0}^{T}(P_{p,t} - P_{s,t}) * Pr_t \tag{4.16}$$

Total electricity cost

In this study, total electricity cost is defined as the sum of the base electricity cost and the cost of BESS operation. This cost does not take the cost of the PV system which is assumed to be constant across all three models discussed in this research. BESS operational cost is estimated by multiplying the count of cycles in the test period and the cost of a single charge/discharge cycle. From the information from the manufacturers about the rated number of cycles and total cost of BESS, it was determined that the cost per cycle is 0.42 Euro/cycle.

$$\text{Total Electricity Cost} = \text{Base Electricity Cost} + \text{No of cycles} \times \text{Cost per cycle} \tag{4.17}$$

Count of battery cycles

The battery life deterioration is assessed by counting the number of complete charge/discharge cycles within the depth of discharge limits during the testing period. This is calculated by summing all the charging SoC changes over the testing period and dividing it by the operating depth of discharge. The BESS in this study operates between SoC limits of 0.2 and 0.8, making the depth of discharge 0.6.

$$\text{Cycle count} = \frac{\sum_{t=0}^{T} \text{Positive } \Delta SoC_t}{\text{Depth of discharge}} \tag{4.18}$$

Load factor

Load factor could be defined as the ratio between the average load to the peak load in a given period. This period could be a day, week or month. This is a widely used metric to measure the efficiency of energy consumption. A high load factor indicates the consumption pattern is consistent and stable whereas a low load factor indicates more fluctuations in consumption. This study chose the daily period as it offers granularity and a real-time view of the consumption pattern to make better decisions. The equation for it is presented in Equation 4.19

$$LF = \frac{\text{Average load in a day}}{\text{Peak load of the day}} \tag{4.19}$$

Daily peak power consumption

The daily peak power consumption is defined as the highest power purchased by the system within a single day. This metric gives insights into understanding the maximum power demand pattern over a time period and helps assess the capacity requirements

of electrical infrastructure. Equation 4.20 shows the mathematical representation of the metric where D is the number of days.

$$P_{\text{peak}} = max(P_{\text{day}}) \tag{4.20}$$

Average ramp rate

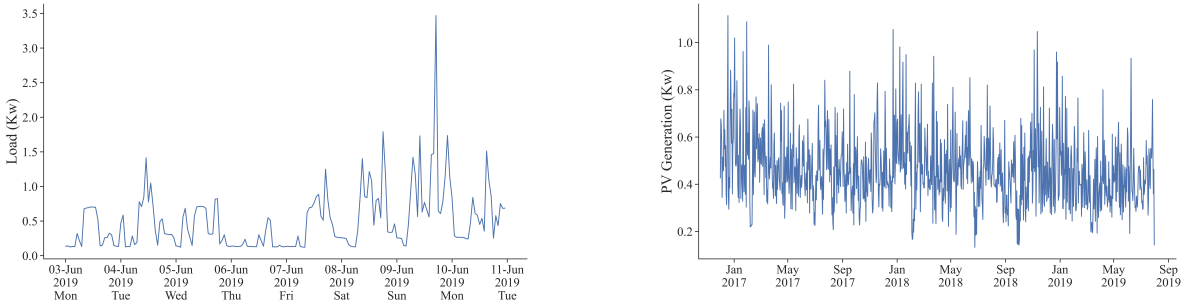
This metric is used to quantify the rate of change in power consumption. It is calculated by differencing the consecutive power purchased as shown in Equation 4.21 over a specific time interval and in this analysis, the hourly period is considered. The average ramp rate gives insights into how fast the power consumed changes in that given time window. This metric is vital for the grid operator to ensure the reliability and stability of the electrical grid.

$$Average\ ramp\ rate = \frac{\sum_{t=0}^{T} P_p(t) - P_p(t-1)}{T} \tag{4.21}$$

5

Exploratory Data Analysis

In this chapter, the initial step before implementing the forecasting models involves analyzing and comprehending the data. The preliminary step before diving into creating forecasting models is to analyze and understand the data, this could be done visually or by analyzing the statistics of the data. A visual examination could use line plots, density plots, box plots, violin, scatter, rolling statistics, histograms, and other various tools to draw a preliminary understanding of the patterns and information in the data.

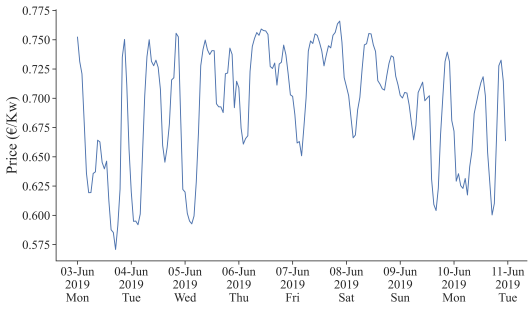


(a) Hourly load consumption for a week

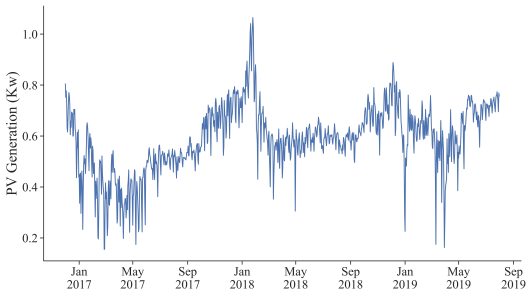
(b) Daily average load across the dataset

Figure 5.1: Household Electricity Consumption

In Fig 5.1, the load consumption of a single household [11] for a week is shown. The first inference is that the load pattern for a single household has more variance and less smooth pattern than the aggregate pattern because of the influence of individual human behavior. Secondly, It could be observed that during the weekends the consumption is relatively higher than on the weekdays, indicating the presence of a weekly pattern.



(a) Hourly market prices for a week

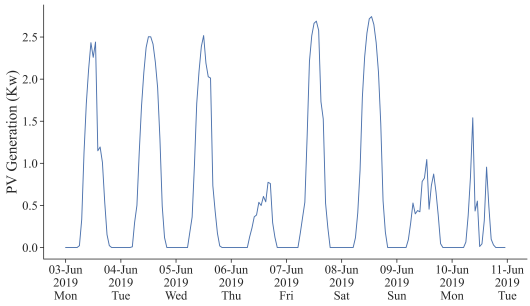


(b) Daily average across the dataset

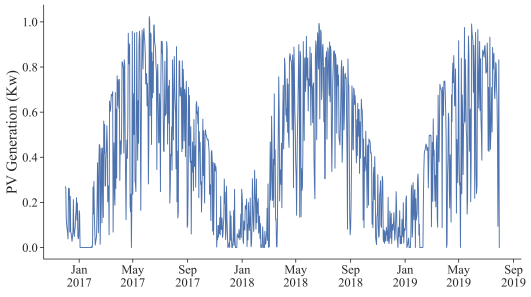
Figure 5.2: Electricity market prices

Thirdly, with a much closer look it could be seen that the consumption has a diurnal pattern with two peaks one in the morning and the other in the late evening, this could be seen much clearly in Fig 5.1a. There is also a yearly pattern which is difficult to visually inspect in Fig 5.1b but it can be subtly seen that the consumption is higher during the winter months around January and February.

The hourly market electricity price for a week is shown in Fig.5.2. Similar to the load, the price has a diurnal and a weekly seasonality as observed in the figure. A diurnal pattern with peaks in the morning and evening when the consumption is higher. On the weekends the cost is less relative to the weekdays which could be due to the lower aggregate consumption on weekends. As higher consumption is a general indicator of higher prices, the yearly seasonality as seen in Fig.5.2b shows that prices are higher during winter when the consumption is also higher.



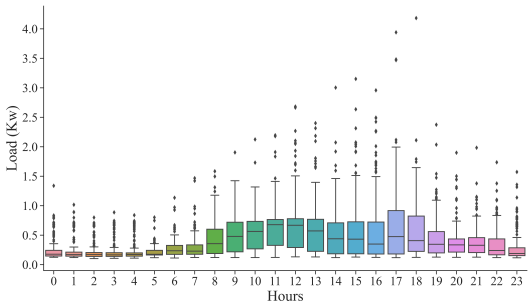
(a) Hourly PV generation for a week



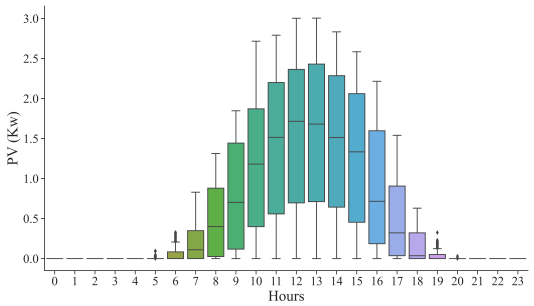
(b) Daily average across the dataset

Figure 5.3: PV Generation

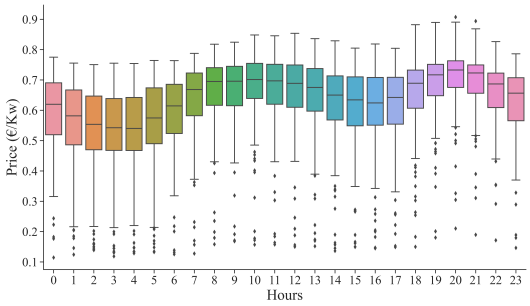
The amount of solar energy generated by the 4000W rooftop panels is shown in Fig.5.3a for a week. It is evident that energy is produced only when there is irradiance in the daytime. There is a diurnal pattern for the power production with the peak power produced around noon. This can observed visually and more detailed in the box plot Fig.5.4b where it peaks at 13.00. The yearly seasonal changes significantly impact the production with higher production in summer and lower in winter, as seen in Fig.5.3b.



(a) Load consumption



(b) PV generation



(c) Electricity market prices

Figure 5.4: Visualizing hourly trends using box plots

As discussed earlier, Fig 5.4a ,5.4b,5.4c has strong diurnal seasonality which could be observed by the hourly mean values of load, PV, and price in each of these plots, respectively. In these plots, the black dots at the end of whiskers represent outliers which signifies extreme values. In Fig 5.4a, it can be seen that outliers are spread relatively more during the day than at night, which shows consumption patterns. In contrast, In Fig 5.4c it could be seen that the outliers are more spread on the lower side of whiskers which emphasizes that wholesale prices fall lower.

6

Results and Discussion

The first part of this chapter discusses the results of the various forecasting models for load, PV generation and dynamic electricity price. The analysis includes the performance, advantages and limitations of different neural network architectures and finally presents the model with the most accurate predictions. The second part of the chapter investigates the performance of RL-based energy management systems against a baseline and a benchmark model. Specifically, it explores the effect of different algorithms on cost savings to the end consumer, effects on grid congestion and battery life degradation.

6.1. Forecasting

This section individually presents the results of forecasting load, PV generated and electricity price using the deep learning methods discussed in the section. The simulations were tested out of the box on data from 01/06/2019 to 01/10/2019, but for the ease of interpretation and understanding of the performance, a week's data (03/06/2019 to 10/06/2019) is plotted and analyzed in detail. Although the performance was evaluated for baseline, statistical and six deep-learning models, only the results of crucial models are plotted. The learning curve of the best-performing model is also added to further assess the performance of the neural network architecture. It should also be noted that the results shown in tables and figures comparing different architectures are for the whole data.

Each forecasting task is evaluated based on MAE, RMSE and R-squared of the out of the testing data set. The prediction performance is shown visually using box plots and the values are presented in the tables.

6.1.1. Load

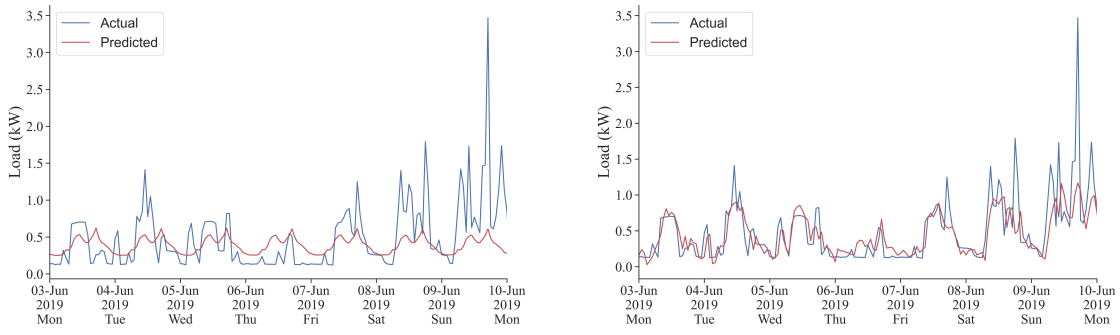
In Table 6.1 results of forecasted electrical consumption load errors for one hour in a household are presented. It should be established that the prediction is for a single household, which increases the model complexity compared to an aggregate level as the profile heavily depends on human behavior. The baseline model, which uses the previous day's value as the forecast shows the most prediction error compared to other employed methods which shows the relative performance increase of other

models. It could be noted in Fig.6.3 that the predictions from the persistence model have a negative correlation with actual value, this could be due to the naive nature of the model and also shows it fails to capture the seasonalities and variations in the data.

Table 6.1: Summary of Prediction errors and R-Squared for Load prediction

Models	**MAE (kW)**	**RMSE (kW)**	**R^2**
Persistence Model (Baseline)	0.251	0.410	-0.356
SARIMA	0.231	0.322	0.149
Deep Learning Models			
Multilayer Perceptron (MLP)	0.195	0.281	0.359
Convolutional Neural Network(CNN)	0.179	0.275	0.363
Recurrent Neural Network (RNN)	0.183	0.277	0.361
Long Short Term Memory (LSTM)	0.175	0.270	0.374
CNN-LSTM Hybrid	0.181	0.271	0.404
Bidirectional LSTM	0.173	0.269	0.409

The results from the SARIMA model are shown in Fig.6.1a, it could be observed that the forecast exhibits a consistent pattern with slight differences in the longer horizon. This shows the model's limitation to capturing only one seasonality and it could be observed that daily periodicity fails to interpret the weekly seasonality. Another inherent shortcoming of the model to learn the non-linear relationships in the data could also seen in Fig.6.1a as it fails to capture the peaks and variations. The key advantage of this statistical method is its simplicity which requires less historical data for training and fewer computational resources compared to machine learning models discussed later. The performance in Table.6.1 is better than the baseline but still lower than the deep learning models.



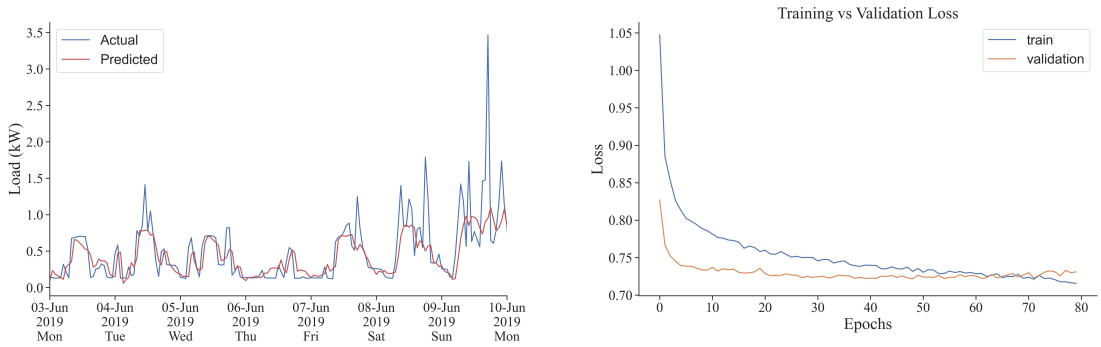
(a) SARIMA **(b)** CNN

Figure 6.1: Forecasted vs. Actual of Load profile for a week

In the deep-learning models, MLP, a simple neural network architecture consisting of only fully connected dense layers, significantly reduced the errors (MAE and RMSE) compared to SARIMA and baseline and captured more variance. However,

this architecture cannot model temporal dependencies as the load has temporal information that could be utilized to make better predictions. This limitation was addressed by using CNN and RNNs architecture.

