Accelerating Machine Learning-Based Memristor Compact Modeling Using Sparse Gaussian Process

Yuta Shintani and Michiko Inoue
Grad. School of Science and Technology
Nara Institute of Science and Technology
Ikoma, Japan
{shintani.yuta.ta4,kounoe}@is.naist.jp

Michihiro Shintani
Grad. School of Science and Technology
Kyoto Institute of Technology
Kyoto, Japan
shintani@kit.ac.jp

***Abstract*—Research on dedicated circuits for multiply and accumulate processing, which is vital to machine learning (ML), using memristors has attracted considerable attention. However, memristors have unknown operating principles, making it challenging to create compact models with sufficient accuracy. This study proposes a compact modeling method based on Gaussian process for memristors. Although various ML-based modeling methods have been proposed, only the reproduction accuracy has been evaluated using SPICE circuit simulator, and long learning times have not been sufficiently discussed. The proposed method reduces the learning and inference times using a Gaussian process with considering sparsity. An evaluation using data from memristor devices obtained by actual measurements demonstrates that the proposed method achieves over 2,629 times faster than conventional method using long short-term memory (LSTM). Moreover, inference on a commercial SPICE simulator can be performed with the same accuracy and computation time. All experimental environments, including the source code, are available at https://github.com/sntnmchr/SGPR-memristor/blob/main/README.md.**

***Index Terms*—Compact modeling, Memristor, Gaussian process, SPICE simulation**

I. Introduction

Information processing using artificial intelligence, represented by deep learning, is vital to many applications. Most circuits for artificial intelligence applications have been implemented using field-programmable gate arrays (FPGAs) and graphics processing units (GPUs); however, conventional von Neumann-type circuits using silicon CMOS transistors have high power consumption and cannot be expected to improve performance owing to the physical limitations of miniaturization. To solve this problem, memristors have attracted attention as new materials to replace silicon CMOS circuits. Memristors are passive devices whose resistance changes nonlinearly according to the current history [1]. Moreover, memory and calculation can be realized in a single device with low power consumption because they can store resistance non-volatilely, even when no input voltage is applied. Therefore, memristor neural networks that use memristors as synapses (weights) are expected to have a circuit architecture with low power consumption and the ability to solve the von Neumann bottleneck [2]. Memristor neural networks can efficiently perform the sum-of-products operation, the primary operation of neural networks, by implementing memristors in a three-dimensional stack called a crossbar array [3].

When designing a circuit using semiconductor devices, verifying its behavior and characteristics is necessary before manufacturing it using circuit simulators [4]. Accurate circuit simulation requires a high-precision compact model, and various models have been developed. For example, for CMOS integrated circuits, the Berkeley short-channel IGFET model (BSIM) has been widely used as an industry standard and is supported by many commercial SPICE simulators [5]. However, in the case of emerging devices, such as memristors with an unknown operating principle compared with CMOS, most of these compact models involves empirical equations and do not always simulate accurately [6], [7]. Nonetheless, waiting for a complete understanding of the physical behavior hinders the early design and development of competitive circuits with sufficient performance and characteristics.

Many machine learning (ML)-based compact modeling methods have been reported in recent years [8]. These methods are advantageous because a compact model can be created if actual measurement data are available. The generated model from the measurement can be reproduced using SPICE simulator. Reference [9] proposed an ML-based power MOSFET compact modeling method using a Gaussian process [10]. The Gaussian process is a nonparametric method based only on the measured data. Moreover, its fitting result is more accurate than that of the empirical model based on the MOSFET threshold voltage; however, it does not support multistate devices, such as memristors, owing to its hysteresis characteristics. Reference [11] proposed a memristor modeling method using a simple multilayer neural network. However, a large amount of data with random numbers added to the measured data was necessary for learning, requiring substantial time and effort for data preparation. Reference [12] proposed an ML-based memristor compact model using a long short-term memory (LSTM) network [13]. Highly accurate modeling was achieved using the LSTM method by considering the state of the memristor. However, LSTM requires cell and gate optimization during learning, resulting in a long learning time. Nonetheless, discussions on learning time are limited.

Evaluating the training and inference times is necessary when generating ML-based device models. However, most studies only discussed the accuracy of the generated compact model on a SPICE simulator. Moreover, discussions about the generation

979-8-3503-4859-0/DATE24/© 2024 EDAA

(i.e., evaluating the training and inference times using a SPICE simulator) are limited. In particular, the extrapolation did not perform well in typical machine learning algorithms. Learning all operation patterns is necessary when modeling analog devices, such as memristors, using ML, which is unsuitable for extrapolation; modeling methods with a long learning time using LSTM in [12] are a serious issue in actual design.

