Functionality of Optimization Techniques in Machine Learning for SoC Estimation

Okemakinde Femi, Kim Jonghoon
Energy Storage Conversion Lab., Chungnam National University

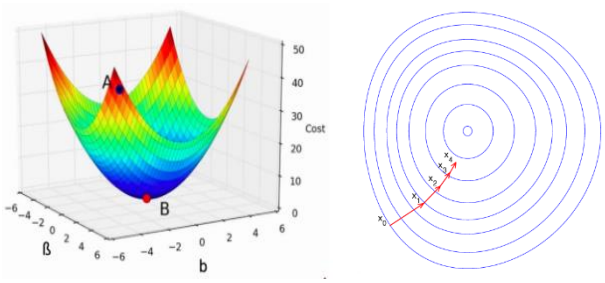
ABSTRACT

Over the years, numerous experiments have been carried out on different methods for accurate state-of-charge estimation. Neural networks have so far been the most utilized machine learning estimation method due to their ability to achieve high estimation performance. In this study, the foundation of a neural network is laid by investigating the most important parameter of the algorithm; the Optimization technique. First, the gradient descent optimizer is examined, then it is developed from scratch with the Python programming language. Finally, the functionality of this developed optimizer is confirmed by implementing it in a single-neuron NumPy machine learning algorithm.

1. Introduction

Of all the roles of the Battery Management System (BMS,) state of charge (SoC) estimation is the primary and most crucial criterion. As of now, the common methods of estimating a battery's state of charge are conventional method, model-based method, and data-driven method.
Unlike the other methods, the data-driven method does not require the simulation of a battery model, or information about the internal parameters of the battery. This method is a direct offline learning of the nonlinear relationship between the SoC and battery variables like voltage, current, and temperature, with the use of a machine learning (ML) model or algorithm.
Neural Networks (NN) has so far been the most utilized algorithm due to its ability to extract more hidden details from the input data which in turn improves the accuracy of the estimation[1]. Node layers, which include an input layer, one or more hidden layers, and an output layer, make up neural networks. Each node, otherwise known as a neuron, is connected to other nodes and has a weight and bias that go along with it. An optimizer that assigns weights and bias to each layer is chosen for a model, and as the model is being trained, the weights and biases of each node are adjusted by the optimizer so as to get the closest possible result to the SoC as possible.
This study aims to lay the foundation of a neural network model by investigating the Optimization technique. The optimization technique is essential in order to achieve the best values for the weights and biases of NN models under various circumstances.

2. Gradient Descent Modelling



(a) (b)

Fig.1 Gradient descent representations: a) 3D representation, and b) 2D representation

The mini-batch gradient descent will be developed in this study. The dataset will be divided into ten batches, and a function will be built to model this optimization technique from scratch. This function will take in seven arguments as input; The dataset (which includes the battery variables), the target SoC values, the initial gradient descent parameters (initial weights and initial bias), the gradient function, the cost function, the learning rate or alpha, and finally the desired number of iterations. Gradient descent starts at the initiation point, moving steadily downward until it reaches the point where the cost function is as minimal as possible as illustrated in Fig. 1.

2.1 Gradient Function

This function is responsible for evaluating the weights and bias changes in relation to the change in cost.
The weight parameter calculation is done with eqs. (1), while the bias parameter calculation is done with eqs. (2).

$$w_n = w_{n-1} - \alpha \frac{d}{dw} J(w_{n-1}, b_{n-1}) \quad (1)$$

$$b_n = b_{n-1} - \alpha \frac{d}{db} J(w_n, b_{n-1}) \quad (2)$$

w_n is the updated weight, w_{n-1} is the initial weight at the start of the computation, α is alpha which is the learning rate of the algorithm, and $\frac{d}{dx} J(w, b)$ is a multi-variable derivative term. Likewise, b_n is the updated bias, and b_{n-1} is the initial bias at the start of the computation.