As seen in Table.6.1, CNN reduces the forecast error and captures better variance than MLP. The better performance of CNN could be attributed to its ability to extract features from sequential data and capture local patterns using filters. The forecast using CNN is shown in Fig.6.1b, it could be observed with the fluctuations that the model is trying to capture the local patterns. However, although this architecture does the feature extraction to learn the underlying pattern it does not have an inherent structure to model sequential data. This characteristic is built in RNNs where they can model data with temporal information and this could be seen with better performance by RNN-based architectures (LSTM and BiLSTM) as seen in Fig.6.3.



(a) Forecasted vs. Actual of Load profile for a week

(b) Learning curve

Figure 6.2: Performance of BiLSTM for load forecasting

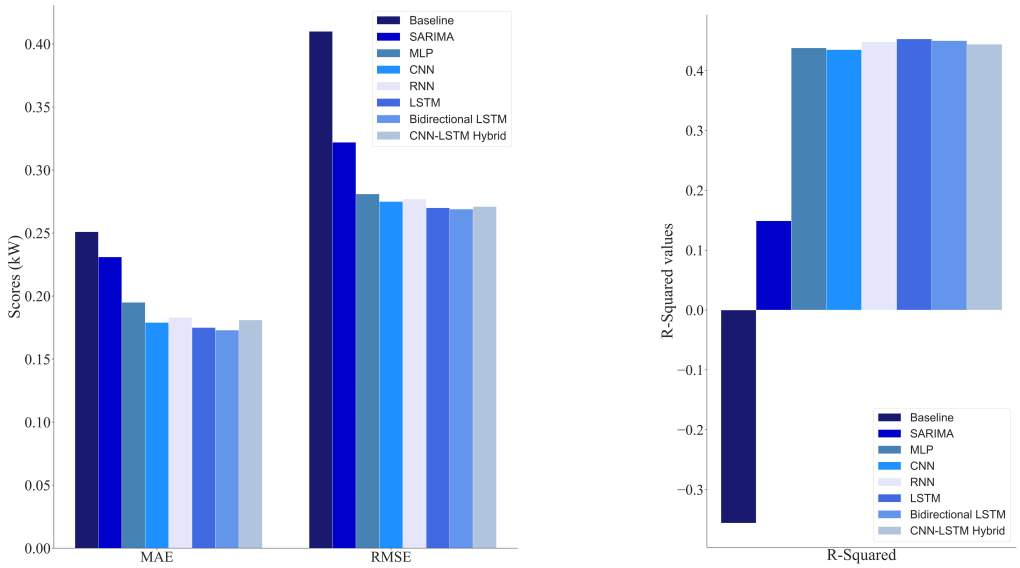


Figure 6.3: Bar Plot of Error metrics for Load forecasting

In this analysis, Bi-directional LSTM has been observed to have the least MAE and RMSE compared to other deep-learning models and captures the most variance

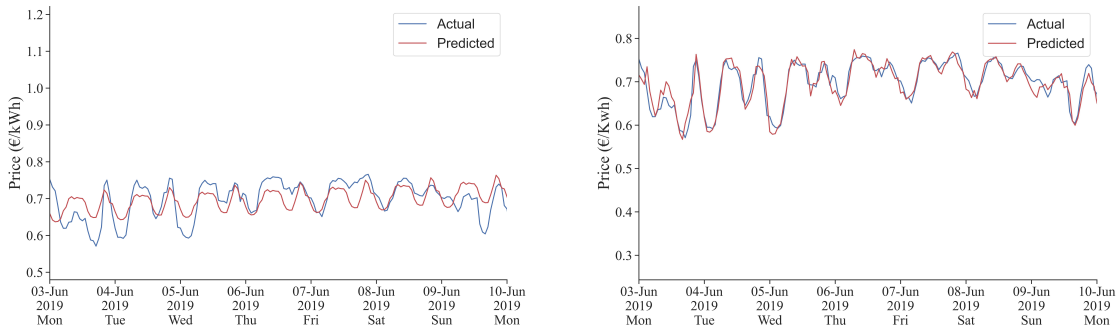
for load forecasting. The differences between the performances of the deep-learning models are relatively low. The learning curve of the implemented BiLSTM network can be seen in Fig.6.2b. The first inference from the loss plot is a steep loss in training and validation loss, indicating that most of the learning takes place at the start and the learning takes place until the training and validation loss is converged.

6.1.2. Electricity price

The electricity price forecast results summary is shown in Table.6.2. Similar to load forecasting, the results for a one-hour forecast horizon and equivalent architectures are evaluated. In the initial observation, it could be seen that the baseline performs better than SARIMA in terms of MAE and R-squared as seen in Fig.6.6. This could be due to the non-stationary nature of the data and the significant amount of random variations or noise in the data which severely limits SARIMA's capability.Fig.6.4a shows the plotted results from SARIMA, as anticipated the model captures only one seasonality and is very limited in capturing variance due to the linear nature.

Table 6.2: Summary of Prediction errors and R-Squared for Electricity price prediction

Models	**MAE Price (€/Mwh)**	**RMSE Price (€/Mwh)**	**R^2**
Persistence Model (Baseline)	43.3	65.9	0.433
SARIMA	49	59.9	0.142
Deep Learning Models			
Multilayer Perceptron (MLP)	19	24.6	0.920
Convolutional Neural Network(CNN)	13.8	19.3	0.950
Recurrent Neural Network (RNN)	13.4	18.1	0.956
Long Short Term Memory (LSTM)	13.6	19.3	0.951
CNN-LSTM Hybrid	17.3	25.2	0.916
Bidirectional LSTM	12.5	17.4	0.960



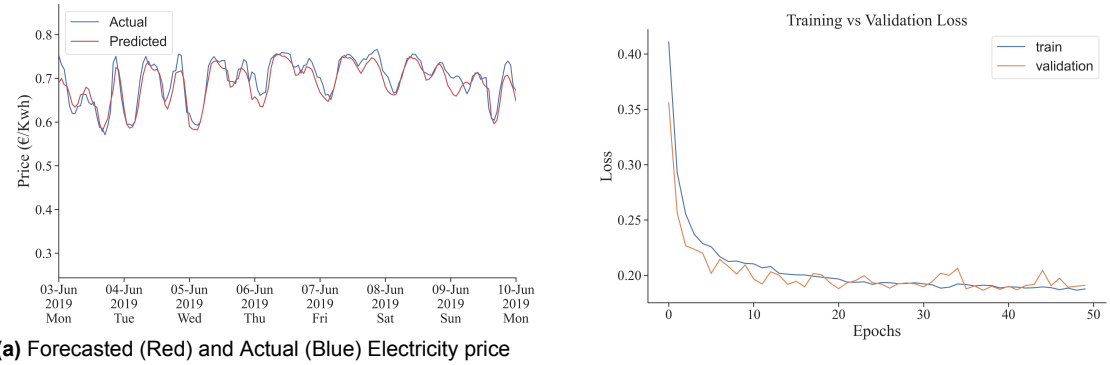
(a) SARIMA **(b)** CNN

Figure 6.4: Forecasted vs. Actual of Electricity price for a week

Deep learning models have much improved performance compared to baseline

and SARIMA as seen in Table.6.2 and visually in Fig.6.6. As expected, the MLP architecture has higher forecast errors than other models, mainly due to its limitation of representing temporal relationships.The results of CNN architecture as seen in Fig.6.4b, display its ability to extract feature information using filters as the network attempts to model the small perturbations.

The ability of RNNs and their variants (LSTM, BiLSTM) to inherently incorporate sequential information has improved the prediction performance compared to MLP. However, the CNN-LSTM hybrid has relatively higher MAE and RMSE which could be attributed to problem overfitting or architecture could be more sophisticated for the electricity price data it tries to model.



(a) Forecasted (Red) and Actual (Blue) Electricity price for a week

(b) Learning curve

Figure 6.5: Performance of BiLSTM for electricity price forecasting

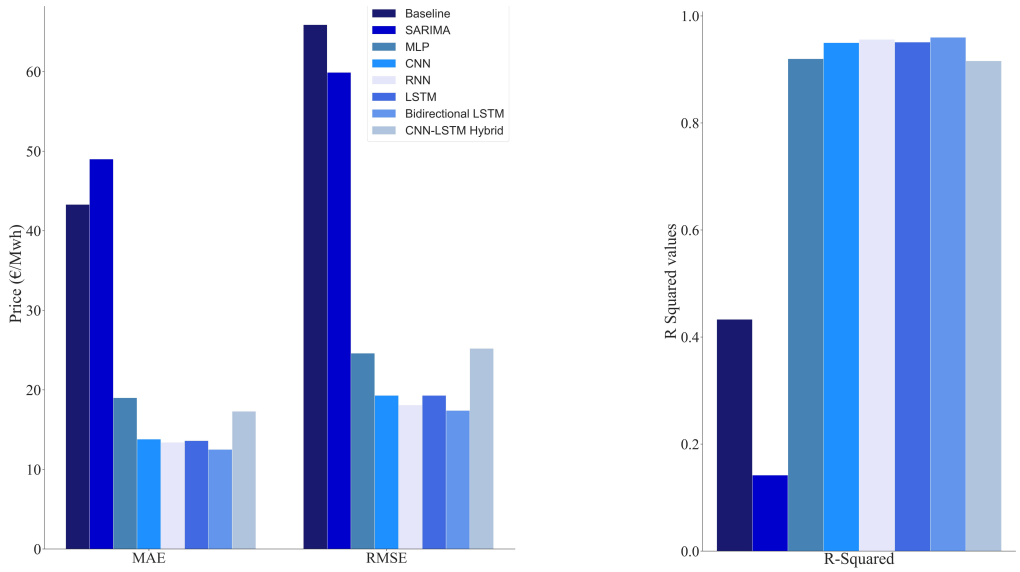


Figure 6.6: Bar Plot of Error metrics for Electricity price forecasting

Bi-directional LSTM has shown the best performance with RMSE of 17.4 kW, MAE of 12.5 kW and Pearson coefficient of 0.960 for the electricity price prediction for a one-hour horizon. The prediction results and the network's learning curve are shown

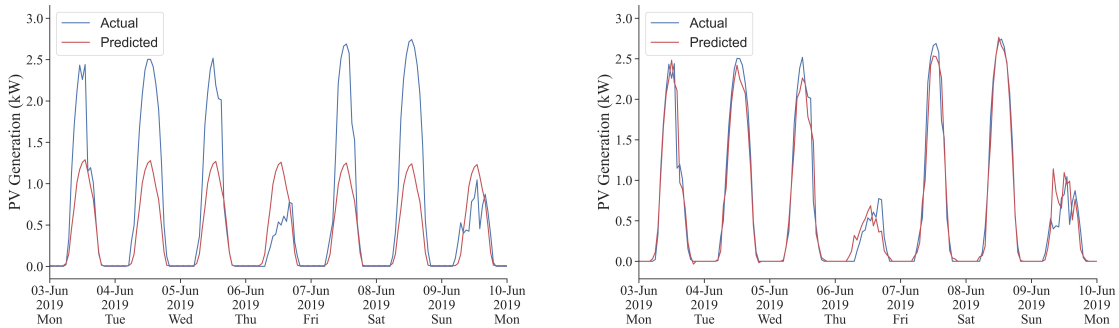
in Fig.6.5a and Fig.6.5b. The fluctuations in the validation loss curve are due to the hyperparameters chosen, explicitly choosing a higher learning rate than the batch size of 32 as the network gets updated in a smaller batch size with a faster learning rate.

6.1.3. PV Generation

Similar to the previous forecasting tasks, the results for PV generated are summarized in Table.6.3. In the testing set, initial analysis shows that the baseline model performs better than SARIMA in terms of MAE and lower in terms of RMSE. This shows that SARIMA models the diurnal seasonality which captures the peaks better. In this specific week plotted in Fig.6.7a, it could seen that average predictions in a day are lower than the actual average which showcases the limitation of modelling only one seasonality as yearly periodicity was not incorporated. This also significantly decreases the explained variance in the SARIMA model as compared to the baseline as seen in Fig.6.9

Table 6.3: Summary of Prediction errors and R-Squared for PV generation prediction

Models	**MAE (kW)**	**RMSE (kW)**	**R^2**
Persistence Model (Baseline)	0.268	0.538	0.593
SARIMA	0.274	0.434	0.188
Deep Learning Models			
Multilayer Perceptron (MLP)	0.124	0.237	0.921
Convolutional Neural Network(CNN)	0.114	0.229	0.926
Recurrent Neural Network (RNN)	0.126	0.237	0.921
Long Short Term Memory (LSTM)	0.117	0.227	0.927
CNN-LSTM Hybrid	0.128	0.233	0.923
Bidirectional LSTM	0.118	0.226	0.928

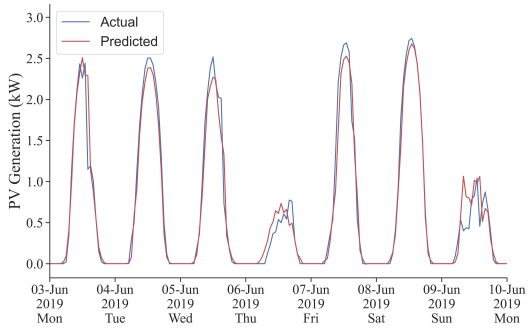


(a) SARIMA **(b)** CNN

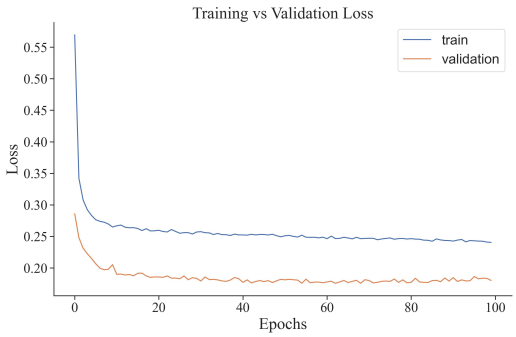
Figure 6.7: Forecasted vs. Actual of PV generation for a week

Concerning MAE, CNN marginally performs better than other deep learning architectures, which could be attributed to its capability to extract features. Regarding RMSE, Bidirectional LSTM shows lower forecast errors compared to other models

and is analogous to the previous prediction tasks which could be ascribed to its ability to model sequential information and interpret the input sequence in both forward and backward direction. The results of CNN and BiLSTM are shown in Fig.6.8a and Fig.6.8a.



(a) Forecasted (Red) and Actual (Blue) PV generation for a week



(b) Learning curve

Figure 6.8: Performance of BiLSTM for PV generation forecasting

In the learning curve Fig.6.8b, it could be seen that the training and validation loss are not converging although the model is optimized for the best performance. This could be associated with the diurnal pattern of PV generation, which has no production during the night or this could also be due to the nature of the data. Regularization using dropout layers was also simulated but with more convergence, other evaluation statistics performed worse. It is inferred here that in some cases better convergence does not necessarily mean improved performance, this could be because after certain learning the network might not be able to generalize well to changing unseen data.

