Proceedings of the ASME 2018 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference
IDETC/CIE 2018
August 26-29, 2018, Quebec City, Quebec, Canada

DETC2018-86347

Online Estimation of Lithium-Ion Battery Capacity Using Deep Convolutional Neural Networks

Sheng Shen
Department of Mechanical Engineering,
Iowa State University
Ames, IA, USA

MK Sadoughi
Department of Mechanical Engineering,
Iowa State University
Ames, IA, USA

Xiangyi Chen
Department of Electrical and Computer Engineering,
University of Minnesota Twin Cities
Minneapolis, MN, USA

Mingyi Hong
Department of Electrical and Computer Engineering,
University of Minnesota Twin Cities
Minneapolis, MN, USA

Chao Hu
Departments of Mechanical Engineering and
Electrical and Computer Engineering,
Iowa State University
Ames, IA, USA

ABSTRACT

Over the past two decades, safety and reliability of lithium-ion (Li-ion) rechargeable batteries have been receiving a considerable amount of attention from both industry and academia. To guarantee safe and reliable operation of a Li-ion battery pack and build failure resilience in the pack, battery management systems (BMSs) should possess the capability to monitor, in real time, the state of health (SOH) of the individual cells in the pack. This paper presents a deep learning method, named deep convolutional neural networks, for cell-level SOH assessment based on the capacity, voltage, and current measurements during a charge cycle. The unique features of deep convolutional neural networks include the local connectivity and shared weights, which enable the model to estimate battery capacity accurately using the measurements during charge. To our knowledge, this is the first attempt to apply deep learning to online SOH assessment of Li-ion battery. 10-year daily cycling data from implantable Li-ion cells are used to verify the performance of the proposed method. Compared with traditional machine learning methods such as relevance vector machine and shallow neural networks, the proposed method is demonstrated to produce higher accuracy and robustness in capacity estimation.

Keywords: State of Health, Li-ion battery, Deep convolutional neural networks

1. INTRODUCTION

Over the past two decades, lithium-ion (Li-ion) rechargeable batteries have been increasingly used in consumer electronics, such as cell phones and laptops, in transportation applications, such as hybrid and electric vehicles, and in emerging grid-scale energy storage applications. Safety and reliability of the Li-ion batteries in these applications have been receiving a considerable amount of attention from both industry and academia. Online state of health (SOH) estimation by battery management systems (BMSs) is essential to ensuring the safety and reliability of individual cells operating in the field [1]. Two major parameters characterizing the SOH of a Li-ion battery cell are the capacity and internal resistance of the cell [2]. When the capacity of a cell is used to indicate its SOH, the cell SOH is often defined as the ratio of the cell capacity at the current charge-discharge cycle to the rated capacity given by the cell manufacturer. Accurate capacity estimation allows a battery cell to be replaced right before the cell capacity fades to an end of life limit, thus allowing cell useful life to be fully exploited while preventing imminent cell failures.

In recent years, literature has reported a variety of machine learning-based methods for online estimating the capacity of Li-ion batteries. In these machine learning-based methods, the SOC estimation can be performed by learning the complex dependency of the battery capacity on the characteristic features that are extracted from the charge voltage and current measurements. This dependency can be learned using different

learning methods, including neural network (NN) [3] and kernel regression [4]. For instance, Hu et al. [5] have extracted several characteristic features that are indicative of the capacity from the charge curves and then employed RVM to learn the relationship between the capacity of a battery and its charge-related features. A RVM regression model, after being trained offline, can be used to infer the unknown capacity of a battery online from a set of charge related features.

Although these traditional machine learning methods can provide the satisfactory accuracy in capacity estimation, they require manually extracting a few number of features from the charge voltage and current measurements. It is difficult to manually identify the characteristic features that carry the most useful information for estimating the battery capacity [6]. In addition, extracting a limited number of features may risk the loss of critical information during the training process.

To avoid the need for human engineers to manually identify the informative features from the raw data, this research will explore the potential of a deep learning method, deep convolutional neural networks (DCNNs), to build a predictive model from large-scale datasets for achieving state-of-the-art accuracy in online estimation of Li-ion battery capacity. DCNNs can easily take advantage of increases in the amount of available computation and data, to the task of online estimation of lithium-ion battery capacity.

This paper is organized as follows. Section 2 elucidates the fundamental principle of deep learning and describes the architecture of DCNNs. Section 3 discusses the use of DCNNs to build a capacity estimation model that approximates the mapping from the multi-dimensional time-series inputs to the one-dimensional capacity output. The effectiveness of the capacity estimation model is demonstrated by leveraging 10-year daily cycling data from 8 implantable Li-ion cells. Section 4 presents and discusses the results from this demonstration. Several concluding remarks of this study are presented in Section 5.

