High-Speed Channel Modeling with Machine Learning Methods for Signal Integrity Analysis

Tianjian Lu, *Member, IEEE,* Ju Sun, *Member, IEEE,* Ken Wu, *Member, IEEE,* and Zhiping Yang, *Senior Member, IEEE*

***Abstract*—In this work, machine learning methods are applied to high-speed channel modeling for signal integrity analysis. Linear, support vector and deep neural network (DNN) regressions are adopted to predict the eye-diagram metrics, taking advantage of the massive amounts of simulation data gathered from prior designs. The regression models, once successfully trained, can be used to predict the performance of high-speed channels based on various design parameters. The proposed learning-based approach saves complex circuit simulations and substantial domain knowledge altogether, in contrast to alternatives that exploit novel numerical techniques or advanced hardware to speed up traditional simulations for signal integrity analysis. Our numerical examples suggest that both support vector and DNN regressions are able to capture the nonlinearities imposed by transmitter and receiver models in high-speed channels. Overall, DNN regression is superior to support vector regression in predicting the eye-diagram metrics. Moreover, we also investigate the impact of various tunable parameters, optimization methods, and data preprocessing on both the learning speed and the prediction accuracy for the support vector and DNN regressions.**

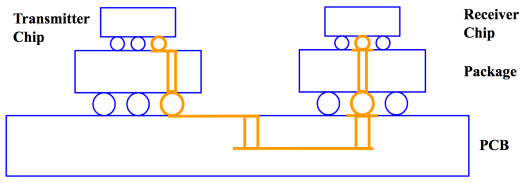
***Index Terms*—Deep Neural Networks, Eye Diagram, High-Speed Channel, Machine Learning, Signal Integrity, Support Vector Regression.**

I. Introduction

A high-speed, chip-to-chip system-level design as shown in Fig. 1(a) entails intensive simulations prior to manufacturing, for both verification and optimization purposes. In the area of signal integrity, there are two major types of simulation techniques: electromagnetic solvers and circuit simulators [1]. Electromagnetic solvers are often used to extract models for high-speed interconnects such as IC packages, printed circuit boards (PCBs), and connectors. To evaluate the performance at the system level, most high-speed designs use the eye diagram. As shown in Fig. 2, the eye diagram is created by first segmenting and then overlaying the transient waveforms obtained from a circuit simulator. In addition to model extraction, simulations are also performed to predict the impact of manufacturing variances on signal integrity, because variations of design parameters during manufacturing may alter the characteristics of a well-designed high-speed channel. Simulations invariably happen in many iterations and across various design stages. Consequently, substantial amounts of simulation data are generated.

Tianjian Lu, Ken Wu, and Zhiping Yang are with Google Inc., 1600 Amphitheatre Pkwy, Mountain View, CA 94043, USA, e-mail: {tianjianlu, kenzwu, zhipingyang}@google.com.

Ju Sun is with the Mathematics Department of Stanford University, 450 Serra Mall, Building 380, Stanford, CA 94305-2125, USA, e-mail: sunju@stanford.edu



(a)

	Transmitter	Interconnect		Receiver
				
Parameters	Pre-emphasis	Trace Width/Spacing	Dielectric Constant	Equalization
	Jitter	Substrate Thickness	Loss Tangent	Jitter

(b)

Fig. 1: (a) The topology of a high-speed channel and (b) the design parameters considered in this work.

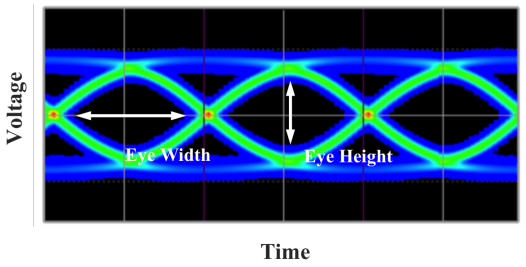


Fig. 2: Illustration of eye height and width in an eye diagram.

The simulation techniques employed in characterizing high-speed channels for signal integrity can be computationally expensive. There are efforts in utilizing domain decomposition schemes and parallel computing to enhance the efficiency of electromagnetic solvers on model extraction [2], [3]. Model-order reductions techniques are also proposed to achieve a fast frequency sweep in extracting models over a broadband [4], [5]. Moreover, there are approaches to efficiently generating eye diagrams by utilizing shorter data patterns as inputs [6] or using statistical methods [7].

