applied sciences

MDPI

Article

Application of Deep Learning Techniques for the State of Charge Prediction of Lithium-Ion Batteries

Sang-Bum Kim [1,†] and Sang-Hyun Lee [2,*,†]

[1] Department of Robotdrone Engineering, Honam University, Gwangsan-gu, Gwangju 62399, Republic of Korea; 2021115@honam.ac.kr
[2] Department of Computer Engineering, Honam University, Gwangsan-gu, Gwangju 62399, Republic of Korea
* Correspondence: leesang64@honam.ac.kr
† These authors contributed equally to this work.

Abstract: This study proposes a deep learning-based long short-term memory (LSTM) model to predict the state of charge (SOC) of lithium-ion batteries. The purpose of the research is to accurately model the complex nonlinear behavior that occurs during the charging and discharging processes of batteries to predict the SOC. The LSTM model was trained using battery data collected under various temperature and load conditions. To evaluate the performance of the artificial intelligence model, measurement data from the CS2 lithium-ion battery provided by the University of Maryland College of Engineering was utilized. The LSTM model excels in learning long-term dependencies from sequence data, effectively modeling temporal patterns in battery data. The study trained the LSTM model based on battery data collected from various charge and discharge cycles and evaluated the model's performance by epoch to determine the optimal configuration. The proposed model demonstrated high SOC estimation accuracy for various charging and discharging profiles. As training progressed, the model's predictive performance improved, with the predicted SOC moving from 14.8400% at epoch 10 to 12.4968% at epoch 60, approaching the actual SOC value of 13.5441%. Simultaneously, the mean absolute error (MAE) and root mean squared error (RMSE) decreased from 0.9185% and 1.3009% at epoch 10 to 0.2333% and 0.5682% at epoch 60, respectively, indicating continuous improvement in predictive performance. In conclusion, this study demonstrates the effectiveness of the LSTM model for predicting the SOC of lithium-ion batteries and its potential to enhance the performance of battery management systems.

Keywords: long short-term memory; lithium ion battery; state of charge estimation; MAE; RMSE

Citation: Kim, S.-B.; Lee, S.-H. Application of Deep Learning Techniques for the State of Charge Prediction of Lithium-Ion Batteries. *Appl. Sci.* **2024**, *14*, 8077. https://doi.org/10.3390/app14178077

Academic Editors: Gerard Ghibaudo and Frede Blaabjerg

Received: 31 July 2024
Revised: 6 September 2024
Accepted: 7 September 2024
Published: 9 September 2024

1. Introduction

Since 2020, major countries, including the United States and European countries, have introduced environmental regulations to reduce greenhouse gas emissions by 20–30%, which has promoted the use of renewable energy and led to rapid growth in the battery market [1]. Batteries play a crucial role in the smart grid and electric vehicle industries, and lithium-ion batteries are widely adopted as replacements for lead-acid batteries due to their high energy density and long lifespan [2].

In particular, batteries for electric vehicles require high capacity and operate in complex environments, making a systematic battery management system essential for safe and reliable management [3–5]. The state of charge (SOC) is a critical indicator in determining the condition of a battery, defined as the ratio of the remaining charge to the rated capacity of the battery in operation [6]. However, due to the battery's nonlinearity and electrochemical reactions, SOC cannot be measured directly, which has led to the study of various estimation methods.

The traditional SOC estimation method, known as the Coulomb counting method, is simple and effective, but it is influenced by factors such as initial value, current sensor error, and battery degradation [7]. The Kalman filter (KF) and its extended form, the

extended Kalman filter (EKF), enable nonlinear system estimation for linear time-varying systems, but they have disadvantages such as implementation complexity and the need to consider many parameters during modeling [8–10]. To address these challenges, various technologies, including artificial intelligence algorithms, have been proposed, and recent advances in artificial intelligence and computer equipment, coupled with the explosive increase in data, have heightened interest in AI [11].

AI-based methods for SOC estimation include artificial neural networks (ANN), support vector machines (SVM), fuzzy algorithms, and recurrent neural networks (RNN), which provide high accuracy across various current ranges [12,13]. However, since the SOC of a battery is calculated as the accumulated amount of current, it is necessary to include sequence information in the estimation network. RNNs can learn data that change over sequences through state variables, and with the introduction of memory cells such as LSTM, it has become possible to learn large-scale data over long periods. Recently, studies have also been conducted to estimate SOC in combination with the convolutional neural network (CNN) method [14,15].

Recently, various AI models have been applied in the field of SOC estimation for lithium-ion batteries, and performance comparisons and improvements of different approaches have been actively conducted. For instance, traditional filter-based models like the Kalman filter (KF) and extended Kalman filter (EKF) have been widely used for SOC estimation; however, these models face limitations in handling nonlinear data and complexity in implementation. Additionally, studies have been conducted to evaluate the accuracy and efficiency of SOC estimation using various machine learning algorithms, providing a broad understanding of SOC estimation model performance optimization, but often lacking a focus on the detailed performance enhancement of specific models [16]. On the other hand, other AI-based models, such as support vector machines (SVM), have shown excellent performance in handling nonlinear characteristics but are limited in their ability to learn from sequence data [17].

In contrast, this study proposes a method for predicting the SOC of lithium-ion batteries using a long short-term memory (LSTM) model. LSTM can learn long-term dependencies from sequence data, effectively modeling the complex nonlinear charging and discharging patterns of batteries. By comparing with existing studies, this paper aims to emphasize the superior prediction performance of LSTM and its potential to enhance battery management systems (BMS). This approach suggests the possibility of overcoming the limitations of traditional filter-based models and achieving high SOC prediction accuracy under various conditions.

Artificial intelligence has become a critical issue globally, and SOC prediction using deep learning can handle the nonlinear characteristics of batteries in real time, demonstrating high adaptability to various battery types [18,19]. In this study, we propose a method to estimate SOC using an RNN network with LSTM memory cells. For this, we downloaded and utilized measured data from the CS_2 lithium-ion battery provided by the University of Maryland School of Engineering, and we obtained training data through experiments on the structure and operation of the LSTM and the electrical discharge capacity. The collected data consisted of a total of four cells: CS_35, CS_36, CS_37, and CS_38.

The structure of this paper is as follows. Section 2 describes the artificial intelligence model and structure proposed in this study, as well as the composition and research methods of datasets for four types of battery cells. Section 3 details the design and structure of the LSTM model used in this paper, while Section 4 presents the experimental results. Finally, the conclusion of this paper is discussed in Section 5.