This study proposes a method for modeling memristor devices using sparse Gaussian process [14]. As Gaussian process is based on Bayesian estimations, accurate modeling can be expected without overfitting. Furthermore, it has the advantage of faster training than the LSTM. However, Gaussian process regression (GPR) requires a long SPICE run time with the generated compact model because the computational order of the inference time is the cube of the number of training samples. Therefore, the proposed method uses sparse GPR, an approximation of GPR, to speed up the inference while maintaining accuracy. The model is generated by combining a block that predicts the next state and a block that predicts the current using the state and voltage of the memristor as the training data. The generated model is implemented in Verilog-A, which is analog hardware description language [15], with the necessary parameters extracted from the trained model and can be executed using a commercial SPICE simulator. Numerical experiments using measured data are performed to compare the performance of the proposed method with that of the LSTM-based method [12], a state-of-the-art method with the same error level and inference time. Unlike the previous works, we comprehensively evaluate the training and inference times as well the prediction accuracy in SPICE simulation.

The remainder of this paper is organized as follows: Section II provides an overview of the Gaussian process regression and sparse Gaussian process regression used in the proposed method and explains the state-of-the-art compact modeling method using LSTM [12] as a conventional method; Section III describes the proposed method; Section IV evaluates the proposed method using the current-voltage (I-V) characteristics of memristor; Finally, Section V concludes the paper.

II. Preliminaries

A. Gaussian process regression

A Gaussian process is a stochastic process with a multivariate Gaussian distribution [10]. The corresponding outputs $\boldsymbol{y} = (y_1, y_2, \cdots, y_N)$ are considered for the inputs $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N)$. When the simultaneous probability $p(\boldsymbol{y})$ can be expressed as a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{K})$ with mean $\boldsymbol{\mu}$ and kernel matrix $\boldsymbol{K}$, the relationship between $\boldsymbol{X}$ and $\boldsymbol{y}$ is called a Gaussian process, where kernel matrix $\boldsymbol{K}$ is given by $K_{nn'} = k(\boldsymbol{x}_n, \boldsymbol{x}_{n'})$ using the kernel function $k(\boldsymbol{x}_n, \boldsymbol{x}_{n'})$ for all input pairs $(\boldsymbol{x}_n, \boldsymbol{x}_{n'})$. In many cases, the data are normalized, and a Gaussian process with mean $\boldsymbol{0}$ is represented as $\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{K})$.

GPR is a model that estimates the function $y = f(\boldsymbol{x})$ from input variable $\boldsymbol{x}$ to output variable y. It is a regression method that can deal with arbitrary functions and a nonparametric method that does not require prior knowledge of the function $f(\cdot)$ because it is a nonlinear model. Moreover, the estimated function is obtained as a predictive distribution because a Bayesian estimation is used, and the uncertainty of the estimation can be expressed.

Algorithm 1 : Gaussian process regression

Input: Training data: $(\boldsymbol{X}_{\rm train}, \boldsymbol{y}_{\rm train})$, Test data: $(\boldsymbol{X}_{\rm test})$, Kernel function: $f_{\rm kernel}$
Output: Mean of predicted values: $\boldsymbol{\mu}$, Variance of predicted values: $\boldsymbol{\upsilon}$

1: Let N be the size of the training data $\boldsymbol{X}_{\rm train}$
2: **for** $n = 1$ to N **do**
3: **for** $n' = 1$ to N **do**
4: $\boldsymbol{K}(n, n') = f_{\rm kernel}(\boldsymbol{x}_n, \boldsymbol{x}_{n'})$
5: **end for**
6: **end for**
7: Optimize hyperparameters of $f_{\rm kernel}$ using $\boldsymbol{X}_{\rm train}, \boldsymbol{y}_{\rm train}$
8: Let M be the size of test data $\boldsymbol{X}_{\rm test}$
9: **for** $m = 1$ to M **do**
10: **for** $n = 1$ to N **do**
11: $\boldsymbol{k}_*(n) = f_{\rm kernel}(\boldsymbol{x}_n, \boldsymbol{x}^*_m)$
12: **end for**
13: $k_{**} = f_{\rm kernel}(\boldsymbol{x}^*_m, \boldsymbol{x}^*_m)$
14: $\boldsymbol{\mu}(m) = \boldsymbol{k}_*^T \boldsymbol{K}^{-1} \boldsymbol{y}_{\rm train}$
15: $\boldsymbol{v}(m) = k_{**} - \boldsymbol{k}_*^T \boldsymbol{K}^{-1} \boldsymbol{k}_*$
16: **end for**

