HDK: Toward High-Performance Deep-Learning-Based Kirchhoff Analysis

Xinying Wang, Olamide Timothy Tawose, Feng Yan, Dongfang Zhao
University of Nevada
Reno, NV 89557, USA
{xinyingw,otawose}@nevada.unr.edu, {fyan,dzhao}@unr.edu

Abstract

The Kirchhoff law is one of the most widely used physical laws in many engineering principles, e.g., biomedical engineering, electrical engineering, and computer engineering. One challenge of applying the Kirchhoff law to real-world applications at scale lies in the high, if not prohibitive, computational cost to solve a large number of nonlinear equations. Despite recent advances in leveraging a convolutional neural network (CNN) to estimate the solutions of Kirchhoff equations, the low performance is still significantly hindering the broad adoption of CNN-based approaches. This paper proposes a high-performance deep-learning-based approach for Kirchhoff analysis, namely HDK. HDK employs two techniques to improve the performance: (i) early pruning of unqualified input candidates and (ii) parallelization of forward labelling. To retain high accuracy, HDK also applies various optimizations to the data such as randomized augmentation and dimension reduction. Collectively, the aforementioned techniques improve the analysis speed by 8× with accuracy as high as 99.6%.

Introduction

In various disciplines such as medical engineering and healthcare devices, scientists can only measure the end-to-end electrical resistance values that are derived from the complex, nonlinear transformation of individual resistances (Niu et al. 2018b). Nonetheless, the objective of many domain-specific applications is to find intrinsic resistance values, and the orthodox approach is to solve the system of large numbers of nonlinear equations formed according to the Kirchhoff law (Kirchhoff Law 2019).

Solving a Kirchhoff-based system of nonlinear equations poses two technical challenges: (i) the root of unknowns is not unique because the unknowns appear at the denominators of the equations; (ii) the computation for finding the root takes a long, sometimes prohibitive, time. For instance, Niu et al. (Niu et al. 2018a) reported that forming the Kirchhoff equations would take hundreds of days even for a small-scale 40×40 electrode array. Although (Niu et al. 2018a) presented an algorithm to reduce the number of equations from exponential to polynomial, how to efficiently solve them remains an open challenge to both the biomedical engineering and parallel computing communities.

Inspired by the recent advances in deep learning (DL), researchers started to seek non-analytic paradigms to *estimate* the solution, e.g., training a convolutional neural network (CNN) to accurately predict the unknown resistor distribution in an electrode array (Tan et al. 2019). This approach demonstrated to be an effective means to "learn" the nonlinear function between inputs and outputs, as the error rate is reported as low as 0.49%.

However, before the CNN-based approach can be practically adopted, one critical issue concerning performance must be addressed. Specifically, the CNN-based approach (Tan et al. 2019) simply *simulates* the input and output data for the neural network, assuming the training set is known a priori, which is not the case in the real world. In fact, it is one of the most challenging problems to efficiently obtain the training data set for an electrode array due to the computational complexity of Kirchhoff nonlinear equations: As shown by Niu et al. (Niu et al. 2018a), a commodity workstation took less than one minute to generate all the nonlinear equations for a 20×20 electrode array, but then the time increased to more than four hours for a 100×100 electrode array—prohibitively slow in the practice. This is partly why in (Tan et al. 2019) authors *simulated* tens of thousands of training data and tested it on a small-scale 16×15 electrode array[1]. The slow performance has significantly hindered CNN-based Kirchhoff analysis from being broadly adopted in real-world applications.

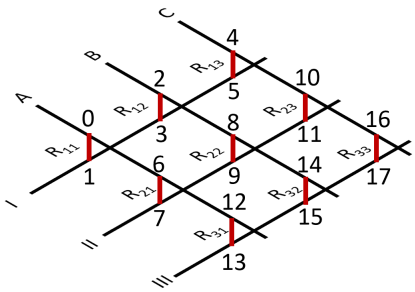
To overcome the above challenge, this paper proposes a high-performance DL-based method for Kirchhoff analysis, namely HDK. HDK employs two specific techniques to improve the performance of DL-based Kirchhoff analysis: (i) early pruning of unqualified candidate inputs and (ii) parallelization of forward labelling. To retain the high accuracy of the DL models, HDK also introduces random errors to augment the training set and at the same time applies dimension

[1] In literature, an electrode array can be organized in a ring or a grid topology. Of note, in (Tan et al. 2019), the array is organized as a 16-electrode ring, which is equivalent to a 16×15 two-dimensional array referred to by (Niu et al. 2018a).

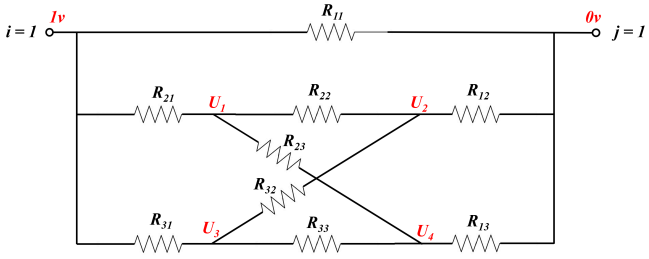
reduction to the original data. Collectively, the aforementioned techniques reduce the analytical time from hours to 15 minutes with accuracy as high as 99.6%.