2.2 Cost Function

Thanks to the cost function, we can improve the algorithm's performance by knowing how well it is performing. Depending on the different values of weights and bias for each iteration,

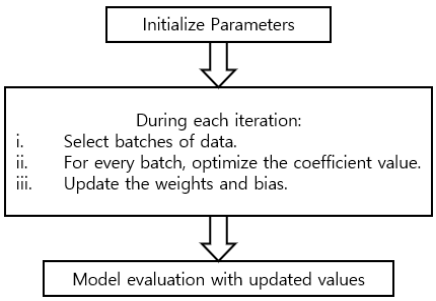


Fig.2 Gradient descent algorithm flowchart

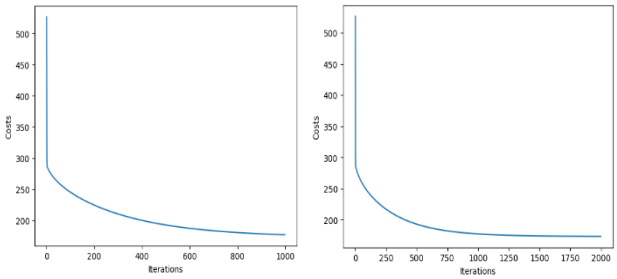


Fig.3 Cost convergence curve: a) DST profile, b) FUDS profile

the cost function will return the cost of utilizing these parameters.

The squared error cost function is utilized in this study because we must account for negative errors and eqs. (3) shows the expression of the cost function.

$$J(w,b) = \frac{1}{m}\sum_{n=1}^{m}(\hat{y_n} - y_n)^2 \quad (3)$$

J(w,b) is the cost of implementing the weights w and bias b, m is the total number of training examples in the dataset, $\hat{y}$ is the predicted SoC value after implementing the updated w and b, and y is the reference SoC value.

3. Gradient Descent Function & Results

At the start of the algorithm, an empty array called "J array" is initialized where the costs, J, of each iteration will be stored for investigation. The first set of weights and bias that which will be implemented is manually inputted, then using these parameters, the gradient function will compute the next set of weights and bias that gets the cost closer to the local minimum. The cost of implementing these parameters will be calculated by the cost function and the result will be recorded in the J array. A single-neuron NumPy function which has a functional composition, $N(f(x_n))$ is embedded in this algorithm. The neuron calculates the weighted sum of the input values as expressed in eqs. (4).

$$f(x_n) = Wx_n + b \quad (4)$$

The procedures carried out by this optimizer are illustrated in Fig. 2. To test and confirm the functionality of this gradient descent algorithm, a battery dataset from the Battery Research Group of the Center for Advanced Life Cycle Engineering (CALCE)[2] was employed. The dataset was compiled from tests performed at various temperatures on an A123 LifePO4 battery. This dataset was selected because experiments were performed by simulating two driving cycles; the dynamic stress test (DST), and the federal urban driving schedule (FUDS).

The gradient descent algorithm was implemented for both driving profiles, and a large number of iterations was chosen for each case because a single-neuron algorithm is used to test the gradient descent algorithm and not a full neural network. 1000 iterations for the DST driving cycle data were completed, and the loss decreased significantly for the first 300 iterations before progressively decreasing till the end of all iterations. After about 920 iterations, the loss was constant at 162.94 which means it has reached the local minima of that neuron. For the FUDS cycle data, 2000 iterations were carried out. Like the DST data, the loss dropped drastically for the first 500 iterations and there was a gradual reduction until the completion of the whole iterations. The cost in the neuron began to converge after 1300 iterations before becoming constant at 168.29. The cost curves for both drive cycles are shown in Fig. 3.

4. Conclusion

This study identifies and investigates the most important parameter of the neural network model; the optimizer. The gradient descent optimization technique was built from scratch and the functionality was tested with a single-neuron NumPy algorithm. The functionality was tested with a battery dataset of two drive cycles. Iterations for both profiles showed cost reduction while converging to the local minimum, and the cost convergence of this algorithm proved that the developed gradient descent algorithm can be effectively implemented in a neural network model for SoC estimation.

This research was supported by the National Institute of Information and Communications Technology Evaluation and Planning with financial resources from the government (Ministry of Science and ICT) in 2022 (No. 2022-1711152629, Functionality of Optimization Techniques in Machine Learning for SoC Estimation)

참 고 문 헌

[1] Shijie Tong, Joseph H. Lacap, Jae W. Park, "Battery state of charge estimation using a load-classifying neural network," Journal of Energy Storage, Vol. 7, No. 2, pp. 236-243, 2016.

[2] Center for Advanced Life Cycle Engineering (CALCE): Lithium-ion Battery Experimental Data. Available: https://www.calce.umd.edu/battery-data (accessed Apr. 07, 2023).