2. Research Model

In this study, a preprocessing step was conducted to prepare the training data for the LSTM model. This step is crucial for enabling long-term learning, which is essential in time series data applications. The LSTM network structure for SOC estimation is illustrated in

Figure 1. The training dataset consisted of input data and the corresponding measured SOC values.

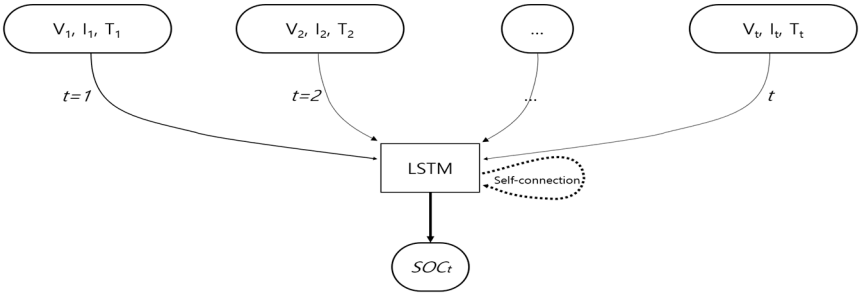


Figure 1. LSTM network structure for SOC estimation.

For SOC estimation using an LSTM network, the input data consists of voltage V_t, current I_t, and temperature T_t at time step t. This causes the network to output SOC_t, an estimate of SOC. This process can be expressed as a formula as follows.

$$\text{Input: } (V_t, I_t, T_t), \text{ Output: } SOC_t, \tag{1}$$

A simple mathematical representation of how an LSTM network works is that the input (V_t, I_t, T_t) at each time step t is processed through an LSTM cell to calculate the SOC estimate SOC_t at that time step. LSTM cells have the ability to estimate SOC for new inputs while remembering information from previous time steps. This process can be expressed as a formula as shown in Equation (2) below.

$$LSTM((V_t, I_t, T_t)) \rightarrow SOC_t, \tag{2}$$

This process is performed iteratively, resulting in an estimate of SOC for time series data. The detailed internal mechanisms of an LSTM network can be expressed in complex equations.

Figure 2 is a schematic diagram of the inside of a forward-propagating LSTM cell, showing the main components of an LSTM cell: the input gate, forget gate, and output gate. Each gate controls the flow of information, and the cell state C_t is responsible for long-term memory. The input x_t and the previous hidden state h_{t-1} combine information through gates and determine which information to remember, delete, and output. The input of the forward propagation LSTM cell is the current data and the state of the previous time, and the state vector is obtained through each gate. The delete gate determines whether to keep or delete past information and has a value between 0 and 1, which is the range of the sigmoid function. Here, the closer it is to 1, the more past information is preserved. The input gate determines the information to be remembered for the current step, and like the delete gate, if it is 1, all information currently entered as an input is remembered. After passing through the LSTM cell, when it reaches the output layer, the estimated SOC is obtained using the state vector and bias. Here, the working principle of the LSTM cell for Figure 2 can be expressed by a formula. The main formula is as follows.

The forget gate determines what information to discard from the cell state at a previous point in time. Here, ft in Equation (3) is the activation value of the forget gate at the current time t, which determines which cell state information to keep and discard. Σ is a sigmoid activation function that converts to a value between 0 and 1 to control the amount of information that passes through the forget gate. W_f is the weight of the forget gate, which is adjusted during the learning process. $[h_{t-1}, x_t]$ is a vector connecting the hidden state h_{t-1} at the previous time and the input x_t at the current time. b_f is the bias of the forget gate, which is adjusted during the learning process.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3}$$

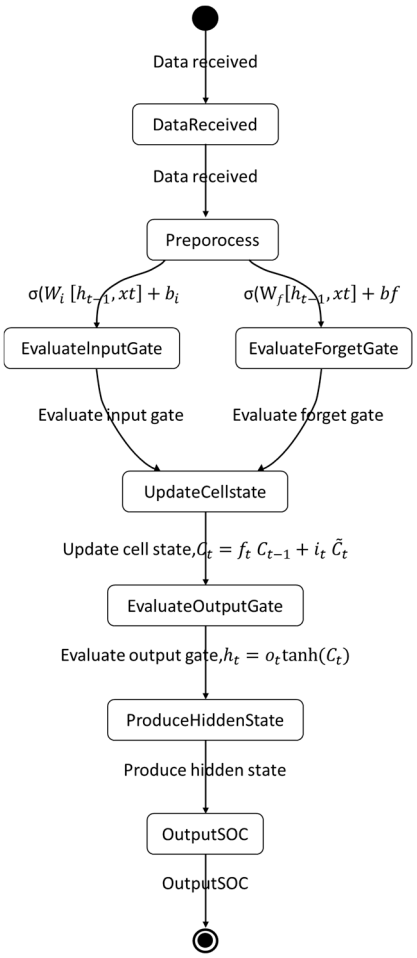


Figure 2. Flow diagram inside a forward-flowing LSTM cell.

The input gate determines whether new information will be stored in the cell state. Here, in Equation (4) it is the activation value of the input gate at current time t, which determines how important the new input is to the cell state. W_i is the weight of the input gate. b_i is the bias of the input gate. $\tilde{C}_t$ is the cell state candidate value at the current time t, which determines how much new information is reflected in the cell state. W_C is the weight for generating cell state candidates. b_C is the bias for generating cell state candidates.

$$i_t = \sigma(W_i \cdot [h_t - 1, x_t] + b_i)\ \tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{4}$$

The cell state update calculates each new cell state. Here, C_t in Equation (5) is the cell state at the current time t, which is responsible for the long-term memory of LSTM. C_{t-1} is the cell state at the previous time point $t-1$ and represents past information.

$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t \tag{5}$$

The output gate determines which part of the cell state to output, which will be used to determine the next hidden state. Here, o_t in Equation (6) is the activation value of the output gate at the current time t, which determines which information to output among the cell states. W_o is the weight of the output gate. b_o is the bias of the output gate. H_t is the hidden state at the current time t and is the output value passed to the next time point.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)\ h_t = o_t \times \tanh(C_t) \tag{6}$$

Here, W and b represent the weights and biases being learned. σ is the sigmoid activation function, and tanh is the hyperbolic tangent activation function. $[h_{t-1}, x_t]$ means concatenating the previous hidden state and the current input.