Problem Formulation

Formally, let vector $\vec{Z}$ represent the set of measured values, the goal is to find the vector $\vec{R}$ that satisfies $\vec{Z} = K(\vec{R})$, where $K(\cdot)$ indicates the Kirchhoff law. The Kirchhoff law states that *the aggregation of the incoming current flows must be equal to the aggregation of the outgoing ones*. The law must be applied to each joint on an electronic device, therefore usually comprising a system of equations, which is why the constraints are represented by vectors.



(a) Electrode array with $n = 3$



(b) Converted vertex-based topology (for one path)

Figure 1: A 3×3 electrode array, where each of the 18 vertices follows the Kirchhoff law.

To make matters more concrete, we exemplify $\vec{Z} = K(\vec{R})$ in a 3×3 electrode array, as shown in Figure 1(a). One set of axes (A, B, and C) are interconnected to another set (I, II, and III) on nine resistances: $R_{11}, \cdots, R_{33}$. In scientific experiments, we could only measure the resistant values from the endpoints, indicated by Z_{ij}'s. For instance, $Z_{B,II}$ is the measured resistance value between axis B and axis II. An equivalent, vertex-oriented topology can be achieved; Figure 1(b) shows the converted topology for a specific path between the axis A (i.e., $i = 1$) and axis I (i.e., $j = 1$). The actual voltage will not matter as the measurement is all on the resistances; for the sake of simplicity, we assume the source voltage is 1 volt, and the end (ground) voltage is 0 volt, as indicated in Figure 1(b). Let R_{ij} indicate the resistance on the i-th x-axis and the j-th y-axis. Without loss of generality, the end-to-end measured resistance on the first x-axis and the first y-axis must satisfy the following system of equations as per the Kirchhoff law:

$$\begin{cases} \frac{1-U_1}{R_{21}} = \frac{U_1-U_2}{R_{22}} + \frac{U_1-U_4}{R_{23}} \\ \frac{U_2}{R_{12}} = \frac{U_1-U_2}{R_{22}} + \frac{U_3-U_2}{R_{32}} \\ \frac{1-U_1}{R_{31}} = \frac{U_3-U_2}{R_{32}} + \frac{U_3-U_4}{R_{33}} \\ \frac{U_4}{R_{13}} = \frac{U_1-U_4}{R_{23}} + \frac{U_3-U_4}{R_{33}} \\ \frac{1}{Z_{11}} = \frac{1}{R_{11}} + \frac{U_2}{R_{12}} + \frac{U_4}{R_{13}} \end{cases} \quad (1)$$

where U_k ($k \in \{1, 2, 3, 4\}$) indicates one of intermediate voltages at joints, as illustrated in Figure 1(b). The first four equations represent the four Kirchhoff constraints, and the last equation calculates the composite resistance between $i = 1$ and $j = 1$.

In practice, only Z's can be measured, and all U's and R's are unknowns. Although our goal is to find R values, voltage values U's have to be found as well. Note that Equation 1 speaks of the relation between a single pair of $i = 1$ and $y = 1$ only. The entire network would have nine systems of nonlinear equations in the form of Equation 1, making the total number of constraints equals 45. Also, note that in each equation we have four unknown U's, and there are nine unknown R's shared by all nine equations. Therefore, there are a total of 45 unknowns as well. That is, we have the same number of unknowns and constraints—a well-defined system of equations to solve. In a more general sense, there are $n^2(2n-1)$ constraints and $n^2(2n-1)$ unknowns for a given $n \times n$ electrode array.

However, the problem is proven ill-posed, meaning that the unknown variables cannot be uniquely and stably reconstructed from measurements alone (Barber and Brown 1984). If we look closer at the unknowns in Equation 1, the majority comes from the voltages, i.e., $n^2(2n-2)$ of U's, and only a small fraction of them are R's: n^2; unfortunately, such a small fraction of R's (i.e., $\frac{1}{2n-1}$) are the root cause of the technical challenge because they all appear in the denominators, which makes the entire system of equations *nonlinear*. The problem then becomes ill-posed because:

- **Non-uniqueness of Root.** According to the rank of an augmented matrix, we could determine whether a system of linear equations has no root, a unique root, or infinite numbers of roots. For a system of nonlinear equations like Equation 1, we know there must be at least one root because they follow the physical Kirchhoff law. Literature (Semenov 2015) shows that there exist ways to find all possible roots; and yet, in this application, there should be only one real root, and we have no computational means to verify whether the found root is the real one.

- **Computational Complexity.** Unlike a system of linear equations, most algorithms for solving nonlinear equations are problem-dependent and a lot more complicated. Literature (Niu et al. 2018a) shows that solving a system of nonlinear equations derived from a 20×20 electrode array takes more than five hours. In addition, the time-scale trend follows an exponential pattern: the solving procedure takes about 10 seconds for a 10×10 electrode array. Note that, in practice, the scale is much larger than 20;

our device prototype is 64×64. The computation time increases 2,000 times from $n = 10$ to $n = 20$, and will not be acceptable at $n = 64$.