Algorithm 1 shows the computational procedure for the predictive distribution using GPR, comprising a training phase (lines 1 to 7) and an inference phase (lines 8 to 16). In the GPR, the training data are defined as $(\boldsymbol{X}_{\rm train}, \boldsymbol{y}_{\rm train}) = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_N, y_N)\}$, comprising input $\boldsymbol{x}$ and output y pairs and a kernel function $f_{\rm kernel}$ that provides the covariance of the Gaussian distribution. There are several types of $f_{\rm kernel}$. For example, the radial basis function (RBF) kernel, a typical kernel function, is expressed as follows [16]:

$$f_{\rm kernel}(\boldsymbol{x}, \boldsymbol{x'}) = \theta_1 \exp\left(-\frac{|\boldsymbol{x} - \boldsymbol{x'}|^2}{\theta_2}\right), \tag{1}$$

where θ_1 and θ_2 are the hyperparameters to be optimized in line 7 of Algorithm 1. GPR returns the predictions $\boldsymbol{y^*} = (y^*_1, y^*_2, \cdots, y^*_M)$ of the test data $\boldsymbol{X}_{\rm test} = (\boldsymbol{x}^*_1, \boldsymbol{x}^*_2, \cdots, \boldsymbol{x}^*_M)$ using the prediction model computed from $(\boldsymbol{X}_{\rm train}, \boldsymbol{y}_{\rm train})$.

Lines 1 to 6 of Algorithm 1 compute an $N \times N$ kernel matrix $\boldsymbol{K}$ for each element $\boldsymbol{X}_{\rm train}$ using a kernel function. In lines 8 to 16, the output y^*_m for a new input $\boldsymbol{x}^*_m$ of the test data input is derived using the probability density function as follows:

$$\begin{aligned} &p(y^*_m | \boldsymbol{x}^*_m, \boldsymbol{X}_{\rm train}, \boldsymbol{y}_{\rm train}) \\ =&\mathcal{N}(\boldsymbol{k}_*^T \boldsymbol{K}^{-1} \boldsymbol{y}_{\rm train}, k_{**} - \boldsymbol{k}_*^T \boldsymbol{K}^{-1} \boldsymbol{k}_*). \end{aligned} \tag{2}$$

From Eq.(2), the mean $\boldsymbol{\mu} = (\mu_1, \mu_2, \cdots, \mu_M)$ and variance $\boldsymbol{\upsilon} = (\upsilon_1, \upsilon_2, \cdots, \upsilon_M)$ of $\boldsymbol{y}^*$ can be derived analytically. The mean and variance values are used for the prediction and its confidence, respectively.

B. Sparse Gaussian regression

Although the Gaussian process provides a flexible model for many applications, it requires a long inference time when calculated straightforwardly. Because the inverse of the kernel matrix $\boldsymbol{K}$ shown in Eq. (2) must be computed during inference, its time complexity is $\mathcal{O}(N^3)$ when the training data size is N, becoming a computational bottleneck [17], [18]. To overcome this issue, the proposed method employs sparse Gaussian process regression (SGPR), an approximate method of GPR, to perform inference at high speed, even when the dataset size N is large [14].

SGPR introduces inducing points $\boldsymbol{X}^{(\mathrm{u})} = (\boldsymbol{x}_1^{(\mathrm{u})}, \boldsymbol{x}_2^{(\mathrm{u})}, \cdots, \boldsymbol{x}_Z^{(\mathrm{u})})$, where Z ($Z \ll N$) denotes the number of inducing points. The output value $\boldsymbol{u} = f(\boldsymbol{X}^{(\mathrm{u})})$ for the inducing points $\boldsymbol{X}^{(\mathrm{u})}$ is the inducing vector $\boldsymbol{u} = (u_1, \cdots, u_Z)^T$, and the GPR is approximated using $\boldsymbol{u}$. In SGPR, the probability density function of the predicted value y^* corresponding to an unknown input $\boldsymbol{x}^*$ is expressed as follows:

$$\begin{aligned} &p(y^*|\boldsymbol{x}^*, \boldsymbol{X}_{\mathrm{train}}, \boldsymbol{y}_{\mathrm{train}}, \boldsymbol{X}^{(\mathrm{u})}) \\ =&\mathcal{N}(\boldsymbol{k}_{Z*}^T \boldsymbol{K}_{ZZ}^{-1} \widehat{\boldsymbol{u}}, k_{**} - \boldsymbol{k}_{Z*}^T \widehat{\Sigma}_{\boldsymbol{u}}^{-1} \boldsymbol{k}_{Z*}), \end{aligned} \tag{3}$$