2. METHODOLOGY

This section describes the proposed approach for online estimating the Li-ion battery capacity. Subsection 2.1 describes the strategy for structuring the input and output variables in online capacity estimation. Subsection 2.2 describes the overall architecture of DCNNs that approximates a non-linear mapping from the multi-dimensional time-series input space to the one-dimensional capacity output space. Subsection 2.3 discusses the algorithm used to train a DCNNs model.

2.1 Input and output structures

The objective of this study is to estimate the capacity of a Li-ion cell based on the input curves (i.e. voltage, current and capacity curves) of the cell during the charge process. It should be mentioned that the voltage and current can be directly measured from the cell, while capacity curve is calculated using the coulomb counting method, which integrates the charge current over time for the entire charge cycle [7]. The simple input curves of an individual cell for one charge cycle is shown in Fig. 1. Each curve is discretized into 25 steps (corresponding to 25 equal time steps) and the discretized values of voltage, current and capacity are considered as the input variables to be fed into the DCNNs model. Therefore, the input variable is a 25×3 matrix, in which the first, second and third columns correspond to the discretized values of voltage, current and capacity, respectively, and can be expressed as:

$$\text{Input} = \begin{bmatrix} \hat{V}_1 & \hat{I}_1 & \hat{C}_1 \\ \vdots & \vdots & \vdots \\ \hat{V}_i & \hat{I}_i & \hat{C}_i \\ \vdots & \vdots & \vdots \\ \hat{V}_{25} & \hat{I}_{25} & \hat{C}_{25} \end{bmatrix}_{25\times 3} \quad (1)$$

where $\hat{V}_i$, $\hat{I}_i$, and $\hat{C}_i$ are the discretized values of voltage, current and capacity at the i^{th} time step (Δt_i) of an individual charge cycle. For each charge cycle, there is a corresponding discharge capacity output, which is a scalar number and calculated using the coulomb counting method, integrating the discharge current over time for the entire discharge cycle.

In order to model this high-dimensional mapping problem (i.e., $R^{75} \rightarrow R^1$), this study utilizes a well-known deep learning method called DCNNs. In the next subsection, a review of DCNNs and the reasons behind choosing this methodology are provided.

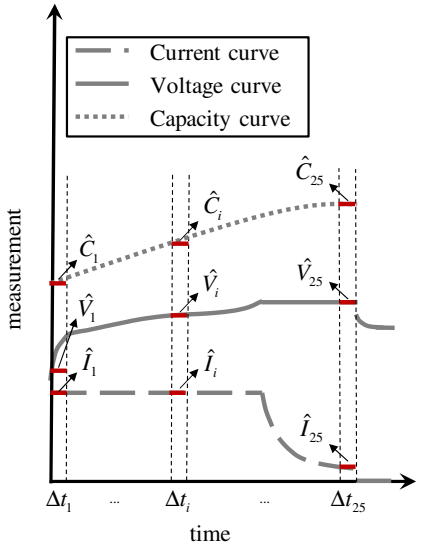

Fig. 1 Simple input curves of an individual cell for one charge cycle

2.2 Overall architecture of DCNNs

The overall architecture of DCNNs, as depicted in Fig. 2, is composed of eight building blocks, the first five being convolution stages and the remaining three being fully-connected stages. In this study, the networks have 28,000 parameters and 5,841 neurons, consist of five convolutional layers, one of which is followed by a max-pooling layer, and three fully-connected layers with a regression layer. BN is applied to the outputs of all convolutional and fully-connected

layers in the proposed DCNNs. Following BN, ReLU is used to introduce nonlinearity to the networks. The max-pooling layer is only applied to the output of ReLU at the first convolutional layer. The output of the last fully-connected layer is fed into a regression layer to predict the target output that is a continuous variable.

