Neural network learns from mock-up operation experience: implementing on a solar energy community distribution system with heat storage

Chih-Hsiang Lee

This dissertation is submitted for the degree of MSc by Research of Engineering

March 2019

Department of Engineering
Declaration

This thesis has not been submitted in support of an application for another degree at this or any other university. It is the result of my own work and includes nothing that is the outcome of work done in collaboration except where specifically indicated. Many of the ideas in this thesis were the product of discussion with my supervisor Dr Dénes Csala.

Chih-Hsiang Lee

Lancaster University, UK
Abstract
Inspired by Imitation Learning, this paper trained a LSTM network by a mock-up operation experience of a solar energy community distribution system. Unlike the conventional method that implements LSTM only to predict features for the control programme to calculate an operation action according to a strategy, the LSTM of the proposed model integrates the strategy into its structure and thus can outputs actions directly. To examine whether the proposed model outperforms the conventional model, this paper first describes an operation strategy, adopted by both models, that aims to decrease total operation cost. Since the strategy needs accurate predictions to work effectively, an expert who can perfectly predict the future is created by historical data. The behaviours of the expert that follows the strategy are used as the training data of the LSTM in the proposed model. During simulation, the proposed model has better performance and computation efficiency than the conventional LSTM model by 25% higher and 75 times faster. Many researches have proposed control models for different systems and implemented LSTM only to predict key uncertainty in those models. To these researches, this paper demonstrates a promising result that the performance of a control model can be improved by integrating the strategy of that model into a neural network with mock-up operation experience.
Acknowledgements

Thank you to Dr Dénes Csala of Lancaster University, UK, for advising and helpful comments on this work. Without your guidance and dedicated involvement, this paper would have never been accomplished.

I would like to express my gratitude to PVOutput online service and Energy Lancaster for providing the historical data for this research, and Gill Fenna of Quantum Strategy and Technology, UK, for helping in data collection and internship opportunity.

I also greatly appreciate the effort devoted by the developers of Keras, Tensorflow, Jupyter Notebook and python libraries.
# Contents

1 INTRODUCTION 1

1.1 Standard Model 2
1.2 Proposed Model 4
1.3 Vanilla Model 6

2 SOLAR ENERGY COMMUNITY DISTRIBUTION SYSTEM 8

2.1 Details of the System 8
2.2 Objective of Operation 10

3 OPERATION STRATEGY FOR THE COMMUNITY SYSTEM 14

4 PYTHON IMPLEMENTATION 20

4.1 Simulation Environment 20
4.2 Standard Model 21
4.3 Proposed Model 21

5 RESULTS AND DISCUSSION 26

5.1 Training result of the networks in Standard Model 26
5.2 Operation Performance 29
5.3 Annual Cost 33
5.4 Training and Computation Efficiency 40

6 CONCLUSION 42

7 REFERENCES 45

8 APPENDICES 47
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Pseudo Code: Operation strategy for optimal operation curve</td>
<td>19</td>
</tr>
<tr>
<td>4.1</td>
<td>Pseudo Code: Simulation Environment</td>
<td>23</td>
</tr>
<tr>
<td>4.2</td>
<td>Pseudo Code: Standard Model</td>
<td>25</td>
</tr>
<tr>
<td>4.3</td>
<td>Pseudo Code: Proposed Model</td>
<td>25</td>
</tr>
<tr>
<td>5.1</td>
<td>Yearly Operation Cost</td>
<td>36</td>
</tr>
<tr>
<td>5.2</td>
<td>Operation effectiveness, $e_{op}$</td>
<td>38</td>
</tr>
<tr>
<td>5.3</td>
<td>Operation effectiveness, $e_{op}$, of each week in Result 1</td>
<td>38</td>
</tr>
<tr>
<td>5.4</td>
<td>$e_{op}$ and $R$ of operation cost during cold weeks in Result 1</td>
<td>39</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1.1 Concept of Standard Model 3

Figure 1.2 Training of Standard Model 4

Figure 1.3 Concept of Proposed Model 6

Figure 1.4 Training of Proposed Model 6

Figure 2.1 Solar Energy Community Distribution System 9

Figure 3.1 Expert’s Operation Curve on a cold day 18

Figure 3.2 Expert’s Operation Curve on a warm day 18

Figure 5.1 Comparisons of predictive and true values in the Standard Model 27

Figure 5.2 One-day simulation (Result 1) 31

Figure 5.3 One-day simulation (Result 2) 32

Figure 5.4 One-day simulation (Result 3) 32

Figure 5.5 One-day simulation (Result 4) 33
List of Appendices

Appendix 1 Comparisons of predictive and true values in the Standard Model 48

Appendix 2 Python Code 58
1 Introduction

This paper presents a practical application of Long Short-Term Memory neural network (LSTM) [1] on a solar energy community distribution system. Unlike other models that predict features individually for supporting operators or control programmes to decide on operation actions, the proposed model in this paper was trained for directly determining the next operation action based on input features.

LSTM is capable of predicting time sequence by learning long-term dependencies in a dataset. It has the power of extracting non-linear relationship between input and output, and the capability of identifying patterns in time sequence. Thus, it has been widely used in electricity systems because key uncertainties, such PV generation, wind speed, demands and electricity price, have a temporal dependency between each time step. Many researches [2, 3, 4, 5, 6, 7] applied LSTM purely to predict key features related to electricity industry, such as weather condition, electricity prices, and energy demands. These predictions can be used to support operator’s decision making, but not directly provide operation actions on the electricity equipment or systems.

In the field of sewer system operation, Zhang (2017, 2018) [8, 9] proposed operation strategies for water managements, and then pointed out key uncertainty in these strategies.
LSTM was implemented only to predict the uncertainty, such as future inflow of each wastewater treatment plant or sewer. Similarly, LSTM predictions made in [8, 9] have no connection to their proposed strategies, but only provide better information to operators who use those strategies.

In this paper, we built three models to compare the performance of conventional and our proposed method. Standard Model adopted the idea discussed above that forecast only serves as a reference in operating the system. Operators or control programmes accept the forecast and run the operation strategy to determine the current action. On the contrary, our Proposed Model integrates the operation strategy into its training set, enabling the model to directly control the system. Simulative results show that the Proposed Model outperforms the Standard Model. Moreover, even though the Proposed Model takes more resources to prepare the training set, no calculation need to be done when processing online. In the long run, the Proposed Model consumes less processing time than the Standard Model. Last, for comparison, the Vanilla Model follows a common strategy that the storage always starts at fixed times to be charged or to be discharged.

Note that when we use the word, ‘operator,’ in this paper, it usually means the same as ‘control programme’ since the three Models are controlled by computer programmes.

### 1.1 Standard Model

Applying the concept mentioned above, we build a Standard Model to provide a basis for comparison to the Proposed Model. This concept is a straightforward implementation of LSTM networks on operation of systems with uncertainty and has been adopted by many researches.

Figure 1.1 depicts the Standard Model of using LSTM predictions to aid operators in operation of a solar energy distribution system. The energy distribution system is showed
in Figure 2.1 and detailed in Chapter 2. Each sub-model (green square) in Figure 1.1 has one LSTM network. In the beginning of every half hour, each sub-model accepts input from historical data to make prediction of five key features relevant to operation decision: PV generation, electricity demand, heat demand, importing price of the grid and exporting price of the grid. A computer programme that follows operating strategy (blue square) then accepts those predictions as input for calculating the operation action in current half hour.

**Figure 1.1 Concept of Standard Model**

![Figure 1.1 Concept of Standard Model](image)

Note that sub-models of the Standard Model can be more complicated, taking more features as input to increase its accuracy. However, since the Proposed Model in this paper only use the five input features, we set sub-model of the Standard Model only take its own feature as input for a fair comparison of the two models. Figure 1.2 shows the training method of sub-models. The LSTM sub-models approximate the relationship between two sets of time sequence. Operators or control programmes accept the output sequence as a guideline to decide their operation action.
1.2 Proposed Model

To design our Proposed Model, we first formulated an operating strategy that determines the target level of heat storage every half hour, based on the five input features. Following our proposed strategy, the heat storage will be charged if its current energy level is less than the target level. Charging can be done not only by PV generation but also electricity imported from the grid if future importing price is expected to become higher. If the current energy level is more than the target level, the heat storage discharges. This proposed strategy is designed to decrease total operation cost, detailed in Section 2.2. To make the comparison meaningful, ‘Operation Strategy (blue square)’ in Figure 1.1 is the same as our proposed operating strategy of heat storage in Figure 1.3.

To utilise the proposed strategy, uncertainties of the five input features must be eliminated. Instead of training five LSTM sub-models that predict these features, we trained only one LSTM model that takes these five features as inputs to directly output a target level for every half hour.

We applied the principle behind Imitation Learning, of which a model learns from expert’s behaviours. For example, when learning self-driving cars, a model is showed with pairs of state and action for it to interpret the policy behind the decision of actions.
Those demonstrated actions are recorded from an expert, such as a human driver. Imitation Learning is usually implemented when calculation of an action is impossible or too expensive, but the task is easy for a human to perform. In our case, although no person can perfectly predict future when operating heat storage, we can create a mock-up expert from historical data. This expert follows the proposed strategy in simulations of operating a system. The expert’s behaviours are then used as the training set for the Proposed Model.

The training method of the Proposed Model is showed in Figure 1.4. In contrast to the training pairs for the Standard Model in Figure 1.2, the Proposed Model learns to interpret the relationship between a time sequence and one output value. The training pairs in Figure 1.4 are ‘handcrafted’ by following the proposed strategy. Consequently, the Proposed Model is connected to the proposed strategy itself, and its output is directly determining the next operation action.

The working process of the Proposed Model is depicted in Figure 1.3. Compared with the Standard Model, the Proposed Model runs the strategy only once during preparation of training pairs, while the computer programme in the Standard Model must repeat calculation of the strategy each half hour every time when it receives new forecasts of its five predictive features.
1.3 Vanilla Model

A common strategy is to charge and discharge heat storage at fixed times. Examining the actions performed by the expert discussed in the Proposed Model, we found that most of the time the expert charges the storage at 13:30 and discharges the storage at 17:00.
Chapter 1 Introduction

Therefore, we set the Vanilla Model always charge and discharge at these two times every day. The storage is charged and discharged with a fix rate.

Another key difference between the Vanilla Model and other two models is that during charging the Vanilla Model never imports electricity from the grid if PV-generation is not enough because the Vanilla Model have no ability to forecast electricity price. It would end up in excessive expenditure if allowing the Vanilla Model to import electricity. When there’s no PV-generation during charging, the Vanilla Model would stop charging the storage until PV-generation resumes.

The remainder of this paper is organized in the following way: Chapter 2 details the components of the solar energy community distribution system and the objective of its operation. Chapter 3 introduces the proposed strategy for operation of the community system and describes how an expert is created and the operation behaviour of this expert. Chapter 4 explains the implementation of the three Models and the simulation process in python environment. Chapter 5 discusses and analyses the outcome of simulation. Chapter 6 summarises the results and provides a suggestion of future researches.
2 Solar Energy Community Distribution System

2.1 Details of the System

Showed in Figure 2.1, the design of this system is based on a real proposed project for a community located in North West of England. PV generation is the only domestic supply in the system. During each time interval, such as a half hour, PV generation is used to meets electricity demand first, and any insufficiency is addressed by importing electricity from the grid. After that, surplus of PV generation, if any, is used to run heat pumps for meeting heat demand. Electricity demand takes priority over heat demand because PV generation would suffer loss due to energy conversion in heat pumps. In Figure 2.1, COP stands for Coefficient of Performance, which defines the conversion factor between electricity energy and heat energy.
If PV generation is insufficient to cover heat demand, the short of heat supply is compensated by importing electricity from the grid to run heat pumps or by discharging heat from the storage. When heat pumps run out of capacity, the only way to provide heat is discharging the storage. In this paper, we assume that heat pump capacity is always sufficient to cover demand peak. The heat pump capacity is set to be a little high than the maximum heat demand in our simulative environment, but not infinite.

Finally, excess PV generation can be sold to the grid, or be used to charge heat storage if heat pumps still has capacity. In this study, we assume that domestic use of PV generation is always more economical than selling to the grid.

Heat storage can be charged by heat pumps that consume PV generation, imported electricity or both. Due to the capacity of heat pumps, charging storage may be limited sometimes.
In each half hour there are two prices: System Sell Price (SSP) and System Buy Price (SBP). When the operator imports electricity from the grid, the operator needs to pay the SBP. Likewise, the grid pays the SSP to the operators who export electricity to the grid. These two prices are called ‘imbalance prices’ and originally designed to tackle the deficit of imbalance energy. In our study, we use a historical data of SSP and SBP around Lancaster area to simulate the price change faced by operators.