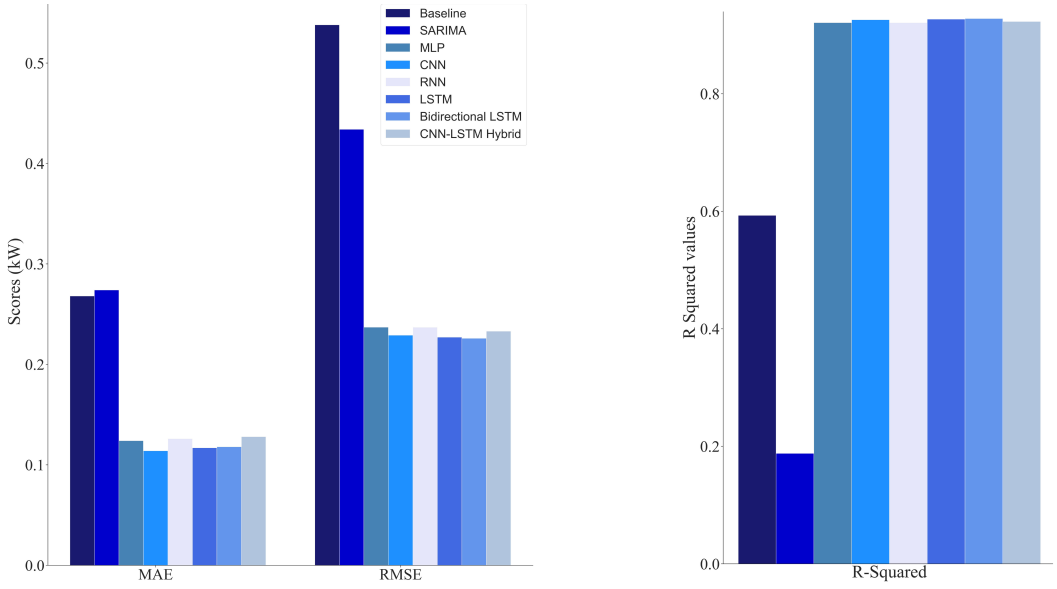
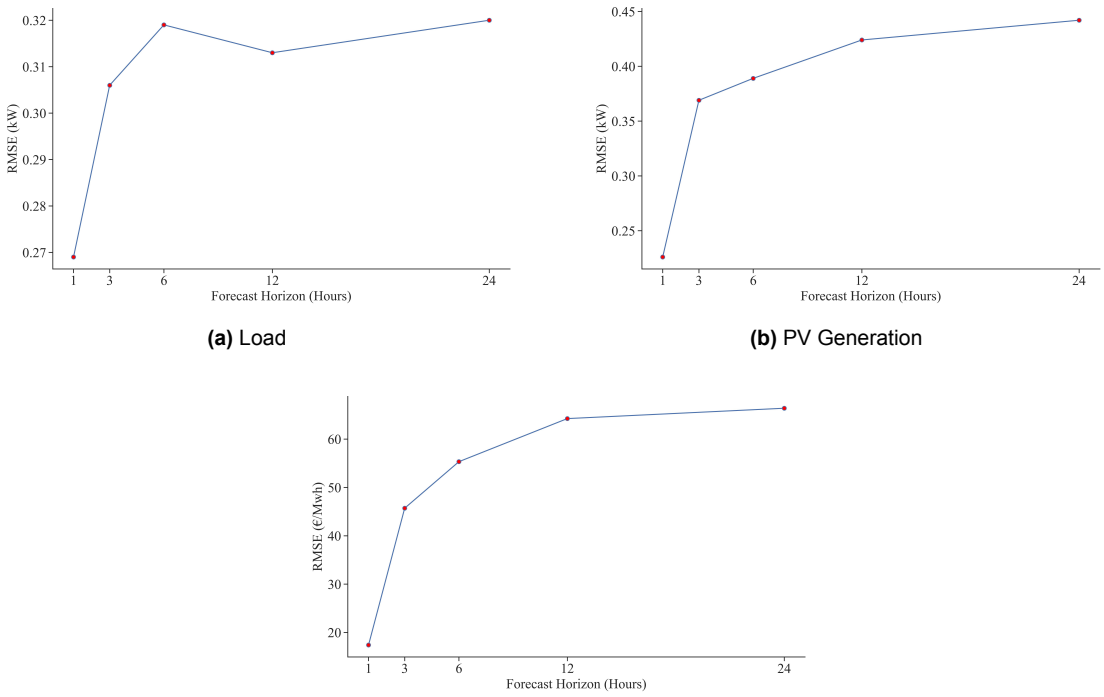


Figure 6.9: Bar Plot of Error metrics for PV generation forecasting

The models explored thus far are evaluating for a one-hour horizon that could be extended to time steps in the future. In contrast to the one-hour horizon here, the

networks are trained on the target variable 3rd,6th,12th, and 24th hour into the future. Leveraging the already done comparison and since the neural networks exhibit similar performances, the Bi directional LSTM is chosen for predicting the future horizons of Load, PV generation and market prices. The RMSE for each forecasted variable for different horizons is shown in Fig.6.10. A comparable trend is observed as the error increases with an extending horizon, this is anticipated as the level of randomness or uncertainty increases over time. The transition from the first to the third-hour horizon is observed with a substantial increase in RMSE.



(a) Load

(b) PV Generation

(c) Electricity Price

Figure 6.10: Forecasting Horizon vs. RMSE for Bidirectional LSTM

The key inferences observed in the forecasting of load, PV generation and electricity price are discussed below

- In all three of the forecasting tasks, BiLSTM was the best-performing prediction model in minimizing the error statistics (MAE and RMSE). However, it was observed that the differences in performance between deep-learning models especially the recurrent neural network variants were relatively low. However, the performance increase from baseline/SARIMA to deep-learning models was significant.
- In the SARIMA model, its inherent limitation to incorporate only seasonality and the inability to learn non-linear relationships has bounded its performance.
- Simple RNN prediction models were consistently observed to be performing less than other recurrent neural network models and this could be due to the vanishing gradient problem.

- Although CNN-LSTM combines the feature extraction and sequential modeling aspects, the performance was lower than other models. This could be due to the overfitting as a result of complex architecture and even with regularization, performances were not improved so it is also possible that the sophisticated architecture is too much for the nature of the data.
- As the forecasting horizon increases, a notable trend is observed where the error (RMSE) consistently increases reflecting the rise in uncertainty over time.

6.2. Energy management system

In this sub-section, Each of the three algorithms (Baseline, Benchmark, RL) is evaluated independently to ensure they adhere to operational constraints related to the state of charge, maximum charge and discharge limits. BESS Dispatch strategies are also analyzed. Additionally, the RL aspect is discussed in detail, focusing on the DDPG agent's learning process. Finally, we compare the performance of all three algorithms using the metrics defined in the section.4.7

6.2.1. Baseline

To set a reference point for the EMS performance, the baseline algorithm discussed in sub-section3.2.1 is analyzed first. During the experiment, the state of charge is within the limits of 0.2 and 0.8 as seen in Fig.6.11. It should also be noted that the battery is not effectively used up to its potential as it is idle for most of the duration after complete discharge. In this model, the battery does not engage in power grid exchanges for arbitrage advantages instead, it imports power when neither the PV nor the battery can meet the load requirements.

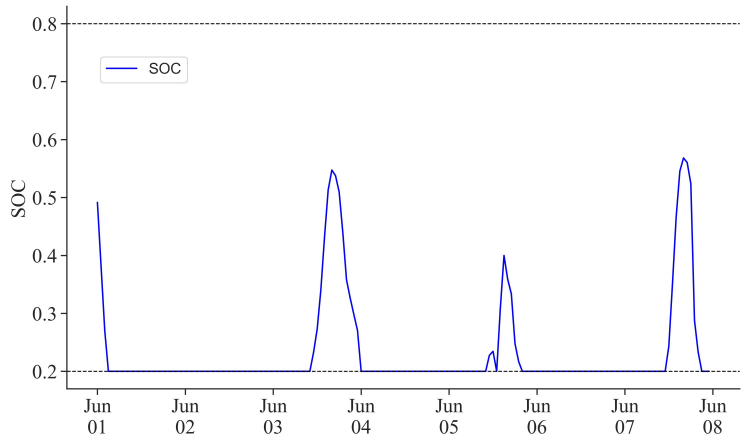


Figure 6.11: Baseline: State of charge plot

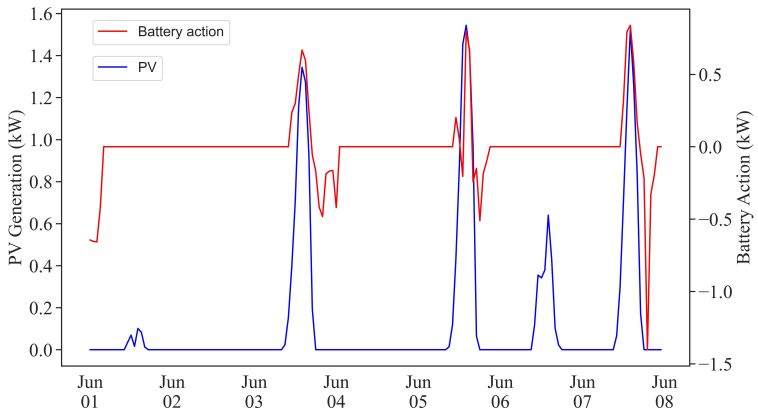


Figure 6.12: Baseline: Battery actions vs PV generation

The battery operates within the prescribed boundaries $[-2.8, 2.8]$ for its charging and discharging power, shown in Fig.6.12. It can also be seen that the battery charges only when the PV generated is more than the required load and discharges at the following time step when there is a deficit. So, it is not modeled to discharge to the load when the price is at its peak which could have benefited the end user. Here, this strictly follows the algorithm to prioritize consumption at the source which can be seen in the battery actions and does not consider the price signal in the decision-making process.

The power balance is maintained as observed in Fig.6.13, It is evident that the majority of the time the load is satisfied by direct grid import. However, there are instances where all three sources (PV, Battery and direct grid import) are used to fulfill the demand.

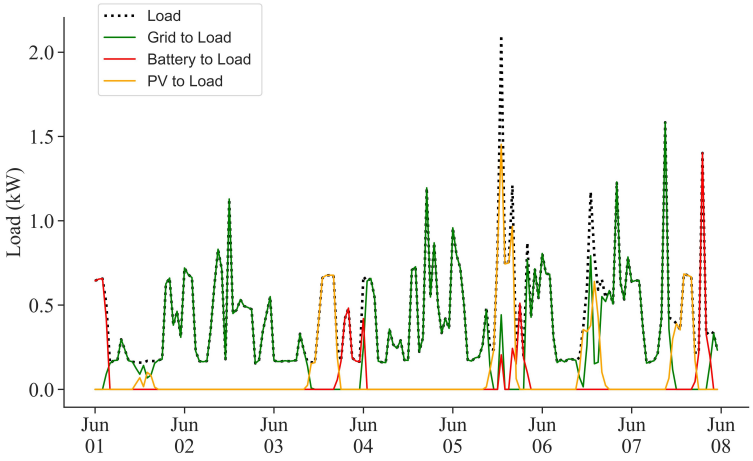


Figure 6.13: Baseline: System Power flow

In addition to the periods when the battery charges and discharges centered around PV generation, there is an observable arbitrage opportunity in Figure 6.14. This opportunity arises from the diurnal pattern of the market price during which the battery is also dormant. This presents an opportunity to charge when the price is low and discharge to the load or back to the grid to maximize cost savings.

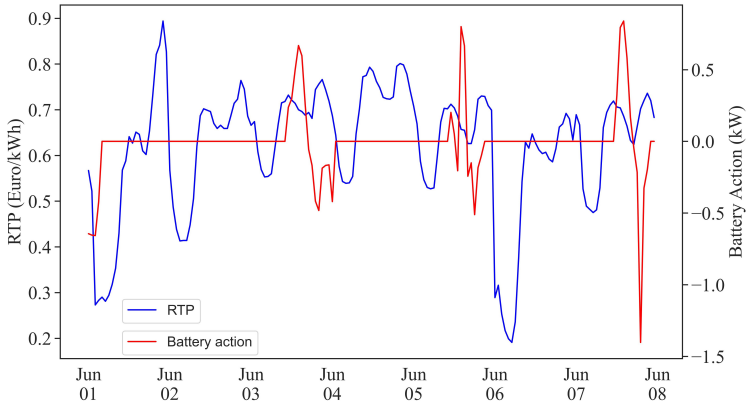


Figure 6.14: Baseline: Battery actions vs. Real-time price

6.2.2. Benchmark

The results of defined benchmark algorithm 3.7 are discussed here. The analysis begins by ensuring that the battery system parameters, specifically the state of charge and the maximum charge and discharge limits fall within acceptable limits. The state of charge is observed to fall within the range of [0.2, 0.8], as depicted in Fig.6.15 and the battery's charging and discharging behavior is also within the boundaries $[-2.8, 2.8]$ as illustrated in the Fig6.17. It is important to emphasize that this model is built on top of the baseline algorithm, It's behavior is expected to be comparable to the former but with an extension to include grid interactions to gain arbitrage opportunities. Threshold hyperparameters used for this simulation was found to be $X_1 = 0.7\ and\ X_2 = 0.2$ through an exhaustive search as described in sub-section.4.6.3.

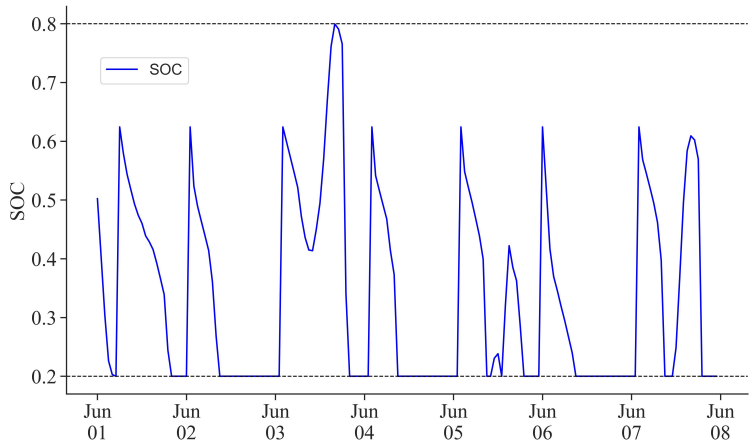


Figure 6.15: Benchmark EMS : State of charge plot

The energy balance for this model was preserved as depicted in Fig.6.16 following the established rule that the load is not satisfied through PV generated and battery, it would be met by importing directly from the grid.

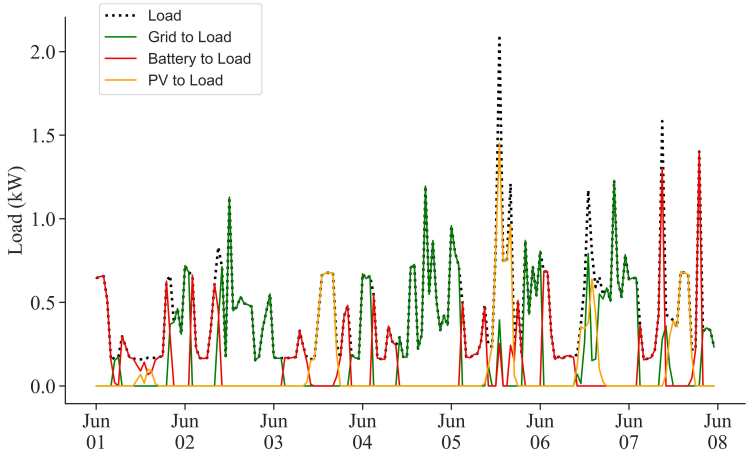


Figure 6.16: Benchmark : System Power flow

As anticipated, it is noticeable from the Fig.6.17 that the battery gets charged when the generated PV exceeds the needed power to meet the load. This is immediately

followed by discharging the total capacity when the system faces a power deficit and reaches the lower SoC limit which could be inferred from Fig.6.15. It may have been financially beneficial for the consumer to utilize the battery to supply power to the load during peak pricing periods.