These inputs play an important role in determining the operation of each gate and the updating of the cell state. Weights and biases are optimized through learning, and the sigmoid function and hyperbolic tangent function provide nonlinear transformations, allowing complex patterns to be learned.

At the last stage of each forward propagation, a loss function is calculated to minimize the error between the estimated value and the measured value, and the mean square error (MSE) is used. The loss function is as follows in Equation (7) [20].

$$Loss = \frac{1}{n}\sum_{t=1}^{n}(SOC_t - SOC_t^*)^2 \quad (7)$$

SOC_t is the measured value of step t, and is a value estimated by the SOC_t^* network. n represents the total length of data. The network weights are updated using the difference between the measured value and the estimated value using the backpropagation technique. This is performed through Adam, an optimization technique. Adam is a method that combines the advantages of AdaGrad [21] and RMSProp [22] and is characterized by simple implementation and high computational efficiency.

SOC represents the amount of charge stored in the battery and is calculated by the battery management system as it is expressed as a percentage or a range from 0% to 100%.

The formula for calculating the state of charge (SOC) through the current integration method is as follows [21,22],

$$SOC_t\ (\%) = SOC_0 - \left(\frac{\int_0^t idt}{C_n}\right) \times 100 \quad (8)$$

where SOC_t is the measured SOC at time step ttt, and SOC_0 is the initial SOC value. In this study, SOC_0 was estimated based on a lookup table derived from the results of the OCV experiment conducted with a SOC 5% pulse. Figure 3 presents graphs showing the relationship between open circuit voltage (OCV) and the state of charge (SOC) for four different battery cells (cell IDs 35, 36, 37, and 38). Each graph illustrates the relationship between the state of charge of a battery cell, expressed as a percentage of SOC, and its voltage, measured as OCV in volts.

The graphs in Figure 3 should ideally demonstrate that OCV increases as SOC increases, reflecting that when a battery is fully charged (SOC 100%), the OCV should be high. Any observed discrepancies or irregularities might be attributed to measurement errors or specific characteristics of the battery cells. Therefore, it is essential to account for these variations accurately when developing models for SOC estimation to ensure the correct relationship is captured.

Additionally, the initial SOC value (SOC_0) estimation relies heavily on accurate OCV measurements and the conditions under which the OCV experiment was conducted, including factors such as temperature and charge/discharge rates. These factors significantly influence the precision of SOC estimation and should be carefully controlled during the modeling process. However, in some intervals, there may be data irregularities, which may be due to errors in the measurement process or the characteristics of specific battery cells. All graphs follow a similar pattern, but there may be subtle differences depending on the characteristics of the battery cells.

Here, C_n is the rated capacity and i is the current at each time step. C_n is a variable value depending on the actual degree of battery aging, but since this paper uses short-cycle data, it is calculated assuming a constant capacity.

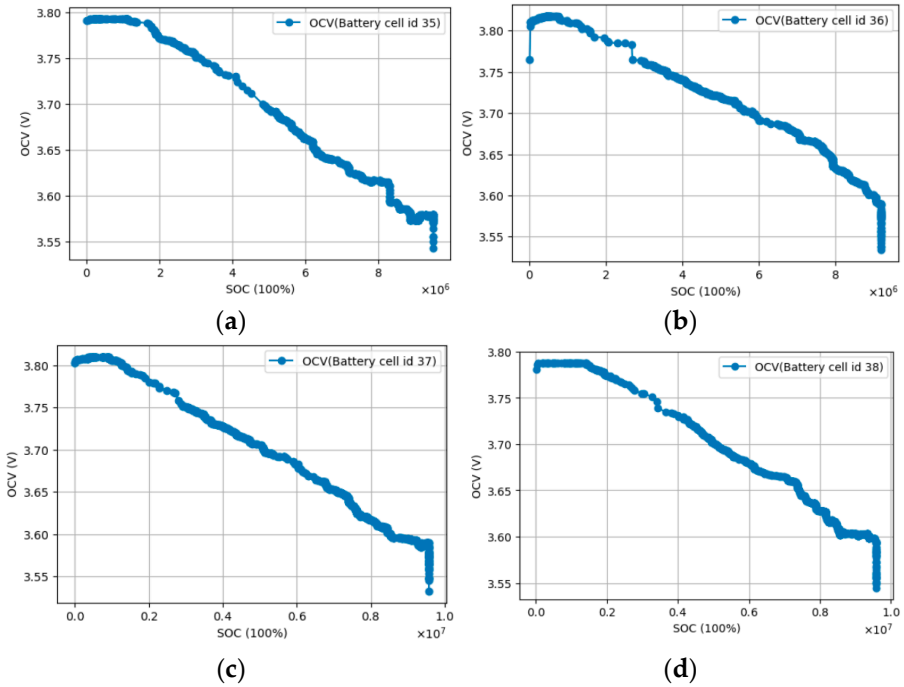


Figure 3. Comparison between SOC and OCV in (**a**–**d**) for CS2 battery cells 35, 36, 37, and 38, respectively.

In this study, we proceeded with the process shown in Figure 4 to train an LSTM model for data preprocessing, model learning, and performance evaluation, predict battery capacity using test data, and evaluated the model's performance using performance indicators. We evaluated prediction performance to determine which model produced the most accurate and reliable SOC estimates.

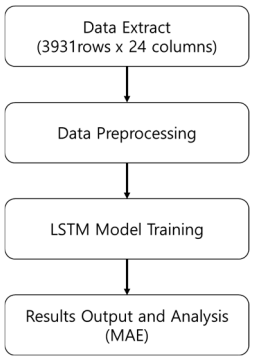


Figure 4. Process of model training and performance evaluation.

3. Structural Design of LSTM Model

This section provides a detailed description of the characteristics of the battery and covers data preprocessing and implementation methods for the LSTM network model. First, we compare the impact of the shape of the model input, and then compare and analyze the learning ability and performance according to the LSTM structure and model parameter settings.

In order to write the internal structure of the LSTM cell shown in Figure 5 in Section 2 of this paper as a Python program, a library such as TensorFlow or keras is used to perform calculations that update each gate and cell state.