2.2 Objective of Operation

In our system, PV generation is always used to meet electricity demand first and then heat demand. After that if any PV generation remains, it can be used to charge the storage or be sold to the grid. Thus, we defined ‘PV Surplus’ as the amount of remaining PV generation we can manipulate:

\[
PV \text{ Surplus} = PV \text{ generation} - electricity \text{ demand} - (heat \text{ demand} ÷ COP) \tag{1}
\]

\[
\text{if } PV \text{ Surplus} < 0, \ PV \text{ Surplus} = 0
\]

When PV generation is unable to cover all heat demand, we defined a term ‘Shortage’ as the amount of remaining heat demand that we need to cope with by heat storage:

\[
Shortage = heat \text{ demand} - [(PV \text{ generation} - electricity \text{ demand}) \times COP] \tag{2}
\]

\[
\text{if } (PV \text{ generation} - electricity \text{ demand}) < 0, Shortage = heat \text{ demand}
\]

\[
\text{if } Shortage < 0, Shortage = 0
\]

Every half hour the operator determines a target level for the heat storage. If current level is high than the target level, the heat storage is discharged until current level drops to the target level. If current level is lower than the target level, the heat storage is charged by PV surplus first. It can also be charged by imported electricity only if importing electricity with current SBP is beneficial, compared to importing electricity with future SBP when the demand actually occurs in the future. In other words, the operator must have the
capability to forecast future electricity prices to know when the best time to buy electricity is. Furthermore, the operator must be able to forecast future PV generation and demands to determine what is the actual amount of heat needed to be prepared in advanced. For example, if a sunny day is expected, the operator has no need to import electricity to charge the storage even though current SBP is low.

The goal of the operator is to reduce operation cost of the system. Operation cost is equal to the expenditure of importing electricity from the grid subtracted by the income of selling PV generation to the grid. In terms of cost, income is negative:

\[
\text{Operation cost} = \text{expenditure of importing} + (-\text{income of exporting})
\] (3)

With heat storage and a good predictor of future PV generation, demands and system prices, the operator can accomplish several tasks to decrease operation cost:

A. If the operator has PV surplus in the current moment and expects a Shortage in a future moment and importing electricity with future SBP is expensive than not selling PV surplus with current SSP, the operator should charge the storage with current PV surplus:

(i) \textit{Selling PV surplus right now, and importing electricity in the future}:

\[
\begin{align*}
\text{income} &= -\text{PV surplus} \times \text{current SSP} \\
\text{expenditure} &= \left[\text{Shortage} \div \text{COP}\right] \times \text{future SBP}
\end{align*}
\]

(ii) \textit{Saving PV surplus right now for the future}:

\[
\begin{align*}
\text{income} &= 0 \\
\text{expenditure} &= 0
\end{align*}
\]

(Assuming: \(\text{PV surplus} \times \text{COP} \times \text{loss}^{(\text{future} - \text{current})} = \text{Shortage}\))
if \( \text{Operation cost}(ii) - \text{Operation cost}(i) < 0: \)

\[
\begin{align*}
&\rightarrow 0 - (-PV \text{ surplus} \times \text{current SSP} + [\text{Shortage} \div \text{COP}] \times \text{future SBP}) < 0 \\
&\rightarrow PV \text{ surplus} \times \text{current SSP} < [\text{Shortage} \div \text{COP}] \times \text{future SBP} \\
&\rightarrow PV \text{ surplus} \times \text{current SSP} < PV \text{ surplus} \times \text{loss}^{(t_{\text{future}} - t_{\text{current}})} \times \text{future SBP} \\
&\rightarrow \text{current SSP} \div \text{loss}^{(t_{\text{future}} - t_{\text{current}})} < \text{future SBP} \\
&\quad \text{(4)}
\end{align*}
\]

, where \( t_{\text{future}} - t_{\text{current}} \) is the difference between current and future time. And \( \text{loss} \) is the heat loss in storage per unit time. In our study, the unit time is equal to a half hour, and transition loss is ignored for simplification.

B. If the operator has no PV surplus in the current moment and expects a Shortage in a future moment and importing electricity with future SBP is expensive than importing electricity with current SBP, the operator should import electricity with current SBP to charge the storage.

(i) Do nothing right now, and importing electricity in the future:

\[
\begin{align*}
\text{expenditure}_{\text{current}} &= 0 \\
\text{expenditure}_{\text{future}} &= [\text{Shortage} \div \text{COP}] \times \text{future SBP}
\end{align*}
\]

(ii) Importing electricity right now for the future:

\[
\begin{align*}
\text{expenditure}_{\text{current}} &= \text{Importing electricity} \times \text{current SBP} \\
\text{expenditure}_{\text{future}} &= 0
\end{align*}
\]

(\text{Assuming: Importing electricity} \times \text{COP} \times \text{loss}^{(t_{\text{future}} - t_{\text{current}})} = \text{Shortage})

if \( \text{Operation cost}(ii) - \text{Operation cost}(i) < 0: \)

\[
\begin{align*}
&\rightarrow \text{Importing electricity} \times \text{current SBP} - [\text{Shortage} \div \text{COP}] \times \text{future SBP} < 0 \\
&\rightarrow \text{Importing electricity} \times \text{current SBP} < [\text{Shortage} \div \text{COP}] \times \text{future SBP} \\
&\rightarrow \text{Importing electricity} \times \text{current SBP} \\
&\quad < \text{Importing electricity} \times \text{loss}^{(t_{\text{future}} - t_{\text{current}})} \times \text{future SBP} \\
&\rightarrow \text{current SBP} \div \text{loss}^{(t_{\text{future}} - t_{\text{current}})} < \text{future SBP} \\
&\quad \text{(5)}
\end{align*}
\]
C. If the operator expects several available electricity sources at $t_1, t_2, t_3, t_4$, and a Shortage at $t_5$, the operator must compare the prices, which are modified by loss and different time spans. The modified prices could be:

\[
\begin{align*}
&\text{current SSP} + \text{loss}^{(t_{\text{future}}-t_{\text{current}})}, \quad \text{if the source is PV surplus} \\
&\text{current SBP} + \text{loss}^{(t_{\text{future}}-t_{\text{current}})}, \quad \text{if the source is importing electricity}
\end{align*}
\]

After comparison, the operator exploits the sources in order of profitability. Consequently, depending on the amount of heat required by Shortage at $t_5$, some of the sources may be exhausted, some never used, and some used only part of their available supply. It is important for the operator not to consume an electricity source more than the requirement; otherwise operation cost would increase. For example, if the operator takes the exact amount of electricity, remaining PV generation can be sold to the grid instead of suffering unnecessary loss in the heat storage and being used in somewhere not actually profitable. Similarly, if the operator imports the exact amount of electricity from the grid, no extra expenditure would be incurred.
Chapter 3 Operation Strategy for the Community System

3 Operation Strategy for the Community System

With historical data, we can assume that there is a perfect predictor, “an expert,” who can forecast all we need in next 24 hours, which is divided equally into $t_0$ to $t_{47}$. Our operation strategy is to analyse the relationship of PV surplus, Shortage, SSP and SBP at $t_0$ to $t_{47}$, to determine the profitability of each available electricity source and to distribute all available electricity sources to all Shortage at $t_0$ to $t_{47}$ accordingly.

Available electricity sources include PV Surplus and importing electricity from the grid.

At the start of $t_0$, the expert holds the values of PV surplus, Shortage, SSP and SBP at $t_0$ to $t_{47}$. First, it creates a profit table, in which each entry is called a ‘profit number’:

\[
\text{profit number} = \begin{cases} 
\frac{\text{SSP}_p + \text{loss}(t_n-t_p)}{\text{SBP}_n}, & \text{if using PV Surplus at } t_p \text{ to charge heat storage} \\
\frac{\text{SBP}_p + \text{loss}(t_n-t_p)}{\text{SBP}_n}, & \text{if importing electricity at } t_p \text{ to charge heat storage} 
\end{cases}
\]

, where $t_n > t_p$ and $t_n, t_p \in t_0$ to $t_{47}$. SSP$_p$ is the SSP at $t_p$, SBP$_p$ is the SBP at $t_p$ and SBP$_n$ is the SBP at $t_n$. We only consider $t_n$ when there is a Shortage at $t_n$.

We set $pf_{P,n}^{PV}$ be the profit number when using PV Surplus at $t_p$ to charge heat storage for future Shortage at $t_n$. Similarly, $pf_{P,n}^{Gd}$ is the profit number when importing electricity from the grid at $t_p$ to charge heat storage for future Shortage at $t_n$. Refer to Equation (4) and (5), it is obvious that if $pf_{P,n} < 1$, it’s profitable to use electricity source at $t_p$. On the other hand, if $pf_{P,n} \geq 1$, it has no need to use electricity source at $t_p$ and this $pf_{P,n}$ would be excluded from the profit table.
Next, the expert distributes all available electricity source to all Shortage, starting from the smallest $p_{f,p,n}$. The expert calculates the exact amount of electricity needed at $t_p$ for the Shortage at $t_n$:

$$\textit{electricity requirement at } t_p = (\textit{Shortage at } t_n ÷ \textit{loss}(t_n - t_p)) \times \textit{COP}$$  \hspace{1cm} (7)$$

The expert then adjusts the electricity requirement at $t_p$ according to heat pump capacity at $t_p$ and heat storage capacity at $t_p, t_{p+1}, t_{p+2}, \ldots$, and $t_n$ because heat pump capacity limits the amount of heat that can be charged, and heat storage capacity limits the amount of heat that can be stored in the heat storage.

Finally, the expert decreases the electricity source at $t_p$ as much as possible according to the modified electricity requirement at $t_p$. If the electricity source is PV Surplus, the expert records how much amount of PV Surplus remains. If the electricity source is from the grid, the expert can import as much as it need, because we assume that the connection to the grid is always available. The amount of electricity consumed at $t_p$ turns into heat, which reduces heat pump capacity at $t_p$. The expert also records the decrease of Shortage at $t_n$ and the decreases of heat storage capacity at $t_p, t_{p+1}, t_{p+2}, \ldots$, and $t_n$.

To increase the efficiency of the algorithm, when a heat pump capacity at $t_p$ is exhausted, all $p_{f,p,n}$ with $t_p$ will be deleted from the profit table. Similarly, when a heat storage capacity at $t_x$ is used up, all $p_{f,p,n}$ with $t_p \leq t_x \leq t_n$ will be deleted. In addition, after a Shortage at $t_n$ is fully fulfilled, all $p_{f,p,n}$ with $t_n$ will be deleted.

After the expert goes through all entries of the profit table, all Shortages that are not fully fulfilled will be coped with importing electricity at their current time. We obtain an optimal operation curve, such as showed in Figure 3.1 and Figure 3.2. A pseudo code is showed in Table 3.1.
In Figure 3.1 and Figure 3.2, the heat level of heat storage (purple dot) of \( t_n \) is the heat level at the start of \( t_n \), and the bars (orange and indigo) show how much amount of heat is charged into the storage at the end of \( t_n \). For example, at the start of \( t_0 \) and \( t_1 \) in Figure 3.1 there is no heat in the storage, and the operator charges the storage by 243.18 kWh during \( t_1 \). Thus, at the start of \( t_2 \) the heat level is equal to 243.18 kWh as showed in the figure.

Note that PV generation in Figure 3.1 and Figure 3.2 has been subtracted by electricity demand first and then converted to heat energy for clearly demonstrating how PV generation is used to charge the storage.

The operation curves in Figure 3.1 and Figure 3.2 demonstrate several behaviours that our Proposed Model must learn:

A. Avoid storing excessive heat:

Comparing the sum of heat demand from \( t_{12} \) and \( t_{17} \) (approx. 771.36 kWh) and the total heat released from the heat storage from \( t_{12} \) and \( t_{17} \) (approx. 762.98 kWh) in Figure 3.1, it can be seen that heat prepared in the storage is slightly less than the heat demand because it can be covered by the PV generation at \( t_{17} \) (approx. 8.38 kWh). After that, heat demand from \( t_{18} \) and \( t_{29} \) is fully covered by PV generation. This behaviour demonstrates that our expert knows the optimal amount of heat that needs to be prepared before a certain time, depending on when PV generation begins and what amount of PV generation occurs in the future.