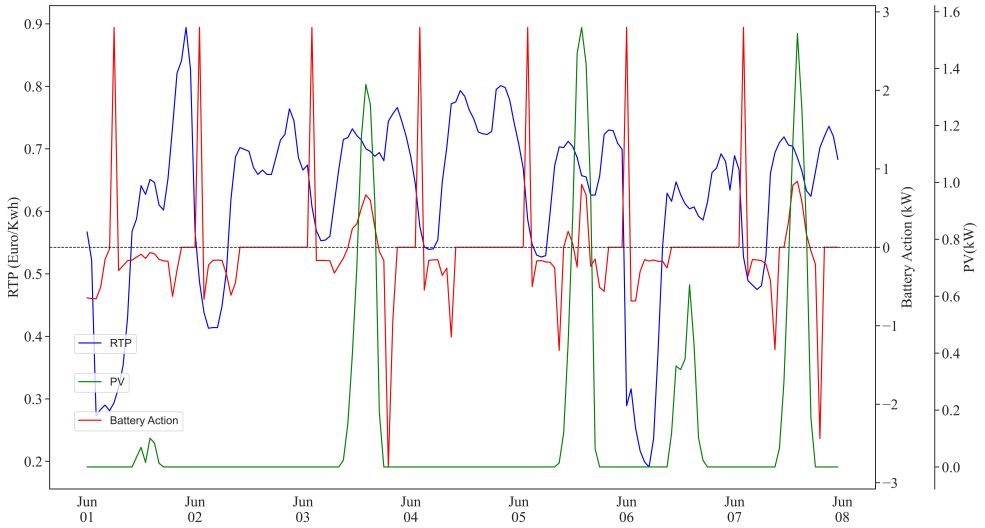


Figure 6.17: Benchmark : Battery action vs. RTP vs. PV generation

Fig.6.18 illustrates the isolated battery exchange with the grid. It is observable that when the threshold to purchase electricity is reached, the algorithm immediately charges to its fullest capacity instead of waiting till the price drops even further before charging. In contrast, during the discharging most of the capacity is discharged to meet the load which restricts it to sell it back to the grid at peak price duration.

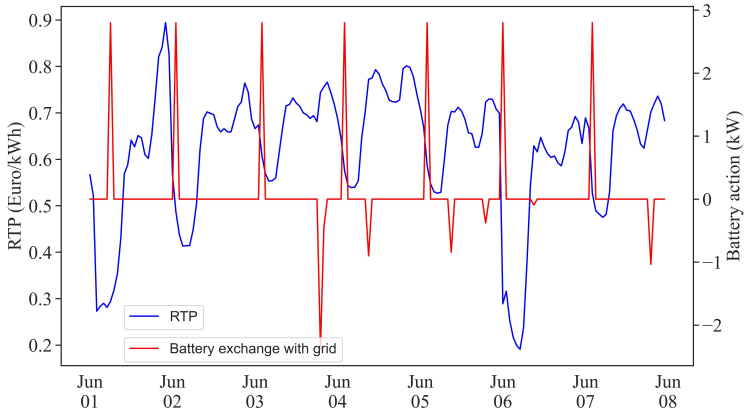


Figure 6.18: Benchmark EMS: Battery exchange with grid vs. RTP

The limitation of this algorithm in predicting price peaks and the practice of fully charging or discharging the battery rather than doing so in proportion to the price fluctuations and load pattern reduces the financial advantages for the end consumer. The RL in the following section addresses these limitations by modeling the complex

relationship between load, market price and PV generated to maximize cost savings for the consumer.

6.2.3. Reinforcement learning

The agents were trained in the custom EMS environment for 20000 episodes with an episode length of 24 hours. An agent tries to maximize the cost savings for a day, this also coincides with the diurnal pattern of the load, market price and PV generation. If the agent makes optimal decisions for a single day, this is expected to be extrapolated for a year. Due to the computational constraints, the agents are trained on 30 days of data and tested on the following seven days. In the training process, all the individual rewards are negative in order to have stable learning whereas having a mix of positive and negative rewards has been observed to have resulted in sub-optimal performance. The whole reward setting is centered around minimizing the base cost of electricity for a day hence the rewards closer to zero the better the performance.

The training process for the DDPG agents can be seen in Fig.6.19, the agents begin with a high degree of exploration leading to very high negative cumulative rewards. Soon, the agents start fine-tuning the policy, leading to better performance. The steep reward increase is a positive sign which indicates that the agent is learning and adapting to the environment. The reward dip around episode 7500 could be due to exploration space or the agents being exposed to unfamiliar observations. In the later episodes, the cumulative rewards are seen to be stabilized but improve slowly. The zoomed-in version shows the cumulative stabilized around -20. It is important to emphasize that the cumulative reward here does not exclusively reflect the cost of electricity but the sum of all individual rewards as discussed in the sub-section 4.5.3.

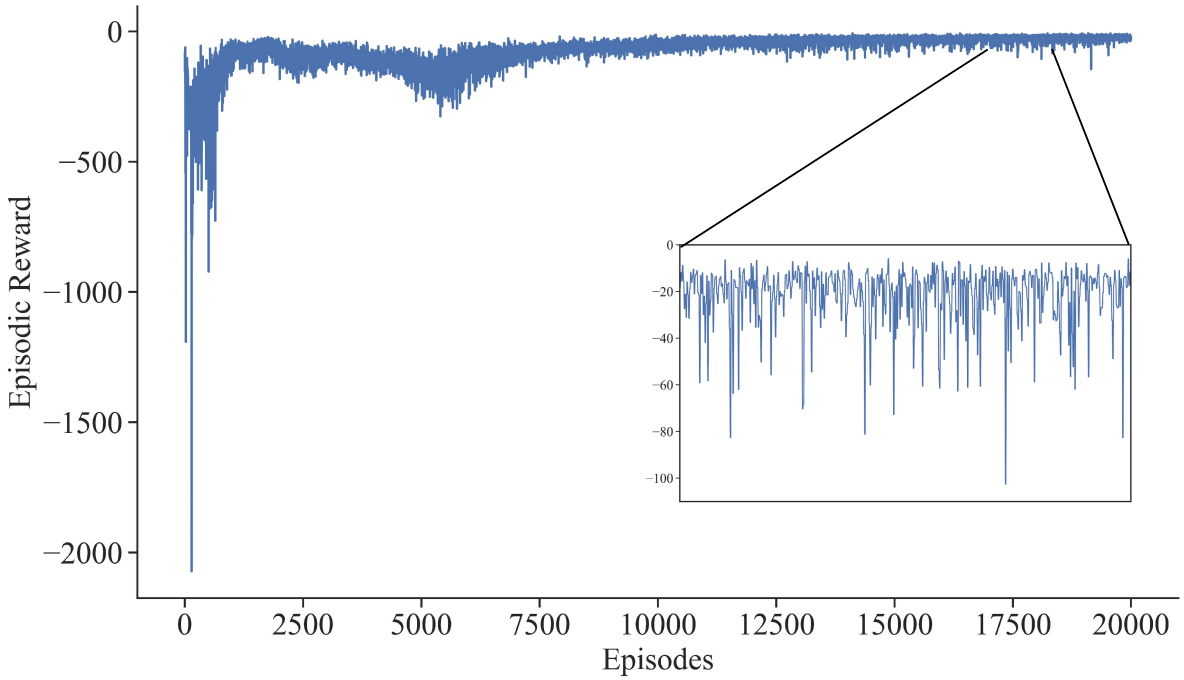


Figure 6.19: DDPG reward training process

As the episodes progress, the agents improve their coordination and learn to refine their policies gradually. This results from the reduction of exploration and agents exploiting the already learned policy. The exploration noise added to the actions from the Ornstein-Uhlenbeck process is illustrated in Fig.6.20

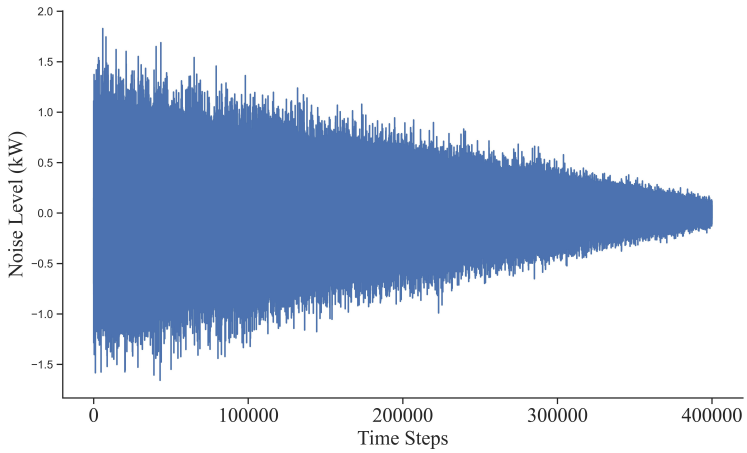
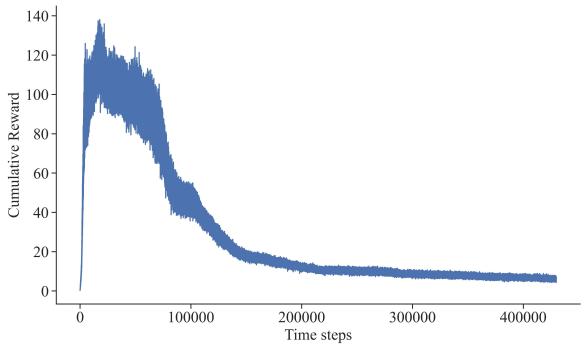


Figure 6.20: Exploration noise using the Ornstein-Uhlenbeck process

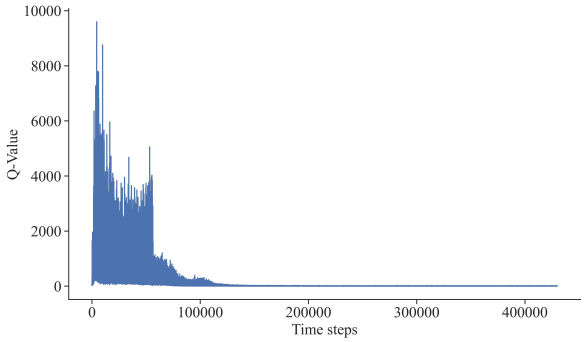
Under the hood, the cumulative episodic rewards are increased because their target actor and critic networks are improving at estimating an action and quantifying how good that action is. This interaction as described in sub-section 4.5.5 improves the agent's decision-making, which is evident in the loss curves of the networks as shown in Fig.6.21.

The critic networks of both agents are observed to have lower losses than their respective actor networks for several reasons. Firstly, the critic network is a value function comparable to a regression task. The actor-network is a policy function that could be interpreted as an optimization problem to estimate the action actions that could be more complex and challenging than the critic network. Secondly, the actor network involves exploration noise to find increased cumulative reward compared to the critic network which does not explicitly involve exploration.

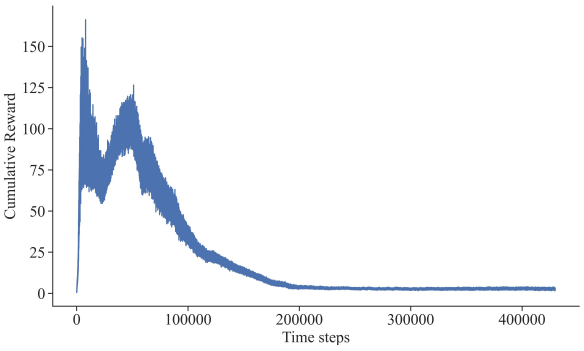
The actions performed by the agents a_1& a_2 are shown in Fig.6.22. In the initial test runs, the agents constantly breached the battery charging and discharging limits in fine-tuning the reward function. However, these drawbacks were mitigated by using a scaled tanh activation function for agent a_1 and a scaled sigmoid activation function in the output layer of the target actor network respectively.



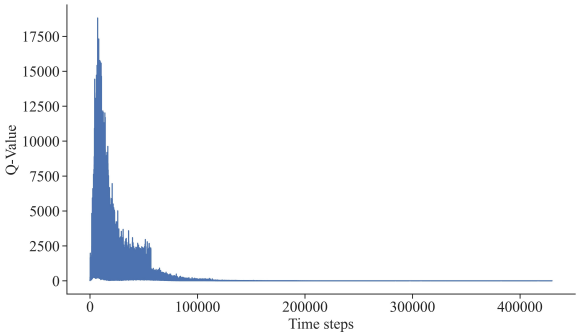
(a) Actor Network of Agent 1



(b) Critic Network of Agent 1

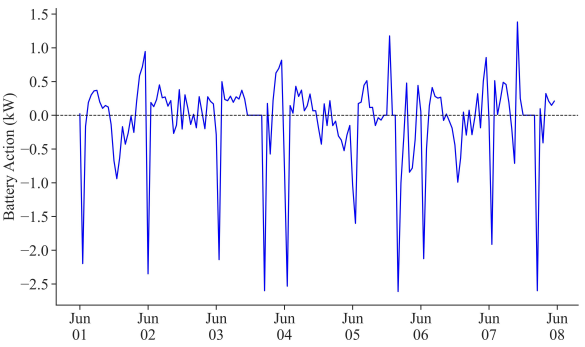


(c) Actor Network of Agent 2

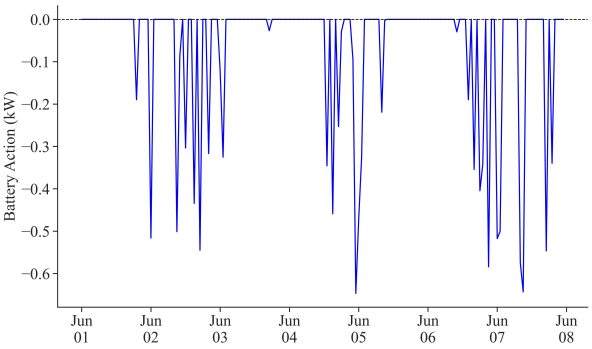


(d) Critic Network of Agent 2

Figure 6.21: Loss curves for the target networks involved in the DDPG algorithm



(a) Agent 1



(b) Agent 2

Figure 6.22: The actions executed by the DDPG agents

As Q-Learning is a model-free approach that learns through trial and error on interaction with the environment, the agents have to cooperatively learn to be within the SoC boundaries which are implemented by an explicit reward function as shown in Equation.4.11. As SoC limits influence other parameters in the system, it becomes crucial for it to be within limits to reduce deterioration of the battery as well as obey the physical laws hence a huge penalty was given for this. Another inference from the state of charge as visualized in Fig.6.23 is that the agent takes action at every time

step and does not stay idle due to the continuous action space.

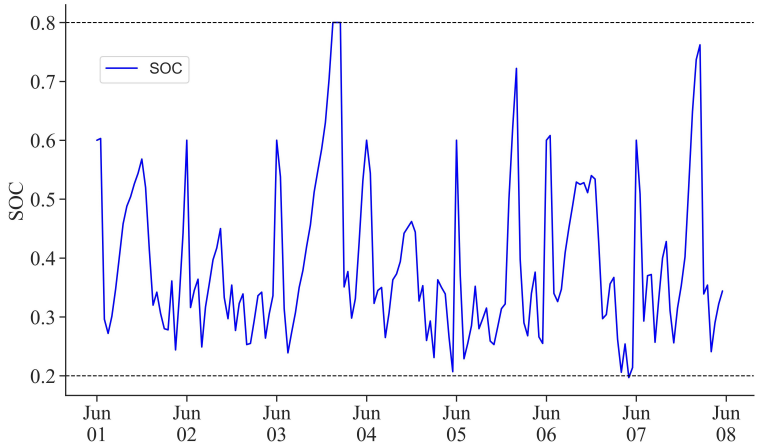


Figure 6.23: RL-Based EMS: State of charge

The system energy balance is preserved as depicted in Fig.6.24. The load is fulfilled with varying amounts from PV, battery and directly from the grid. Compared to the baseline and benchmark algorithm discussed earlier, RL-based EMS involves more battery discharge to meet the load.