where $\widehat{\boldsymbol{u}}$ and $\widehat{\Sigma}_{\boldsymbol{u}}$ are the mean and variance of posterior probability $p(\boldsymbol{u})$ of $\boldsymbol{u}$. Because the inverse matrix $\boldsymbol{K}_{ZZ}^{-1}$ is a $Z \times Z$ matrix, the inverse matrix time complexity $\mathcal{O}(Z^3)$ can be reduced by keeping Z small. Therefore, the SGPR can reduce the inference time by reducing the number of inducing points Z within the required regression accuracy limit.

The SGPR accuracy depends on the number of inducing points and their arrangement. The approximation of inducing points depends on whether $\boldsymbol{u}$ represents the entire distribution of $f(\boldsymbol{X})$. Therefore, arranging $\boldsymbol{X}^{(\mathrm{u})}$ appropriately is necessary to reduce the number of inducing points and maintain a high approximation accuracy. The $\boldsymbol{X}^{(\mathrm{u})}$ arrangement can be optimized simultaneously with the kernel function hyperparameters. In this case, the optimization is performed in line 7 of Algorithm 1.

C. LSTM-based memristor modeling

An ML-based memristor device modeling method [12] based on LSTM [13], a state-of-the-art method, was proposed. This method constructs a state-aware memristor model by preparing blocks to predict the memristor state and current. A compact model from the constructed model is implemented in Verilog-A. The model can be simulated using a commercial SPICE simulator.

LSTM is effective in predicting time-series data and well-suited for modeling transiently changing memristors. The authors reported that highly accurate models were obtained in an evaluation using actual measured data of the manufactured memristors. However, LSTM has a disadvantage because the longer the time-series pattern, the longer the learning time. This is because the calculation results are transmitted from the past to the future in order in the hidden-layer network. Although the model training time should be one evaluation metric in ML-based compact model generation, training and inference times were not discussed in [12].

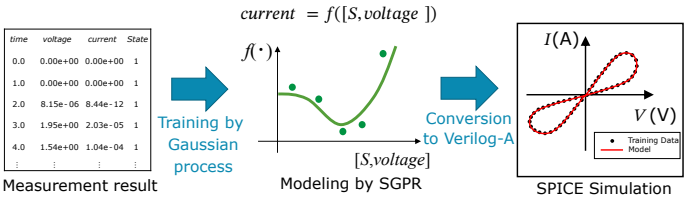


Fig. 1. Overview of the proposed method.

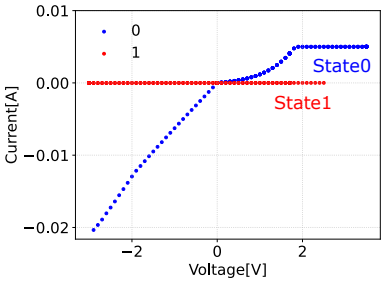


Fig. 2. I-V characteristic of memristor devices.

III. Sparse Gaussian Process-based Memristor Modeling

This study proposes a novel ML-based device modeling method for memristors using SGPR. Fig. 1 shows an overview of the proposed method. The method conducts modeling using the measurement data as training data, followed by conversion to Verilog-A. Applying Gaussian process enables faster learning than the conventional method of [12]. However, using the model generated by the Gaussian process for inference increases computational time. Thus, the execution time of the circuit simulation in SPICE is increased. Therefore, the proposed method introduces SGPR to accelerate the computation time required for the inference. During conversion to Verilog-A, the necessary data for inference are extracted from the trained model and added to Verilog-A, enabling SPICE circuit simulation. Note that SGPR can also calculate the confidence of prediction as shown in Eq. (2). This allows the compact model engineer to verify whether the number and location of measured data samples are sufficient, which is a significant advantage from a practical viewpoint over LSTM used in [12].

Section III-A elaborates on a detailed method for constructing a memristor model using SGPR. Section III-B delineates the specifics of implementing the generated model in Verilog-A.

A. Model generation using SGPR

As shown in Fig. 2, the I-V characteristics of a memristor take different currents between the off and on states, even when the same voltage is applied. Therefore, by defining the off and on states of the memristor as 1 and 0, respectively, the state S is introduced to model the I-V characteristics of the memristor. Two blocks are designed to develop this model employing state S: one to predict the next state S_{t+1}^P, and the other to forecast

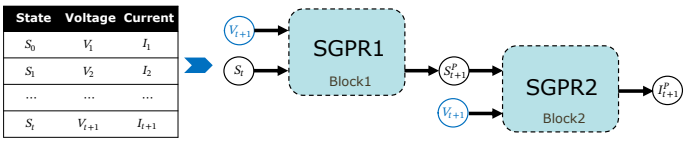


Fig. 3. Structure of memristor model considering state.