Likewise, expecting a low demand during the evening in Figure 3.2, the expert fills the storage to a sufficient amount of heat (approx. 794.52 kWh), but not to its full capacity (1500 kWh). This shows the expert’s capability of operating the storage optimally by knowing PV generation and heat demand in advanced.
B. Charge the storage economically:

Knowing how much amount of heat needs to be prepared is not enough. The expert must figure out how to charge the storage in a cost-effective way. In Figure 3.1, the expert imports electricity at $t_1, t_2, t_5$ and $t_{11}$ to meet the target level at $t_{12}$ because SBPs at $t_1, t_2, t_5$ and $t_{11}$ are lower than other SBPs between $t_1$ to $t_{11}$. Note that even though SBP at $t_1$ (0.03232 £/kWh) is lower than SBP at $t_{11}$ (0.03472 £/kWh), the expert still chooses to import electricity at $t_{11}$ due to the modification of SBP made by heat loss, as discussed in Equation (5). Similarly, in Figure 3.2, the expert consumes PV generation at $t_{23}, t_{24}, t_{25}, t_{28}$ and $t_{32}$ because of low modified prices. From $t_{19}$ to $t_{32}$ in Figure 3.1, the expert has several different electricity sources from PV generation or from the grid for meeting the target level at $t_{33}$. The expert exploits PV generation as much as possible from $t_{29}$ to $t_{24}$ and stop using PV generation at $t_{23}$ because the modified SSP starts to be higher than modified SBP at $t_{29}$ to $t_{32}$. Note that PV generation between $t_{27}$ and $t_{29}$ is not fully used by the heat pump because PV generation need to meet heat demand first. On the other hand, PV generation between $t_{24}$ and $t_{26}$ is not fully used due to the maximum capacity of heat pump.
Figure 3.1 Expert’s Operation Curve on a cold day

![Expert's Operation Curve (cold day)](image)

Figure 3.2 Expert’s Operation Curve on a warm day

![Expert's Operation Curve (warm day)](image)
Table 3.1 Pseudo Code: Operation strategy for optimal operation curve

```
Algorithm 1 operation strategy for optimal operation curve
(Note that COP must be considered in actual codes)

1. Input PV generation, Electricity Demand, Heat Demand, SSP and SBP of \( t_0 \) to \( t_{47} \)
2. \( PV\ Surplus = \max(0, PV\ generation - Electricity\ Demand) \)
3. Create Cost Table:
   \[
   \text{Table} = []
   \]
   for \( n \in t_n \) when there is a Shortage do
   for \( p = n - 1, n - 2, n - 3, \ldots \) do
     calculating \( p_{f_{p,n}} \)
     if \( p_{f_{p,n}} < 1 \):
       Table.append(\( p_{f_{p,n}} \))
     else:
       break
   Table.sort(ascending)
4. Distribute energy:
   for \( p_{f_{p,n}} \) in Table do
     supply = Shortage at \( t_n \)
     A. check remaining heat pump capacity at \( t_p \),
        reduce supply if need
     B. check remaining heat storage capacity at \( t_p, t_{p+1}, t_{p+2}, \ldots \), and \( t_n \) (heat loss considered)
        reduce supply if need
     C. According to supply:
        update PV Surplus or Importing electricity at \( t_p \)
        if remaining PV Surplus at \( t_p = 0 \):
          delete all \( p_{f_{p,n}}^{PV} \) at \( t_p \) in Table
        update remaining heat pump capacity at \( t_p \)
        if remaining heat pump capacity at \( t_p = 0 \):
          delete all \( p_{f_{p,n}} \) at \( t_p \) in Table
        update remaining heat storage capacity at \( t_p, t_{p+1}, t_{p+2}, \ldots \), and \( t_n \) (heat loss considered)
        if any remaining heat storage capacity at \( t_p, t_{p+1}, t_{p+2}, \ldots \), or \( t_n = 0 \):
          delete all \( p_{f_{p,n}} \) that use heat storage at \( t_p, t_{p+1}, t_{p+2}, \ldots \), or \( t_n \) in Table
        update Shortage at \( t_n \)
        if Shortage at \( t_n = 0 \):
          delete all \( p_{f_{p,n}} \) at \( t_n \) in Table
```
Chapter 4 Python Implementation

4 Python Implementation

We use Python and Jupyter Notebook to create the Models and to conduct simulations. The implement of LSTM networks is constructed by Keras, a neural networks API of Python [10].

4.1 Simulation Environment

The pseudo code of simulation environment is showed in Table 4.1.

We first set up a four-year database of the five features (PV generation, electricity and heat demand, SSP and SBP):

A. PV generation is based on a four-year real data.

B. We assumed that electricity demand per dwelling per year is set to be 3000 kWh and there are 180 houses in the community. Electricity demand curve is based on a one-year real data.

C. Heat demand per dwelling is set to be 4500 kWh. Heat demand curve is based on a one-year estimated data.

D. SBP and SSP are based on a one-year real data. The average of SBP is 0.04756 £/kW, and of SSP is 0.0366 £/kWh. SBP is always greater than or equal to SSP.

After picking a day, the simulation environment loads the five features at $t_{-48}$, $t_{-47}$, $t_0$, $t_1$, ..., and $t_{47}$, of which $t_0$ is 12:00 AM of that day. The output of simulation is an operation curve of each Model between $t_0$ and $t_{47}$, and the total operation cost of each Model during $t_0$ to $t_{47}$. 


4.2 Standard Model

In Standard Model, we trained five networks to predict each feature (PV generation, heat demand, electricity demand, SSP and SBP). Each network receives a value sequence of \( t_{n-48} \) to \( t_{n-1} \) to forecast the sequence of \( t_n \) to \( t_{n+47} \), as showed in Figure 1.2, in which \( p = 48 \) and \( m = 47 \). The operator then put these predicted sequences of \( t_n \) to \( t_{n+47} \) into Algorithm 1 (Table 3.1) to determine the target level of heat storage at \( t_n \). The pseudo code of Standard Model is showed in Table 4.2.

The training sets of Standard Model are prepared by pairing the sequences of \( t_{n-48} \) to \( t_{n-1} \) with the sequences of \( t_n \) to \( t_{n+47} \) for each feature in the four-year database.

These five networks have the same figuration that the first layer is a LSTM layer with a hard-sigmoid function as its activation function. The second layer is a dropout layer with a dropout rate equal to 0.5, connected to the last layer which is a simple Dense layer with hard-sigmoid function. The cost function is MSE. Input of the first layer is scaled to a range of 0 to 1, and the output of the Dense layer is also between 0 to 1, which will be transformed back to the original range based on the training set. This is because normalization can make learning process faster.

4.3 Proposed Model

In Proposed Model, we trained only one network. The network receives five sequences of \( t_{n-48} \) to \( t_{n-1} \) to forecast one value: the target level for \( t_n \), as showed in Figure 1.4, in which \( p = 48 \). The operator has no need to run Algorithm 1 (Table 3.1) repeatedly. The pseudo code of Proposed Model is showed in Table 4.3.

The training set of Proposed Model is prepared by putting five sequences of \( t_{n-48} \) to \( t_{n-1} \) of the four-year database into Algorithm 1 (Table 3.1) to obtain the target level for \( t_n \).
The figuration of the network in Proposed Model has similar structure of which the first layer is a LSTM layer with a hard-sigmoid function as its activation function. The second layer is a dropout layer with a dropout rate equal to 0.3, connected to the last layer which is a simple Dense layer with hard-sigmoid function. Similarly, the cost function is MSE, input of the first layer is scaled to a range of 0 to 1, and the output of the Dense layer is also between 0 to 1, which will be transformed back to the original range based on the training set. This is because normalization can make learning process faster.
Table 4.1 Pseudo Code: Simulation Environment

**Algorithm 2** Simulation Environment
(Note that COP must be considered in actual codes)

1. **Load** Features (PV generation, Electricity Demand, Heat Demand, SSP and SBP) of \( t_{n-48} \) to \( t_{n-1} \)
2. **Set** \( \text{Cost}_S = 0, \text{Cost}_P = 0, \text{Cost}_V = 0 \)  
   
   \[ \text{## Cost}_x \text{ is the operation cost, } x = S, P \text{ or } V \]
   
   \[ \text{## } S = \text{Standard Model}, P = \text{Proposed Model}, V = \text{Vanilla Model} \]
3. \( \text{heat level}_S = [ ], \text{heat level}_P = [ ], \text{heat level}_V = [ ] \)
4. **For** \( n = 0, 1, 2, \ldots, 46, 47 \) **do**
   
   **(A). Input** Features of \( t_{n-48} \) to \( t_{n-1} \) **into** Standard Model (Algorithm 3)
   
   **Return** \( \text{Target}_S \)  
   
   **## target level predicted by Standard Model**
   
   **Do** step (B), with \( \text{Target} = \text{Target}_S, \text{Cost} = \text{Cost}_S, \text{heat level} = \text{heat level}_S \)

   **(B). Load** \( HP \text{ capacity}_n, Storage \text{ capacity}_n, Shortage_n, PV \text{ Surplus}_n \)
   
   \( HP \text{ capacity} \leftarrow HP \text{ capacity}_n \)
   
   \( Storage \text{ capacity} \leftarrow Storage \text{ capacity}_n \)
   
   \( Shortage \leftarrow Shortage_n \)
   
   \( PV \text{ Surplus} \leftarrow PV \text{ Surplus}_n \)
   
   **Let** \( HP \text{ capacity}, Storage \text{ capacity}, Shortage, PV \text{ Surplus} \) **always \( \geq 0 \)**
   
   \[ \text{## whenever it’s a negative number, set it to be } 0 \]

   if \( \text{Target} > \text{heat level}_{n-1} \):
   
   \[ \text{charge}_n \leftarrow \min( \text{Target} - \text{heat level}_{n-1}, HP \text{ capacity}, Storage \text{ capacity}) \]
   
   \( \text{export}_n \leftarrow 0 \)
   
   **Let** \( \text{charge}_n, \text{export}_n \) **always \( \geq 0 \)**
   
   \( \text{heat level}_n \leftarrow \text{heat level}_{n-1} + \text{charge}_n \)
   
   if \( PV \text{ Surplus} > 0 \):
   
   while \( \text{charge}_n > 0 \):
   
   \( \text{export}_n \leftarrow PV \text{ Surplus} - \text{charge}_n \)
   
   \( \text{charge}_n \leftarrow \text{charge}_n - PV \text{ Surplus} \)
   
   \( \text{import}_n \leftarrow \text{charge}_n \)
   
   \( \text{charge}_n \leftarrow 0 \)
   
   \( \text{Cost} \leftarrow \text{Cost} + \text{import}_n \times SSP_n \)
   
   \( \text{Cost} \leftarrow \text{Cost} - \text{export}_n \times SSP_n \)
   
   else if \( Shortage > 0 \):
   
   \( \text{Cost} \leftarrow \text{Cost} + Shortage \times SSP_n \)
   
   while \( \text{charge}_n > 0 \):
   
   \( \text{import}_n \leftarrow \text{charge}_n \)
   
   \( \text{charge}_n \leftarrow 0 \)
   
   \( \text{Cost} \leftarrow \text{Cost} + \text{import}_n \times SSP_n \)
else if Target \leq \text{heat level}_{n-1}:
    \text{discharge}_n \leftarrow \min(\text{heat level}_{n-1} - \text{Target}, \text{Discharge Rate})

Let \text{discharge}_n \text{ always} \geq 0
\text{heat level}_n \leftarrow \text{heat level}_{n-1} - \text{discharge}_n
if \text{PV Surplus} > 0:
    \text{Cost} \leftarrow \text{Cost} - \text{PV Surplus} \times \text{SSP}_n
else if \text{Shortage} > 0:
    \text{Shortage} \leftarrow \text{Shortage} - \text{discharge}_n
    \text{Cost} \leftarrow \text{Cost} + \text{Shortage} \times \text{SBP}_n

(C). Input Features of \text{t}_{n-48} to \text{t}_{n-1} into Proposed Model (Algorithm 4)
Return Target_p, \text{## target level predicted by Proposed Model}
Repeat step (B)., 
but with Target = Target_p, \text{Cost} = \text{Cost}_p, \text{heat level} = \text{heat level}_p

(D). Repeat step (B)., 
but with Target_v, \text{Cost}_v, \text{heat level} = \text{heat level}_p, and this condition:
if n<27:
    Target_v = \text{heat level}_{n-1}
else if n<34:
    Target_v = \text{Full Storage Capacity}
    Set import_n always = 0
else:
    Target_v = 0

5. Print \text{Cost}_s, \text{Cost}_p, \text{Cost}_v
Plot \text{heat level}_s, \text{heat level}_p, \text{heat level}_v, and features of \text{t}_0 to \text{t}_{47}
Table 4.2 Pseudo Code: Standard Model