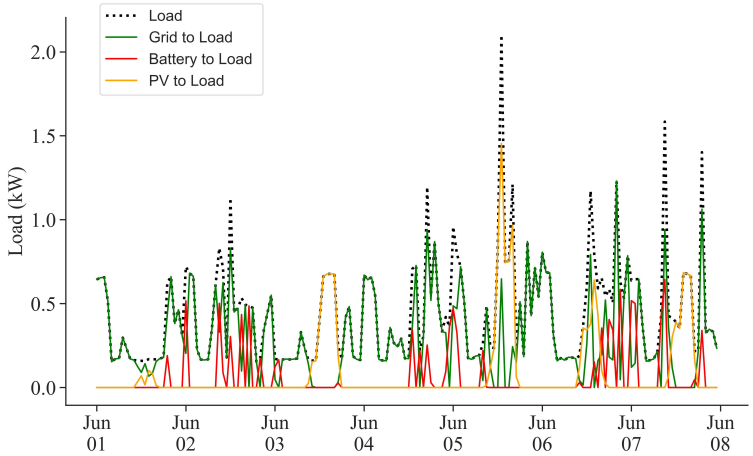


Figure 6.24: RL-Based EMS: System Power Flow

The BESS dispatch decisions made by DDPG agents a_1& a_2 are shown in are shown in Fig.6.25 along with real-time price, load and PV generation. The battery actions are the cumulative sum of both RL agent's actions as defined in sub-section4.5.5. The initial inference shows that the battery is charged when the PV generated exceeds the required load due to the partly rule-based structure of the implemented algorithm. The actions taken by DDPG agents result from the cumulative reward function and the weight of individual reward components. As this research is centered around maximizing financial benefit for the end consumer, reducing the base electricity cost, isolating how the battery interacts with the grid and when it discharges to the load is imperative to analyze the DDPG agent's actions.

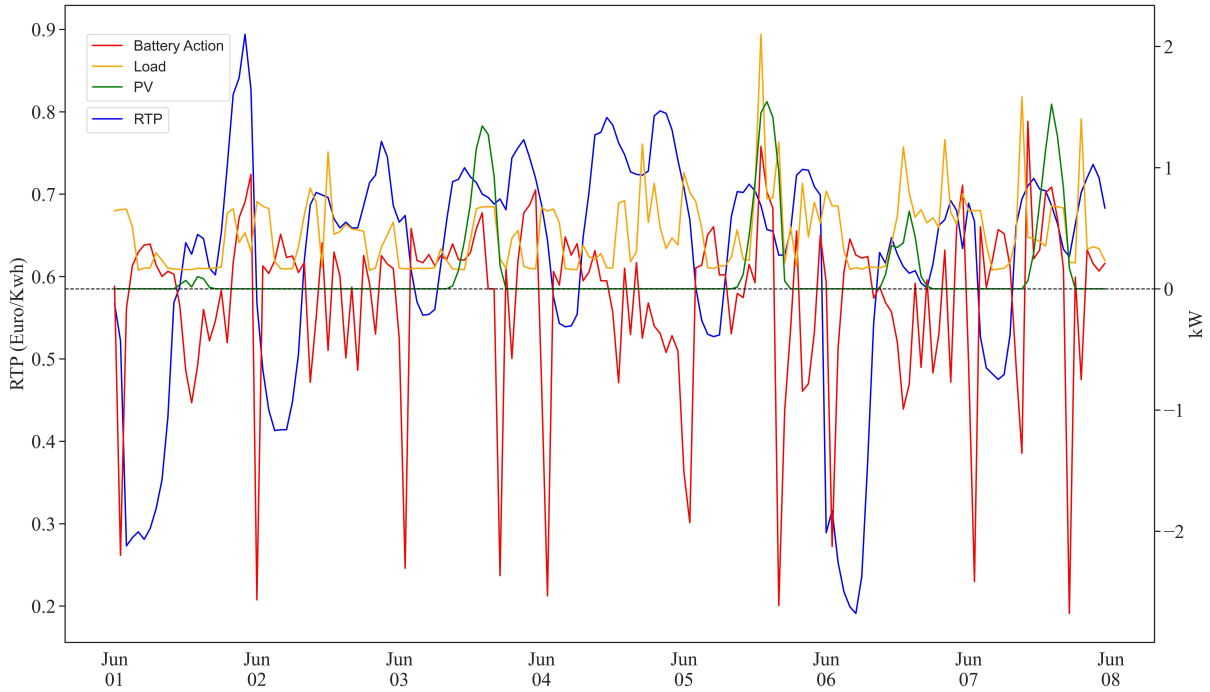


Figure 6.25: RL-Based EMS: PV generated vs Battery Action Vs RTP

Focusing only on the battery interaction with the load Fig.6.26, agent 2 is solely responsible for making this decision. Here, it should be noted that although the battery action is depicted in a positive sign in the figure for clarity, the red line represents the amount of power discharged to the load. Upon closer examination of Fig.6.26, it becomes evident that agent a_2 supplies to the load when the market prices are higher and actively seeks to avoid discharging when the prices are relatively lower. This is an anticipated strategy to reduce the base cost of electricity by leveraging the low market price duration to charge the battery.

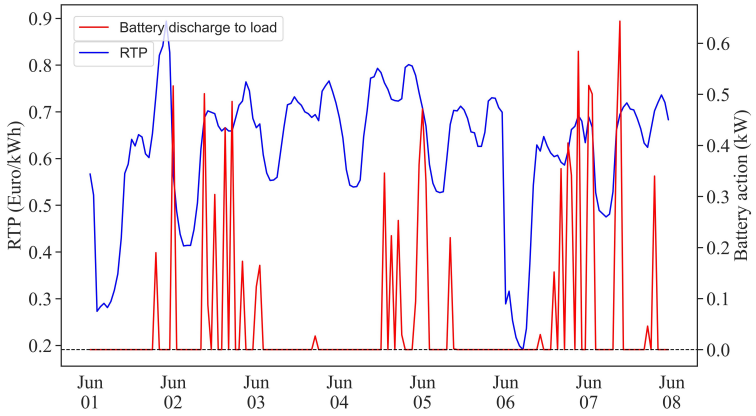


Figure 6.26: RL-based EMS : Battery discharge to the load vs RTP

Another important aspect of a financially optimal EMS is leveraging the fluctuations in market prices to benefit from arbitrage opportunities. Fig.6.27 shows the grid interaction with the battery. This part represents the actions taken solely by agent a_2.

A broad idea of arbitrage is to buy electricity when the prices are lower and sell it back to the grid at a higher price. Here, it is observable that almost every time there is a dip in the market price, usually the lowest during the night the agent charges the battery. A critical interpretation is that right before the market price anticipates the day's lowest price, the agent sells a considerable amount back to the grid to make space for purchasing at the lowest price.

In a few cases, the agent sells it back to the grid when the market price is higher, which is counter-beneficial. This could be interpreted as the agent anticipating a higher price signal. The other small-scale charge discharge cycles could be interpreted as the agent trying to take advantage of small fluctuations in the market price for arbitrage benefits.

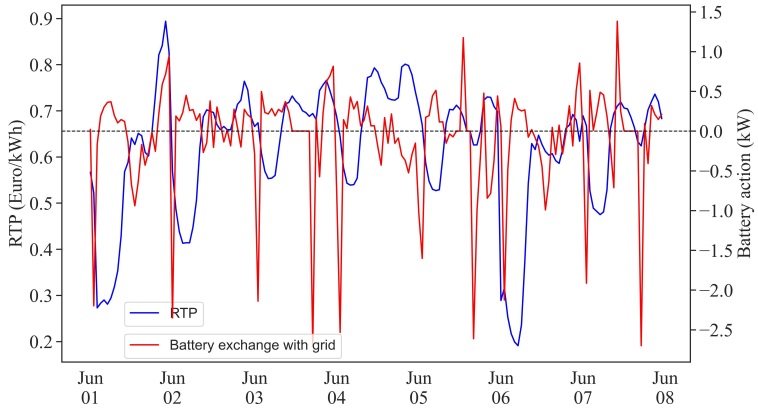


Figure 6.27: RL-based EMS : Battery exchange with grid vs RTP

6.2.4. Comparative analysis

The proposed RL-based energy management system is evaluated against the baseline and benchmark models across three key aspects : cost savings for the end consumer, benefits to the grid operator and the effect on the life of BESS.

The effects on the grid operator are analyzed through daily peak power consumption, daily load factor and ramping power. Peak shaving is a critical part of grid congestion management to improve grid reliability by reducing strain on the grid and for better resource management. However, the load studied here is for a single household which is expected to have minimal peak shaving impact if the operator incentivizes demand side products for homeowners which would significantly influence overall benefits to the consumer. For this study, it assumed that the grid operator does not provide any financial incentives for peak shaving and only benefit is through trading with the grid.

The peak power of the daily purchased electricity over the testing period is shown in Fig.6.28. It is evident that the daily peak power is significantly lower for the baseline compared to the other two approaches, this is because the BESS does not trade with the grid to leverage arbitrage and consumption at source is given precedence. When observing the benchmark and RL algorithms, they exhibit higher peaks as the purchased electricity includes fulfilling the load stored in BESS for selling at the higher price point.

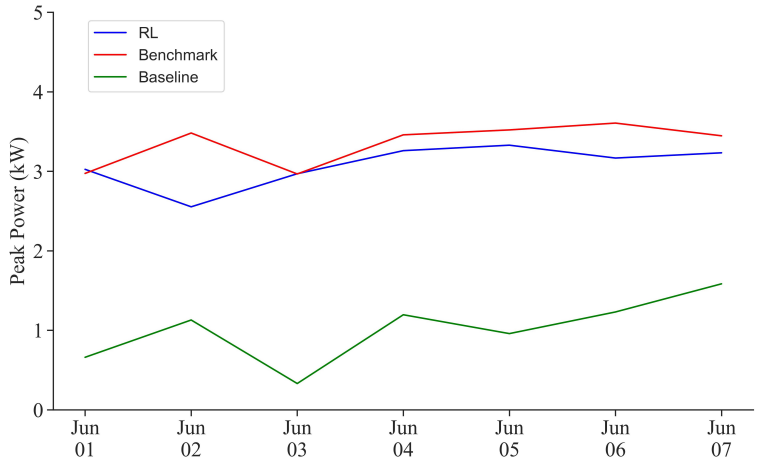


Figure 6.28: Comparison of Daily Peak Power Consumption

The Fig.6.29 illustrates the daily load factor of the power purchased and it is evident that RL-based EMS has a higher load factor than the other two models. The load factor indicates the electrical system's efficiency, consistency and predictability of the load profile. Here, this could be inferred as the RL-based EMS is more consistent with its power purchasing pattern than the other two which is suggested to have a more fluctuating pattern. Having a higher load factor helps the grid operator improve the grid's reliability as it exhibits a predictable load profile and better capacity planning.

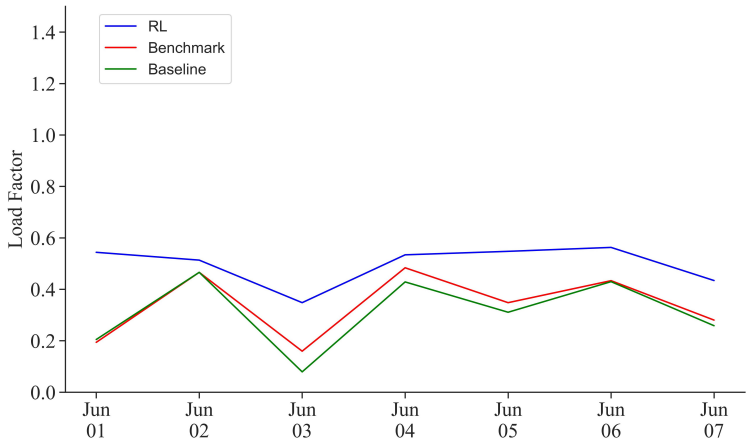


Figure 6.29: Comparison of Daily Load Factor

At an aggregate level, high ramping fluctuations cause system frequency imbalances, pushing the grid operator to activate reserve and ancillary capacities. A lower demand fluctuation would enhance the grid's stability and reduce ancillary costs for grid operator. It can be noticed in Fig.6.30 that the RL-based EMS has a higher median ramping relative to the other two algorithms but a lower extreme variability compared to the benchmark model. As RL takes sequential decisions with future uncertainty signals, it can spread the charging of the battery over multiple time steps as opposed to the benchmark which charges the battery to the fullest at the first instance when it reaches the threshold.

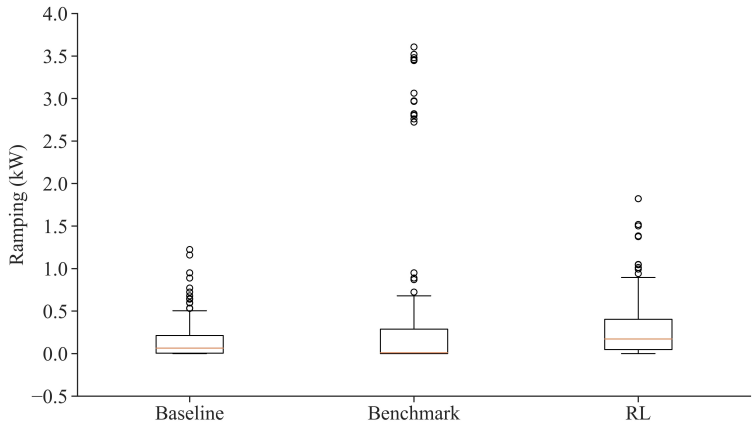


Figure 6.30: Box Plot Analysis of Ramping

The focus of the EMS is centered around optimizing the dispatch of the BESS. Various strategies involve using the battery for different durations, potentially leading to its degradation. The count of cycles was determined using Equation.4.19 and the results are compared in Fig.6.31, there is a significant increase in cycle count for RL and this could be ascribed to the increased exchange with the grid to leverage arbitrage opportunities. The increased cycle count is also evident in the agent's actions compared to the baseline seen in the utilization of the battery capacity in Fig.6.11 and Fig.6.23. The increase in cycle count would lead to faster deterioration of the battery, for which the system would incur replacement costs over its lifetime.

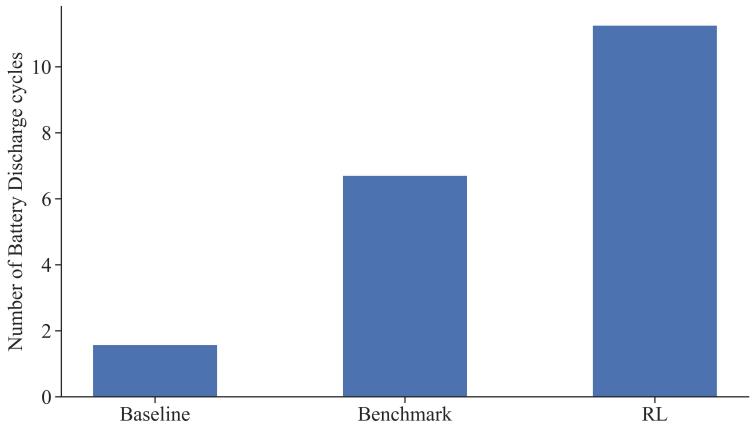


Figure 6.31: Count of Battery Charge and Discharge Cycles

The project's primary goal was to assess the cost-effectiveness of the DDPG-based EMS, which was reflected in the reward function of the RL problem. It was found that the DDPG-based EMS minimized the base electricity cost by 26.65% compared to the baseline and by 14.5% compared to the set benchmark model as illustrated in Fig.6.32. Additionally, the cost of using the battery was incorporated using the cost per cycle which was found to be 0.41 Euros/cycle. It also should be noted that the total electricity cost here does not include the LCOE of PV generated as it is assumed to be constant across all algorithms and the focus is on evaluating the efficacy of the DDPG agents. When factoring in the cost associated with battery usage,

the results reflected a similar trend to the base electricity cost with RL minimizing the total electricity cost by 14.2% compared to the baseline and by 11% compared to the benchmark. The increase in financial benefit to the consumer by agent-based EMS could be attributed to the sequential decision making which led to beneficial arbitrage decisions. The summary of the results are shown in Table 6.4

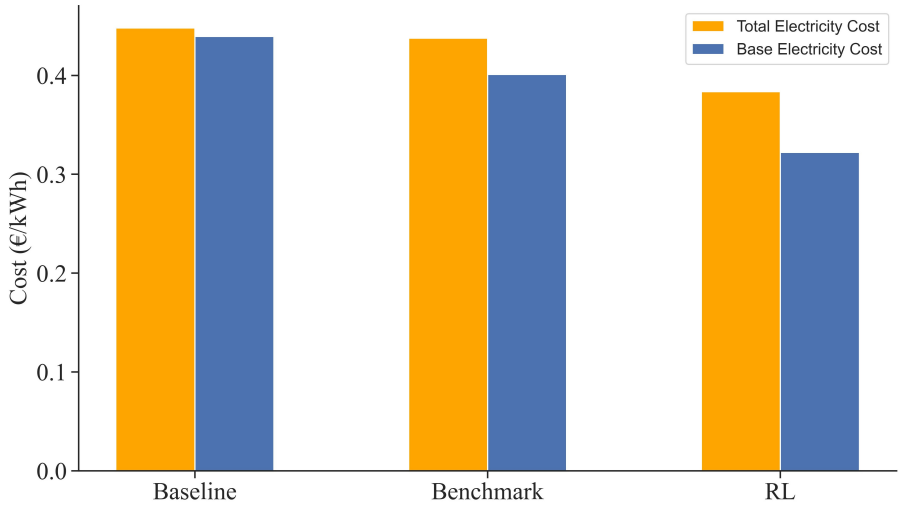


Figure 6.32: Comparative Analysis of Cost-Effectiveness

Table 6.4: Summary of comparative evaluation of EMS

Model	Base Electricity cost (Euro/kWh)	Total Electricity cost (Euro/kWh)	Cycle count	Average Ramping (kW)
Baseline	0.43	0.44	1.58	0.15
Benchmark	0.40	0.43	6.70	0.37
DDPG	0.32	0.37	9.94	0.29

7

Conclusion

Given the need for adaptable and efficient energy management solutions due to the increasing intermittency and complexity of the grid, this research aimed to model an RL agent-based EMS to minimize the electricity cost for a single residential household consisting of uncontrollable load, PV generation and BESS under a dynamic electricity pricing structure. The research consists of two parts. The first part explores the various deep-learning methods to forecast the uncertain variables, and the second part involves feeding these values into the RL-based EMS to make optimal BESS dispatch decisions.