Algorithm 2 Selection method for inducing points

Input: Training dataset: $(\boldsymbol{S}, \boldsymbol{V})$,
Number of inducing points: Z.
Output: Inducing points:$\boldsymbol{X}^{(\mathrm{u})} = (\boldsymbol{x}_1^{(\mathrm{u})}, \boldsymbol{x}_2^{(\mathrm{u})}, ..., \boldsymbol{x}_Z^{(\mathrm{u})})$

1: Let $\boldsymbol{V}^0$ be the voltage data set when S is 0
2: Obtain $\boldsymbol{X}^{V0}$ values with the number $Z/2$ equally spaced between $\max(\boldsymbol{V}^0)$ and $\min(\boldsymbol{V}^0)$
3: Let $\boldsymbol{V}^1$ be the voltage data set when S is 1
4: Obtain $\boldsymbol{X}^{V1}$ values with the number $Z/2$ equally spaced between $\max(\boldsymbol{V}^1)$ and $\min(\boldsymbol{V}^1)$
5: **for** $z = 1$ to Z **do**
6: **if** $z \leq Z/2$ **then**
7: $\boldsymbol{x}_z^{(\mathrm{u})} = (0, \boldsymbol{X}_z^{V0})$
8: **else**
9: $\boldsymbol{x}_z^{(\mathrm{u})} = (1, \boldsymbol{X}_z^{V1})$
10: **end if**
11: **end for**

the subsequent current I_{t+1}^P. Fig. 3 shows the structures of these two blocks. Their corresponding equations are as follows:

$$f_{\mathrm{SGPR1}}(S_t, V_{t+1}) = S_{t+1}^P \quad (4)$$

$$f_{\mathrm{SGPR2}}(S_{t+1}, V_{t+1}) = I_{t+1}^P, \quad (5)$$

where S_t and S_{t+1}^P represent the prior and ensuing states, respectively. The first block serves as a prediction module for the next state and is a function of S_t and V_{t+1}. The second block operates as a current prediction module that acts as a function of S_{t+1}^P and V_{t+1}. The proposed method assigns state S to the measured I-V characteristics in advance.

The proposed method uses SGPR to model the predictions as per Eqs. (4) and (5). SGPR reduces inference time by incorporating inducing points $\boldsymbol{X}^{(\mathrm{u})}$. The number of inducing points Z is set to be less than the number of training data points N to minimize the computational time. This method adopts the approach of optimizing the inducing points simultaneously with the kernel function hyperparameters. The extent to which the inducing points represented the entire distribution determined the accuracy of the approximation of the inducing points. Therefore, the initial positions of the inducing points were selected uniformly from the training data, considering the optimization. Uniform selection is expected to improve convergence speed during optimization.

Algorithm 2 outlines the method for selecting the inducing points. The first line of the algorithm extracts the voltage dataset $\boldsymbol{V}^0$, where S equals 0, from the training data set $(\boldsymbol{S}, \boldsymbol{V})$. The second line procures values at equal intervals from the maximum and minimum values of $\boldsymbol{V}^0$; specifically, $Z/2$ number of values. Lines 3 and 4 replicate the procedures of lines 1 and 2, respectively, for instances where S equals 1. Lines 5 to 11 set the inducing points according to the obtained equal-interval voltage values $\boldsymbol{X}^{V0}$ and $\boldsymbol{X}^{V1}$ and the state of S. The positions of the selected inducing points were optimized concurrently with the hyperparameters $\boldsymbol{\theta}$ during the training process. After optimization, Eq. (3) can be used to predict the current values for specific states and voltages.