<table>
<thead>
<tr>
<th>Algorithm 3 Standard Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. <strong>Receive</strong> <em>Features</em> of <em>t_{x-48} to t_{x-1}</em> from Algorithm 2</td>
</tr>
<tr>
<td>2. <strong>Load</strong> <em>Predictor_{PV}, Predictor_{ED}, Predictor_{HD}, Predictor_{SSP}, Predictor_{SBP}</em> ## five trained LSTM network</td>
</tr>
<tr>
<td>3. <strong>For</strong> (item, label) in [ (PV generation, PV), (Electricity Demand, ED), (Heat Demand, HD), (SSP, SSP), (SBP, SBP) ] <strong>do</strong></td>
</tr>
<tr>
<td>input Feature(item) of <em>t_{x-48} to t_{x-1}</em> into <em>Predictor_{label}</em></td>
</tr>
<tr>
<td>return Feature(item) of <em>t_{x} to t_{x+47}</em></td>
</tr>
<tr>
<td>4. <strong>input</strong> <em>Features</em> of <em>t_{x} to t_{x+47}</em> into Algorithm 1</td>
</tr>
<tr>
<td>return <em>operation curve</em> of <em>t_{x} to t_{x+47}</em></td>
</tr>
<tr>
<td>5. <em>Target_{S} ← the value of operation curve at t_{x}</em></td>
</tr>
<tr>
<td>6. <strong>return</strong> <em>Target_{S} to Algorithm 2</em></td>
</tr>
</tbody>
</table>

Table 4.3 Pseudo Code: Proposed Model

<table>
<thead>
<tr>
<th>Algorithm 4 Proposed Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. <strong>Receive</strong> <em>Features</em> of <em>t_{x-48} to t_{x-1}</em> from Algorithm 2</td>
</tr>
<tr>
<td>2. <strong>Load</strong> <em>Predictor_{P}</em> ## trained LSTM network of Proposed Model</td>
</tr>
<tr>
<td>3. <strong>input</strong> <em>Features</em> of <em>t_{x-48} to t_{x-1}</em> into <em>Predictor_{P}</em></td>
</tr>
<tr>
<td>return <em>Target_{P} of t_{x}</em></td>
</tr>
<tr>
<td>4. <strong>return</strong> <em>Target_{P} to Algorithm 2</em></td>
</tr>
</tbody>
</table>
5 Results and Discussion

5.1 Training result of the networks in Standard Model

Figure 5.1 demonstrates six comparisons of predictive values and true values. More examples can be found in Appendix A. Blue lines in the figures are the true values of one day and red lines are the values predicted by the five networks in Standard Model. Networks that predict PV generation, electricity and heat demands show the ability to match a rough pattern to the curve of true values. However, the networks are unable to fit those small and rapid changes on the curve delicately.

Predictions made for SBP and SSP are unsatisfying. Predictive values always fluctuate around the average number. This means that the networks are not trained enough, resulting in a bad approximation that sticks around average number to bring a smaller MSE.

One reason could be that the networks need more features to better define an approximation between input and output of the prices. Many factors influence the variations of SBP and SSP, such as real-time changes of generation and consumption, unexpected shutdowns of some units and grid imbalance caused by other occurrence.

In our study, we did not improve the SBP and SSP predictors of the Standard Model because we aim to demonstrate the difference of performance between the Standard Model and the Proposed Model. Therefore, the Standard Model can only receive the same five features as used in the Proposed Model.
Figure 5.1 Comparisons of predictive and true values in the Standard Model

(1) PV Generation

*The x-axis shows feature values (PV generation in this case), which is varied in the range of 0 and 1 since we’ve normalized the data. The y-axis is between 0 and 48, which denotes $t_0$ and $t_{48}$ respectively. However, $t_0$ is not always match 12:00 AM because all input sequences have been randomized.*
(2) Electricity Demand Prediction

(3) Heat Demand Prediction

(4) SBP Prediction
5.2 Operation Performance

Showed in Figure 5.2, the Standard Model and the Proposed Model exhibit a similar behaviour of the expert. Both Models identified the two demand peaks in the morning and the evening. It is obvious that the Vanilla Model has no ability to predict future heat demand. Therefore, the Vanilla Model saved more PV generation than the evening demand and lost the income of exporting PV generation to the grid. The Vanilla Model can be improved by setting two sets of on-and-off time, one for summer and another for winter, since the averages of heat demand in summer and winter are different.

We can conclude that accurate predictions of heat demand are crucial to the operation of heat storage. Figure 5.3 shows one example that the Standard Model incorrectly predicts two demand peaks. Consequently, it prepared more heat than the actual need. The excess use of heat storage in the morning leads to extra import of electricity. Another excess use in the evening consumes PV generation unnecessarily.

Correct prediction of SBP and SSP is another key factor of a good performance. Even though a Model accurately identifies the heat demand, its performance still can be compromised by inaccurate prediction of price. In Figure 5.4, the Standard Model predicts heat demand in the morning with accuracy to a certain extent. However, it
expects a low SBP at $t_3$; accordingly, the Standard Model starts to charge the heat storage too early, hence unnecessary heat loss in the heat storage occurred and, more importantly, the Standard Model imports electricity with a relative high SBP at $t_3$, as showed in Figure 5.4 in which the SBP (red dot) at $t_3$ (approx. 0.048 £/kWh) is much higher than $t_{10}$ (approx. 0.036 £/kWh), of which time the expert starts to charge the storage in the morning.

The same behaviour of the Standard Model can be seen in Figure 5.5 during the morning. Since outputs of the unreliable SBP predictor in the Standard Model are stuck around the average of SBPs, it’s hard for the Standard Model to detect the sudden drop of SBP at $t_{10}$ in Figure 5.5.

In addition, incorrect prediction of PV generation can also weaken the performance of the Standard Model. In Figure 5.4, there are two PV generation peaks at $t_{23}$ and $t_{29}$. Unlike the Proposed Model and the expert, the Standard Model charges no heat into the storage during the peak at $t_{29}$ because it does not expect this PV generation peak. It uses PV generation peak at $t_{23}$ to charge the storage, and hence suffers from unnecessary heat loss in the heat storage.

Note that in Figure 5.4 the true values of SSP during the midday are nearly the same. Thus, the reason for the expert to choose to consume PV generation at $t_{29}$, instead of $t_{23}$, is not because of a notable difference of SSP but considering on heat loss over the course of time. The predictive SSPs provided by SSP predictor in the Standard Model are almost the same as the average number and therefore we can conclude that in Figure 5.4 the Standard Model uses PV generation peak at $t_{23}$ because it didn’t expect another peak at $t_{29}$, but not because it expects a higher SSP around $t_{29}$.
The operation curve of the Proposed Model demonstrates roughly the same pattern as of the expert. Unlike the Standard Model, the network in the Proposed Model is trained to directly predict a target level. We cannot discuss the behaviour of the Proposed Model like we do with the Standard Model in above paragraphs because the network in the Proposed Model does not predict each feature separately.

**Figure 5.2 One-day simulation (Result 1)**

The blue, red, indigo and yellow lines are the operation curves of the expert, the Proposed Model, the Standard Model and the Vanilla Model, respectively. Green dash line is the PV generation that has been subtracted by electricity demand and converted into heat by COP. Pink dash line is the heat demand. Red and Blue dots are SBP and SSP.
Figure 5.3 One-day simulation (Result 2)

![Graph showing one-day simulation (Result 2)](image1)

Figure 5.4 One-day simulation (Result 3)

![Graph showing one-day simulation (Result 3)](image2)
Chapter 5 Results and Discussion

Figure 5.5 One-day simulation (Result 4)

5.3 Annual Cost

One way to examine the performance is to compare the operation costs of each Model in simulation. We run three one-year simulations for all the Models and summed the daily operation cost according to Equation (3). The results are showed in Table 5.1. Note that the last Model in the table has no heat storage. It sells all PV Surplus to the grid, and whenever there is a Shortage, it imports electricity.

Negative operation cost indicates that the system exported more electricity than imported from the grid in a year. Model without storage has the highest income of importing electricity in all three simulations, as showed in Column (A) in Table 5.1.

Expenditure of importing in Equation (3) can be further separated depending on its purpose, as showed in Column (B) and (C). Since the Vanilla Model and the Model without storage cannot charging the heat storage by importing electricity, both shows zero in Column (C).

Column (D) shows that the expert outperforms other four Models. Our Proposed Model has close performance to the Model without heat storage. The Standard Model and the
Vanilla Model fail to reduce overall operation cost, compared to the Model without heat storage.

To compare the performance of these Models, we defined a number, $e_{op}$, that describes the effectiveness of operation. Operating the heat storage, a Model decreases the total revenue of exporting electricity and increases the total expenditure of importing electricity from the grid for charging the heat storage, as showed in Equation (8) and (9). Similarly, the operation of heat storage reduces the total expenditure of importing electricity for meeting the heat demand, as Equation (10). The Models aim to decrease $E_{PV} + E_{Grid}$ and increase $R$ as much as possible because a higher $e_{op}$ indicates that a Model profits from its operation more effectively, as Equation (11). It is profitable to implement a Model only if the $e_{op}$ of that Model is larger than 1.

\[
E_{PV} = \text{Expenditure of PV Surplus for charging the storage}
\]

\[
= (\text{Revenue of a Model}) - (\text{Revenue of the Model without storage}) \tag{8}
\]

\[
E_{Grid} = \text{Expenditure of importing electricity for charging the storage} \tag{9}
\]

\[
R = \text{Reduction in Expenditure of importing electricity for heat demand}
\]

\[
= (\text{Expenditure of the Model without storage}) - (\text{Expenditure of a Model}) \tag{10}
\]

\[
e_{op} = \frac{R}{E_{PV} + E_{Grid}} \tag{11}
\]

Table 5.2 shows each $e_{op}$ of each Model in the three simulations. As the same we observe from the comparison of total operation cost of each Model, the expert has the highest $e_{op}$ around 1.55. Our Propose Model nearly meets the requirement with a $e_{op}$ around 0.98. The Standard Model and the Vanilla Model fails with $e_{op}$ around 0.75 and 0.45 respectively.
We also calculated different $e_{op}$ in each week in the simulation result 1, as showed in Table 5.3, to examine how PV generation and heat demand affects $e_{op}$ of each Model.

During the cold weeks, such as week 1, 2, 12 and 13, we have smaller amount of PV generation to meet the heat demand directly or to be charged into the heat storage in advanced. Since SBP are always larger or equal to SSP, using PV Surplus is usually more economical than importing electricity. Consequently, with less amount of economical PV generation, the operation costs of these weeks are positive.

It should be note that the term, ‘cold’ or ‘warm,’ does not mean that the weather is colder or warmer in those weeks. ‘Cold’ means the system must import more electricity from the grid because the total PV generation is relative lower, and/or the total heat demand is relative higher.

The $e_{op}$ of the Proposed Model, Standard Model and Vanilla Model is greater than 1 during the cold weeks. In addition, $e_{op}$ of the expert during the cold weeks are greater than during the warm weeks. This is because most of the time during the cold weeks the Models has no need to predict PV generation correctly since PV generation in cold weeks is relative less and has less influence on operation. Consequently, the Models need only reliable predictions on demand and prices, and thus it is easier for the Models to make a better decision. Since the price predictors of the Standard Model are less effective, the $e_{op}$ of the Standard Model is lower than of others in the cold weeks.

The Vanilla Model sometimes has better $e_{op}$ during cold weeks because most of the time in a cold week the remaining PV Surplus is usually small, and the heat demand is usually large. Therefore, with a lower risk of suffering from unnecessary heat loss in the storage, it’s tolerable to always store all remaining PV Surplus for the heat demand in the evening.
Note that even though the $e_{op}$ of the Vanilla Model is greater, it does not guarantee that the Vanilla Model can outperform other Models because the Vanilla Model has no concern with price prediction and importing electricity. Table 5.4 shows the $e_{op}$ and the total reduction, $R$, of operation cost during cold weeks. In week 1, the $e_{op}$ of the Vanilla Model (1.54) is greater than the Proposed Model (1.32). However, $R$ of the Vanilla Model (£65) is less than the Proposed Model (£197). The same occurs in week 13.