7.1. Answer to the research question

As laid out in the introductory chapter, the main research question is divided into five parts. Here, the sub-questions are addressed first, followed by an in-depth answer to the main research question.

1. **What deep learning-based models could be employed to forecast household load, PV generation, and electricity prices?**
   In the literature survey conducted in Chapter 2, various neural network based forecasting architectures are published and most of the foundational algorithms can solve a prediction problem, but recurrent neural networks were found to be most effective when dealing with time series as they have an inherent capability to model temporal data. In particular, RNN variants of Long short-term memory architectures were most suitable for the task as they contain an internal and a hidden state to learn short-term and long-term patterns in the data.
2. **What deep learning prediction algorithm achieves the highest accuracy when forecasting load, market price and PV generation for integrating into the HEMS?**
   The neural network architecture of MLP, CNN, Simple RNN, LSTM, Bi-directional LSTM, and CNN-LSTM hybrid were optimized and tested individually for forecasting Load, Electricity market prices and PV generation. It was found that Bi-Directional LSTM was the most accurate algorithm for all three tasks with the least root mean squared error (RMSE), which was chosen to be integrated into the EMS. It was also observed that the performance differences between

different networks were negligible, especially between recurrent neural network variants. Another observation was that the accuracy was reduced progressively with the increase in the forecast horizon.

3. **What type of reinforcement learning algorithm is most suitable for solving energy management system problems**
   In investigating the existing literature in Chapter 2, several studies in the field were primarily done using a value-based model-free RL architecture, Deep Q-learning (DQN). However, recent works have introduced actor-critic architectures that work in continuous action space which becomes essential, especially when the BESS action space is continuous as opposed to DQN which outputs discrete actions. Moreover, Deep deterministic policy gradient (DDPG), which belongs to the actor-critic framework which generates deterministic and continuous actions making it a suitable choice for energy management solutions.
4. **How to model an RL-based EMS with load, BESS, and PV to increase cost savings for the consumer?**
   The problem was framed as a partially observable Markov decision process (POMDP) and modeled as a multi-agent DDPG (MADDPG) problem. Two coordinated agents were defined, with one agent solely managing the exchanges between the BESS and the grid whereas the other agent only decides on discharging the BESS to the load. The MADDPG framework employed has a deterministic rule-based approach for dispatching PV and was not incorporated as part of its RL internal processes. The observation space in the EMS environment includes Load, PV generated, Market price and their respective forecasted values at future horizons. The reward function is comprised of elements to minimize the base electricity cost while ensuring that the BESS operates within its operational constraints. The episode length of 24 hours was chosen for the agent to learn to minimize cost per day. The agents were trained on 30 days of real-world data and tested on out of the box sample of a 7-day period on various evaluation metrics.
5. **What is the performance of the RL-based EMS when compared to a rule-based deterministic model in terms of cost savings for the end consumer?**
   The RL-based EMS was compared against two models one is deterministic rule-based and the following is an extension of the latter which incorporates the price signal in the decision-making process which is referred to as the benchmark. The results simulated for 7 days showed that the RL-based EMS increased the cost savings of the consumer by 14.2% compared to the baseline and by 11% compared to the benchmark. This reduction in total electricity cost was achieved because of the BESS leveraging arbitrage opportunities with the grid. RL approach led to a sixfold increase in battery cycle count compared to the baseline and 1.5 fold compared to the benchmark which accelerates the deterioration of the battery and hence increases the replacement costs over the lifetime. On the effects observed on the grid operator, the RL-based EMS exhibited a higher load factor than the other algorithms, indicating a higher but stable demand pattern. On examining the peak power consumption, the RL-based EMS showed significantly higher peaks and required higher ramping than the baseline.

The main research question was constructed as

How can deep learning-based models and reinforcement learning algorithms be effectively employed to optimize energy management in a household and how do these models compare to traditional rule-based approaches in terms of cost savings for the consumer?

The first part of the study revealed that the Bidirectional LSTM produced the most accurate predictions of hourly load, electricity market prices and PV generation to be integrated into the HEMS assessed in terms of RMSE. The performance differences between the compared deep learning models were observed to the minimal, especially among the recurrent neural networks. Additionally, It was also noticed that forecast errors significantly increase with the prediction horizon.

In the second part, to enhance real-time BESS decision-making for HEMS, a multi-agent DDPG reinforcement model was proposed and modeled to increase the cost savings for the end consumer. This algorithm was compared against two models, one without the price signal and one with the price signal. It was found that the RL-based EMS increased the cost savings for the consumer by 14.2% compared to the baseline but increased the stress on the grid which is evident from higher peak consumption, higher ramping and load factor.DDPG displayed high sensitivity to hyperparameters which led to breaking the physical boundaries of the BESS in the test runs. Finding a balance between enabling the agents to learn the physical constraints of the BESS while also maintaining stable and efficient learning is a challenge. It should be noted that these experiments were conducted in offline training mode and extending such a system online could offer greater robustness in adapting to recent changes highlighting the strength of RL.

7.2. Recommendations and Future Work

This study demonstrated the financial effectiveness of the proposed multi-agent DDPG EMS approach. However, further research is required to explore and deploy the discussed algorithm effectively in a real-world setting. The following recommendations are given for future work.

- The current problem definition could be expanded to include the reward functions to find an optimal policy taking grid congestion alleviation into account.
- Training the agents on an extensive dataset and assessing their performance in different seasons to make it a robust model.
- Two agent DDPG architecture could be extended to add two more agents to coordinate the optimal dispatch of PV between load, grid and the BESS.
- Compare the RL EMS against MILP-based model predictive control to have a comprehensive assessment of its performance.
- Hierarchical RL with hybrid action spaces could be implemented to avoid periods of battery action which is not beneficial by making the BESS inactive.
- Perform sensitivity analysis of DDPG agents with respect to different coefficients of reward components and hyperparameters to study how it impacts cost savings.

References

[1] "Entsoe transparency platform." (), [Online]. Available: https://ec.europa.eu/eurostat/web/products-eurostat-news/-/ddn-20230613-1#:~:text=Energy%20use%20in%20households%20up%206%25%20in%202021%20%2D%20Products%20Eurostat%20News,-Back%20Energy%20use&text=In%202021%2C%20households%20accounted%20for,%25)%20and%20electricity%20(24.6%25). (visited on 06/15/2021).

[2] "Renewable energy targets." (), [Online]. Available: https://energy.ec.europa.eu/topics/renewable-energy/renewable-energy-directive-targets-and-rules/renewable-energy-targets_en (visited on 06/15/2022).

[3] "Annual rooftop and utility scale installations in the eu." (), [Online]. Available: https://www.solarpowereurope.org/advocacy/solar-saves/fact-figures/annual-rooftop-and-utility-scale-installations-in-the-eu (visited on 06/15/2021).

[4] R. Khezri, A. Mahmoudi, and M. H. Haque, "Optimal capacity of solar pv and battery storage for australian grid-connected households," *IEEE Transactions on Industry Applications*, vol. 56, no. 5, pp. 5319–5329, 2020.

[5] R. Dufo-López, "Optimisation of size and control of grid-connected storage under real time electricity pricing conditions," *Applied Energy*, vol. 140, pp. 395–408, 2015.

[6] M. Beaudin and H. Zareipour, "Home energy management systems: A review of modelling and complexity," *Renewable and sustainable energy reviews*, vol. 45, pp. 318–335, 2015.

[7] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[8] E. Mocanu, D. C. Mocanu, P. H. Nguyen, *et al.*, "On-line building energy optimization using deep reinforcement learning," *IEEE transactions on smart grid*, vol. 10, no. 4, pp. 3698–3708, 2018.

[9] P. Lissa, C. Deane, M. Schukat, F. Seri, M. Keane, and E. Barrett, "Deep reinforcement learning for home energy management system control," *Energy and AI*, vol. 3, p. 100 043, 2021.

[10] A. Gasparin, S. Lukovic, and C. Alippi, "Deep learning for time series forecasting: The electric load case," *CAAI Transactions on Intelligence Technology*, vol. 7, no. 1, pp. 1–25, 2022.

[11] P. Huber, M. Ott, M. Friedli, A. Rumsch, and A. Paice, "Residential power traces for five houses: The ihomelab rapt dataset," *Data*, vol. 5, no. 1, p. 17, 2020.

[12] H. Golmohamadi, R. Keypour, B. Bak-Jensen, and J. R. Pillai, “Optimization of household energy consumption towards day-ahead retail electricity price in home energy management systems,” *Sustainable Cities and Society*, vol. 47, p. 101 468, 2019.

[13] W. Kong, Z. Y. Dong, D. J. Hill, F. Luo, and Y. Xu, “Short-term residential load forecasting based on resident behaviour learning,” *IEEE Transactions on power systems*, vol. 33, no. 1, pp. 1087–1088, 2017.

[14] Y. Chakhchoukh, P. Panciatici, and L. Mili, “Electric load forecasting based on statistical robust methods,” *IEEE Transactions on Power Systems*, vol. 26, no. 3, pp. 982–991, 2010.

[15] A. K. Dubey, A. Kumar, V. García-Díaz, A. K. Sharma, and K. Kanhaiya, “Study and analysis of sarima and lstm in forecasting time series data,” *Sustainable Energy Technologies and Assessments*, vol. 47, p. 101 474, 2021.

[16] H. Hewamalage, C. Bergmeir, and K. Bandara, “Recurrent neural networks for time series forecasting: Current status and future directions,” *International Journal of Forecasting*, vol. 37, no. 1, pp. 388–427, 2021.

[17] W. Kong, Z. Y. Dong, Y. Jia, D. J. Hill, Y. Xu, and Y. Zhang, “Short-term residential load forecasting based on lstm recurrent neural network,” *IEEE transactions on smart grid*, vol. 10, no. 1, pp. 841–851, 2017.

[18] M. Alhussein, K. Aurangzeb, and S. I. Haider, “Hybrid cnn-lstm model for short-term individual household load forecasting,” *Ieee Access*, vol. 8, pp. 180 544–180 557, 2020.

[19] G. Dudek, P. Pełka, and S. Smyl, “A hybrid residual dilated lstm and exponential smoothing model for midterm electric load forecasting,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 7, pp. 2879–2891, 2021.

[20] D. Rangelov, M. Boerger, N. Tcholtchev, P. Lämmel, and M. Hauswirth, “Design and development of a short-term photovoltaic power output forecasting method based on random forest, deep neural network and lstm using readily available weather features,” *IEEE Access*, 2023.

[21] M. J. Mayer and G. Gróf, “Extensive comparison of physical models for photovoltaic power forecasting,” *Applied Energy*, vol. 283, p. 116 239, 2021.

[22] F. Mulder, “Implications of diurnal and seasonal variations in renewable energy generation for large scale energy storage,” *Journal of Renewable and Sustainable Energy*, vol. 6, no. 3, p. 033 105, 2014.

[23] S. Sobri, S. Koohi-Kamali, and N. A. Rahim, “Solar photovoltaic generation forecasting methods: A review,” *Energy conversion and management*, vol. 156, pp. 459–497, 2018.

[24] Y. Li, Y. Su, and L. Shu, “An armax model for forecasting the power output of a grid connected photovoltaic system,” *Renewable Energy*, vol. 66, pp. 78–89, 2014.

[25] A. Jędrzejewski, J. Lago, G. Marcjasz, and R. Weron, “Electricity price forecasting: The dawn of machine learning,” *IEEE Power and Energy Magazine*, vol. 20, no. 3, pp. 24–31, 2022.

[26] P.-H. Kuo and C.-J. Huang, “An electricity price forecasting model by hybrid structured deep neural networks,” *Sustainability*, vol. 10, no. 4, p. 1280, 2018.

[27] A. Vagale, L. Šteina, and V. Vēciņš, “Time series forecasting of mobile robot motion sensors using lstm networks,” *Applied Computer Systems*, vol. 26, no. 2, pp. 150–157, 2021.

[28] R. Balakrishnan and V. Geetha, “Review on home energy management system,” *Materials Today: Proceedings*, vol. 47, pp. 144–150, 2021.

[29] K. Mason and S. Grijalva, “A review of reinforcement learning for autonomous building energy management,” *Computers & Electrical Engineering*, vol. 78, pp. 300–312, 2019.

[30] Y. Zou, T. Liu, D. Liu, and F. Sun, “Reinforcement learning-based real-time energy management for a hybrid tracked vehicle,” *Applied energy*, vol. 171, pp. 372–382, 2016.

[31] W. Zhang, J. Wang, S. Du, H. Ma, W. Zhao, and H. Li, “Energy management strategies for hybrid construction machinery: Evolution, classification, comparison and future trends,” *Energies*, vol. 12, no. 10, p. 2024, 2019.

[32] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, “Deep reinforcement learning: A brief survey,” *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[33] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.

[34] D. Silver, A. Huang, C. J. Maddison, *et al.*, “Mastering the game of go with deep neural networks and tree search,” *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[35] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, pp. 279–292, 1992.

[36] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.

[37] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” *arXiv preprint arXiv:1511.05952*, 2015.

[38] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” in *International conference on machine learning*, Pmlr, 2014, pp. 387–395.

[39] T. P. Lillicrap, J. J. Hunt, A. Pritzel, *et al.*, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.

[40] L. Yu, W. Xie, D. Xie, *et al.*, “Deep reinforcement learning for smart home energy management,” *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 2751–2762, 2019.

[41] D. O’Neill, M. Levorato, A. Goldsmith, and U. Mitra, “Residential demand response using reinforcement learning,” in *2010 First IEEE international conference on smart grid communications*, IEEE, 2010, pp. 409–414.

[42] J. Achiam, “Spinning Up in Deep Reinforcement Learning,” 2018.