Methodology

Early Pruning: Upper-Bound of Input Parameters

In theory, the input data Rs can be arbitrary, although the values are usually within a specific range depending on the applications. That is, the parameter space of Rs is infinite: any positive real numbers are eligible. How to select a meaningful input data range remains an open problem. What we need here is to prune off the impossible candidates, in the context of electrode arrays, before selecting the training data set Rs. We thus must explore the intrinsic properties embedded in this specific engineering application. In the remainder of this section, we will pursue an analytical model for estimating the input Rs' range derived from the measured Z values.

To start with, we introduce a set of parameters, which constructs a parameterized model for the topology shown in Figure 1(b). Instead of having a 3×3 array, we assume the dimension n is sufficiently large for practical usages, e.g., $n = 64$ in our prototype device. It follows that there are $(n-1)$ horizontal paths between a starting and ending endpoints. Each of these horizontal paths comprises three resistors with two distinct voltages in between, namely U^k_{left} and U^k_{right}, where k ($1 \le k \le n-1$) indicates the path index, $left$ indicates the left voltage and $right$ indicates the right voltage (assuming the current flows from left to right). It should be clear that the three resistors on those $(n-1)$ have nothing to do with the dimension of the sample 3×3 electrode array: all these paths would have three resistors anyways regardless of the dimension n (Niu et al. 2018a). In addition to the horizontal paths, there are $(n-2)$ "cross" paths starting from each U^k_{left}.

If we look closely at the topology from endpoint i to endpoint j in Figure 1(b), the current at U_1 is diluted into two currents toward R_{22} and R_{23}. On the other hand, the current at U_2 is aggregated from currents through R_{22} and R_{32}. Therefore, we observe a distributive and symmetric divide-and-conquer workflow in this engineering application. If we assume, for the sake of analysis simplicity, the resistors follow a uniform distribution in a general $n \times n$ electrode array, then the current must satisfy the following

$$I_{ik} = (n-1) \cdot I_{kk} = I_{kj}$$

where we denote the start point as i, end point as j, and I_{ik} as the current between i and U^k_{left}. According to Ohm law, it follows that

$$U^k_{right} = (n-1) \cdot (U^k_{left} - U^k_{right})$$

Consequently, if n is sufficiently large in practice, we then have $U^k_{left} \approx U^k_{right}$. An important implication is then the resistors between both intermediate voltage points are computationally negligible. Formally, we have

$$\lim_{n \to \infty} R_{kk} = 0 \tag{2}$$

for a current topology from i to j where $k \neq i$ and $k \neq j$. It should be clear that this conclusion is for the computation between a specific pair of endpoints; it does not imply the physical resistor is cut down to zero at large scales.

Equation 2 lays out a cornerstone for further analysis of the targeting parameter space. With Equation 2, we could safely eliminate all the Rs in the intermediate positions, i.e., those that are not adjacent to either endpoint. Back to the example in Figure 1(b), it means that we can remove R_{22}, R_{23}, R_{32}, and R_{33} when we compute the current from $i = 1$ to $j = 1$. It follows that the simplified topology now has only n horizontal paths: the top one comprises a single R_{ij}, and each of the remaining $(n-1)$ horizontal paths comprises two resistors R_{ik} and R_{kj}. It follows that, again, if we assume a uniform distribution of R, then according to the Kirchhoff law between endpoints i and j:

$$\frac{1}{Z} = \frac{1}{R} + (n-1) \cdot \frac{1}{2R}$$

or,

$$R = \frac{n+1}{2} \cdot Z \tag{3}$$

Therefore, we can pick the training data Rs in the range of $\left(0, \frac{(n+1) \cdot Z}{2}\right]$, where Z is the measured value and n is the array size.

Parallel Forward Labelling

We propose to label the data set in the forward direction, which is parallelized with multiple processes. Conventionally, if we have a general relationship $K(\cdot)$ between $\vec{R}$ and $\vec{Z}$ s.t. $\vec{R} = K(\vec{Z})$ where $\vec{Z}$ is the set of measured values and $\vec{R}$ is the set of unknown individual resistances, then we must have obtained the $\vec{R}$ calculated from $K(\cdot)$ along with the $\vec{Z}$. As discussed above, $K(\cdot)$ is prohibitively expensive to calculate. Therefore, we ask: can we obtain both $\vec{R}$ and $\vec{Z}$ through the inverse function of $K(\cdot)$, namely $K'(\cdot)$, such that:

$$\forall r \in \vec{R} : z = K'(r) \;\; \texttt{iff} \;\; r = K(z)$$

and we hope that the inverse function $K'(\cdot)$ is significantly easier to solve than $K(\cdot)$. This is known as a forward problem.

If we reexamine Equation 1, the inverse function $\vec{Z} = K'(\vec{R})$ consists of only one unknown in the denominator: Z_{11}. In fact, we could solve all the unknown U's in the first four equations and calculate Z_{11} trivially. In other words, the inverse function $K'(\cdot)$ can be built by solving n^2 systems, each of which comprises $(2n-2)$ linear equations and $(2n-2)$ unknowns. This is the simplest form for solving systems of equations: all unknown coefficients are linear, and the number of unknowns is equal to the number of equations. Therefore, a much simpler (inverse) function becomes available for generating the training set.