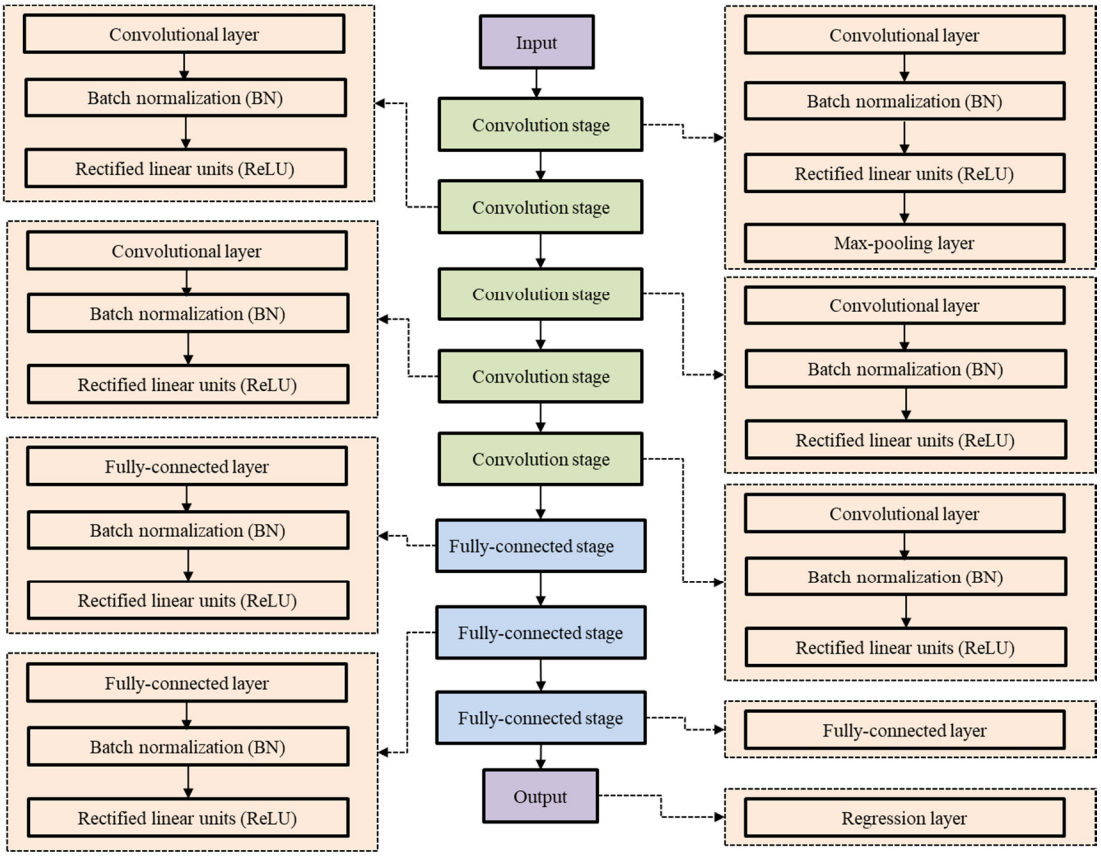

Fig. 2 An illustration of the architecture of the proposed model

As discussed in Subsection 2.1, the inputs to the DCNNs are a sample matrix with fixed size $25 \times 3 \times 1$ (i.e., the height, width, and number of channels of the sample matrix). Thus, the networks' inputs consist of 75 measurements (i.e., F_1^1, F_1^2, …, F_{25}^1) that can be divided into three groups: 1) voltage $\mathbf{F}^1 = \{F_1^1, F_2^1, \ldots, F_{25}^1\}$, 2) current $\mathbf{F}^2 = \{F_1^2, F_2^2, \ldots, F_{25}^2\}$, and 3) charge capacity $\mathbf{F}^3 = \{F_1^3, F_2^3, \ldots, F_{25}^3\}$. These measurements represent the entire dataset collected from a cell during a partial charge cycle. In the 1st convolution stage, the convolutional layer filters inputted samples with 16 kernels (also called filter) of size $1 \times 2 \times 1$, a stride of size 1×1, and the padding $[0,0,1,1]$. The filter moves along the input sample matrix vertically and horizontally, repeating the same computation for each region, that is, convolving the input. The step size with which it moves is called the stride. Then, the samples, in turn, are fed into BN, ReLU, and the max-pooling layer of this stage. The 2nd convolution stage receives the outputs from the 1st convolution stage and feeds it into the convolutional layer of this stage, and filters it with 32 kernels of size $3 \times 1 \times 1$. The 3rd, 4th, and 5th convolution stages are connected sequentially through taking the responses from the ReLU of the previous stage as the inputs to the next stages. They all possess 40 kernels of size $3 \times 1 \times 1$ with a stride of size 1×1. The fully-connected layers have 40 neurons, each of which connects to all neurons in the previous layer. After passing through a stack of convolution and fully-connected stages, the inputted sample reaches the destination, regression layer, which outputs an estimated target corresponding to the input sample.

In this study, the values of the kernels, neurons, and numbers of fully-connected layers are identified empirically through a large number of parametric experiments. Based on the experimental results, two takeaway points can be summarized and are listed as follows:

- The use of a very small filter size throughout the entire networks could improve the performance of the proposed model.
- Removing fully-connected layers or changing the neurons of fully-connected layers may result in degraded performance of the proposed model.