B. Implementation using Verilog-A

After the training, the memristor model in Fig. 3 is implemented in Verilog-A for SPICE simulation. The model using SGPR requires implementing Eq. (3) for the prediction, implying that this equation must be incorporated into Verilog-A. The unknown input value is denoted as $\boldsymbol{x}^*$ and its predicted value as y^*. Simplifying Eq. (3), the mean μ of the predictive result $p(y^*)$ of SGPR can be expressed as follows:

$$\mu = \boldsymbol{k}_{Z*}^T \boldsymbol{W}, \quad (6)$$

where the vector $\boldsymbol{W}$ is the calculated value of $\boldsymbol{K}_{ZZ}^{-1}\widehat{\boldsymbol{u}}$. In addition, $\boldsymbol{k}_{Z*}^T$ is a matrix obtained by applying the kernel function f_{kernel} to the inducing points $\boldsymbol{X}^{(\mathrm{u})}$ and test data $\boldsymbol{x}^*$. Thus, the three data points required for computing μ using SGPR are as follows:

- The inducing points $\boldsymbol{X}^{(\mathrm{u})}$ optimized during modeling
- Hyperparameters $\boldsymbol{\theta}$ of the kernel function f_{kernel}
- The vector $\boldsymbol{W}$

In the Verilog-A model, $\boldsymbol{k}_{Z*}^T$ is first calculated, followed by $\boldsymbol{k}_{Z*}^T \boldsymbol{W}$. In Verilog-A, the above calculations are implemented by combining arrays and loops. The Verilog-A code used in Section IV can be found at https://github.com/sntnmchr/SGPR-memristor/blob/main/verilog/rram_gp.va.

IV. Numerical Experiments

Experiments were performed using measured data from the memristor used in [12] to verify the effectiveness of the proposed method. For comparison, the memristor was modeled using a conventional method with LSTM [12], and the learning time, error, and inference time were compared. The generated compact model was implemented in Verilog-A and simulated using a commercial SPICE simulator [19].

A. Setup

The measured data for the memristor used in [12] can be obtained online as described in [20]. The measured results for the TaN/HfOx/Pt memristor are presented here. In the measurements, positive and negative bias voltages were applied to the memristor, and low-and high-resistance states were sequentially observed. The measurement data comprised four values for each measurement point: time, voltage, current, and status. The input voltage and output current ranged from $+1.2\,\mathrm{V}$ to $-1.5\,\mathrm{V}$ and from $+1.0\,\mathrm{mA}$ to $-1.0\,\mathrm{mA}$, respectively. Two data types were used: 812 and 2,456 measurement points, which are shown in Section IV-B. Data from 812 measurement points were used as training data for the modeling. The other is used as inference data. Note here that the inference data is larger and

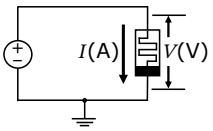


Fig. 4. Simulation circuit of memristor in SPICE simulator.

TABLE I
COMPARISON OF TRAINING TIME AND ERROR

Method	Training time [sec]	RMSE [mA]	RMSLE
LSTM [12]	1412.139	0.0115	0.0115
GPR	2.283	0.0124	0.0122
SGPR(600)	1.375	0.0127	0.0125
SGPR(200)	0.603	0.0128	0.0126
SGPR(50)	0.537	0.0148	0.0148

more complex than the training data, considering its application to real-world compact modeling.

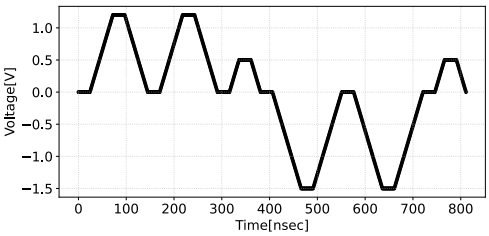
In this experiment, as described in Section III-A, inducing points $\boldsymbol{X}^{(\mathrm{u})}$ were introduced to use the SGPR. A value smaller than 812 points was selected because the number Z of $\boldsymbol{X}^{(\mathrm{u})}$ must be smaller than the number of training data points. The experiments were performed with $Z = 600, 200$, and 50. The proposed method used the RBF kernel for modeling, shown in Eq. (1).

All programs were implemented in Python and ran on a computer equipped with an Intel(R) Xeon(R) Platinum 8160 CPU @ 2.10 GHz. The Pyro package was used to build the SGPR model [21], and the TensorFlow package was used to implement the conventional method [22]. For the inference evaluation of the generated compact model, data from 2,456 measurement points were used as the inference data. The SPICE simulation was employed to perform a transient circuit analysis for the inference, as shown in Fig. 4.

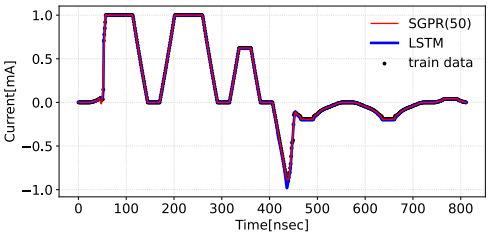
B. Results

1) Training: The conventional and proposed methods generated the models using the training data. Fig. 5 shows the simulation results calculated with the training data. A case in which the SGPR had 50 inducing points is presented as an example. The black dots represent the training data, the blue line is the conventional method model, and the red line is the proposed method model. Figs. 5(a) and 5(b) show the applied voltage and output current in the training data, respectively. In addition, Fig. 5(c) shows the I-V characteristics, and Fig. 5(d) shows a three-dimensional graph with the state variable added as the axis. In Fig. 5, the points where the current changed rapidly are those where the memristor state changed; however, the models were successfully built according to the training data.