With a linear number of unknowns and the new topology, our next optimization is to parallelize the axis-oriented Kirchhoff equations. There are n^2 possible end-to-end measured resistance values from an $n \times n$ electrode array. Even if we have reduced the number of equations from n^2 to $2n-2$,

there are still a total of $2n^2(n-1)$ equations from the entire array—we hope that parallelization could significantly reduce the time for processing these many equations.

The key challenge is how to isolate the shared z-axes between different paths across pair-wise endpoints, such that these paths can be calculated in parallel. Although each z-axis is touched by a single x- and y-axis, respectively, when those unknowns are boiled down to the Kirchhoff equations, all of the unknowns are interconnected and cannot be trivially parallelized.

Our approach to parallelizing the Kirchhoff equations is to leverage an essential property of circuits: as long as the voltage and underlying individual resistance are kept unchanged, the measured end-to-end resistance and circuits value are also unchanged. That is to say, regardless of how many times we measure the network, the results should be the same; it does not matter whether we measure all the n^2 end-to-end resistance values at once or n^2 times—basically one single pair of (i, j) each time, $1 \leq i, j \leq n$. Therefore, we could parallelize the $2n^2(n-1)$ equations along with the combinations of x- and y-axis. Indeed, this parallelization is still somewhat coarse-grained: we can parallelize the n^2 factor and each thread still needs to process $(2n-2)$ equations. In practice, however, n^2 could be a fairly large number and implies a high degree of parallelism. Take our device prototype for example again where $n = 64$, the proposed approach can be parallelized by up to 4,096 threads. Although linear scalability is unlikely in practice due to various overhead, n^2 threads imply strong parallelism; we will report quantitative results in the evaluation section when we apply the message passing interface (MPI) to the equations.

Augmentation through Random Errors

In real-world engineering applications, measurement errors are the norm to exist. To take this factor into account, we employ a procedure to randomized the $\vec{Z}$'s when we train the model. Another benefit of doing so is the fast augmentation of the training set. As discussed before, the training set is parallelly generated from the inverse calculation based on the Kirchhoff law. Nevertheless, it still takes tens of minutes to find a single mapping between a 64×64 $\vec{R}$ and $\vec{Z}$, as we will see in the evaluation section.

One tricky question for taking this approach is how much randomization should be applied. In this context, the degree of randomization is two-fold: (i) for a single element, what is the upper and lower bound of the randomized input and output; and (ii) how many new inputs and outputs should be generated from the randomization. In engineering applications, 5% errors are usually acceptable; we control all the randomized data within 1% of the exact values from Equation 1. From the application's perspective, there is no hard requirement over the number of new inputs and outputs. However, the number of extended mappings between $\vec{R}$ and $\vec{Z}$ might significantly influence the model accuracy, as we will see in the evaluation.

Dimension Reduction

One common challenge in data analysis of real-world engineering applications lies in the high dimension. For instance, the electrode arrays from our wet lab are 64×64, totaling in 4,096 dimensions. Instead of directly training over these 4,096 features, we apply principal component analysis (PCA) to the measurements and hope to improve the accuracy of the model.

The critical question in the proposed approach is how many dimensions we want to reduce the original data. In theory, the dimensions should be collectively tuned by both PCA and DL models. That is, the PCA should remain much information (usually 95+%) with a small subset of dimensions and at the same time, such a small set of dimensions result in a few neurons in the DL model leading to high accuracy in predicting the intrinsic resistance values R's. We will experimentally show that the optimal choice of the reduced dimensions will be application-dependent, and usually resides close to the point where less than 1% – 5% information is lost from the PCA.

Evaluation

System Implementation

We have implemented the proposed method, mainly with Python and MPI, and released the source code hosted at Github: https://github.com/hpdic/HDK_On_Electrode_Arrays.

Experimental Setup

Test Bed Our experiments are primarily carried out on the Amazon Web Services (AWS). Table 1 lists the instance types for the evaluation. For the parallel forward labelling experiment, we use both of the *t2.2xlarge* and the *c5.18xlarge* instances. For model training, we use only the *t2.2xlarge* instance. All instances are installed with Ubuntu 18.04, Anaconda 2019.3, Python 3.7, NumPy 1.15.4, SciPy 0.17.0, mpi4py v2.0.0, and mpich2 v1.4.1.

Table 1: AWS instances used for evaluation.

Instance	CPU	Memory
t2.2xlarge	Intel Xeon E5-2686	32 GB
c5.18xlarge	Intel Xeon Platinum 8124M	144 GB

Data Sets and Baseline Systems We evaluate the proposed approach using two data sets. Both data sets are collected from our wet lab with measured Zs using 64×64 electrode arrays. The core data set comprises 100 data points, which are augmented into 1,400 data points through up-to-1% random errors.

According to Equation 3, the input Rs should be in the range of $\frac{64+1}{2} = 32.5\times$ of the measured Zs. We name the first data set as *cell medium*, representing the estimated original range of data in a biological experiment. Specifically, the *cell medium* data set has its R's about $15\times$–$25\times$ of the measured values. The second data set is called *wound surface*, representing an application where the electrode array

is applied to a patient's wound skin in a clinic. The typical R's values are about 25×–35× of the measured Zs.