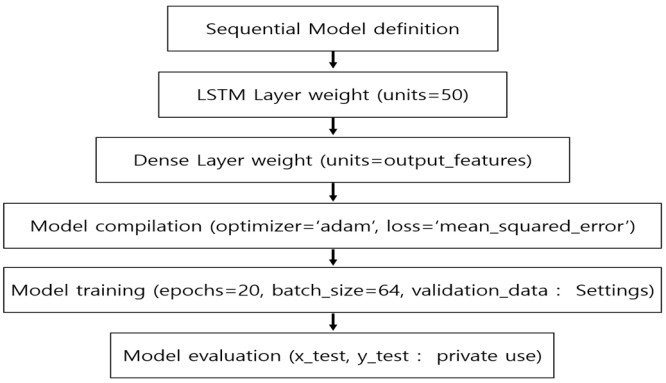


Figure 5. Flowchart for LSTM structure design.

Figure 4 is a flow chart showing the process of constructing an LSTM model using the Python code and keras library.

The LSTM model accepts input data, processes it through LSTM cells, and finally produces output data. Inside an LSTM cell, information is processed through several gates. Since the basic structure of the LSTM model was designed by substituting the equations in Section 2, the pseudocode in Figure 3 also shows the structure of the LSTM model.

The pseudocode in Algorithm 1 defines the process of calculating each gate and cell state of an LSTM cell as a function. Sigmoid and hyperbolic tangent stand for the sigmoid activation function and hyperbolic tangent activation function, respectively, which combine weights and inputs to output values between 0 and 1, or −1 and 1.

Algorithm 1: Pseudocode for LSTM structure design algorithm

Function LSTM_Cell(input x_t, previous_hidden_state h_{t-1}, previous_cell_state C_{t-1}):
    # Calculate the forget gate
    f_t = Sigmoid(forget_weights W_f * [h_{t-1}, x_t] + forget_bias b_f)
    # Calculate the input gate
    i_t = Sigmoid(input_weights W_i * [h_{t-1}, x_t] + input_bias b_i)
    $\tilde{C}_t$ = HyperbolicTangent(cell_state_weights W_C * [h_{t-1}, x_t] + cell_state_bias b_C)
    # Update the cell state
    $C_t = f_t * C_{-1} + i_t * \tilde{C}_t$
    # Calculate the output gate
    o_t = Sigmoid(output_weights W_o * [h_{-1}, x_t] + output_bias b_o)
    $h_t = o_t$ * HyperbolicTangent(C_t)
    return h_t, C_t
Weights and biases are optimized during the model training process.
This pseudocode represents the operations within an LSTM cell at a single time step.
These operations are repeated sequentially over the entire sequence.

[h_{t-1}, x_t] means a vector that concatenates the previous hidden state and the current input. * indicates element-wise multiplication, and in actual implementation matrix operations are used.

These operations are designed to enable LSTM networks to learn long-term dependencies in sequence data. Deep learning libraries abstract these complex calculations from the user, making it easier to construct and experiment with models at a higher level.

Algorithm 1 shows Python code for constructing an LSTM model using the keras library. Each part of the code is directly related to the internal workings of the LSTM cell.

As an explanation of Algorithm 2, where input_time_steps refers to the time steps of the input data, input_features refers to the number of features in the input data, and output_features refers to the number of features in the output data. x_train, y_train, x_val, y_val, x_test, and y_test represent training, validation, and test data and labels, respectively. Rather than directly implementing the actual LSTM internal calculations

and gate update logic, this code demonstrates how to leverage the LSTM layer provided by TensorFlow to create a model that processes sequence data. The loss function and optimization algorithm used when compiling the model are used to learn the weights and biases inside the LSTM cells.

Algorithm 2: Structural design algorithm for LSTMs using pseudocode

```
# Define the model.
model = Sequential()
# Add an LSTM layer. Here, the number of units refers to the number of neurons.
model.add(LSTM(units=50, input_shape=(input_time_steps, input_features)))
# Add a dense layer for final output.
model.add(Dense(units=output_features))
# Compile the model. We use MSE as the loss function.
model.compile(optimizer='adam', loss='mean_squared_error')
# Train the model.
model.fit(x_train, y_train, epochs=20, batch_size=64, validation_data=(x_val, y_val))
# Evaluate the model.
model.evaluate(x_test, y_test)
```

A more detailed explanation of the source code in Algorithm 2 is as follows. LSTM (units=50, input_shape=(input_time_steps, input_features)) is the part that adds an LSTM layer to the model, where units=50 means the output dimension of the cell; that is, the size of the hidden state h_t. input_shape defines the shape of the input data expected by the model. Where input_time_steps is the length of the sequence (number of time steps), and input_features is the number of features (data dimensions) at each time step.

The model.add(Dense(units=output_features)) code is the last layer of the model, adding a dense layer (dense layer) that uses the output h_t of the LSTM cell to calculate the final output, and output_features is the final layer of the model, which indicates the number of dimensions. model.compile(optimizer='adam', loss='mean_squared_error') is the part that compiles the model, where optimizer='adam' uses the Adam optimization algorithm, and loss='mean_squared_error' specifies the MSE as the loss function. This corresponds to Equation (7) in Section 2. The code of model.fit(x_train, y_train, epochs=20, batch_size=64, validation_data=(x_val, y_val)) is the part that trains the model, where x_train and y_train are the training data and the corresponding target value, respectively. epochs=20 means learn by repeating the entire training data 20 times, and batch_size=64 indicates the number of data samples delivered to the network at one time. validation_data=(x_val, y_val) sets data to verify the model's performance during the training process. mod-el.evaluate(x_test, y_test) evaluates the performance of a trained model using test data x_test and label y_test.4.

4. Implementation and Results

The development environment for this study was constructed. In this study, the software environment for the experiment was Python version 3.10, the artificial intelligence library used was the PyTorch-based MMDetection API, and the hardware environment was Windows 10 for OS, with an i9-9900 k for CPU, 128 GB of RAM, and 128 GB for the GPU. NVIDIA RTX 6000 was used.

4.1. DataSet

The experiments on the CS2 lithium-ion batteries used in this paper were conducted in the environment shown in Table 1. All CS2 cells underwent the same charging profile, a standard constant current/constant voltage protocol with a constant current rate of 0.5 C until the voltage reached 4.2 V, followed by 4.2 V until the charge current dropped below 0.05 A. Unless specified, the discharge cutoff voltage for these batteries was 2.7 V. All CS2 cells were randomly numbered and named accordingly. The nth CS2 cell is given the name 'CS2_n' [23].

Table 1. Specifications of CS2 lithium-ion batteries.