2.3 Training algorithm

Like any other machine learning method, DCNNs contain a set of unknown parameters that need to be identified using a training dataset. To this end, a cost function J is defined to quantify the differences between the DCNNs' predictions and

the true outcomes of the training dataset. The stochastic gradient descent (SGD) method is utilized to minimize the expected generalization error given by the cost function. SGD is probably the most widely used optimization algorithm in deep learning, particularly when the training process is very slow due to the large amount of available data. In this study, SGD with momentum (SGDM), originally designed by B.T. Polyak [8], is employed as the optimization algorithm to accelerate the learning process. SGDM updates the parameters, $\boldsymbol{\theta}$ (weights, $\boldsymbol{\omega}$, and biases, b), to minimize the generalization error by taking small steps in the direction of the negative gradient of the cost function. In other words, SGDM accumulates an exponentially decaying moving average of the past gradients and continues to move in their direction. The cost function $J(\boldsymbol{\theta})$ with the regularization term (also called weight decay) can be expressed as:

$$J_R(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda\Omega(\boldsymbol{\omega}) = \frac{1}{2m}\sum_{i=1}^{m}(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2}\boldsymbol{\omega}^T\boldsymbol{\omega} \quad (2)$$

where $J_R(\boldsymbol{\theta})$ is the cost function with the regularization term $\lambda\Omega(\boldsymbol{\omega})$, λ is the L_2 regularization factor that weighs the relative contribution of the norm penalty term, $\Omega(\boldsymbol{\omega})$, $h_{\boldsymbol{\theta}}(\mathbf{x})$ is the hypothesis function, m is the number of training samples used at each iteration, $\mathbf{x}^{(i)}$ is the feature vectors of the i^{th} training sample, and $y^{(i)}$ is the associated target value. The hypothesis function $h_{\boldsymbol{\theta}}(\mathbf{x})$ is defined as:

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = b_0x_0 + \omega_1x_1 + \cdots + \omega_nx_n \quad (3)$$

where n is the input dimension, and x_n and ω_n denote the n^{th} input variable and its parameter, respectively ($x_0 = 1$, and b_0 is the bias value). Then, the parameter $\boldsymbol{\theta}$ can be iteratively updated according to the following:

$$\boldsymbol{\theta}_{j+1} = \boldsymbol{\theta}_j - \alpha\nabla J_R(\boldsymbol{\theta}) + \gamma(\boldsymbol{\theta}_j - \boldsymbol{\theta}_{j-1}) - \lambda\alpha\boldsymbol{\theta}_j \quad (4)$$

where α is the initial learning rate (or step size), γ is the momentum that determines the contribution of gradients from the previous iteration to the current iteration, $\nabla J_R(\boldsymbol{\theta})$ is the gradient of the cost function, $J_R(\boldsymbol{\theta})$, and $\boldsymbol{\theta}_j$ denotes the parameter estimate at the j^{th} iteration, and j is the iteration index.

The weights in each layer are randomly initialized according to a Gaussian distribution with the mean of 0 and standard deviation of 0.01, while the values of biases of all convolutional and fully-connected layers are initialized at 0.

3. IMPLEMENTATION OF DEEP LEARNING

The efficiency of the proposed deep learning method was verified based on experimental data from a 10 years' continuous cycling test (i.e., repeated full charge/discharge cycles) on eight Li-ion prismatic cells. This section first introduces the extraction of partial charge data from the full charge data, and then discusses the implementation of training, validation, and test for DCNNs in this experimental verification.

3.1 Data generation for capacity estimation

It is noted that, in practice, a Li-ion battery cell often does not experience a complete discharge process before being installed on a battery charger. It is often that a user wishes to charge the cell before its complete depletion. In this case, the cell starts to be charged from a partially discharged state with a certain amount of remaining capacity. In order to simulate this real case where the cell starts to be charged at the partially discharged state and ends up being fully charged, we generated a partial charge curve from a full charge curve by truncating the full charge curve below a pre-assigned initial charge voltage (V_{initial}) [9,10]. The value of V_{initial} was randomly drawn from a uniform distribution between a lower bound V_{low} and an upper bound V_{high}. Two settings with different voltage ranges that correspond to a low initial SOC and a high initial SOC were considered in this study to investigate how the initial SOC affects the accuracy of capacity estimation. Note that the high initial SOC setting indicates a cell undergoes a shallower discharge, as compared to the low initial SOC. Table 1 summarizes the voltage and SOC ranges for these two initial SOC settings.