For both data sets, we hold out 20% of the data for prediction, and the remaining 80% for training. We use 10-fold cross-validation and report the variance in the error bar if it is noticeable.

We compare the proposed work with two baseline systems recently published: (i) the conventional Kirchhoff analysis on electrode arrays (Niu et al. 2018a) and (ii) a CNN-based Kirhohoff analysis on electrode rings (Tan et al. 2019). The metrics include scalability, accuracy, and performance.

DL Models The models are trained through Keras (Keras 2019), a high-level API built upon TensorFlow (TensorFlow 2019). Models in Keras are defined as a sequence of layers. We create a sequential model and add layers one at a time until the network reaches high accuracy. The number of inputs are initially set to $input\_dim = 512$ for both models. In both models, we use a fully-connected network structure with two hidden layers, and the number of neurons is 2,048. Table 2 illustrates the network architecture of the DL models.

Table 2: DL network architecture for 64×64 arrays.

Layer	Input	Dense-1	Dense-2	Output
Neuron	100–1,400	2,048	2,048	100–1,400

During the training, we initialize the network weights with a custom initializer ($He\ initialization$) as we use the rectifier $relu$ activation function on the first two layers for a reduced likelihood of vanishing gradient. We apply no activation function to the output layer because we aim to train a regression model instead of a classification model. When compiling, we use the $mean\_squared\_error$ loss function and the gradient descent algorithm $adam$ that appears to be highly efficient for our data.

Speedup from Parallel Forward Labelling

This section reports the performance of the parallelized generation of training data set following the Kirchhoff law. Specifically, we report that to solve a system of equations derived from a 64×64 array, it takes almost two hours (6,787 seconds) to solve the system of 126 equations using a serial implementation, as shown in Figure 2. The majority of computation time comes from the generation of the equations. To this end, we explore the potential parallelism at a fine-granularity. Specifically, we break down the equations into smaller batches, each of which is assigned to a rank in the message passing interface (MPI) (MPICH Accessed 2019). This parallelism is only possible because the converted, vertex-oriented equations can be formed independently.

Figure 2 also reports the performance when various levels of parallelism are applied. Of note, we observe significant speedup on 2, 4, 8, and 16 threads—exhibiting almost linear scalability. However, the improvement becomes marginal when more threads join the computation, and the speedup

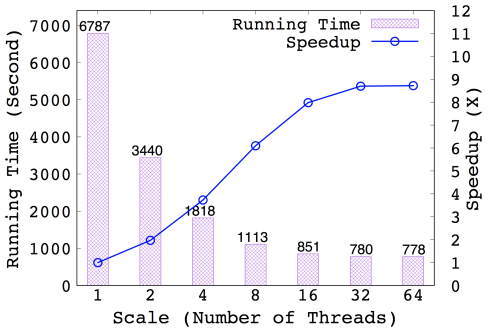


Figure 2: Time for labelling data from $\vec{R}$ to $\vec{Z}$.

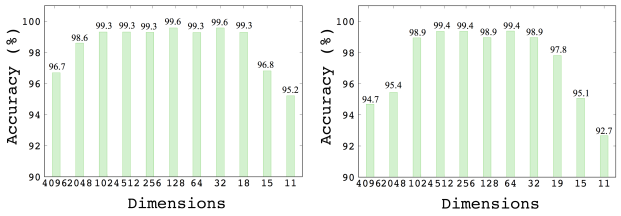
stops at 8.7× with 64 threads. This can be best explained by the fact that the underlying I/O overhead is maximally amortized; that is, the I/O bandwidth is almost saturated by 16 concurrent threads. Therefore, we believe scaling out the application into multiple physical nodes would further improve the performance by having multiple I/O devices; we will leave this as our future work that focuses on the scalability of DL-based Kirchhoff analysis.

It should be noted that this conclusion should not be generalized as the experiment consists of 126 equations, and the underlying hardware is a commodity solid-state disk (SSD) drive. And yet, regardless of specific applications and hardware, the resource is expected to be saturated at some point. A general model would be useful and is an interesting research question, which is beyond the scope of this paper.

Sensitivity and Overhead of Dimension Reduction

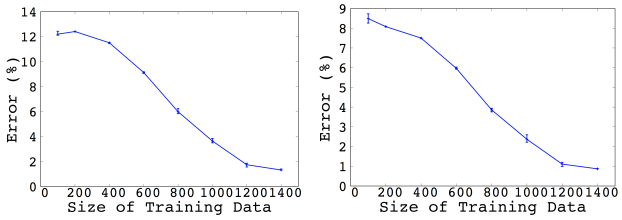
Figure 3 reports the accuracy when we reduce the original 4,096-dimensional data into various levels. The total number of data points is 1,200 for all cases; as we discuss before, 1,200 is sufficient to train accurate models (see Figure 5). The left-most columns indicate the accuracy of the models with no dimension reduction over the original data: 96.7% and 94.7% for the cell medium and wound surface data, respectively. After reducing the dimension to 1,024, both models are significantly improved to be around 99% accuracy. The high accuracy then does not change much until a very low dimensionality is reached: for the cell medium data set, the accuracy drops to 96.8% on 15 dimensions; for the wound surface data set, the accuracy drops to 97.8% on 19 dimensions. This can be best explained by the over-reduction where too much information is lost after PCA. Both experiments exhibit a convex curve over the accuracy, which is not accidental: neither high dimensionality nor overly-reduced dimensionality would be the optimal choice. We believe this is an interesting research question to find out the optimal dimensionality for ANN models and will address this in our future work.