Battery (Parameters)	Specifications (Value)
Capacity Rating	1100 mAh
Cell Chemistry	LiCoO2 cathode, EDS results also showed trace elements of Manganese
Weight (w/o safety circuit)	21.1 g

Table 1 utilizes the dataset from the measurements of CS2 lithium-ion batteries provided by the College of Engineering at the University of Maryland to evaluate the AI model. The Excel data measured for the CS_2 lithium-ion batteries were composed of four cells: CS35, CS36, CS37, and CS38, as shown in Figure 6, and these were consolidated into a single dataset. Additionally, columns for the voltage generated using calculated values for each cell [3.7, 3.8, 3.9, 4.0, 4.1, 4.2], SOC, OCV, ID, Date_Time, Cycle_Index, Charge_Capacity (Ah), and Discharge_Capacity (Ah) were added.

	(3.7, 3.8)	(3.7, 3.9)	(3.7, 4.0)	(3.7, 4.1)	(3.7, 4.2)	(3.8, 3.9)	(3.8, 4.0)	(3.8, 4.1)	(3.8, 4.2)	(3.9, 4.0)	...	(4.0, 4.1)	(4.0, 4.2)	(4.1, 4.2)	OCV	Charge_Capacity(Ah)	Discharge_Capacity(Ah)	SOC	ID	Date_Time	Cycle
0	630.318828	2221.123148	4292.154768	5552.792227	6424.641171	1590.804319	3631.830462	4862.440358	5719.476953	1920.976540	...	1230.609896	2087.646491	857.036595	3.494803	2.150882	1.137092	1.40e+07	35.000000	2010-08-19 17:59:42	1.000000
1	690.350301	2221.124363	4292.154768	5552.792227	6424.641171	1590.805052	3631.830462	4862.440358	5719.476953	1950.990479	...	1230.622600	2096.910698	866.285848	3.496422	3.291456	2.268441	1.40e+07	35.000000	2010-08-19 21:35:39	2.000000
2	720.366060	2221.124363	4292.154768	5552.792227	6424.641171	1590.805052	3631.830462	4862.440358	5719.476953	2011.001505	...	1230.622600	2103.515569	871.848944	3.497231	4.422278	3.397807	1.40e+07	35.000000	2010-08-20 01:09:04	3.000000
3	630.319311	2221.124363	4232.141792	5462.764392	6339.892086	1590.805052	3601.822481	4832.445081	5709.572775	1981.001672	...	1230.622600	2103.515569	872.893141	3.522809	5.536041	4.521028	1.38e+07	35.000000	2010-08-20 04:41:25	4.000000
4	600.303530	2221.123211	4202.124883	5432.749840	6307.437369	1590.804017	3601.790381	4832.412809	5705.305950	2011.001341	...	1230.622428	2103.515569	872.893141	3.556481	6.646490	5.632064	1.31e+07	35.000000	2010-08-20 08:13:30	5.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3926	0.000000	0.000000	90.045232	360.181250	909.984363	0.000000	90.045232	360.181250	909.984363	90.045232	...	300.151055	819.939131	535.022114	3.952868	16.162456	16.064774	-6.58e+05	38.000000	2011-02-03 13:41:24	1050.000000
3927	0.000000	0.000000	90.043387	360.181242	898.248338	0.000000	90.043387	360.181242	898.248338	90.043387	...	270.136018	814.923551	538.068994	3.954650	16.456239	16.355996	-8.07e+05	38.000000	2011-02-03 15:09:14	1051.000000
3928	0.000000	0.000000	90.043387	360.181242	898.248338	0.000000	90.043387	360.181242	898.248338	90.043387	...	270.136018	814.923551	538.068994	3.954650	16.456239	16.355996	-8.07e+05	38.000000	2011-02-03 15:09:14	1052.000000
3929	0.000000	0.000000	60.030187	360.179344	874.953738	0.000000	60.030187	360.179344	874.953738	60.030187	...	270.135957	808.204951	514.772496	3.954650	16.751501	16.651220	-7.03e+05	38.000000	2011-02-03 16:36:06	1053.000000
3930	0.000000	0.000000	60.030187	360.179344	874.953738	0.000000	60.030187	360.179344	874.953738	60.030187	...	270.135957	808.204951	514.772496	3.954650	16.751501	16.651220	-7.03e+05	38.000000	2011-02-03 16:36:06	1054.000000

3931 rows × 22 columns

Figure 6. Format of the dataset.

Table 2 below shows four data item numbers and added columns: CS_35, CS_36, CS_37, and CS_38. This table shows the change in actual state of charge (SOC) values and Validation Loss over each epoch during the training of the LSTM model. An epoch means that the entire training dataset was fed to the model once, and this table represents data from epochs 10 through 60. In particular, the model seems to be performing best at Epoch 60, with the lowest validation loss (0.3192%). Overall, both the true SOC value and the verification loss tend to decrease as the epoch increases, showing that the model's prediction performance is gradually improving with training.

Table 2. Types of battery data.

Decision	Battery Type	Number of Data	Training Data	Test Data
Battery Dataset	CS35	911	3146	787
	CS36	950		
	CS37	1016		
	CS38	1056		
Total	4 Cell	3933		

Figure 7 shows the training loss and validation loss according to the epochs for the generalized dataset. Both losses declined rapidly initially, and then remained at a stable low level.

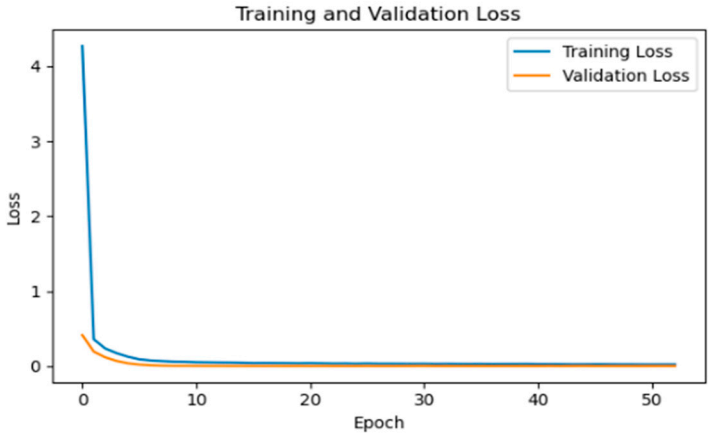


Figure 7. Graph of training and validation loss.