[43] G. Han, S. Lee, J. Lee, K. Lee, and J. Bae, “Deep-learning-and reinforcement-learning-based profitable strategy of a grid-level energy storage system for the smart grid,” *Journal of Energy Storage*, vol. 41, p. 102 868, 2021.

[44] Y. Liu, D. Zhang, and H. B. Gooi, “Optimization strategy based on deep reinforcement learning for home energy management,” *CSEE Journal of Power and Energy Systems*, vol. 6, no. 3, pp. 572–582, 2020.

[45] G. Henri, T. Levent, A. Halev, R. Alami, and P. Cordier, “Pymgrid: An open-source python microgrid simulator for applied artificial intelligence research,” *arXiv preprint arXiv:2011.08004*, 2020.

[46] J. R. Vázquez-Canteli, S. Dey, G. Henze, and Z. Nagy, “The citylearn challenge 2020,” in *Proceedings of the 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, 2020, pp. 320–321.

[47] A. Tealab, “Time series forecasting using artificial neural networks methodologies: A systematic review,” *Future Computing and Informatics Journal*, vol. 3, no. 2, pp. 334–340, 2018.

[48] J. Brownlee, *Deep learning for time series forecasting: predict the future with MLPs, CNNs and LSTMs in Python*. Machine Learning Mastery, 2018.

[49] J. C. B. Gamboa, “Deep learning for time-series analysis,” *arXiv preprint arXiv:1701.01887*, 2017.

[50] S. E. Razavi, A. Arefi, G. Ledwich, G. Nourbakhsh, D. B. Smith, and M. Minakshi, “From load to net energy forecasting: Short-term residential forecasting for the blend of load and pv behind the meter,” *IEEE Access*, vol. 8, pp. 224 343–224 353, 2020.

[51] M. Massaoudi, S. S. Refaat, I. Chihi, M. Trabelsi, F. S. Oueslati, and H. Abu-Rub, “A novel stacked generalization ensemble-based hybrid lgbm-xgb-mlp model for short-term load forecasting,” *Energy*, vol. 214, p. 118 874, 2021.

[52] S. Albawi, T. A. Mohammed, and S. Al-Zawi, “Understanding of a convolutional neural network,” in *2017 international conference on engineering and technology (ICET)*, Ieee, 2017, pp. 1–6.

[53] C.-J. Huang and P.-H. Kuo, “A deep cnn-lstm model for particulate matter (pm2. 5) forecasting in smart cities,” *Sensors*, vol. 18, no. 7, p. 2220, 2018.

[54] H. Dong, H. Dong, Z. Ding, S. Zhang, and Chang, *Deep Reinforcement Learning*. Springer, 2020.

[55] T. Tiong, I. Saad, K. T. K. Teo, and H. bin Lago, “Deep reinforcement learning with robust deep deterministic policy gradient,” in *2020 2nd International Conference on Electrical, Control and Instrumentation Engineering (ICECIE)*, IEEE, 2020, pp. 1–5.

[56] J. Snoek, H. Larochelle, and R. P. Adams, “Practical bayesian optimization of machine learning algorithms,” *Advances in neural information processing systems*, vol. 25, 2012.

[57] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization.,” *Journal of machine learning research*, vol. 13, no. 2, 2012.

[58] J. Brownlee, *Probability for machine learning: Discover how to harness uncertainty with Python*. Machine Learning Mastery, 2019.

[59] “Entsoe transparency platform.” (), [Online]. Available: `https://transparency.entsoe.eu/` (visited on 06/15/2023).

[60] S. B. Kotsiantis, D. Kanellopoulos, and P. E. Pintelas, “Data preprocessing for supervised leaning,” *International journal of computer science*, vol. 1, no. 2, pp. 111–117, 2006.

[61] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, 2nd. Otexts, 2018.

[62] G. E. Uhlenbeck and L. S. Ornstein, “On the theory of the brownian motion,” *Physical review*, vol. 36, no. 5, p. 823, 1930.

[63] C. Bergmeir and J. M. Benítez, “On the use of cross-validation for time series predictor evaluation,” *Information Sciences*, vol. 191, pp. 192–213, 2012.

A

IEEE Conference paper

A Comparison of Various Deep Learning Methods for Household Load Forecasting

Karthikeyan Deivamani
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Delft, The Netherlands
K.Deivamani@student.tudelft.nl

Farshid Norouzi
Department of DC Systems Energy Conversion and Storage
Delft University of Technology
Delft, The Netherlands
f.norouzi@tudelft.nl

Aditya Shekhar
Department of DC Systems, Energy Conversion and Storage
Delft University of Technology
Delft, The Netherlands
a.shekhar@tudelft.nl

Pavol Bauer
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Delft, The Netherlands
P.Bauer@tudelft.nl

***Abstract*—Forecasting energy consumption is vital for smart grid operations to manage demand, plan loads, and optimize grid operations. This work aims at reviewing and experimentally evaluating six univariate deep learning architectures to forecast load for a single household using a real-world dataset. Multi-layer perceptron (MLP), Convolutional neural network (CNN) and recurrent neural networks (Simple RNN, Long Short Term Memory (LSTM)) were the neural network methods that were analysed along with robust LSTM architectures like Bi-directional LSTM and CNN-LSTM Hybrid. All the models were tuned using Bayesian optimization and evaluated using root mean squared error (RMSE) as the metric. In addition to neural network models, Seasonal ARIMA (SARIMA) a statistical model is also presented to observe the performance. As a result, Bi-directional LSTM was observed to have achieved the best performance with the smallest value of RMSE; however, it was also observed that differences in performances between other neural network models were quite low, especially between the RNN architectures. Additionally, although machine learning methods performed better than SARIMA the former model was more complex and computationally intensive.**

***Index Terms*—electric load forecasting, smart grid, time-series forecasting, univariate, deep learning**

I. Introduction

Microgrids are a promising solution in making the electric grid more reliable and green by improving energy reliability, energy sharing and demand-side management aspects. To leverage the full capabilities of a microgrid, accurate load forecasting becomes a critical task either from a consumer perspective to reduce consumption or from a grid operator perspective for a better decision-making process or for efficient energy storage system management. With the rise in advanced monitoring infrastructure more granular and extensive data is being collected. Deep learning forecasting methods have demonstrated significant potential in effectively managing larger and more intricate datasets [1].

979-8-3503-9678-2/23/$31.00 ©2023 IEEE

Short-term load forecasting (STFL) is the process of predicting the power demand of a power system over a short-term period, typically ranging from a few minutes to a few hours. Deep learning has demonstrated improved performance in modeling complex patterns for individual household load profiles, which tend to be more volatile due to their dependence on individual behavior, as opposed to aggregate level modeling [2].

In literature, different types of models both linear and non-linear have been used for STFL. Family of Auto-regressive moving average (ARMA) models were pioneers in STFL [3] which was then evolved into SARIMA to account for seasonal variance [4]. The limitation of this set of statistical methods is that it assumes a linear system whereas most often real-world cases exhibit non-linear properties. In order to solve this shortcoming, models like feed-forward neural networks have become attractive as they show capabilities in modelling complex non-linear systems such as load forecasting [1]. Neural network techniques range from simple MLP to convolutional methods to recurrent neural networks [5] along with their variants LSTM [6] and Gated recurrent units (GRU). Hybrid architectures have also been proposed in the literature between neural networks as well as between statistical and machine learning methods such as CNN-LSTM hybrid presented in [7] and a hybrid LSTM-Exponential smoothening [8] respectively.

The scope of this paper is to provide a comparative analysis of basic deep-learning architectures for STFL of a single household and compare using standard error metrics such as RMSE and MAE to input into an energy management system.

This paper is organized as follows: Section II reviews the load dataset, preprocessing steps, error metrics and the details of the models being evaluated. Next, section III presents the results and discussion of parameter search and load forecasting models. Finally, the conclusions are provided in section IV.

II. Methodology

A. Data Collection and preparation

The dataset used in this analysis is the hourly load from a single household which was retrieved from the IHomeLab RAPT dataset [9]. The data is from a household in the greater Lucerne region in Switzerland and the data spans from 1 December 2016 to 31 July 2019 which is around two and a half years.

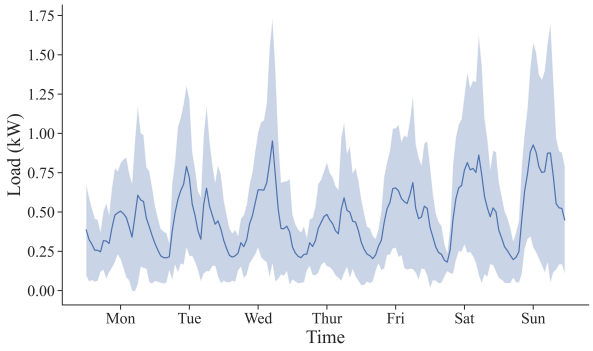


Fig. 1. Weekly statistics for the load in the whole IHomeLab dataset. The bold line is the mean and the blue area covers one standard deviation from the mean

In Fig. 1 the bold line represents the mean of that hour over every week in the dataset, here it could be observed that there are high fluctuations and a higher mean during Sunday,Saturday and Wednesday. However, higher consumption during the weekend is expected but Wednesday is an interesting observation.

The missing values in the dataset were located and filled using the previous value and to deal with non-stationarity the data was differenced from the previous value [10]. Stationarity of the data was checked using Augmented Dickey-Fuller (ADF) test [11].

B. Forecasting models

1) Baseline:

A seasonal naive model was used which is using the same value of the previous season as the predicted value this is also called a persistent model. In this analysis, since the daily seasonality is strong as inferred from auto-correlation values the previous day were used as the forecasting value for the coming day.

2) Statistical Model (SARIMA):

SARIMA is a class of time series forecasting techniques that predicts the future values based only on the past behavior of the variable being modelled along with accounting for seasonalities [4]. The mathematical representation of the model is given below.

$$y_t = c + \sum_{n=1}^{p} \phi_n y_{t-n} + \sum_{n=1}^{q} \theta_n \epsilon_{t-n} + \sum_{n=1}^{P} \Phi_n y_{t-sn} + \sum_{n=1}^{Q} \Theta_n \epsilon_{t-sn} + \epsilon_t \quad (1)$$

SARIMA (p,d,q)(P, D, Q)s is a parametric model, and (p,d,q) is the auto-regressive lag order, order of differencing, moving average lag order respectively and (P, D, Q) is for the seasonal terms. In (1) the c is a constant term, ϵ_t is the error term and ϕ_n,θ_n,Θ_n,Φ_n are the coefficients of lag terms .

3) Artificial Neural Network (ANN) Methods:

Neural Networks are part of a family of machine-learning techniques inspired by the functioning of the human brain which consists of interconnected nodes (neurons) compartmentalized into layers [12]. A typical neural network architecture consists of an input, output layer along with hidden layers where the abstractsa are learned . For this forecasting problem, 24 lags were utilized as the feature due to its strong diural pattern [13]. Different architectures of neural networks built on the foundational structure are discussed below.

a) Multi layer perceptron (MLP):

Multi-layer perceptron is a type of feed-forward ANN where the information flows in only one direction from the input layer to the output layer. This brings the limitation of not capturing the temporal dependencies regardless MLP has shown to have competitive performance in a few load forecasting use cases [14].

b) Convolutional Neural Network(CNN):

Convolutional neural networks are a family of ANN which works with a grid-like structure and have been extensively used in image recognition and natural language processing [15]. The key property of CNNs to extract features has been leveraged for univariate time-series forecasting, where a filter/kernel is passed through the series to extract relevant features. Although CNN has shown to be effective in NLP and image recognition, they are not widely used in time series forecasting as CNN cannot model sequential data, which has been addressed recently by combining them with recurrent neural networks [2]. A vanilla CNN and a CNN-LSTM hybrid architectures have been analyzed in this work.

c) Recurrent Neural Network (RNN):

Recurrent neural networks is a NN architecture that modifies feed-forward neural networks to handle sequential data and capture patterns better, which makes it a powerful tool for time series forecasting [5]. Compared to feed-forward neural networks, RNNs maintain an internal state that allows them to remember information from previous inputs.RNN's internal feedback loop allows it to use its previous outputs as inputs to the current step, making it capable of modeling sequences and capturing temporal dependencies in the data.

d) Long Short Term Memory (LSTM):

LSTM is a variant of RNN that was developed to address the issue of vanishing or exploding gradients during backpropagation. This problem occurs when the gradient values become too small or too large, making it difficult for the network to update the weights effectively [16].LSTMs have a hidden state and a cell state which store short-term dependencies and long-term information. This ability of LSTM to store long-term dependencies has it a popular choice in load forecasting [6]. One extension of LSTM is the Bi-directional LSTM, as unidirectional LSTM processes the input sequence only in the

TABLE I
HYPERPARAMETERS AND PARAMETER RANGES FOR DEEP LEARNING MODELS

Hyperparameter	Range
Number of hidden layers	[1,2,3]
Number of neurons	(1,120)
Dropout ratio	(0,0.5)
Learning rate	(0.00001,0.01)

forward direction, and Bi-LSTM processes both ways forward and backward [17]. This allows the model to capture a better sequential relationship in the input sequence. Unidirectional and Bi-directional LSTMs both have been explored in this analysis.

C. Hyperparameter Tuning

Hyperparameter tuning is important in time series forecasting to find the best model hyperparameters that minimize forecast error. RMSE is a commonly used target metric for this optimization process.

For SARIMA, the autoregressive, differencing, moving average terms $(p, d, q)x(P, D, Q)$ were analyzed using a correlogram, after which a grid search was done to obtain best-fit hyperparameters [4].

Two common methods for hyperparameter tuning are grid search and random search. However, as the number of hyperparameters to tune increases, the number of possible combinations can grow exponentially, making the search computationally intensive and sometimes infeasible. Bayesian optimization is a probabilistic model which finds an optimal set of hyperparameters in fewer evaluations. After every iteration, it updates the search algorithm and avoids the low-performing region [18]. For this comparative study, a simple Bayesian algorithm was performed for the hyperparameter in the range as mentioned in table II.

D. Performance Metrics

The accuracy of models and neural network architectures is typically assessed using scale-dependent error metrics like root mean squared error (RMSE) and mean average error (MAE) [19]. RMSE is particularly effective as it places greater weight on large errors and can handle values close to zero. Since they are scale-dependent, they cannot be used to compare different datasets. R-squared (R^2) is a statistical measure that represents the proportion of the variance in predicted values is explained by the true values, (4) shows the mathematical representation.

1) Mean Absolute Error (MAE):

$$MAE = \frac{1}{n}\sum |\hat{\mathbf{y}}_\mathbf{t} - \mathbf{y}_\mathbf{t}| \tag{2}$$

2) Root Mean Squared Error (RMSE):

$$RMSE = \sqrt{\frac{1}{n}\sum |\hat{\mathbf{y}}_\mathbf{t} - \mathbf{y}_\mathbf{t}|^\mathbf{2}} \tag{3}$$

3) R-Squared:

$$R^2 = 1 - \frac{\sum(\mathbf{y_i} - \hat{\mathbf{y}})^2}{\sum(\mathbf{y_i} - \bar{\mathbf{y}})^2} \tag{4}$$

In (2) and (3), $\hat{\mathbf{y}}_\mathbf{t}$ is the predicted value and $\mathbf{y}$ is the actual value. To assess model accuracy, a combination of metrics is needed to inspect different aspects of the forecast. In this paper, RMSE is chosen as the metric over which models are optimized as well as for evaluation.

A holdout validation was undertaken for evaluation where the dataset was split into 70% for training 20% for validation and 10% for testing without shuffling of data points.

III. RESULTS AND DISCUSSION

In this section, the results of the above mentioned models using the IHomeLab dataset and accuracy on the test data using RMSE and MAE are analyzed.