We report the reduced dimensions with the corresponding information retained from the PCA in Table 3. The minimums of input and output neurons (i.e., dimensionality) for both data sets are 8 and 10, respectively, when 45% information is lost; in order to maintain 99% of the information,



(a) Cell Medium (b) Wound Surface

Figure 3: Accuracy with various dimension reductions.



(a) Cell Medium (b) Wound Surface

Figure 4: Error rate of 10-fold cross-validation.

both input and output in two data sets require 19 neurons. This result is aligned well to our findings on accuracy as reported in Figure 6: in cell medium, the accuracy reaches 99.3% with only 18 dimensions; in the wound surface, although the accuracy reaches a bit lower than Cell Medium (97.8%), a slightly higher dimensionality (i.e., 32) leads to an accuracy of 98.9%, and the 64-dimension becomes highly accurate, 99.4%.

Table 3: Retained information at reduced dimensions.

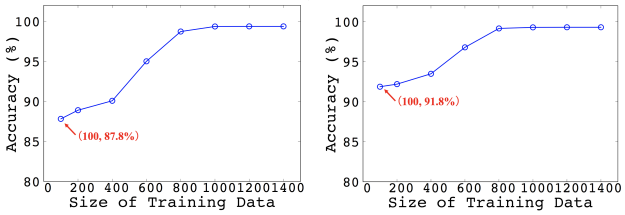
Information (%)	55	65	75	85	95	99
Cell Input Dim. (#)	8	10	12	14	17	19
Cell Output Dim. (#)	10	12	14	16	18	19
Wound Input Dim. (#)	8	10	12	15	18	19
Wound Output Dim. (#)	10	12	14	16	19	19

The overhead to reduce the dimensions is 135 seconds for all cases. The reason why it takes noticeable time is that mainstream libraries (e.g., scikit-learn (Scikit-learn 2019)) do not support PCA at specific low dimensions, which is the case for electrode arrays. To that end, we implemented a PCA module from scratch specifically for this application. It should be clear that the overhead at this level, i.e., 100s seconds, is negligible compared to the overall training time for the application. The performance of our implementation is highly dependent on the dimensions of the input data. When we apply the PCA implementation to smaller matrices, $2,048 \times 2,048$ matrices take 19 seconds, and $1,024 \times 1,024$ matrices take only four seconds.

Effectiveness of Augmentation

Figure 4 shows the error rates of 10-fold cross-validation at different augmentation scales. The variances are also plotted as error bars at each point. When the data set is augmented into 1,400 points, the error rate reaches around 1% with unnoticeable variances.

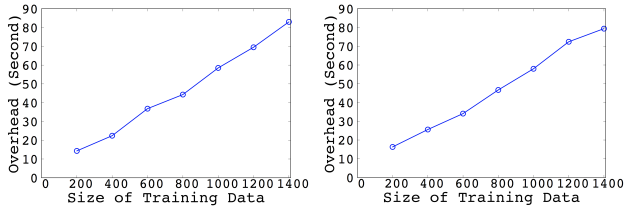
Figure 5 reports the prediction accuracy when various degrees of augmentation is applied to both data sets (with dimensionality = 512), respectively. Both original data sets comprise of only 100 data points, and we apply the randomness to augment the data sets into 200, 400, 600, 800, 1,000, 1,200, and 1,400 data points. With only 100 data points, we can only achieve 87.8% and 91.8% accuracy, respectively, which are unacceptable in real-world engineering applications. However, the accuracy gets drastically improved and passes 99% when 800 data points are available. Over-fitting issues start to appear when more than 800 data points are involved. How to find the optimal number of augmented data points is beyond the scope of this paper; in practice, because the randomness can be quickly achieved, a trial-and-error approach is feasible.



(a) Cell Medium (b) Wound Surface

Figure 5: Accuracy over various sizes of training data.

Figure 6 reports the overhead to augment two data sets through random errors. The linear correlation between the overhead and the size of the randomized data points is expected: the computational and I/O time should be proportional to the augmentation of the input data. We also observe that the overhead is in terms of tens of seconds. It should be noted that this overhead is counted as part of the training phase; Given that the core training data might take hours to collect (i.e., 24 hours for about 100 points), this overhead is negligible in the entire training phase.



(a) Cell Medium (b) Wound Surface

Figure 6: Performance overhead of augmentation.

Put Everything Together: End-to-End Comparison

We compared the proposed HDK method with two baseline systems recently published: (i) the polynomial Kirchhoff analysis (PKA) method on electrode arrays (Niu et al. 2018a) and (ii) a CNN-based Kirchhoff analysis (CKA) on electrode rings (Tan et al. 2019). We re-implemented the baseline systems and deployed them to the same testbed when comparing them with the proposed HDK method. Table 4 illustrates their accuracy and performance on the same testbed along with the largest scales initially reported by the respective work.