Table 3 and Figure 7 show the training loss and validation loss of epoch 10, 20, 30, 40, 50, and 60 in the training process of the LSTM model. Loss reduction, both training and validation losses, tend to decrease consistently as the epoch progresses. This means that the model is learning from your data, and its performance is gradually improving. There was no significant difference between the loss of training and the loss of validation. This indicates that overfitting was fairly well managed. If there is overfitting, the training loss will continue to decrease, but the verification loss will tend to increase. Performance gains went from epoch 10 to 60, while training loss decreased from 2.5654 to 1.1306 and validation loss decreased from 1.5072 to 0.3192. This suggests that the model was increasingly making accurate predictions. Optimal model selection had the lowest validation loss at epoch 60, so it can be assumed that the model had the best predictive performance at this point. However, it is necessary to consider the point at which the decline in losses will moderate, and decide at what point to proceed further with the learning or stop early.

Table 3. Actual SOC values versus verification losses over epochs.

Epoch	Actual SOC (%)	Validation Loss (%)
Epoch 10	2.5654	1.5072
Epoch 20	1.6975	0.7171
Epoch 30	1.4155	0.4085
Epoch 40	1.3254	0.3496
Epoch 50	1.1692	0.3253
Epoch 60	1.1306	0.3192

In conclusion, the performance of the model was improving as it learned, and it seemed that it was learning stably without overfitting. The model from epoch 60 showed the best performance.

4.2. LSTM Model for SOC Inference

RMSE (root mean square error) and MAE were used as indicators to check the performance of the LSTM model used in this paper. Here, RMSE was used to determine how accurate the model's prediction was. The lower the value, the higher the model's prediction performance. The calculation formula is as shown in Equation (9) below.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(SOC_i - SOC_i^*)^2} \quad (9)$$

Here, n is the total number of data points, and SOC_i is the SOC value of the ith actual observed data point. SOC_i^* is the SOC value of the ith data point predicted by the model. For each observation, calculate the difference between the actual value SOC_i and

the predicted value SOC_i^*. This difference is squared to avoid negative values. Because the square emphasizes the size of the error, larger errors are given greater weight, and all calculated squared errors are added together. Calculate the mean square error by dividing that sum by the number of data points n. Finally, we take the square root of this average value to obtain the RMSE value. The RMSE can be directly compared to actual values with the same units, helping to intuitively understand the size of the model's prediction error. MAE is a measure of mean absolute error, which is the average of the absolute differences between predicted and actual values. MAE is an indicator of how incorrectly the size of the error was predicted on average, and its calculation formula is as shown in Equation (10) below.

$$MAE = \frac{1}{n}\sum_{n-1}^{n}|SOC_i - SOC_i^*| \quad (10)$$

where n is the total number of data points. SOC_i is the SOC value of the ith actual observed data point. SOC_i^* is the SOC value of the ith data point predicted by the model. For each observation, calculate the difference between the actual value SOC_i and the predicted value SOC_i^*. Here, we take the absolute value of each difference and only consider the magnitude, regardless of the direction (positive or negative) of the error. After adding up all of these absolute errors, divide the sum by the number of data points n to obtain the average. MAE represents the average size of the error, and like RMSE, the lower the value, the better the model's prediction performance.

4.3. Results of Applying the LSTM Model for SOC Inference

In this paper, model parameters are parameters necessary for model construction or training process and include layers, units, batch sizes, forks, etc. In this chapter, we will compare and analyze the actual SOC and predicted SOC according to epochs 10, 20, 30, 40, 50, and 60 to estimate the appropriate SOC value from the dynamic profile. All model parameters except the parameters selected here are set the same.

Looking at Table 4 and Figure 8 together, we see that the model tends to predict closer to the actual SOC value as the epoch progresses, but tends to slightly underestimate or overestimate the SOC value in certain epochs. Epoch 60 shows the most accurate predictions, suggesting that the model is gradually optimizing with sufficient training. However, the predicted values remained very close to the actual values in all epochs, indicating that the model's performance was stable.

Table 4. Comparative analysis of actual and predicted SOC values according to exposition.

Epoch	Actual SOC (%)	Predicted SOC (%)
Epoch 10	13.5441	14.8400
Epoch 20	13.5441	12.6617
Epoch 30	13.5441	12.4947
Epoch 40	13.5441	12.6203
Epoch 50	13.5441	12.5185
Epoch 60	13.5441	12.4968

This study proposes a deep learning-based approach using an LSTM model to predict the state of charge (SOC) of lithium-ion batteries. The LSTM model is well-suited for modeling the complex, nonlinear patterns that occur during the charging and discharging processes of batteries due to its ability to learn long-term dependencies from sequential data.

The data used in this experiment was obtained from the University of Maryland, consisting of measurements from CS2 lithium-ion batteries collected under various temperature and load conditions. The input variables for training the LSTM model included voltage (V_t) and current (I_t) at each time step.