Table 1. The two initial SOC settings considered in generation of partial charge data

Setting	Setting name	Voltage range	SOC range
Setting I	Low initial SOC	V_{low} = 3.65 V V_{high} = 3.80 V	Roughly 3%—23%
Setting II	High initial SOC	V_{low} = 3.80 V V_{high} = 3.85 V	Roughly 23%—43%

In addition to the effect of the initial SOC, the effect of current measurement error was also investigated in this study. This investigation considered two scenarios: Scenario I — No bias in current measurement and Scenario II — 2% positive bias in current measurement. In Scenario I, no change was made to the current and charge capacity measurements, whereas in Scenario II, the current-related features, the current and charge capacity, were artificially increased by 2% to simulate a 2% positive bias in the current measurement. The two scenarios were employed to study the influences of current measurement error on the accuracy of capacity estimation. Based on these two scenarios and the two voltage settings, a total of 4 different cases were considered in this study and are listed as follows:

- Case 1 — Low initial SOC (3%—23%), no bias in current measurement;
- Case 2 — Low initial SOC (3%—23%), 2% positive bias in current measurement;
- Case 3 — High initial SOC (23%—43%), no bias in current measurement;
- Case 4 — High initial SOC (23%—43%), 2% positive bias in current measurement.

3.2 Implementation of training for DCNNs

The objective of the proposed deep learning method is to reduce the expected generalization error given by Eq. (2). In order to meet this objective, we trained our DCNNs model using SGDM with a minibatch of 128 examples and their associated targets. An initial learning rate of 0.01 was set for all layers and this rate decreases by a factor of 5 for every 7 training epochs. The values of several important parameters used in training the DCNNs model are listed in Table 2.

Table 2. List of parameter values used in training

Parameter	Value
Initial learning rate, α	0.01
Minibatch size	128
Momentum, γ	0.9
L_2 Regularization, λ	0.0001
Number of epochs	35

During the training process, the training and validation datasets were shuffled for each epoch to achieve a relatively unbiased estimator of the true gradient of the cost function and to give all samples an equal chance to be used. Specifically, the minibatch with 128 samples in one epoch, used for updating the parameters **θ** (weights and biases) and computing the validation RMSE, is different from that in another epoch. Due to several reasons including the randomness in the data shuffling and parameter initialization, two runs with the same training and validation datasets may give rise to two different local minima in the cost function. However, this is often not a real concern and we therefore usually settle for finding a point in the parameter space that has a low cost but not the minimal cost [11,12,13].

In this study, computations were carried out on a processor Intel Core i7-8700 CPU 3.2 GHz and 64 RAM, and a NVIDIA TITAN XP graphics processing unit (GPU) with 12 GB of GDDR5X memory. With the support from the advanced CPU and GPU, our network takes between 5 and 7 minutes to train.

3.3 Implementation of validation and test for DCNNs

The conventional data splitting strategy first shuffle the entire dataset and then split it into training, validation, and test subsets to ensure that samples be selected randomly for training, validation, and test. This is mainly due to the fact that many datasets are naturally arranged in some forms where successive samples are more likely correlated with their neighbors than those arranged farther away. The dataset in this study consists of cycling test data from eight cells, arranged in a descending order of the charge/discharge cycle number within a cell and in an ascending order of the cell number (i.e., cell 1 to cell 8). If samples in a minibatch were drawn following the order of this dataset, the minibatch would likely consist of samples from one specific cell and thus be biased toward the cell. In such a case, whereas the order of the dataset holds some physical significance, it is necessary to shuffle the dataset before splitting it for training and validation [13].

For the task of online capacity estimation, it is of practical significance for a trained DCNNs model to be capable of accurately estimating the capacity of one specific cell from the beginning to the end of life. In other words, the test dataset should be made up of complete data from one or multiple cells rather than partial data from the cell(s). Consequently, the traditional splitting strategy needs to be adjusted for the specific task considered in this study. To this end, instead of shuffling the entire dataset, we first took the data from one cell out of the entire dataset, to be used as the test set, and then shuffled the remaining data from the other seven cells to create the training (70%) and validation (30%) sets. We therefore evaluated the performance of the proposed method using a test dataset that consists of the complete data from one cell and is separated from the data used for training and validation.

Although the data splitting strategy adopted in this study differs from those usually used by researchers in the deep learning community, it allows us to evaluate the performance of a trained deep learning model against a complete set of measurement data from the beginning to the end of life for any battery cell.

3.4 Definition of error measures

The accuracy of a trained DCNNs model was evaluated by using the eight-fold cross validation, as shown in Fig. 3. The completed dataset was first split into eight mutually exclusive subsets or folds, cell 1, cell 2, …, cell 8, corresponding to the eight experimental cells. Subsequently, eight trials of training, validation, and test were performed such that within each trial a fold of the data was held out for test, while the remaining seven folds were pulled together, randomly shuffled, and divided into a training dataset (70% of all samples in the seven folds) and a validation dataset (30%).