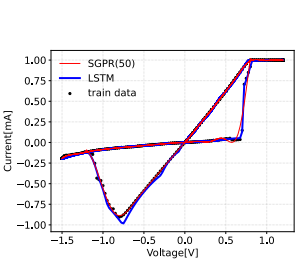
Table I compares the training times and errors for each method. The results for the proposed method are presented for the $Z = 600, 200$, and 50 for the SGPR. In addition, the results of a standard GPR that does not consider sparsity is presented as a reference. The training time of the conventional method was more than 1,400 s as discussed in Section II-C, whereas that of the proposed SGPR method was less than 1 s when the numbers of inducing points were 200 and 50. By contrast,



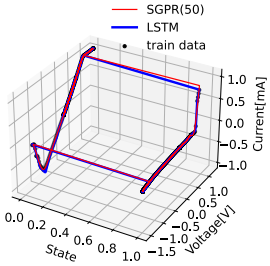
(a) Input voltage.



(b) Output current.



(c) Current-Voltage.



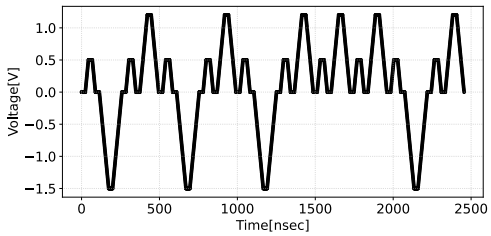
(d) Current-Voltage-State.

Fig. 5. Simulation results for training data.

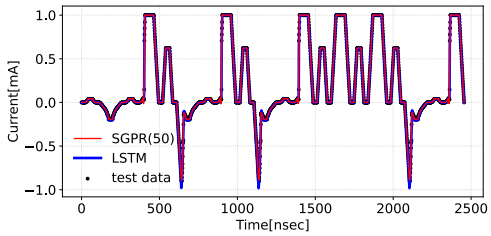
the root mean squared error (RMSE) and root mean squared logarithmic error (RMSLE), which are learning errors, were comparable. Thus, the proposed method improved the training time by 2,629 times that of the conventional method with the same accuracy level.

2) Inference: The SPICE circuit simulation results were compared using the inference data for each model generated by the conventional and proposed methods. Fig. 6 shows the simulation results obtained using SPICE with the inference data. A case in which the SGPR had 50 inducing points is presented. The black dots represent the inference data, the blue lines represent the conventional method model, and the red lines represent the proposed method model. In Fig. 6, the transient changes of the applied voltage and output current and the I-V characteristics are shown as well as Fig. 5. These figures show that the I-V characteristics of the actual measurement data can be reproduced by inference using the training model. In addition, the switching of states can be reproduced, as in the measured data.

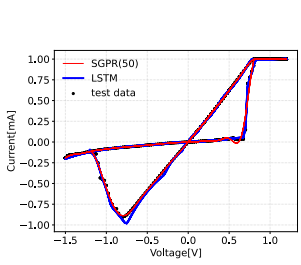
Table II summarizes the comparison results of the inference time and error using SPICE for each method. The conventional method and GPR inference time were 0.45 s and 1.85 s, respectively. However, the inference time can be shortened by introducing sparsity and reducing the number of inducing points; the inference time was approximately the same with 50 inducing



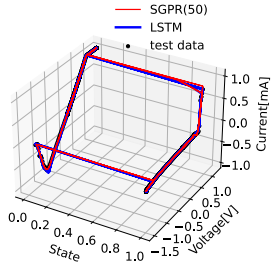
(a) Input voltage.



(b) Output current.



(c) Current-Voltage.



(d) Current-Voltage-State.

Fig. 6. Simulation results for inference data.

TABLE II
COMPARISON OF INFERENCE TIME AND ERROR BY SPICE SIMULATION

Method	Inference time [sec] (SPICE runtime)	RMSE [mA]	RMSLE
LSTM [12]	0.45	0.0120	0.0120
GPR	1.85	0.0141	0.0138
SGPR(600)	1.35	0.0144	0.0142
SGPR(200)	0.70	0.0145	0.0143
SGPR(50)	0.39	0.0167	0.0167

points. Moreover, the RMSE and RMSLE, representing the error, remained approximately the same. Thus, the inference time of the proposed method was approximately the same as that of the conventional method.