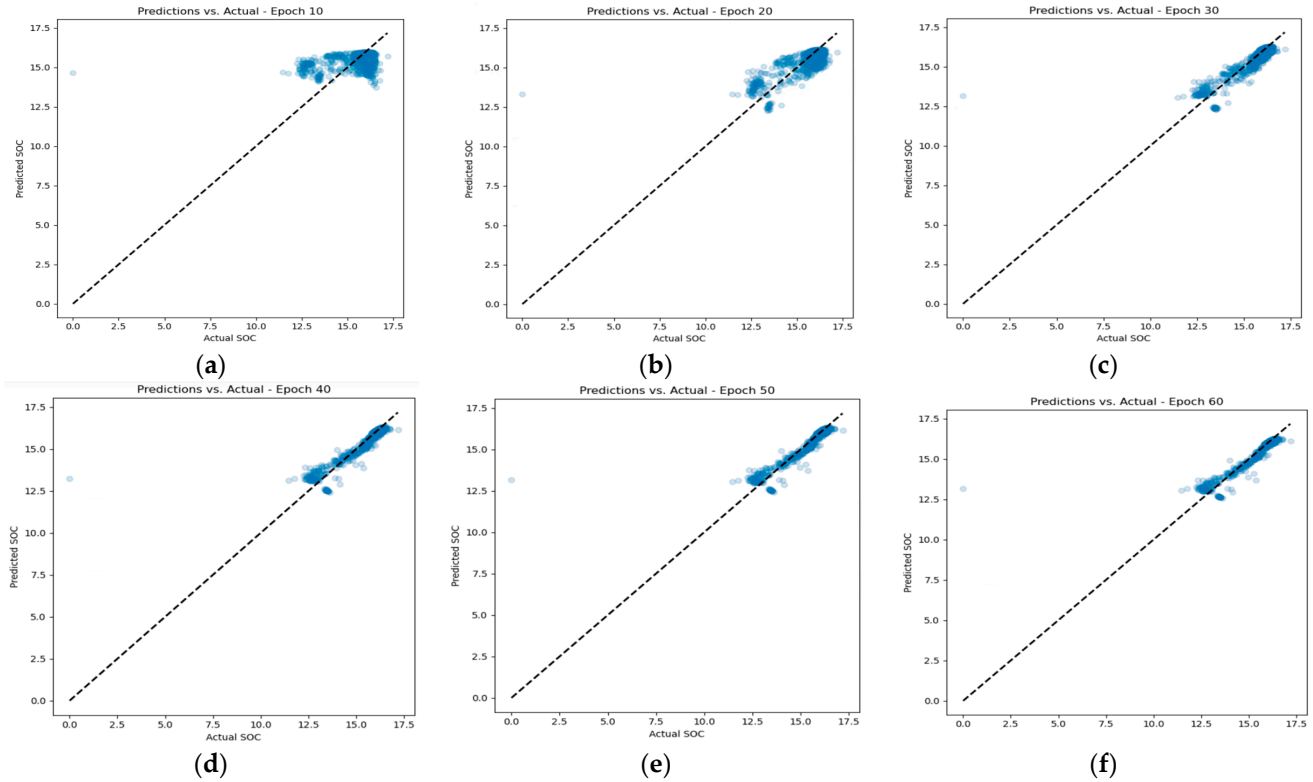


Figure 8. Actual and predicted battery SOC values for (**a**) epoch 10, (**b**) epoch 20, (**c**) epoch 30, (**d**) epoch 40, (**e**) epoch 50, and (**f**) epoch 60. A graph showing the difference between values.

The LSTM network was designed to predict SOC at each time step based on the input data: voltage, current, and temperature. The LSTM cell, which consists of input gates, forget gates, and output gates, decides whether to retain or discard information, thus enhancing the model's learning capability. The model was trained using data from various charge and discharge cycles, and the performance was evaluated at each epoch (training iteration) to determine the optimal configuration settings.

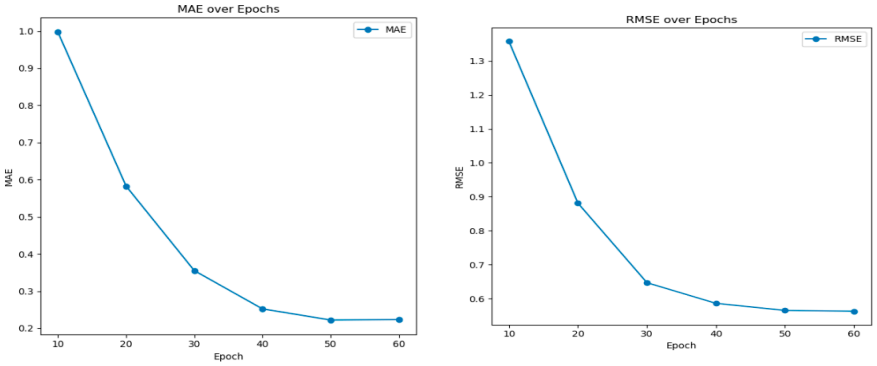
Table 5 and Figure 9 present the predicted SOC, MAE, and RMSE values for each epoch, allowing us to analyze the model's training process and performance. In the early stages, such as epoch 10, the predicted SOC was 14.8400%, with MAE at 0.9185 and RMSE at 1.3009. These higher error values indicate that the model was not yet optimized. However, as the training progressed to epoch 20, the predicted SOC decreased to 12.6617%, MAE reduced to 0.5693, and RMSE to 0.8812, demonstrating that the model was learning and improving.

Table 5. LSTM model performance across epochs.

Epoch	Predicted SOC	MAE	RMSE
Epoch 10	14.8400	0.9185	1.3009
Epoch 20	12.6617	0.5693	0.8812
Epoch 30	12.4947	0.3403	0.6511
Epoch 40	12.6203	0.2706	0.5968
Epoch 50	12.5185	0.2331	0.5793
Epoch 60	12.4968	0.2333	0.5682

By epoch 30, the predicted SOC was 12.4947%, with further reductions in MAE and RMSE to 0.3403 and 0.6511, respectively. This trend of increasing accuracy and decreasing error continued, with epoch 40 showing a predicted SOC of 12.6203, MAE of 0.2706, and RMSE of 0.5968. The continuous decline in errors indicates that the model was becoming increasingly precise. By epoch 50, the predicted SOC was 12.5185%, with MAE

at 0.2331 and RMSE at 0.5793, further indicating ongoing improvements in performance. Finally, at epoch 60, the predicted SOC stabilized at 12.4968%, with MAE at 0.2333 and RMSE at 0.5682, suggesting that the model was stabilizing and the margin of error was consistently decreasing.



(**a**) (**b**)

Figure 9. Comparison of (**a**) MAE and (**b**) RMAE across epochs.

Overall, as the number of epochs increased, the predicted SOC values gradually stabilized, and both MAE and RMSE showed a declining trend. This indicates that the model was training effectively and could make increasingly accurate predictions over time. The continuous decrease in MAE and RMSE reflects a reduction in the model's prediction error, signifying steady improvement in performance.

5. Conclusions

Accurate condition prediction (SOC) of lithium-ion batteries plays an important role in various applications such as electric vehicles and smart grids. Accurate estimation of SOC is essential to ensure efficient use, longevity, and safety of batteries. However, accurately estimating SOC is a challenge due to the complex nonlinear nature of batteries and the varying operating environments. In this study, we propose a method to estimate the SOC of lithium-ion batteries using deep learning, especially short- and LSTM models. LSTM has a great ability to learn long-term dependencies from sequence data and can effectively model temporal patterns in battery data. This study used battery data collected from different charge and discharge cycles to train an LSTM model and evaluated the model's performance by epoch to derive the optimal model configuration. Through this method, this study aimed to improve the accuracy and reliability of SOC estimation of lithium-ion batteries. In this paper, we propose a method for estimating the SOC (condition prediction) of lithium-ion batteries using the LSTM model, and explore the possibility of improving the performance of the battery management system through it. The proposed model has shown highly accurate SOC estimations for a variety of charge and discharge profiles, especially when concluding based on the data provided that the predicted SOC value tends to be closer and closer to the actual SOC value of 13.5441% as the epoch progresses. In epoch 10, the predicted SOC was 14.8400%, which was significantly higher than the actual value, but as we progressed to epoch 60, the predicted SOC was 12.4968%, which was closer to the actual value. This means that as training progresses, the accuracy of the model was improving. At the same time, the MAE and root mean squared error (RMSE) were also decreasing. In epoch 10, MAE was 0.9185 and RMSE was 1.3009, but in epoch 60 MAE was 0.2333 and RMSE was 0.5682. This decreasing trend indicates that the model's predictive performance was improving.