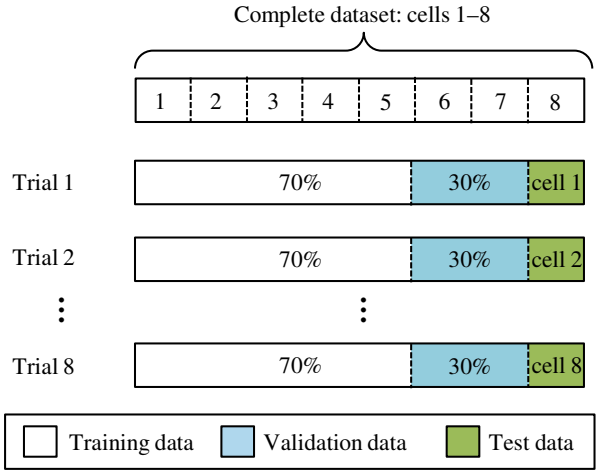

Fig. 3 Procedure of eight-fold cross validation

In the k^{th} cross validation trial, the trained DCNNs model was used to estimate the capacities of the samples in the test dataset (i.e., the complete data from the k^{th} cell). After performing all the eight cross validation trials, the overall test error $\varepsilon_{\text{RMS}}^{\text{All}}$ by DCNNs was then estimated by taking the average of the individual test errors $\varepsilon_{\text{RMS}}^{\text{k}}$ across the eight trials.

$$\varepsilon_{\text{RMS}}^{k} = \sqrt{\frac{1}{N_k}\sum_{i=1}^{N_k}(y^k(\mathbf{x}_i^k) - \hat{y}^k(\mathbf{x}_i^k))^2} \tag{5}$$

$$\varepsilon_{\text{RMS}}^{\text{All}} = \sqrt{\frac{1}{\sum_{k=1}^{8} N_k}\sum_{k=1}^{8}\sum_{i=1}^{N_k}(y^k(\mathbf{x}_i^k) - \hat{y}^k(\mathbf{x}_i^k))^2} \tag{6}$$

where N_k indicates the number of samples used for test in the k^{th} cross validation trial, $\mathbf{x}_i^k$ is the input feature vector of the i^{th} sample in the k^{th} trial, and $y^k(\mathbf{x}_i^k)$ and $\hat{y}^k(\mathbf{x}_i^k)$ are respectively the measured (or true) and estimated capacities for the i^{th} sample in the k^{th} trial.

4. RESULTS AND DISCUSSION

4.1 Capacity estimation results

In order to estimate the accuracy of the proposed methodology for capacity estimation, a traditional machine learning method, RVM, was also employed for the four cases explained in Subsection 3.1. Tables 3–6 show the comparison results between RVM and DCNNs in terms of both the RMSE and maximum error (ME). It should be mentioned that the capacity estimation of DCNNs is based on the best hypotheses of the proposed model (i.e. the best model with the lowest validation error among 150 runs).

Table 3. Capacity estimation results by RVM and DCNNs for Case 1

Model	Item	#1	#2	#3	#4	#5	#6	#7	#8	Overall
RVM	RMSE	0.3379	0.3545	0.5230	0.4419	0.4691	0.4025	0.4203	0.3988	0.4185
	ME	4.5305	2.1429	3.0123	2.6030	2.9161	2.4677	2.4883	2.8807	4.5305
DCNNs	RMSE	0.3022	0.2980	0.3089	0.3677	0.4098	0.3117	0.2770	0.2619	0.3171
	ME	2.4910	2.6828	3.3294	3.5241	2.2963	2.5342	2.8852	2.0273	3.5241

Table 4. Capacity estimation results by RVM and DCNNs for Case 2

Model	Item	#1	#2	#3	#4	#5	#6	#7	#8	Overall
RVM	RMSE	0.3606	0.4411	0.3843	0.5339	0.5062	0.3636	0.3816	0.3706	0.4177
	ME	4.0473	3.2248	2.9912	3.0050	3.2329	3.4885	2.6736	2.9266	4.0473
DCNNs	RMSE	0.3583	0.2599	0.3217	0.4451	0.3923	0.3374	0.2787	0.2714	0.3331
	ME	2.8154	2.3472	4.4404	4.9244	2.1295	2.8925	4.2610	2.0590	4.9244

Table 5. Capacity estimation results by RVM and DCNNs for Case 3

Model	Item	#1	#2	#3	#4	#5	#6	#7	#8	Overall
RVM	RMSE	0.4734	0.4924	0.5742	0.5582	0.8408	0.5601	0.7613	0.6037	0.6080
	ME	5.4080	2.0123	6.2421	6.2263	5.6261	7.6505	3.6241	2.2397	7.6505
DCNNs	RMSE	0.3396	0.3604	0.3384	0.4700	0.3922	0.2776	0.3228	0.3076	0.3511
	ME	4.3461	2.8369	6.2855	6.4279	4.5568	5.8600	2.6983	2.1923	6.4279