Table 4: Comparison between state-of-the-art methods.

Method	PKA (2018)	CKA (2019)	This Work
Scale	20×20	16×15	64×64
Accuracy	100%	92.5%	99.6%
Time	5+ hrs	2.1 hrs	15 mins

In terms of accuracy, the CNN model (CKA) described in (Tan et al. 2019) yielded an accuracy of 92.5%, while HDK achieved accuracy higher than 99%. In the electrode array applications, 1% or lower error rate is acceptable due to the existence of measurement errors (which we leverage to augment the training data, actually). PKA always calculates the exact root of the Kirchhoff equations, thus exhibiting 100% accuracy.

The overall end-to-end execution time of the proposed HDK method takes about 15 minutes for processing 64 × 64, or 4,096-dimensional, electrode arrays. The majority portion comes from preparing the training data. CKA (Tan et al. 2019) was a proof-of-concept over simulated 16-dimensional data and did not report the real execution time. Since CKA was not open source and did not have a phase for collecting real training data, we re-implemented a neural network with a forward model and deployed it to our testbed. Overall, CKA took about 2.1 hours to collect the data and train the CNN model. PKA (Niu et al. 2018a) reported its analytics time as 5.6 hours over a 20×20, or 400-dimensional, array; its training time over a 64×64 array took about seven minutes. We re-implemented PKA with parallelized data training: we were able to reduce the training time from seven minutes to three minutes, and yet the calculating (i.e., Kirchhoff equations) takes a similar time, i.e., more than five hours, on only a 20×20 sub-array and does not stop for larger scales in a reasonable time (a couple of days). In summary, the proposed HDK is orders of magnitude faster than PKA with negligible accuracy loss and delivers more than 8× (2.1 hours / 15 minutes) performance improvement over CKA without compromising the accuracy.

Related Work

Optimizations of Kirchhoff-Based Analyses

Several studies in the literature proposed effective methods to optimize Kirchhoff-based analyses. In (Golden 2000), the MRFSPICE algorithm—a combination of the Metropolis and Besag's ICM (Iterated Conditional Modes) algorithms—was proposed for optimizing highly nonlinear and non-continuous analog circuits using the Kirchoff law. In addition, based on the Kirchhoff Law about an arbitrary sinusoidal steady-state circuit, a principle of dynamic optimization method was adopted by Lin et al. in (Lin et al. 2015) to compute complicated alternating-current circuit network. In (Guo, Ma, and Zhang 2018), a simple transformation was proposed to efficiently obtain the solutions of the autonomous Kirchhoff equation or system using the known solutions of the corresponding local equation or system. Similarly, an iterative method was proposed in (Dang and Huong 2018) for solving a nonlinear biharmonic equation following the Kirchhoff law. In (Cimatti, Mover, and Sessa 2017), an automated Satisfiability Modulo Theories (SMT)-based method was proposed for the formal analysis of Switching Multi-Domain Linear Kirchhoff Networks (SMDLKN).

Neural Networks for Engineering Applications

Neural networks have been found useful for a wide range of real-world engineering applications. For instance, in automobile applications, neural networks were used to provide an accurate estimation of the remaining energy in high-capacity electric vehicle batteries. Neural networks enabled rapid adaptation to battery nonlinearities as well as changes in drivers and driving conditions that are difficult to model or characterize (Taylor 2014). In the oilfield services and equipment industry, Coveney et al. (Coveney, Hughes, and Fletcher 1996) utilized neural networks to predict cement compositions, particle size distributions, and thickening-time curves from the diffuse reflectance infrared Fourier transform spectrum of neat cement powders. This, in turn, allowed the estimation of cement quality and detected batch-to-batch variability in cement reliably. In medical engineering, Lim et al. (Lim et al. 2014) employed neural networks to efficiently and inexpensively screen large populations for diabetic retinopathy, which may lead to blindness if left untreated. The efficiency and accuracy of the neural networks introduced appropriate transformations that exploit general knowledge of the target classes. Similarly, Nigam et al. (Nigam and Graupe 2004) proposed a method for the automated detection of epileptic seizures from electroencephalograms (EEG) signals using neural networks. The authors asserted that in comparison to the manual approach, the application of neural networks significantly reduced the time required to analyze lengthy recordings.

Compared to the related work discussed above, this work falls in the category of leveraging neural networks (i.e., deep learning) to estimate the computationally-prohibitive solutions to a complex system of nonlinear equations in the context of electrode arrays. Although the idea of employing neural networks in electrode array is not new, this work represents the very first study on efficiently collecting, as opposed to *simulating*, the training data at practical scales. In addition, this work applies various optimization techniques such as parallelization, augmentation, and dimension reduction, to further improve the analysis performance without com-

promising the accuracy. This work, thus, lays out the road to widely adopting neural networks in real-world engineering applications at scale.