Overall, as the model learned, it approached the actual SOC value more accurately, and the error rate was constantly decreasing. Therefore, this model can be evaluated as a useful tool for SOC prediction.

Author Contributions: S.-B.K. and S.-H.L.: writing—original draft, data curation, software, and visualization. S.-B.K.: writing—review and editing. S.-H.L.: conceptualization, validation, writing—review and editing, and project administration. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by "Regional Innovation Strategy (RIS)" through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (MOE) (2021RIS-002).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: All data, models, and codes generated or used during the study are available from the corresponding author upon request.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Birky, A.K. Modeling for Light and Heavy Vehicle Market Analysis. Energetics, 2015 Department of Energy. 2015. Available online: https://www.energy.gov/sites/prod/files/2015/07/f24/van012_birky_2015_p.pdf (accessed on 10 December 2023).
2. Hannan, M.A.; Lipu, M.H.; Hussain, A.; Mohamed, A. A review of lithium-ion battery state of charge estimation and management system in electric vehicle application: Challenges and recommendations. *Renew. Sustain. Energy Rev.* **2017**, *78*, 834–854. [CrossRef]
3. Lu, L.; Han, X.; Li, J.; Hua, J.; Ouyang, M. A review on the key issues for lithium-ion battery management in electric vehicles. *J. Power Sources* **2013**, *226*, 272–288. [CrossRef]
4. Hu, X.; Li, S.E.; Yang, Y. Advanced Machine Learning Approach for Lithium-Ion Battery State Estimation in Electric Vehicles. *IEEE Trans. Transp. Electrif.* **2016**, *2*, 140–149. [CrossRef]
5. Zhang, Y.Z.; Ziong, R.; He, H.W.; Liu, Z. A LSTM-RNN method for the lithium-ion battery remaining useful life prediction. In Proceedings of the Prognostics and System Health Management Conference (PHM-Harbin), Harbin, China, 9–12 July 2017.
6. Huang, C.; Wang, Z.; Zhao, Z.; Wang, L.; Lai, C.S.; Wang, D. Robustness evaluation of extended and unscented Kalman filter for battery state of charge estimation. *IEEE Access* **2018**, *6*, 27617–27628. [CrossRef]
7. Ng, K.S.; Moo, C.-S.; Chen, Y.-P.; Hsieh, Y.-C. Enhanced coulomb counting method for estimating state-of-charge and state of health of lithium-ion batteries. *Appl. Energy* **2009**, *86*, 1506–1511. [CrossRef]
8. Sepasi, S.; Ghorbani, R.; Liaw, B.Y. A novel on-board state-of-charge estimation method for aged Li-ion batteries based on model adaptive extended kalman filter. *J. Power Sources* **2014**, *245*, 337–344. [CrossRef]
9. Plett, G.L. Extended Kalman filtering for battery management systems of LiPB-based HEV battery packs: Part 2. Modeling and identification. *J. Power Sources* **2004**, *134*, 262–276. [CrossRef]
10. Plett, G.L. Extended Kalman filtering for battery management systems of LiPB-based HEV battery packs: Part 3. State and Parameter Estimation. *J. Power Sources* **2004**, *134*, 277–292. [CrossRef]
11. Dai, B.; Zhang, Y.; Lin, D. Detecting Visual Relationships with Deep Relational Networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 3076–3086.
12. Rivera-Barrera, J.P.; Muñoz-Galeano, N.; Sarmiento-Maldonado, H.O. SOC Estimation for Lithium-ion Batteries: Review and Future Challenges. *Electronics* **2017**, *6*, 102. [CrossRef]
13. Kim, S.-B.; Lee, S.-H. Battery Balancing Algorithm for an Agricultural Drone Using a State-of-Charge-Based Fuzzy Controller. *Appl. Sci.* **2020**, *10*, 5277. [CrossRef]
14. Anton, J.C.A.; Nieto, P.J.G.; Viejo, C.B.; Vilán, J.A.V. Support Vector Machines Used to Estimate the Battery State of Charge. *IEEE Trans. Power Electron.* **2013**, *28*, 5919–5926. [CrossRef]
15. Song, X.; Yang, F.; Wang, D.; Tsui, K.-L. Combined CNNLSTM Network for State-of-Charge Estimation of Lithium-Ion Batteries. *IEEE Access* **2024**, *7*, 88894–88902. [CrossRef]
16. Lee, S.-H. Performance Evaluation of Machine Learning and Deep Learning-Based Models for Predicting Remaining Capacity of Lithium-Ion Batteries. *Appl. Sci.* **2023**, *13*, 9127. [CrossRef]
17. Chemali, E.; Preindl, M.; Malysz, P.; Emadi, A. Electrochemical and Electrolytic Cell Modeling for SOC Estimation: A Comparative Analysis of SVM and Neural Network Models. *Energy* **2017**, *125*, 375–388.
18. Sutskever, I. Training Recurrent Neural Networks. Ph.D. Thesis, University of Toronto, Toronto, ON, Canada, 2013.
19. Lee, S.-H. A Study on the Performance Evaluation of the Convolutional Neural Network–Transformer Hybrid Model for Positional Analysis. *Appl. Sci.* **2023**, *13*, 11258. [CrossRef]
20. Duchi, J.; Hazan, E.; Singer, Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *J. Ma-Chine Learn. Res.* **2011**, *12*, 2121–2159.
21. Tieleman, T.; Hinton, G. Double Sarsa and Double Expected Sarsa with Shallow and Deep Learning. *J. Data Anal. Inf. Process.* **2016**, *4*, 26–30.

22. Hong, S.R.; Kang, M.; Kim, G.W.; Jeong, H.G.; Beak, J.B.; Kim, J.H. Comparison of Learning Techniques of LSTM Network for State of Charge Estimation in Lithium-Ion Batteries. *J. Inst. Korean Electr. Electron. Eng.* **2019**, *23*, 1328–1336.
23. Li, Z.; Huang, J.; Liaw, B.Y.; Zhang, J. On state-of-charge determination for lithium-ion batteries. *J. Power Sources* **2017**, *348*, 281–301. [CrossRef]