Table 6. Capacity estimation results by RVM and DCNNs for Case 4

Model	Item	#1	#2	#3	#4	#5	#6	#7	#8	Overall
RVM	RMSE	0.4189	0.5540	0.5691	0.6474	0.8813	0.6394	0.6619	0.6007	0.6216
	ME	5.0855	2.1475	6.1685	6.0097	3.5672	8.6117	3.3122	2.5260	8.6117
DCNNs	RMSE	0.3408	0.3710	0.3937	0.4204	0.4919	0.2876	0.2704	0.4252	0.3752
	ME	4.6058	2.7902	5.9427	6.5492	4.7832	4.1072	3.7108	2.4969	6.5492

Three significant observations can be made from these results and are listed as follows:

- First, based on the overall RMSEs and Max errors from all eight cells, the performance of DCNNs is evidently better than that of RVM regardless of the SOC and current bias conditions. From the perspective of RMSEs and Max errors of individual cells, the results suggest that DCNNs are capable of making a more accurate capacity estimation than RVM for all test cells. Although there are several situations that the Max error of RVM is slightly lower than the corresponding Max error of DCNNs, the overall performance of DCNNs is clearly better than RVM in all four Cases. Furthermore, we can observe that both DCNNs and RVM are able to adapt the regression model to changes in the input vectors for all four cases, and the performance improvement of DCNNs over RVM is more significant in the high SOC than the low SOC.
- Second, introducing the 2% positive bias in current measurement does not bring a significant influence on the accuracy of DCNNs (compare Case 1 versus Case 2 and Case 3 versus Case 4).
- Third, the RMSE by the proposed method did not show large increases when the initial SOC range is elevated (10.7% increase from Case 1 to Case 3 and 12.6% increase from Case 2 to Case 4). However, much larger increases were observed in the RMSE by RVM (45.3% increase from Case 1 to Case 3 and 48.8% increase from Case 2 to Case 4). Therefore, it can be concluded that neither deep (low initial SOC 3%–23%) nor shallow (high initial SOC 23%–43%) discharge conditions affect the capacity of the DCNNs to make an accurate capacity estimation.

In summary, the verification results based on 10 years' cycling data suggest that the DCNNs model is able to achieve a more accurate capacity estimation than RVM regardless of the initial charge level and the current measurement noise level. Indeed, DCNNs are more effective than RVM, because they possess a higher generalization ability and can leverage a larger body of information.

4.2 Effect of number of layers

A convolutional neural network can consist of one or multiple convolutional layers. The optimal number of convolutional layers depends on the amount and complexity of the data. A parametric study was implemented to empirically investigate the effect of the number of convolutional layers on the accuracy of the DCNNs. For each cross validation trial of Cases 1 and 3, we conducted 150 independent optimization runs with the number of convolutional layers varied from 0 to 6. Two box plots of the overall RMSE versus the number of convolutional layers were produced to graphically summarize the simulation results and are shown in Fig. 4. Note that the overall RMSEs were based on the best 30 results out of the 150 runs. In DCNNs, a large number of convolutional layers may lead to a very small size of outputs at the last layer. In order to prevent the output size of the last convolutional layer from becoming too small, the maximum number of convolutional layers attempted in this study was 6.

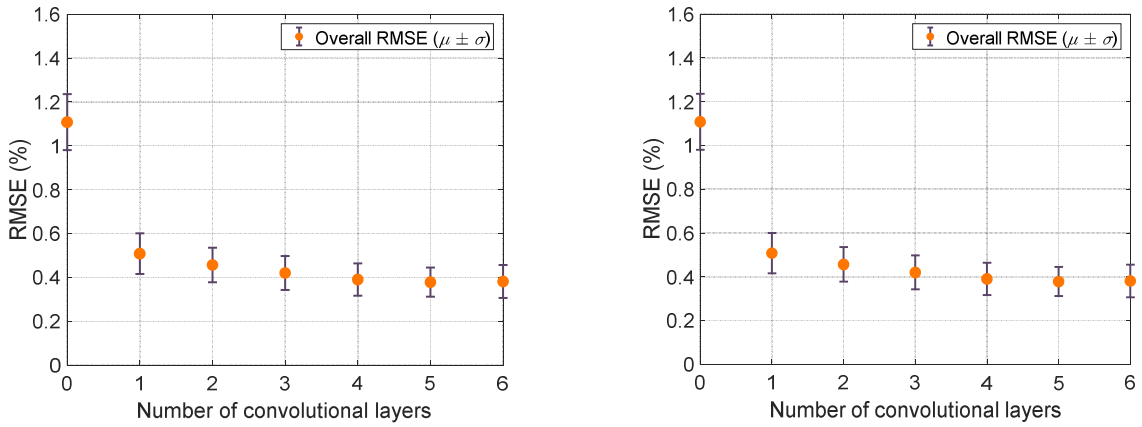

Fig. 4 Overall RMSE on different number of convolutional layers for Case 1 (a) and Case 3 (b).