TABLE II
SUMMARY OF PREDICTION ERRORS (RMSE AND MAE) FOR ABOVE MODELS

Models	MAE (kW)	RMSE (kW)	$\mathbf{R^2}$
Persistence Model (Baseline)	0.251	0.410	-0.356
SARIMA	0.231	0.322	0.149
Deep Learning Models			
Multilayer Perceptron (MLP)	0.195	0.281	0.359
Convolutional Neural Network(CNN)	0.179	0.275	0.363
Recurrent Neural Network (RNN)	0.183	0.277	0.361
Long Short Term Memory (LSTM)	0.175	0.270	0.374
CNN-LSTM Hybrid	0.181	0.271	0.404
Bidirectional LSTM	0.173	0.269	0.409

A. SARIMA

For the SARIMA model, after analyzing autocorrelation and partial autocorrelation plot followed by a grid search the optimum parameters were found to be $p = 2, d = 1, q = 1, P = 3, D = 0, Q = 2$ with the seasonality of 24 hours. As seen in the table II RMSE and MAE were 0.231 kW and 0.322 kW, on comparing this with the baseline performance SARIMA had lower RMSE and MAE. In Fig. 2, the predicted values capture the daily variation but fail to capture the increase during the weekend as the model is limited to one seasonality. As it assumes a linear relationship and stationarity

in data which states constant variance so the model fails to capture the non-linearity. A key advantage of SARIMA and statistical models, in general, is that they need less data to train compared to machine learning methods which makes them robust when fewer data is available. This model is introduced to compare a well-popular statistical model with deep learning methods.

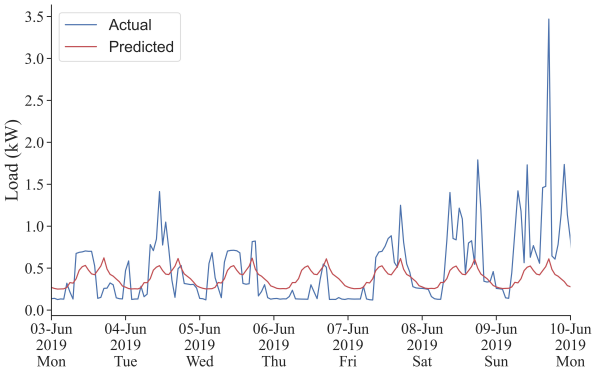


Fig. 2. Forecasted (Red) and Actual (Blue) Load profile for a week using SARIMA Model

B. Multilayer Perceptron (MLP)

In Fig. 3, it can be seen that the MLP model is able to capture the trend of the actual data and fit better than the SARIMA model with a RMSE of 0.281 kW. It can also be noticed that the model does capture variance significantly better visually and by inferring R-squared value in Table II . The parameter chosen were 2 dense layers with the first layer of 90 neurons and the second layer of 20 neurons, a learning rate of 0.0001, dropout ratio of 0.2 for 100 epochs.

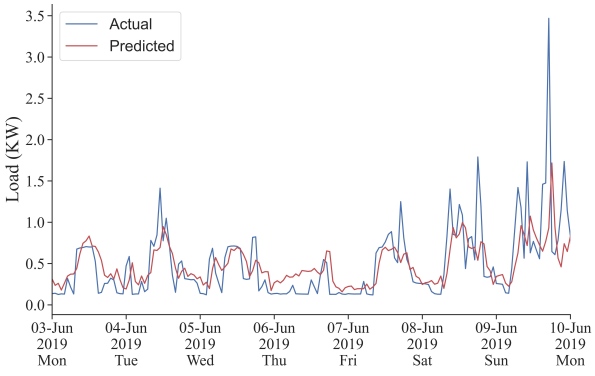


Fig. 3. Forecasted (Red) and Actual (Blue) Load profile for a week using Multilayer Perceptron (MLP)

One of the main drawbacks of using MLP for load forecasting is that it does not model the long-term dependencies and the temporal relationships between the data. Compared to other architectures mentioned, MLPs are heavily parameterized due to the fully connected nature of dense layers which lead to overfitting of the training data. Regularization by a dropout layer was introduced to tackle this problem and generalize the model.

C. Convolutional Neural Network (CNN)

Optimization of CNN hyperparameters yielded 16 filters, kernel size of 4 and 30 neurons in the dense layer as the optimal hyperparameters from bayesian optimization. It performed better than MLP with a RMSE of 0.275 kW with a slight reduction in RMSE and better capturing of vairance as seen in Fig.4 and Table II . The architecuture of CNN is to capture local patterns through filters explains the variance in the predicted data and it also emphasizes the limitation of not capturing the trend and long-term dependencies in the data.

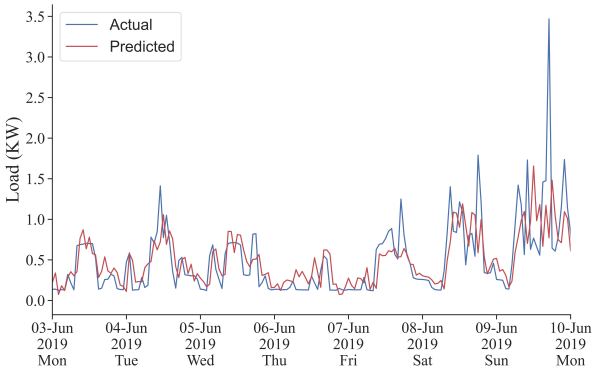


Fig. 4. Forecasted (Red) and Actual (Blue) Load profile for a week using Multilayer Perceptron (CNN)

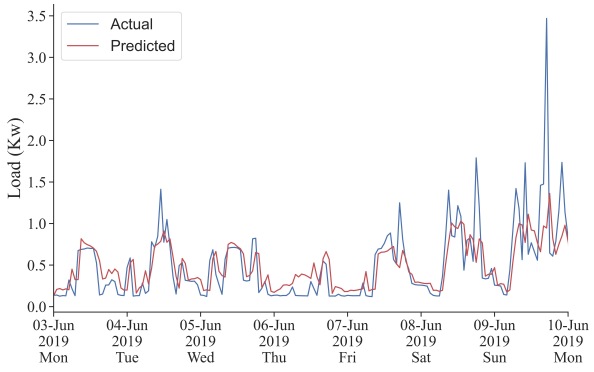


Fig. 5. Forecasted (Red) and Actual (Blue) Load profile for a week using Multilayer Perceptron (Bi-Directional LSTM)

D. Bi-Directional LSTM

In this analysis, Bidirectional LSTM has shown to be the model with the least RMSE of 0.269kW, but the difference between this and other recurrent neural networks (LSTM, CNN-LSTM) is very low as seen in Table II. In Fig. 5, it can be observed that some predictions clearly reflect the influence of the actual value of the previous time step. This is mainly due to the sequential nature of the data and high correlation with the previous time step even after differencing and removing non-stationarity. Adding external features that

influence the load could improve the performance further by providing additional information to the model like for instance weather conditions, holidays, and special events can all impact the electricity consumption of the household. A shallow architecture with a Bi-directional LSTM with 64 neurons and a learning rate of 0.001 was chosen and trained for 80 epochs.

Although, LSTMs have shown good results they suffered from overfitting during experimentation due to smaller train data and a high number of parameters. This was addressed by introducing a regularization dropout layer with a dropout ratio of 0.2.

In this analysis, it was observed that RNNs outperformed the baseline and other neural network methods as seen by their lower RMSE. It was also noted that LSTM based models performed better than simple RNNs with Bi-directional LSTM having a slight edge over others in terms of RMSE, as shown in 6. The problem of overfitting was constantly experienced during the empirical testing for all the models as the data set used to train was limited to only 2 years, and neural networks are complex models that require large amounts of data to achieve better performance.

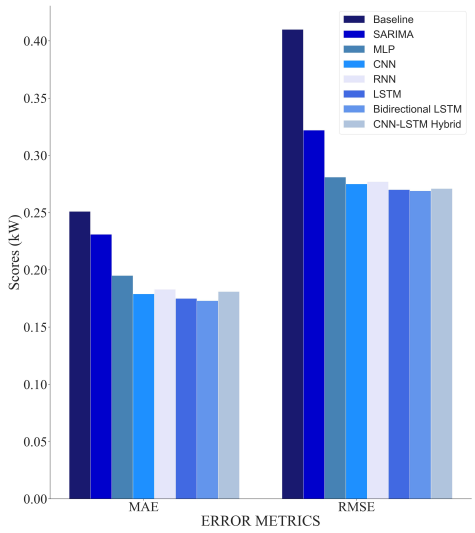


Fig. 6. Comparison of RMSE and MAE for all the discussed models

IV. Conclusion

In this paper, we have compared the univariate single-step load forecasting performance of a statistical model (SARIMA) and primary neural network architectures such as MLP, CNN, RNN and LSTM using rRMSE as the evaluation metric and Bi-Directional LSTM performance was superior to other models.

Although there were only slight variations in the performance of different neural network models, Bi-LSTM exhibited the best results in terms of RMSE, as depicted in Fig. 6 and Table II. Nonetheless, the differences in the performance among these models were not significant. It should also be noted that neural network methods on average performed 12% better than SARIMA but the implementation for the former was complex and computationally intensive than the later. Overall, the findings suggest that deep learning approaches are effective for energy consumption forecasting and can be useful in smart grid operations for managing demand, planning loads, and optimizing grid operations.

References

[1] A. Gasparin, S. Lukovic, and C. Alippi, "Deep learning for time series forecasting: The electric load case," *CAAI Transactions on Intelligence Technology*, vol. 7, no. 1, pp. 1–25, 2022.

[2] W. Kong, Z. Y. Dong, D. J. Hill, F. Luo, and Y. Xu, "Short-term residential load forecasting based on resident behaviour learning," *IEEE Transactions on Power Systems*, vol. 33, no. 1, pp. 1087–1088, 2017.

[3] Y. Chakhchoukh, P. Panciatici, and L. Mili, "Electric load forecasting based on statistical robust methods," *IEEE Transactions on Power Systems*, vol. 26, no. 3, pp. 982–991, 2010.

[4] A. K. Dubey, A. Kumar, V. García-Díaz, A. K. Sharma, and K. Kanhaiya, "Study and analysis of sarima and lstm in forecasting time series data," *Sustainable Energy Technologies and Assessments*, vol. 47, p. 101474, 2021.

[5] H. Hewamalage, C. Bergmeir, and K. Bandara, "Recurrent neural networks for time series forecasting: Current status and future directions," *International Journal of Forecasting*, vol. 37, no. 1, pp. 388–427, 2021.

[6] W. Kong, Z. Y. Dong, Y. Jia, D. J. Hill, Y. Xu, and Y. Zhang, "Short-term residential load forecasting based on lstm recurrent neural network," *IEEE transactions on smart grid*, vol. 10, no. 1, pp. 841–851, 2017.

[7] M. Alhussein, K. Aurangzeb, and S. I. Haider, "Hybrid cnn-lstm model for short-term individual household load forecasting," *Ieee Access*, vol. 8, pp. 180 544–180 557, 2020.

[8] G. Dudek, P. Pełka, and S. Smyl, "A hybrid residual dilated lstm and exponential smoothing model for midterm electric load forecasting," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 7, pp. 2879–2891, 2021.

[9] P. Huber, M. Ott, M. Friedli, A. Rumsch, and A. Paice, "Residential power traces for five houses: the ihomelab rapt dataset," *Data*, vol. 5, no. 1, p. 17, 2020.

[10] J. C. López, M. J. Rider, and Q. Wu, "Parsimonious short-term load forecasting for optimal operation planning of electrical distribution systems," *IEEE transactions on power systems*, vol. 34, no. 2, pp. 1427–1437, 2018.

[11] B. Nepal, M. Yamaha, A. Yokoe, and T. Yamaji, "Electricity load forecasting using clustering and arima model for energy management in buildings," *Japan Architectural Review*, vol. 3, no. 1, pp. 62–76, 2020.

[12] J. C. B. Gamboa, "Deep learning for time-series analysis," *arXiv preprint arXiv:1701.01887*, 2017.

[13] M. Elsaraiti and A. Merabet, "Solar power forecasting using deep learning techniques," *IEEE Access*, vol. 10, pp. 31 692–31 698, 2022.

[14] M. Massaoudi, S. S. Refaat, I. Chihi, M. Trabelsi, F. S. Oueslati, and H. Abu-Rub, "A novel stacked generalization ensemble-based hybrid lgbm-xgb-mlp model for short-term load forecasting," *Energy*, vol. 214, p. 118874, 2021.

[15] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 international conference on engineering and technology (ICET)*. Ieee, 2017, pp. 1–6.

[16] Z. Hu, J. Zhang, and Y. Ge, "Handling vanishing gradient problem using artificial derivative," *IEEE Access*, vol. 9, pp. 22 371–22 377, 2021.

[17] H. Jahangir, H. Tayarani, S. S. Gougheri, M. A. Golkar, A. Ahmadian, and A. Elkamel, "Deep learning-based forecasting approach in smart grids with microclustering and bidirectional lstm network," *IEEE Transactions on Industrial Electronics*, vol. 68, no. 9, pp. 8298–8309, 2020.

[18] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012.

[19] C. Bergmeir and J. M. Benítez, "On the use of cross-validation for time series predictor evaluation," *Information Sciences*, vol. 191, pp. 192–213, 2012.

B

Battery Data sheet

RESU

LG Chem

CHANGE YOUR ENERGY, CHARGE YOUR LIFE

48V

Models		RESU3.3	RESU6.5	RESU10	RESU13
Total Energy [kWh] 1)		3.3	6.5	9.8	13.1
Usable Energy [kWh] 2)		2.9	5.9	8.8	12.4
Capacity [Ah]		63	126	189	252
Nominal Voltage [V]		51.8			
Voltage Range [V]		42.0~58.8			
Max Power [kW]		3.0	4.2	5.0	5.0
Peak Power [kW] (for 3 sec.)		3.3	4.6	7.0	7.0 11.0 (Backup Mode)
Dimension [W x H x D, mm]		452 x 403 x 120	452 x 656 x 120	452 x 484 x 227	452 x 626 x 227
Weight [kg]		31	52	75	99
Enclosure Protection Rating		IP55			
Communication		CAN2.0B			
Certificates	Cell	UL1642			
	Product	UL1973 / TUV (IEC 62619) / CE / FCC / RCM			TUV(IEC 62619)/CE/FCC/RCM

Compatible Inverter Brands : SMA, SolaX, Ingeteam, GoodWe, Sungrow, Victron Energy, Selectronic - More brands to be added
1) Total Energy is measured at the initial stage of battery life under the condition as follows : Temperature 25℃
2) Usable Energy is based on battery cell only

RESU PLUS

RESU Plus is an expansion kit specially designed for 48V models of the RESU series. With RESU Plus, all 48V models can be cross-connected with each other.

- Dimension : 216 x 156 x 121 (W x H x D, mm)
- Number of Expandable Battery Units : Up to 2EA
- IP55

400V

Models		RESU7H		RESU10H	
		Type-R	Type-C	Type-R	Type-C
Total Energy [kWh] 1)		7.0		9.8	
Usable Energy [kWh] 2)		6.6		9.3	
Capacity [Ah]		63		63	
Voltage Range [V]		350~450	430~550	350~450	430~550
Max Power [kW]		3.5		5.0	
Peak Power [kW]		5.0 (for 5 sec.)	5.0 (for 10 sec.)	7.0 (for 10 sec.)	
Dimension [W x H x D , mm]		744 x 692 x 206	744 x 907 x 206	744 x 907 x 206	744 x 907 x 206
Weight [kg]		75	87	97	99.8
Enclosure Protection Rating		IP55			
Communication		RS485	CAN2.0B	RS485	CAN2.0B
Certificates	Cell	UL1642			
	Product	TUV (IEC 62619) / CE / FCC / RCM	TUV (IEC 62619) / CE / RCM	UL1973 / TUV (IEC 62619) / CE / FCC / RCM	

Compatible Inverter Brands : SMA, SolarEdge, Fronius, Huawei - More brands to be added
1) Total Energy is measured at the initial stage of battery life under the condition as follows : Temperature 25℃
2) Usable Energy is based on battery cell only