**Table 5.1 Yearly Operation Cost**

Result 1:

<table>
<thead>
<tr>
<th>Model</th>
<th>(D) Operation Cost (£)</th>
<th>(A) Sell to the Grid</th>
<th>(B) Buy for Heat Demand</th>
<th>(C) Buy for Charging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expert</td>
<td>-48999</td>
<td>-53150</td>
<td>1494</td>
<td>2657</td>
</tr>
<tr>
<td>Proposed Model</td>
<td>-46429</td>
<td>-53377</td>
<td>3889</td>
<td>3059</td>
</tr>
<tr>
<td>Standard Model</td>
<td>-44744</td>
<td>-52577</td>
<td>3212</td>
<td>4621</td>
</tr>
<tr>
<td>Vanilla Model</td>
<td>-44433</td>
<td>-51245</td>
<td>6812</td>
<td>0</td>
</tr>
<tr>
<td>Without Storage</td>
<td>-46459</td>
<td>-54913</td>
<td>8454</td>
<td>0</td>
</tr>
</tbody>
</table>
Chapter 5 Results and Discussion

Result 2:

<table>
<thead>
<tr>
<th>Model</th>
<th>Operation Cost (£)</th>
<th>(A) Sell to the Grid</th>
<th>(B) Buy for Heat Demand</th>
<th>(C) Buy for Charging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expert</td>
<td>-51713</td>
<td>-55934</td>
<td>1428</td>
<td>2793</td>
</tr>
<tr>
<td>Proposed Model</td>
<td>-49154</td>
<td>-56131</td>
<td>3763</td>
<td>3214</td>
</tr>
<tr>
<td>Standard Model</td>
<td>-47452</td>
<td>-55345</td>
<td>3144</td>
<td>4749</td>
</tr>
<tr>
<td>Vanilla Model</td>
<td>-47151</td>
<td>-54062</td>
<td>6911</td>
<td>0</td>
</tr>
<tr>
<td>Without Storage</td>
<td>-49272</td>
<td>-57608</td>
<td>8336</td>
<td>0</td>
</tr>
</tbody>
</table>

Result 3:

<table>
<thead>
<tr>
<th>Model</th>
<th>Operation Cost (£)</th>
<th>(A) Sell to the Grid</th>
<th>(B) Buy for Heat Demand</th>
<th>(C) Buy for Charging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expert</td>
<td>-42620</td>
<td>-46884</td>
<td>1619</td>
<td>2645</td>
</tr>
<tr>
<td>Proposed Model</td>
<td>-40072</td>
<td>-47185</td>
<td>4103</td>
<td>3010</td>
</tr>
<tr>
<td>Standard Model</td>
<td>-38491</td>
<td>-46326</td>
<td>3325</td>
<td>4510</td>
</tr>
<tr>
<td>Vanilla Model</td>
<td>-38236</td>
<td>-45116</td>
<td>6880</td>
<td>0</td>
</tr>
<tr>
<td>Without Storage</td>
<td>-40223</td>
<td>-48689</td>
<td>8466</td>
<td>0</td>
</tr>
</tbody>
</table>
### Table 5.2 Operation effectiveness, $e_{op}$

<table>
<thead>
<tr>
<th>Model</th>
<th>Result 1</th>
<th>Result 2</th>
<th>Result 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expert</td>
<td>1.57</td>
<td>1.55</td>
<td>1.54</td>
</tr>
<tr>
<td>Proposed Model</td>
<td>0.99</td>
<td>0.97</td>
<td>0.97</td>
</tr>
<tr>
<td>Standard Model</td>
<td>0.75</td>
<td>0.74</td>
<td>0.75</td>
</tr>
<tr>
<td>Vanilla Model</td>
<td>0.45</td>
<td>0.40</td>
<td>0.44</td>
</tr>
</tbody>
</table>

### Table 5.3 Operation effectiveness, $e_{op}$, of each week in Result 1

<table>
<thead>
<tr>
<th>Week</th>
<th>Expert</th>
<th>Proposed Model</th>
<th>Standard Model</th>
<th>Vanilla Model</th>
<th>Operation Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.74</td>
<td>1.32</td>
<td>1.16</td>
<td>1.54</td>
<td>positive</td>
</tr>
<tr>
<td>2</td>
<td>1.6</td>
<td>1.16</td>
<td>0.99</td>
<td>1.47</td>
<td>positive</td>
</tr>
<tr>
<td>3</td>
<td>1.43</td>
<td>0.89</td>
<td>0.75</td>
<td>0.77</td>
<td>negative</td>
</tr>
<tr>
<td>4</td>
<td>1.43</td>
<td>0.71</td>
<td>0.58</td>
<td>0.31</td>
<td>negative</td>
</tr>
<tr>
<td>5</td>
<td>1.42</td>
<td>0.68</td>
<td>0.47</td>
<td>0.12</td>
<td>negative</td>
</tr>
<tr>
<td>6</td>
<td>1.63</td>
<td>0.62</td>
<td>0.41</td>
<td>0.07</td>
<td>negative</td>
</tr>
</tbody>
</table>
### Table 5.4 $e_{op}$ and $R$ of operation cost during cold weeks in Result 1

<table>
<thead>
<tr>
<th>Week</th>
<th>Expert</th>
<th>Proposed Model</th>
<th>Standard Model</th>
<th>Vanilla Model</th>
<th>Operation Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1.46</td>
<td>0.36</td>
<td>0.26</td>
<td>0.05</td>
<td>negative</td>
</tr>
<tr>
<td>8</td>
<td>1.56</td>
<td>0.46</td>
<td>0.39</td>
<td>0.06</td>
<td>negative</td>
</tr>
<tr>
<td>9</td>
<td>1.43</td>
<td>0.65</td>
<td>0.49</td>
<td>0.13</td>
<td>negative</td>
</tr>
<tr>
<td>10</td>
<td>1.46</td>
<td>0.81</td>
<td>0.62</td>
<td>0.29</td>
<td>negative</td>
</tr>
<tr>
<td>11</td>
<td>1.45</td>
<td>0.85</td>
<td>0.75</td>
<td>0.49</td>
<td>negative</td>
</tr>
<tr>
<td>12</td>
<td>1.59</td>
<td>1.19</td>
<td>1.01</td>
<td>1.01</td>
<td>positive</td>
</tr>
<tr>
<td>13</td>
<td>1.72</td>
<td>1.27</td>
<td>1.06</td>
<td>1.50</td>
<td>positive</td>
</tr>
</tbody>
</table>
### 5.4 Training and Computation Efficiency

Since the Standard Model and the Proposed Model follow the different concept as showed in Figure 1.1, Figure 1.2, Figure 1.3 and Figure 1.4, it is interesting to examine the training and computation efficiency of the two Models.

#### 5.4.1 Preparation of Training Dataset

For the five predictors in Standard Model, time spent for preparing the training dataset is neglectable because it is only a rearrangement of values according to each time steps. On

<table>
<thead>
<tr>
<th></th>
<th>Expert</th>
<th>Proposed Model</th>
<th>Standard Model</th>
<th>Vanilla Model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Week 2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$e_{op}$</td>
<td>1.6</td>
<td>1.16</td>
<td>0.99</td>
<td>1.47</td>
</tr>
<tr>
<td>$R$</td>
<td>379</td>
<td>90</td>
<td>-7</td>
<td>103</td>
</tr>
<tr>
<td><strong>Week 12</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$e_{op}$</td>
<td>1.59</td>
<td>1.19</td>
<td>1.01</td>
<td>1.01</td>
</tr>
<tr>
<td>$R$</td>
<td>363</td>
<td>113</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td><strong>Week 13</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$e_{op}$</td>
<td>1.72</td>
<td>1.27</td>
<td>1.06</td>
<td>1.50</td>
</tr>
<tr>
<td>$R$</td>
<td>440</td>
<td>169</td>
<td>46</td>
<td>63</td>
</tr>
</tbody>
</table>
the contrary, it took approx. 6 hours to prepare the dataset for the Proposed Model due to the computation caused by running Algorithm 1 for a four-year historical data.

**5.4.2 Training of Models**

It is meaningless to compare the training time of each LSTM networks because the total number of trainable weights/variables is different in different Model. In addition, the training time can also be influenced by the complexity of the dataset, which is different for each predictor.

**5.4.3 Computation Efficiency**

For a one-day simulation, it took approx. 0.8 second for the Proposed Model to make decision, while for the Standard Model it took approx. 1 minute. The difference between 0.8 second and 1 minute is neglectable compared to one day (24 hours), though it demonstrates to what extend an improvement of computation efficiency can be achieved if we build models and train networks in a different way, as discussed in Figure 1.1, Figure 1.2, Figure 1.3 and Figure 1.4.
6 Conclusion

In this paper we proposed a LSTM model for the operation of heat storage in a solar energy community distribution system with PV generation as the only domestic generation and a connection to the main grid. Unlike conventional LSTM model that the networks are only used to predict features for supporting an operator or a control programme to make a decision, our proposed model integrates the operation strategy into the network, and thus provide an operation action directly.

With historical data, we created an expert who can perfectly predict future. This expert follows the operation strategy we proposed in this paper, and then the operation behaviours of this expert are used to train a LSTM network in our proposed model.

We set up three different Models:

A. The Standard Model has five LSTM networks that receive past values of PV generation, electricity demand, heat demand, SSP and SBP to predict future values. These predictive values are then passed to a control programme that follows the operation strategy proposed in this paper to calculate the current target level of the heat storage.

B. The Proposed Model has only one LSTM network that is trained by the operation behaviour of the mock-up expert. This network receives past values of PV generation, electricity demand, heat demand, SSP and SBP to provide the current target level of the heat storage.

C. The Vanilla Model always starts to charge and to discharge the heat storage at fixed times every day. This model has no LSTM network.
We conducted one-year simulations for the expert, the three Models and a system without heat storage. To decrease the total cost of importing electricity to meet the heat demand, each model consumes PV generation that could have been sold to the grid or imports electricity to charge the heat storage when SBP is relative low. We defined a number, $e_{op}$, to describe the operation effectiveness of a Model:

$$e_{op} = \frac{R}{E_{PV} + E_{grid}}$$

$R = \text{Reduction in Expenditure of importing electricity for heat demand}$

$E = \text{Expenditure of PV Surplus or importing electricity for charging heat storage}$

The results of one-year simulations show that the expert has the highest $e_{op}$ around 1.55, and the Propose Model has $e_{op}$ around 0.98. The Standard Model and the Vanilla Model fails with $e_{op}$ around 0.75 and 0.45 respectively. The performance of our Proposed Model is nearly to be profitable if its $e_{op}$ can be further improved to be greater than 1.

We found that during the weeks when the PV generation is low, and the heat demand is high, the $e_{op}$ of the Proposed Model, Standard Model and Vanilla Model is greater than 1. This is because the accuracy of prediction on PV generation has less influence on the performance of a Model. Thus, it is easier for a Model to operate the heat storage during a ‘colder’ week.

Since the Standard Model and the Proposed Model introduces different concepts of implementing LSTM networks, computation efficiency of each Model during the simulation is different. The Standard Model first runs its five LSTM networks to predict features related to operation, and then run the operation strategy to decide an operation action. On the other hand, the Proposed Model directly predicts an operation action. The computation time spent by the Standard Model is 75 times larger than the Proposed Model.
Chapter 6 Conclusion

With the same input (five features at \( t_{n-48} \) to \( t_{n-1} \)), our Proposed Model has a better operation efficiency and less computation time in simulation than the Standard Model that follows the conventional way of implement LSTM networks in decision making of system operation.

In further studies, we intend to create other experts by new operation strategies or by real experience of human operator. By introducing new operation strategy, the number of input features may increase or decrease and further affect \( e_{op} \) of the model. On the other hand, if we introduce human operation experience, the selection of input features would be the key decision for constructing the model. Alternatively, the model can learn directly from extracting a policy from the human operation experience [11] without conducting a supervised learning.