Three observations can be made from the plots: (i) deep neural networks (i.e., DCNNs with 4 or more convolutional layers) significantly improve the accuracy in capacity estimation over the conventional neural networks (0 of convolutional layers); (ii) although each convolutional layer in a DCNNs model contains less than 1% of the networks' parameters, the number of convolutional layers plays a significant role in the estimation accuracy; and (iii) an inappropriate selection of the number of convolutional layers may lead to a low estimation accuracy. The DCNNs model with 5 convolutional layers produced a slightly lower error as compared with the model with any other number of layers. Thus, the final DCNNs model was chosen to contain five convolutional layers and this network depth seemed to ensure satisfactory performance of the proposed method. The results dealt earlier (Tables 3–6) were derived using this chosen value of 5.

5. CONCLUSION

In this study, a deep convolutional neural networks (DCNNs) model was constructed to explore online estimation of Li-ion battery capacity. Stochastic gradient descent with momentum (SGDM) was employed to train the DCNNs model which approximates the unobservable, complex data-to-health relationship. To our knowledge, this research is the first attempt to apply deep learning to online SOH assessment of Li-ion batteries. The performance of the proposed method was verified using 10 years' continuous cycling data acquired from eight Li-ion battery cells. The verification demonstrates that the proposed method achieves promising accuracy, suggesting that the method is an efficient tool for online health management of Li-ion batteries. Our future work includes the applications of our method to non-constant charge and discharge profiles, for instance, the urban dynamic driving schedule (UDDS). These applications would allow us to thoroughly test the robustness of the proposed method in more practical conditions.

ACKNOWLEDGEMENT

This research was in part supported by the US National Science Foundation (NSF) Grant Nos. CNS-1566579 and ECCS-1611333, the NSF I/UCRC Center for e-Design, and the U.S. Department of Transportation, Office of the Assistant Secretary for Research and Technology (USDOT/OST-R) through the Midwest Transportation Center (MTC). Any opinions, findings or conclusions in this paper are those of the authors and do not necessarily reflect the views of the sponsoring agencies. The authors would also like to express special thanks to Dr. Gaurav Jain and Dr. Hui Ye at Medtronic, Inc. for sharing the long-term cycling data for this study.

REFERENCES

1. Andrea, Davide. Battery management systems for large lithium ion battery packs. Artech house, 2010.
2. Lu, Languang, Xuebing Han, Jianqiu Li, Jianfeng Hua, and Minggao Ouyang. "A review on the key issues for lithium-ion battery management in electric vehicles." Journal of power sources 226 (2013): 272-288.

3. G. Bai, P. Wang, C. Hu, M. Pecht, A generic model-free approach for lithiumion battery health management, Appl. Energy 135 (2014) 247e260.
4. B. Saha, K. Goebel, S. Poll, J. Christophersen, Prognostics methods for battery health monitoring using a Bayesian framework, IEEE Trans. Instrum. Meas. 58 (2) (2009) 291e296.
5. Hu, Chao, Gaurav Jain, Craig Schmidt, Carrie Strief, and Melani Sullivan. "Online estimation of lithium-ion battery capacity using sparse Bayesian learning." Journal of Power Sources 289 (2015): 105-113.
6. LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." nature 521, no. 7553 (2015): 436..
7. C. Hu, G. Jain, P. Tamirisa, T. Gorka, Method for estimating capacity and predicting remaining useful life of lithium-ion battery, Appl. Energy 126 (2014)182-189.
8. Polyak, Boris T. "Some methods of speeding up the convergence of iteration methods." USSR Computational Mathematics and Mathematical Physics 4.5 (1964): 1-17.
9. Hu, Chao, Gaurav Jain, Craig Schmidt, Carrie Strief, and Melani Sullivan. "Online estimation of lithium-ion battery capacity using sparse Bayesian learning." Journal of Power Sources 289 (2015): 105-113.
10. Hu, Chao, Gaurav Jain, Puqiang Zhang, Craig Schmidt, Parthasarathy Gomadam, and Tom Gorka. "Data-driven method based on particle swarm optimization and k-nearest neighbor regression for estimating capacity of lithium-ion battery." Applied Energy 129 (2014): 49-55.
11. Saxe, Andrew M., James L. McClelland, and Surya Ganguli. "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks." arXiv preprint arXiv:1312.6120 (2013).
12. Dauphin, Yann N., Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization." In Advances in neural information processing systems, pp. 2933-2941. 2014.
13. Dauphin, Yann, Harm de Vries, and Yoshua Bengio. "Equilibrated adaptive learning rates for non-convex optimization." Advances in neural information processing systems. 2015.