Conclusion and Future Work

This paper proposes a new deep-learning-based approach to efficiently conduct Kirchhoff analyses that are widely used in various engineering fields. The new approach employs multiple techniques including early pruning of unqualified input data, parallelization of forward labelling, data augmentation through random errors and dimension reduction, all of which collectively enable an efficient and accurate mechanism for large-scale Kirchhoff analyses. We implement the proposed approach with the latest machine learning framework and the parallel computing library, and evaluate it with real-world engineering applications showing promising results: the accuracy is as high as 99.6% with up to 8× performance improvement over the state-of-the-art.

Our future work will emphasize on further performance improvement for the data labelling. As per Figure 2, the performance bottleneck of the proposed system lies in the I/O subsystem when parallelizing the data labelling: about 13 out of 15 minutes, or $\frac{13}{15}$ = 87% of the time, was spent on labelling the data. We plan to exploit in-memory processing techniques to alleviate the I/O pressure, for example, by extending our prior works on scalable big data systems (Qin et al. 2019; Wang et al. 2018; Mehta et al. 2017).

Acknowledgement

This work is in part supported by a research award from Amazon Web Services and a research award from Google Cloud Platform. This work is in part supported the National Science Foundation under contracts CCF-1756013 and IIS-1838024 (using resources provided by Amazon Web Services as part of the NSF BIGDATA program).

References

Barber, D., and Brown, B. 1984. Applied potential tomography. *Journal of Physics E: Scientific Instruments* 17:723–733.

Cimatti, A.; Mover, S.; and Sessa, M. 2017. Smt-based analysis of switching multi-domain linear kirchhoff networks. In *2017 Formal Methods in Computer Aided Design (FMCAD)*, 188–195.

Coveney, P. V.; Hughes, T. L.; and Fletcher, P. 1996. Using artificial neural networks to predict the quality and performance of oil-field cements. *AI Magazine* 17(4):41–53.

Dang, Q. A., and Huong, N. T. 2018. Existence results and iterative method for solving a nonlinear biharmonic equation of kirchhoff type. *Computers and Mathematics with Applications* 76:11–22.

Golden, R. M. 2000. Kirchoff law markov fields for analog circuit design. In *Advances in Neural Information Processing Systems (NIPS)*.

Guo, J.; Ma, S.; and Zhang, G. 2018. Solutions of the autonomous kirchhoff type equations in r^n. *Applied Mathematics Letters* 82:14–17.

Keras. 2019. https://keras.io.

Kirchhoff Law. 2019. https://en.wikipedia.org/wiki/Kirchhoff_circuit_laws.

Lim, G.; Lee, M. L.; Hsu, W.; and Wong, T. Y. 2014. Transformed representations for convolutional neural networks in diabetic retinopathy screening. In *Workshops at the Twenty-Eighth Association for the Advancement of Artificial Intelligence (AAAI) Conference on Artificial Intelligence*.

Lin, H. X.; Qi, L.; Cong, C.; and Hong, S. 2015. Study and application of optimization algorithm about sinusoidal steady-state circuit. In *Proceedings of the 34th Chinese Control Conference (CCC)*.

Mehta, P.; Dorkenwald, S.; Zhao, D.; Kaftan, T.; Cheung, A.; Balazinska, M.; Rokem, A.; Connolly, A.; Vanderplas, J.; and AlSayyad, Y. 2017. Comparative evaluation of big-data systems on scientific image analytics workloads. In *Proceedings of the 43rd International Conference on Very Large Data Bases (VLDB)*.

MPICH. Accessed 2019. http://www.mpich.org/.

Nigam, V. P., and Graupe, D. 2004. A neural-network-based detection of epilepsy. *Neurological research* 26(1):55–60.

Niu, Y.; Al-Mamun, A.; Lin, H.; Li, T.; Zhao, Y.; and Zhao, D. 2018a. Toward scalable analysis of multidimensional scientific data: A case study of electrode arrays. In *IEEE International Conference on Big Data (BigData)*.

Niu, Y.; Qi, L.; Zhang, F.; and Zhao, Y. 2018b. Geometric screening of core/shell hydrogel microcapsules using a tapered microchannel with interdigitated electrodes. *Biosensors and Bioelectronics* 112:162–169.

Qin, H.; Zawad, S.; Zhou, Y.; Yang, L.; Zhao, D.; and Yan, F. 2019. Swift machine learning model serving scheduling: A region based reinforcement learning approach. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 13:1–13:23.

Scikit-learn. 2019. https://scikit-learn.org.

Semenov, V. Y. 2015. A method to find all the roots of the system of nonlinear algebraic equations based on the krawczyk operator. *Cybernetics and Systems Analysis* 51(819).

Tan, C.; Lv, S.; Dong, F.; and Takei, M. 2019. Image reconstruction based on convolutional neural network for electrical resistance tomography. *IEEE Sensors Journal* 19(1):196–204.

Taylor, M. D. 2014. Joule counting correction for electric vehicles using artificial neural networks. In *Twenty-Eighth Association for the Advancement of Artificial Intelligence (AAAI) Conference on Artificial Intelligence*.

TensorFlow. 2019. https://www.tensorflow.org/.

Wang, X.; Xu, C.; Wang, K.; Yan, F.; and Zhao, D. 2018. Toward cost-effective memory scaling in clouds: Symbiosis of virtual and physical memory. In *IEEE 11th International Conference on Cloud Computing (CLOUD)*.