We also aim to examine different scenario for this solar energy community distribution system, such as an increase or decrease in the number of houses or solar panels. This would affect \( e_{op} \) of the models because it changes the amount of PV generation and heat demand in certain weeks, and thus makes a week ‘warmer’ or ‘colder,’ as we discussed in Chapter 5.3. Another scenario is that we can put the solar energy community distribution system into another electricity market which is different from the imbalance prices we used in this paper. We can also consider how carbon tax or subsidy for solar energy influences the operation strategy and the performance of our Proposed Model.
7 References


Chapter 8 Appendices

8 Appendices

Appendix 1 Comparisons of predictive and true values in the Standard Model 48

Appendix 2 Python Code 58
Appendix 1 Comparisons of predictive and true values in the Standard Model

(1) PV generation Prediction
Chapter 8 Appendices
(2) Electricity Demand Prediction
Chapter 8 Appendices

(3) Heat Demand Prediction
Chapter 8 Appendices
Chapter 8 Appendices

(4) SBP Prediction

![Graph 1](image1)

![Graph 2](image2)

![Graph 3](image3)
(5) SSP Prediction
Chapter 8 Appendices
Appendix 2 Python Code

```python
1 2 import os
3 import pandas as pd
4 import numpy as np
5 import time
6 import math
7 8 class GridParameter():
9     def __init__(self):
10         self.HP_CAPACITY = 250.0
11         self.HP_COP = 3.667
12         self.MAX_DISCHARGE_ABS = 250.0
13         self.DATA_FILENAME = 'Historical_Data.pkl'
14     # Historical_Data.pkl is a pd.DataFrame with six columns:
15     # Date Time    PV    ElecDemand HeatDemand SSP SBP
16     # 0 2014-09-17 00:00:00 0.000000 18.059880 2.998558 0.033500 0.039760
17     # 1 2014-09-17 00:30:00 0.000000 15.479212 2.998558 0.036650 0.037740
18     # 2 2014-09-17 01:00:00 0.000000 13.544318 2.998558 0.037390 0.038200
19     #                  ....
20     self.TIME_INTERVAL_PER_DAY = 48
21     self.TANK_CAPACITY = 1500.0
22     self.TANKLOSS_PER_DAY = 0.7
23     self.TANKLOSS_PER_T = pow(self.TANKLOSS_PER_DAY,
24                                  1/self.TIME_INTERVAL_PER_DAY)
25     self.BENEFIT_THRESHOLD = 1.0
26     # a threshold of the ratio(of cost)
27     # = [stored heat from other time / import electricity right now]  
28     # only ratio smaller than BENEFIT_THRESHOLD would be taken into
29     # consideration. Thus, we can avoid storing heat(namely occupying
30     # tank capacity) that doesn’t bring much benefit
31     self.DECISION_TIME = 1
32     # if set to be 24, the storage would only be charged after
33     # 12:00 AM everyday
34     self.PRIME_PV = 7
35 36 class Expert(GridParameter):
37     def __init__(self):
38         GridParameter.__init__(self)
39         self.Data = pd.read_pickle(self.DATA_FILENAME)
40         self.Data_len = self.Data.shape[0]
41         self.DAYS = self.Data_len / self.TIME_INTERVAL_PER_DAY
42         np.random.seed(4944)
43         noise = 1 + np.random.normal(0, 1, self.Data_len)*0.03
44         self.Data.loc[:, 'ElecDemand'] = self.Data.loc[:, 'ElecDemand']*noise
45         np.random.seed(4944)
46         noise = 1 + np.random.normal(0, 1, self.Data_len)*0.05
47         self.Data.loc[:, 'HeatDemand'] = self.Data.loc[:, 'HeatDemand']*noise
48         self.reset_dataframe()
49 50     def reset_dataframe(self):
51         self.Data['Tank'] = 0.0
52         self.Data['Prediction'] = 0.0
```
def create_train_data(self):
    self.reset_dataframe()

    for index in range(0, self.Data_len, self.TIME_INTERVAL_PER_DAY):
        generate_episode(index)
        print('day:', index/self.TIME_INTERVAL_PER_DAY, end='\n')

        if index%(500*48) == 0:
            filename = 'Historical_Data_expert.done.checkpoint.pkl'
            self.Data.to_pickle(filename)
            print('checkpoint at ', filename)
            print(')

        filename = 'Historical_Data_expert.done.pkl'
        self.Data.to_pickle(filename)
        print('pickle as ', filename)
        print(')

    def generate_episode(self, t):
        episode = self.Data[t : t + self.TIME_INTERVAL_PER_DAY].copy()
        episode.reset_index(inplace=True)
        self.tank_cp = self.TANK_CAPACITY

        Profit = self.CreateProfitTable(episode)

        ## ---- filling the episode, according to Profit table

        for _ in range(Profit.shape[0]):
            try:
                Profit.index[0]
            except:
                try:
                    print('All entries in Profit table are done.')
                    break
            else:
                t_curr = Profit.loc[Profit.index[0], 't_curr']
                t_past = Profit.loc[Profit.index[0], 't_past']
source = Profit.loc[Profit.index[0], 'source']
Profit, trimmed = self.DistributeEnergy(t_curr, t_past,  
    source, episode, Profit)

if trimmed == False:
    Profit = Profit.drop(Profit.index[0])
    ## --delete first row because it's done.

episode.set_index('index', inplace=True)
self.Data.update(episode)

def CreateProfitTable(self, episode):

    ## profit = stored price / current import price, smaller better  
    ## t_curr = current time  
    ## t_past = between t_0 and t_curr  
    ## source should be a string, either "PV" or "Grid"
    list1 = ['profit', 't_curr', 't_past', 'source']
    Profit = pd.DataFrame(columns = list1)
    rows = 0

    for t_curr in range(self.DECISION_TIME, self.TIME_INTERVAL_PER_DAY):
        if episode.loc[[t_curr, 'Shortage h']] <= 0:
            pass  ## no need to 'buy' heat, no need to calculate profit
        else:
            denominator = episode.loc[t_curr, 'SBP']

        t_past = t_curr

        while t_past >= self.DECISION_TIME:
            ## (1) importing electricity at t_past to charge tank  
            ## for later use at t_curr
            numerator=episode.loc[t_past, 'SBP'] /  
            (self.TANK_LOSS_PER_T**((t_curr-t_past))

            ratio = numerator / denominator
            ## alternative source / importing right now
            if ratio < self.BENEFIT_THRESHOLD:
                Profit.loc[rows] = [ratio, t_curr, t_past, 'Grid']
                rows += 1

            ## (2) using PV surplus at t_past to charge tank  
            ## for later use at t_curr
            if episode.loc[[@past, 'PVSurplus h']] > 0:
                numerator=episode.loc[t_past, 'SSP'] /  
                (self.TANK_LOSS_PER_T**((t_curr - t_past))

                ratio = numerator / denominator
                if ratio < self.BENEFIT_THRESHOLD:
                    Profit.loc[rows] = [ratio, t_curr, t_past, 'PV']
                    rows += 1

            t_past -= 1

    ## break tie by t_curr, we deal with the energy distribution
    ## at early time first. This would probably reduce the occurrence of
Chapter 8 Appendices

```python
# occupying tank capacity for too long
Profit = Profit.sort_values(by = ['profit', 't_curr'])
Profit = Profit.reset_index(drop = True)

# this prime number is for sorting and deleting specific data
# PV = 7, a prime number, making any [label = t * 7] unique
# whenever the solar surplus runs out, or whenerer the solar surplus is
# still available but HP capacity or tank capacity runs out
# we can use this unique label to delete all related entries in profit
# table at once.
# If we delete entries that are no longer feasible,
# overall calculation will be more efficient
for index in range(Profit.shape[0]):
    if Profit.loc[index, 'source'] == 'PV':
        Profit.loc[index, 'PV_label'] = \n        Profit.loc[index, 't_past'] * self.PRIME_PV

return Profit

def DistributeEnergy(self, t_curr, t_past, source, episode, Profit):
    trimmed = False
    # If Profit has been altered in this function,
    # its first rows would have been deleted before being return.
    # Consequently, its first two rows would be deleted in one main loop.
    # (one happens in this function, another in the main loop)
    # Shortage should be positive
    if episode.loc[t_curr, 'Shortage_h'] < 0.0:
        raise ValueError('Shortage should be positive at ', t_curr)

    # toTake = how much we "aim" to take at t_past.
    # Tank loss must be considered.
    toTake = \n    episode.loc[t_curr, 'Shortage_h']/(self.TANK_LOSS_PER_T**(t_curr-t_past))

    #----- checking feasibility -----------------------------

    # Grid is unlimited, but PV is not.
    if source == 'PV':
        if episode.loc[t_past, 'PVSurplus_h'] < 0.0:
            raise ValueError('PVSurplus should be positive at ', t_past)

        if episode.loc[t_past, 'PVSurplus_h'] < toTake:
            # PV surplus at t_past is less than we aim to take
            toTake = episode.loc[t_past, 'PVSurplus_h']
            # so we can only aim to take all the PV surplus
            # else, we can keep our original aim

        # HP capacity has a limit
        if episode.loc[t_past, 'HP'] < toTake:
            # HP capacity at t_past is less than our aim
            toTake = episode.loc[t_past, 'HP']
            # so we can only aim to take all the PV surplus
            # else, we can keep our original aim

    # now, check all the Tank capacity from t_past+1 to t_curr
```

Chih-Hsiang Lee - March 2019
## Chapter 8 Appendices

```python
# Note: any change to the heat level of storage at t reflects on 'Tank' at t+1
# for example, 'storage' at t=0 is a result from all events happened before t=0,
# which means 'storage' at t represents the status of tank "at the start of t"
# Thus, if we change tank at time t_past,
## we check tank capacity from t_past+1 to t_curr

```
t = t_past + 1
toStore = toTake

```python
while t <= t_curr:
    toStore = toStore * self.TANK_LOSS_PER_T
    EmptyTank = self.tank_cp - episode.loc[t, 'Tank']
    if EmptyTank < toStore:
        # Tank capacity at t is less than our aim
        toStore = EmptyTank
        # so we can only aim to use all the remaining capacity
        # else, we can keep our original aim for t

    t += 1
```

## after all above,
## now toStore =
## a feasible amount of heat that can be stored at the start of t_curr

```
##--- updating episode data  ---------------------------------------------

## let 'Shortage_h' at t_curr consumes the heat we stored for it
## note that this doesn't means HeatDemand at t has consumed the stored heat yet,
## nor the number of 'storage' at t_curr need to be decreased.
## We can see that HeatDemand at t has been fulfilled only at the data of t+1 on
## the final episode table.
## 'Shortage_h' is just a variable for calculation of how to distribute available
## energy.

```
episode.loc[t_curr, 'Shortage_h'] -= toStore

```python
if abs(episode.loc[t_curr, 'Shortage_h']) < 1e-6:
    trimmed = True
    episode.loc[t_curr, 'Shortage_h'] = 0.0
    Profit = Profit[Profit.t_curr != t_curr]
    # delete all entries of t_curr in profit table
    # because Shortage at t_curr has been fulfilled

## put toStore into storage at (the start of) t_curr
episode.loc[t_curr, 'Tank'] += toStore

```python
if abs(self.tank_cp - episode.loc[t_curr, 'Tank']) < 1e-6:
    trimmed = True
    episode.loc[t_curr, 'Tank'] = self.tank_cp
    Profit = Profit[\n        np.logical_or(\n            np.logical_and(Profit.t_curr < t_curr, Profit.t_past < t_curr ),\n            np.logical_and(Profit.t_curr > t_curr, Profit.t_past > t_curr ))\n        ]
    # because storage is full at this t,
    # any attempt from profit table that tries to "cross" t would fail
    # "cross" means storing heat at time< and reserving it until another time>
```python
## put toStore into storage in each t, retrospectively
## (TANK LOSS_PER_T should be considered)
t = t_curr - 1

while t > t_past:
    ## charging storage at time t_past only affects tank from
    ## t_past+1 to t_curr
    toStore = toStore / self.TANK LOSS_PER_T
    episode.loc[t, 'Tank'] = toStore

    if abs(self.tank_cp - episode.loc[t, 'Tank']) < 1e-6:
        trimmed = True
        episode.loc[t, 'Tank'] = self.tank_cp
        Profit = Profit[np.logical_or(
            np.logical_and(Profit.t_curr < t, Profit.t_past < t),
            np.logical_and(Profit.t_curr > t, Profit.t_past > t))]

    ## because storage is full at t,
    ## Any attempt from profit table that tries to "cross" t would fail.
    ## "Cross" means storing heat at time < t and reserving it until
    ## another time > t.
    t = 1

    ## after above,
    ## now toStore = the amount of heat would be stored at the end of t_past

## use HP capacity at t_past to generate toStore
episode.loc[t_past, 'HP'] = toStore
if abs(episode.loc[t_past, 'HP']) < 1e-6:
    trimmed = True
    episode.loc[t_past, 'HP'] = 0.0
    Profit = Profit[Profit.t_past != t_past]

    ## because HP can no long generate heat at t_past,
    ## any attempt from profit table that tries to
    ## charge storage at t_past would fail

    ## spend and record the source
    if source == 'PV':
        episode.loc[t_past, 'PV surplus h'] = toStore
        episode.loc[t_past, 'PV charge h'] = toStore
        if abs(episode.loc[t_past, 'PV surplus h']) < 1e-6:
            trimmed = True
            episode.loc[t_past, 'PV surplus h'] = 0.0
            PV_label = t_past * self.PRIME PV
            Profit = Profit[Profit.PV_label != PV_label]

            ## because there is no PV surplus at t_past,
            ## any attempt from profit table that tries to
            ## charge tank by PV at t_past would fail
        else:
            episode.loc[t_past, 'Grid charge h'] = toStore

    return Profit, trimmed
```