In summary, the experimental results, including the training and inference, show that the proposed method can learn more than 2,629 times faster than the conventional method and the accuracy and SPICE runtime of the proposed method are comparable to the conventional method in inference.

V. CONCLUSION

This study considered a fast modeling method for memristor devices using SGPR. SGPR enables faster learning than existing methods and less inference time than GPR. The proposed method uses the state and voltage of the memristor as training data. Moreover, it combines a block for predicting the next state and another for predicting the current to realize a state-aware model. The generated model can be integrated into commercial SPICE simulators by extracting the necessary parameters and implementing them in Verilog-A. An evaluation of the model using measured data from the memristor demonstrated that the proposed method achieved results 2,629 times faster than the conventional method using LSTM, with a comparable error level and inference time. The proposed method is significant for designing and developing competitive circuits with sufficient performance and characteristics even when the device physics is unknown for compact modeling.

ACKNOWLEDGMENT

This work was partially supported by JSPS KAKENHI Grant, No. 22K11954.

REFERENCES

[1] L. Chua, "Memristor—the missing circuit element," *IEEE Trans. on Circuit Theory*, vol. 18, no. 5, pp. 507–519, 1971.
[2] M. N. Bojnordi and E. Ipek, "Memristive boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning," in *Proc. HPCA*, 2016, pp. 1–13.
[3] J. J. Yang *et al.*, "Memristive devices for computing," *Nat. Nanotech.*, vol. 8, pp. 13–24, 2013.
[4] C. Hu, "Compact modeling for the changing transistor," in *Proc. SISPAD*, 2013.
[5] Y. S. Chauhan *et al.*, "BSIM6: Analog and RF compact model for bulk MOSFET," *IEEE Trans. on Electron Devices*, vol. 61, no. 2, pp. 234–244, 2014.
[6] P.-Y. Chen and S. Yu, "Compact modeling of RRAM devices and its applications in 1T1R and 1S1R array design," *IEEE Trans. on Electron Devices*, vol. 62, pp. 4022–4028, 2015.
[7] Y. Zhao *et al.*, "A compact model for drift and diffusion memristor applied in neuron circuits design," *IEEE Trans. on Electron Devices*, vol. 65, pp. 4290–4296, 2018.
[8] F. Klemme *et al.*, "Modeling emerging technologies using machine learning: Challenges and opportunities," in *Proc. ICCAD*, 2020.
[9] K. Shimozato and T. Sato, "dGPLVM: A nonparametric device model for statistical circuit simulation," in *Proc. ICMTS*, 2022.
[10] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006. [Online]. Available: http://gaussianprocess.org/gpml/chapters/RW.pdf
[11] J. Hutchins *et al.*, "A generalized workflow for creating machine learning-powered compact models for multi-state devices," *IEEE Access*, vol. 10, pp. 115 513–115 519, 2022.
[12] A. S. Lin *et al.*, "RRAM compact modeling using physics and machine learning hybridization," *IEEE Trans. on Electron Devices*, vol. 69, no. 4, pp. 1835–1841, 2022.
[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[14] M. K. Titsias, "Variational learning of inducing variables in sparse Gaussian processes," in *Proc. AISTATS*, 2009.
[15] C. C. McAndrew *et al.*, "Best practices for compact modeling in Verilog-A," *IEEE J. Electron Devices Soc.*, vol. 3, no. 5, pp. 383–396, 2015.
[16] D. Duvenaud, "The kernnnnnel cookbook." [Online]. Available: https://www.cs.toronto.edu/~duvenaud/cookbook/
[17] S. Park and S. Choi, "Hierarchical Gaussian process regression," in *Proc. ACML*, 2010, pp. 95–110.
[18] D.-T. Nguyen, M. Filippone, and P. Michiardi, "Exact Gaussian process regression with distributed computations," in *Proc. SAC*, 2019, pp. 1286–1295.
[19] *HSPICE User Guide: Basic Simulation and Analysis Version P-2019.06*, Synopsys, Inc., 2019.
[20] A. S. Lin, "rram_lstm." [Online]. Available: https://github.com/albertlin11/RRAMunif
[21] E. Bingham *et al.*, "Pyro: Deep universal probabilistic programming," *Journal of Machine Learning Research*, vol. 20, pp. 28:1–28:6, 2019.
[22] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/