********** Object: predictors **********
from keras.models import load_model
from sklearn.preprocessing import MinMaxScaler

class FORECASTER():
    def __init__(self):
        self.agent_PV = self.PV_AGENT()
        self.agent_HD = self.HD_AGENT()
        self.agent_ED = self.ED_AGENT()
        self.agent_SSP = self.SSP_AGENT()
        self.agent_SBP = self.SBP_AGENT()
        self.expert_forecast = self.EXPERT_FOR_FORECAST()
        self.TIME_INTERVAL_PER_DAY = 48  # should have avoided 'magic number'
        self.set_cutoff()

    def produce_hat(self, ndarray, agent):
        ndarray = ndarray.reshape([ndarray.shape[0], ndarray.shape[1], 1])
        return agent.predict(ndarray)

    def set_cutoff(self, start_from = 24):
        self.start_from = start_from

    def _forecast_each(self, input_arr):
        for i in range(input_arr.shape[0]):
            half_hour = int(self.TIME_INTERVAL_PER_DAY

            if half_hour == 0:
                dataframe = pd.DataFrame(columns=
                                    ['PV', 'ElecDemand', 'HeatDemand', 'SSP', 'SBP'])
                dataframe.loc['PV'] = self.hatPV[i]
                dataframe.loc['ElecDemand'] = self.hatED[i]
                dataframe.loc['HeatDemand'] = self.hatHD[i]
                dataframe.loc['SSP'] = self.hatSSP[i]
                dataframe.loc['SBP'] = self.hatSBP[i]
                self.tank_target.append(0.0)

            else:
                dataframe.loc[:, 'PV'] = self.re_inputPV[i][:half_hour]
                dataframe.loc[:, 'ElecDemand'] =
                self.re_inputED[i][:half_hour]
                dataframe.loc[:, 'HeatDemand'] =
                self.re_inputHD[i][:half_hour]
                dataframe.loc[:, 'SSP'] = self.re_inputSSP[i][:half_hour]
                dataframe.loc[:, 'SBP'] = self.re_inputSBP[i][:half_hour]

        return dataframe
Chapter 8 Appendices

```python
if half_hour < self.start_from:
    self.tank_target.append(0.0)
else:
    self.expert_4forecast.generate_episode(dataframe)
    self.tank_target.append(dataframe.loc[half_hour, 'Tank'])

self.SBPthat.append(dataframe.loc[half_hour, 'SBP'])
self.SSPthat.append(dataframe.loc[half_hour, 'SSP'])

self.yhat = np.asarray(self.tank_target)
sel_yhat = self.yhat.reshape(input_arr.shape[0], 1)

return sel_yhat

def forecast(self, input_arr):
    # PLZ check columns=['PV', 'ED', 'HD', 'SSP', 'SBP']
    self.inputPV = input_arr[:, :, 0]  # scaled value
    self.inputHD = input_arr[:, :, 2]
    self.inputED = input_arr[:, :, 1]
    self.inputSSP = input_arr[:, :, 3]
    self.inputSBP = input_arr[:, :, 4]

    self.re_inputPV = self.agent_PV.rescaler.inverse_transform(self.inputPV)
    self.re_inputHD = self.agent_HD.rescaler.inverse_transform(self.inputHD)
    self.re_inputED = self.agent_ED.rescaler.inverse_transform(self.inputED)
    self.re_inputSSP = self.agent_SSP.rescaler.inverse_transform(self.inputSSP)
    self.re_inputSBP = self.agent_SBP.rescaler.inverse_transform(self.inputSBP)

    self.hatPV = self.produce_yhat(input_arr[:, :, 0], self.agent_PV)
    self.hatHD = self.produce_yhat(input_arr[:, :, 2], self.agent_HD)
    self.hatED = self.produce_yhat(input_arr[:, :, 1], self.agent_ED)
    self.hatSSP = self.produce_yhat(input_arr[:, :, 3], self.agent_SSP)
    self.hatSBP = self.produce_yhat(input_arr[:, :, 4], self.agent_SBP)

    self.yhat = self._forecast_each(input_arr)

return self.yhat

class PV_AGENT:
    def __init__(self):
        self.model = load_model('lstm_model_PV.h5')
        self.scale_base = pd.read_pickle('PV_scale_base.pkl')
        self.scale_base.drop(columns=['DateTime'], inplace=True)
        self.rescaler= MinMaxScaler()
        self.rescaler.fit(self.scale_base)
```
def predict(self, array):
    yhat = self.model.predict(array, verbose=0)
    yhat = self.rescaler.inverse_transform(yhat)
    return yhat

class HD_AGENT():
    def __init__(self):
        self.model = load_model('lstm_model_HD.h5')
        self.scale_base = pd.read_pickle('HD_scale_base.pkl')
        try:
            self.scale_base.drop(columns=['DateTime'], inplace=True)
        except:
            pass
        self.rescaler = MinMaxScaler()
        self.rescaler.fit(self.scale_base)

def predict(self, array):
    yhat = self.model.predict(array, verbose=0)
    yhat = self.rescaler.inverse_transform(yhat)
    return yhat

class ED_AGENT():
    def __init__(self):
        self.model = load_model('lstm_model_ED.h5')
        self.scale_base = pd.read_pickle('ED_scale_base.pkl')
        try:
            self.scale_base.drop(columns=['DateTime'], inplace=True)
        except:
            pass
        self.rescaler = MinMaxScaler()
        self.rescaler.fit(self.scale_base)

def predict(self, array):
    yhat = self.model.predict(array, verbose=0)
    yhat = self.rescaler.inverse_transform(yhat)
    return yhat

class SSP_AGENT():
    def __init__(self):
        self.model = load_model('lstm_model_SSP.h5')
        self.scale_base = pd.read_pickle('SSP_scale_base.pkl')
        try:
            self.scale_base.drop(columns=['DateTime'], inplace=True)
        except:
            pass
        self.rescaler = MinMaxScaler()
        self.rescaler.fit(self.scale_base)

def predict(self, array):
    yhat = self.model.predict(array, verbose=0)
    yhat = self.rescaler.inverse_transform(yhat)
    return yhat

class SBP_AGENT():
    def __init__(self):
        self.model = load_model('lstm_model_SBP.h5')
        self.scale_base = pd.read_pickle('SBP_scale_base.pkl')
        try:
            self.scale_base.drop(columns=['DateTime'], inplace=True)
        except:
Chapter 8 Appendices

```python
    pass
    self.rescaler = MinMaxScaler()
    self.rescaler.fit(self.scale_base)

def predict(self, array):
    yhat = self.model.predict(array, verbose=0)
    yhat = self.rescaler.inverse_transform(yhat)
    return yhat

class EXPERT_FOR_FORECAST(Expert):
    def __init__(self):
        GridParameter.__init__(self)

    def reset_episode(self, episode):
        episode['Tank'] = 0.0
        episode['HP'] = self.HP_CAPACITY - episode['HeatDemand']

        episode['PVOutput_h'] = \n            episode['PV'] - episode['ElecDemand']) * self.HP_COP
        updateby = episode[episode['PVOutput_h'] < 0].copy()
        updateby.loc[:, 'PVOutput_h'] = 0.0
        episode.update(updateby)

        episode['PVSurplus_h'] = episode['PVOutput_h'] - episode['HeatDemand']
        episode['Shortage_h'] = -episode['PVSurplus_h']

        updateby = episode[episode['PVSurplus_h'] < 0].copy()
        updateby.loc[:, 'PVSurplus_h'] = 0.0
        episode.update(updateby)

        updateby = episode[episode['Shortage_h'] < 0].copy()
        updateby.loc[:, 'Shortage_h'] = 0.0
        episode.update(updateby)

        episode['GridCharge_h'] = 0.0
        # how much heat generated by imported elec for charging tank
        episode['PVCharge_h'] = 0.0
        # how much heat generated by PV elec for charging tank

    def generate_episode(self, episode):
        self.tank_cp = self.TANK_CAPACITY
        self.reset_episode(episode)

        Profit = self.CreateProfitTable(episode)

        # ---- filing the episode, according to Profit table

        for _ in range(Profit.shape[0]):
            # --works like a while-loop,
            try:
                # --try to pick up first row
                Profit.index[0]
            except:
                # --if there's no first row, all is done
```
```python
## print('All entries in Profit table are done.')
break
else:
t_cur = Profit.loc[Profit.index[0], 't_cur']
t_past = Profit.loc[Profit.index[0], 't_past']
source = Profit.loc[Profit.index[0], 'source']
Profit, trimmed =
    self.DistributeEnergy(t_cur, t_past,
                            source, episode, Profit)

if trimmed == False:
    Profit = Profit.drop(Profit.index[0])
    ## --delete first row because it's done.

return episode.loc[0, 'Tank']

import pickle
import os
import pandas as pd
import numpy as np
import time
import math
from matplotlib import pyplot
from sklearn.preprocessing import MinMaxScaler
from keras.models import load_model
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout

class Simulator(GridParameter):
    def __init__(self, model, forecaster,
                 environment='historical_data_expert_done.pkl'):
        ## Historical_data_expert_done.pkl = Historical_Data.pkl + expert's operation
        ## columns = ["DateTime", "PV", "ElecDemand", "HeatDemand", "SSP", "SBP", "Tank",
        ## "Prediction", "HP", "PVOutput_h", "PVSurplus_h", "Shortage_h",
        ## "GridCharge_h", "PVCharge_h"]
        ## Note that we didn't delete the column 'Prediction' even though in our actual
        ## implementation we never use this column.

        GridParameter.__init__(self)
        self.MODEL = model
        self.FORECASTER = forecaster
        self.HISTORY = pd.read_pickle(environment)
        self.EPISODE_LEN = 7 * self.TIME_INTERVAL_PER_DAY

        self.rescaler = MinMaxScaler()
        drops = ["DateTime", "Tank", "Prediction", "HP", "PVOutput_h",
                 "PVSurplus_h", "Shortage_h", "GridCharge_h", "PVCharge_h"]
        self.rescaler_history = 
        self.rescaler.fit_transform(self.HISTORY.drop(columns=drops))

```

```python
self.rscd_history = pd.DataFrame(self.rscd_history)
selself.rscd_history.columns = \nself.HISTORY.drop(columns=drops).columns.tolist()

### vanilla strategy
self.VANI_CHARGE = 25
self.VANI_DISCHARGE = 34
self.VANI_MAX_TARGET = 1200.0

def run(self, start, figure=False, network=True, forecast=True,
vanilla=True, no_tank=True, day=1):
    self.EPISODE_LEN = day*self.TIME_INTERVAL_PER_DAY
    curve_expert, cost_expert = self.expert_perform(start)
    print('expert', curve_expert.shape, cost_expert)
    
    if network:
        curve_network, cost_network = \n        self.network_perform(start, self.MODEL, scaled=True)
        print('network operation', curve_network.shape, cost_network)
        
    if forecast:
        curve_forecast, cost_forecast = \n        self.network_perform(start, self.FORECASTER, scaled=False)
        print('forecaster operation', curve_forecast.shape, cost_forecast)
    
    if vanilla:
        curve_vanilla, cost_vanilla = self.vanilla_perform(start)
        print('vanilla operation[25,43]', cost_vanilla)
    
    if no_tank:
        cost_no_tank = self.no_tank_perform(start)
        print('no Tank', cost_no_tank)

    if figure is True:
        duration = self.EPISODE_LEN
        
        curve_pv_output_heat = \n        self.HISTORY[start:start+duration]['PVOutput_h'].values
        curve_heat_demand = \n        self.HISTORY[start:start+duration]['HeatDemand'].values
        point_SB = self.HISTORY[start:start+duration]['SBP'].values
        point_SSP = self.HISTORY[start:start+duration]['SSP'].values
        
        fig, axl = plt.subplots()
        axl.set_xlabel('time (half hour)')
        axl.set_ylabel('heat(kW)')
        axl.plot(curve_heat_demand, 'm', label='curve_heat_demand')
        axl.plot(curve_pv_output_heat, 'g', label='curve_pv_output_heat')
        axl.plot(curve_expert, 'b', label='curve_expert')
        try:
            axl.plot(curve_network, 'r', label='curve_network')
        except:
            pass
        
        try:
```

---

Chih-Hsiang Lee - March 2019
```python
ax1.plot(curve_forecast, 'c', label='curve_forecast')
except:
    pass
try:
    ax1.plot(curve_vanilla, 'y', label='curve_vanilla')
except:
    pass

t2 = ax1.twiny()
# instantiate a second axes that shares the same x-axis
ax2.set_ylabel('price(pound/kW)')
ax2.plot(point_SP, 'rD', label='point_SP')
ax2.plot(point_SP, 'b+', label='point_SP')

fig.tight_layout()  # otherwise the right y-label is slightly clipped
fig.set_size_inches(10, 12, forward=True)
pyplot.legend(prop={'size': 18})
pyplot.savefig('output.png')
pyplot.show()

def __provide__(self, need, have):
    assert need >= 0
    assert have >= 0

    if need >= have:
        need -= have
        provide = have
        have = 0.0
    else:
        have -= need
        provide = need
        need = 0.0

    return need, have, provide

```

```python
# def network_perform(self, start, agent, scaled=True, confident=True):
#     episode = self.HISTORY[start:start+simulator.EPISODE_LEN].copy()
#     episode['Tank'] = 0.0
#     episode['HP'] = self.HP_CAPACITY - episode['HeatDemand']
#     episode['PVSurplus_h'] = episode['PVOutput_h'] - episode['HeatDemand']
#     updated = episode[episode['PVSurplus_h'] < 0].copy()
#     updated['PVSurplus_h'] = 0.0
#     episode.update(updated)
#     del updated
#     episode[episode['HeatDemand'] - episode['PVOutput_h']]
#     updated = episode[episode['Shortage_h'] < 0].copy()
#     updated['Shortage_h'] = 0.0
#     episode.update(updated)
#     del updated
```
Chapter 8 Appendices

```python
episode[GridCharge_h'] = 0.0
episode[PVCharge_h'] = 0.0

index_start = start - self.TIME_INTERVAL_PER_DAY
index_end_exclude = start + self.EPISODE_LEN
rcsd_episode = self.rcsd_history[index_start:index_end_exclude].copy()
input_arr = np.zeros((self.EPISODE_LEN, 48, 5))

for i in range(0, self.EPISODE_LEN):
    input_arr[i] = rcsd_episode[i:i+1].TIME_INTERVAL_PER_DAY.values

if scaled:
    self.yhat = agent.predict(input_arr, verbose=0) * self.TANK_CAPACITY
    keyword = 'proposed'
else:
    self.yhat = agent.forecast(input_arr)
    keyword = 'standard'

for i in range(self.EPISODE_LEN):
    index = i + start
    if confident:
        episode = self.step_confident(episode, index, self.yhat[i][0])
    else:
        episode = self.step(episode, index, self.yhat[i][0])

    ## record how much PV-generated electricity remains
    ## that can be exported at time t
    episode[Export_e'] = episode[PVSurplus_h']/ self.HP_COP

    ## record how much shortage of electricity remains
    ## that we need to import for it at time t
    episode[Import_e'] = episode[Shortage_h']/ self.HP_COP

    ## record how much electricity imported to charge
    ## the tank at time t
    episode[Import_forTank'] = episode[GridCharge_h']/ self.HP_COP

    Export = -episode[SSP'] * episode[Export_e']
    Import = episode[SSP'] * episode[Import_e']
    Import_charge = episode[SSP'] * episode[Import_forTank']

    curve_network = episode[Tank'].values
    cost_network = \
        Export.values.sum() + Import.values.sum() + Import_charge.values.sum()

    return curve_network, cost_network

def step_confident(self, episode, index, target):
    ## always fulfill the larger value requested,
    ## no matter using PV generation or imported electricity
    ## to charge heat storage

    ## during good weather, step_naive() shows better performance than step(),
    ## because there are less time that the system need to import electricity
    ## for charging.
    ## Also, there are more opportunities for the system to

    assert target >= 0
```

Chih-Hsiang Lee - March 2019
assert target <= self.TANK_CAPACITY

tank_curr = episode.loc[index, 'Tank']
action = target - tank_curr

if action > episode.loc[index, 'HP']:
    action = episode.loc[index, 'HP']

## charge
if action >= 0 and episode.loc[index, 'PVSurplus_h'] > 0:
    tank_next = tank_curr * self.TANK_LOSS_PER_T + action
    episode.loc[index, 'PVSurplus_h'] = self._provide(action,
                                            episode.loc[index, 'PVSurplus_h'])
    if action > 0:
        episode.loc[index, 'GridCharge_h'] = action

elif action >= 0 and episode.loc[index, 'Shortage_h'] >= 0:
    tank_next = tank_curr * self.TANK_LOSS_PER_T + action
    episode.loc[index, 'GridCharge_h'] = action

## discharge
elif action < 0 and episode.loc[index, 'PVSurplus_h'] > 0:
    action = abs(action)
    if action > self.MAX_DISCHARGE_ABS:
        action = self.MAX_DISCHARGE_ABS
    tank_next = (tank_curr - action) * self.TANK_LOSS_PER_T

elif action < 0 and episode.loc[index, 'Shortage_h'] >= 0:
    action = abs(action)
    if action > self.MAX_DISCHARGE_ABS:
        action = self.MAX_DISCHARGE_ABS
    tank_next = (tank_curr - action) * self.TANK_LOSS_PER_T
    episode.loc[index, 'Shortage_h', __, __] = 
    self._provide(episode.loc[index, 'Shortage_h'], action)
else:
    raise ValueError('Error')

if index+1 > episode.index[-1]:
    pass
else:
    episode.loc[index+1, 'Tank'] = tank_next

return episode

def step(self, episode, index, target):
    assert target >= 0
    assert target <= self.TANK_CAPACITY
    tank_curr = episode.loc[index, 'Tank']
    action = target - tank_curr

    ## charge
Chapter 8 Appendices

```python
if action == 0 and episode.loc[index, 'PVSurplus_h'] > 0:
    action, episode.loc[index, 'PVSurplus_h'], episode.loc[index, 'PVCharge_h'] = self._provide(action, episode.loc[index, 'PVSurplus_h'])
    tank_next = tank_curr * self.TANK_LOSS_PER_T + episode.loc[index, 'PVCharge_h']

elif action > 0 and episode.loc[index, 'Shortage_h'] >= 0:
    tank_next = tank_curr * self.TANK_LOSS_PER_T #+ action
    episode.loc[index, 'GridCharge_h'] += action

## discharge
elif action < 0 and episode.loc[index, 'PVSurplus_h'] > 0:
    action = abs(action)
    if action > self.MAX_DISCHARGE_ABS:
        action = self.MAX_DISCHARGE_ABS
    tank_next = (tank_curr - action) * self.TANK_LOSS_PER_T

elif action < 0 and episode.loc[index, 'Shortage_h'] >= 0:
    action = abs(action)
    if action > self.MAX_DISCHARGE_ABS:
        action = self.MAX_DISCHARGE_ABS
    tank_next = (tank_curr - action) * self.TANK_LOSS_PER_T
    episode.loc[index, 'Shortage_h'] += self._provide(episode.loc[index, 'Shortage_h'], action)

else:
    raise ValueError('Error')

if index+1 > episode.index[-1]:
    pass
else:
    episode.loc[index+1, 'Tank'] = tank_next

return episode

#########################################################################

def expert_perform(self, start):
    episode = self.HISTORY[start:start+self.EPISODE_LEN].copy()
    print(episode.loc[start, 'DateTime'])
    ## record how much PV-generated electricity remains
    ## that can be exported at time t
    episode['Export_e'] = episode['PVSurplus_h'] / self.HP_COP
    ## record how much shortage of electricity remains
    ## that we need to import for it at time t
    episode['Import_e'] = episode['Shortage_h'] / self.HP_COP
    ## record how much electricity imported to charge
    ## the tank at time t
    episode['Import_forTank'] = episode['GridCharge_h'] / self.HP_COP
```

Chih-Hsiang Lee - March 2019 73
```python
def vanilla_perform(self, start):
    episode = self.HISTORY[start:start+simulator.EPISODE_LEN].copy()
    episode['Tank'] = 0.0
    episode['HP'] = self.HP_CAPACITY - episode['HeatDemand']
    episode['PVOutput_h'] = episode['PVSurplus_h'] - episode['HeatDemand']
    updated = episode[episode['PVSurplus_h'] < 0].copy()
    updated['PVSurplus_h'] = 0.0
    episode.update(updated)
    del updated
    episode['Shortage_h'] = episode['HeatDemand'] - episode['PVOutput_h']
    updated = episode[episode['Shortage_h'] < 0].copy()
    updated['Shortage_h'] = 0.0
    episode.update(updated)
    del updated
    episode['GridCharge_h'] = 0.0
    episode['PVCharge_h'] = 0.0
    for i in range(self.EPISODE_LEN):
        index = i + start
        episode = self.step_vanilla(episode, index)

    # record how much PV-generated electricity remains
    # that can be exported at time t
    episode['Export_e'] = episode['PVSurplus_h'] / self.HP_COP
    # record how much shortage of electricity remains
    # that we need to import for it at time t
    episode['Import_e'] = episode['Shortage_h'] / self.HP_COP
    # record how much electricity imported to charge
    # the tank at time t
    episode['Import_forTank'] = episode['GridCharge_h'] / self.HP_COP
    Export = -episode['SSP'] * episode['Export_e']
    Import = episode['SBP'] * episode['Import_e']
    Import_charge = episode['SBP'] * episode['Import_forTank']

    curve_vanilla = episode['Tank'].values
    cost_vanilla = \
        Export.values.sum() + Import.values.sum() + Import_charge.values.sum()
    return curve_vanilla, cost_vanilla
```

---

Chapter 8 Appendices

74
Chih-Hisang Lee - March 2019
```python
Chapter 8 Appendices

tank_currr = episode.loc[index, 'Tank']

if index % self.TIME_INTERVAL_PER_DAY < self.VANI_CHARGE:
    action = 0.0
elif index % self.TIME_INTERVAL_PER_DAY < self.VANI_DISCHARGE:
    action = self.VANI_MAX_TARGET - tank_currr
    if action > episode.loc[index, 'HP']:
        action = episode.loc[index, 'HP']
    else:
        target = tank_currr - self.MAX_DISCHARGE_ABS
        if target < 0:
            target = 0
        action = target - tank_currr

## charge
if action >= 0 and episode.loc[index, 'PVSurplus_h'] > 0:

    action, episode.loc[index, 'PVSurplus_h'], episode.loc[index, \
    'PVCharge_h'] = self._provide(action,
    episode.loc[index, 'PVSurplus_h'])
    tank_next = \
    tank_currr * self.TANK_LOSS_PER_T + episode.loc[index,'PVCharge_h']

elif action <= 0 and episode.loc[index, 'Shortage_h'] >= 0:
    tank_next = tank_currr * self.TANK_LOSS_PER_T #+ action (Disabled line)
    #episode.loc[index, 'GridCharge_h'] = action (Disabled line)
    ## Disable two lines above in order to avoid
    ## incorrect prediction that costs extra expenditure

## discharge
elif action < 0 and episode.loc[index, 'PVSurplus_h'] > 0:
    action = abs(action)
    if action > self.MAX_DISCHARGE_ABS:
        action = self.MAX_DISCHARGE_ABS
    tank_next = (tank_currr - action) * self.TANK_LOSS_PER_T

elif action < 0 and episode.loc[index, 'Shortage_h'] >= 0:
    action = abs(action)
    if action > self.MAX_DISCHARGE_ABS:
        action = self.MAX_DISCHARGE_ABS
    tank_next = (tank_currr - action) * self.TANK_LOSS_PER_T
    episode.loc[index, 'Shortage_h'], _, _ = \
    self._provide(episode.loc[index, 'Shortage_h'], action)

else:
    raise ValueError('Error')

if index+1 > episode.index[-1]:
    pass
else:
    episode.loc[index+1, 'Tank'] = tank_next
```

Chih-Hsiang Lee - March 2019
```python
return episode
```
```python
def no_tank_perform(self, start):
    episode = self.HISTORY[start:start+self.EPISODE_LEN].copy()
    ## record how much PV-generated electricity remains
    episode['Export_e'] = \n        (episode['PWOutput_h'] - episode['HeatDemand'])/ self.HP_COP
    pick_positive_export = episode[episode['Export_e'] > 0].copy()
    ## record how much shortage of electricity remains
    ## that we need to import for it at time t
    episode['Import_e'] = episode['Export_e'] * -1
    pick_positive_import = episode[episode['Import_e'] > 0].copy()
    Export = -pick_positive_export['SSP'] * pick_positive_export['Export_e']
    Import = pick_positive_import['SSP'] * pick_positive_import['Import_e']
    cost_no_tank = Export.values.sum() + Import.values.sum()
    return cost_no_tank
```
```python
expert_history = 'Historical_Data_expert_done.pkl'
model = load_model('lstm_model_Proposed_Model.h5')
forecaster_for_simu = FORECASTER()
simulator = Simulator(model, forecaster_for_simu, expert_history)
simulator.set_cutoff(start_from=simulator.DECISION_TIME)
simulator.run(1918*48, figure=True, network=True,
              forecast=True, vanilla=True, no_tank=False, day=1)