In this paper, we take a different route and propose addressing the efficiency issue by taking advantage of the existing simulation data. A learning-based model is trained on the prior data. Once the training is completed, we obtain a reasonable model characterizing the underlying high-speed channel—performance metrics such as eye height and width

can be predicted from the trained model for varying design parameters. The proposed approach requires no complex circuit simulations or substantial domain knowledge. The model training can be achieved within a reasonable amount of time over modern computing hardware, and the obtained models can be reused for future designs, which amortizes the training cost. Once the training concludes, prediction can be performed in a highly efficient manner.

We consider eight design parameters for a high-speed channel in this work; they are tabulated in Fig. 1(b). We use eye height and eye width as the performance metrics, which are systematically used to assess the system-level performance of high-speed channels. The technical task here is to predict the eye height and eye width for given design specifications. For a given high-speed channel, suppose there is a deterministic function that maps the design parameters to the eye height (width). The learning problem here is to learn this function from the available data samples. In machine learning term, this is a *regression* problem.

We study three regression methods in this work, namely, linear regression [8], support vector regression (SVR) [9], and deep neural network (DNN) regression [10]. Due to the nonlinearity imposed by the transmitter and receiver models, linear regression, which assumes a linear model for the underlying function, fails to make accurate predictions of eye height and width. SVR is able to handle nonlinearities with kernel mappings [9], [11], [12]. However, DNN regression is superior to SVR in terms of empirical prediction accuracy. [1]

Deep neural networks have recently made great progress in selecting relevant results for web search, making recommendations in online shopping, identifying objects in images, and transcribing speeches into texts, to name a few [10], [14]. To model high-speed channels, a DNN takes in raw design parameters as the input and passes them through layers of linear and nonlinear transformations; see Fig. 4. With adequate number of such transformations, the DNN approximates the underlying parameter-performance function reasonably well. The better performance of DNN over SVR observed in our numerical examples may be partly explained by the universal approximation property of DNN [15], which (roughly) says any function with reasonable regularity can be approximated with controlled errors by DNN with appropriate parameters.

II. Regression Methods

The regression-based learning process specialized to our problem consists of three components [12]: a generator, a supervisor, and a regressor, as shown in Fig. 3. The generator first produces one set of input parameters (see Fig. 1(b)) of a high-speed channel, contained in a vector $\{x\}$. The supervisor, which acts as a knowledgeable oracle, returns the eye height or width y corresponding to $\{x\}$. The learning process is essentially the selection of the right regression function $f(\{x\},\{\theta\})$, where $\{\theta\}$ contains the parameters to be learned, such that the prediction made by f approximates the value returned by the supervisor uniformly over all possible input $\{x\}$. The selection is based on a set of N training examples, denoted as $(\{x_i\}, y_i)$ for $i = 1,\ldots,N$. In this section, we describe the three regression methods used in this work, namely, linear, support vector, and DNN regressions.

[1]Other than the higher accuracy, DNN regression is more natural than SVR in handling the multi-output situation [13], in which the eye height and width can be predicted simultaneously through the same neural network without any additional change in the algorithm framework.

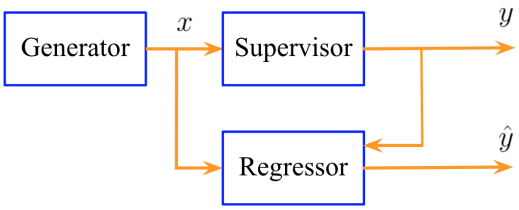


Fig. 3: Regression-based learning [12] consists of a generator, a supervisor, and a regressor. At training, the regressor sees sufficiently many $(\{x\}, y)$ pairs generated by the generator and tries to match the supervisor in predicting y given $\{x\}$. After training, given an input x the regressor returns $\hat{y}$, which should be as close as the supervisor's output y.

A. Linear Regression

The linear regression accommodates multiple input parameters of a high-speed channel and predicts its the eye height or eye width through

$$y = \{\beta\}\{x\}^{\mathrm{T}} + \beta_0, \tag{1}$$

where $\{x\}$ represents the input parameters, $\{\beta\}$ contains the weights, and β_0 is a constant offset. The weights $\{\beta\}$ and the offset β_0 are chosen by solving the following least-squares problem:

$$\text{minimize}_{\{\beta\},\beta_0} \quad \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 . \tag{2}$$

In other words, one seeks to minimize the total squared prediction errors over the training examples. This optimization problem can be reliably solved to large scales (i.e., when more design parameters are considered such that $\{x\}$ is a huge vector) by mature numerical solvers [16].

B. Support Vector Regression

In ε-SVR [11], [12], instead of minimizing the total squared errors, one hopes to make the prediction errors uniformly bounded by a parameter $\varepsilon > 0$, meanwhile to keep the model as "simple" as possible. We start with the case when the prediction model is still linear, i.e.,

$$y = \{w\}\{x\}^{\mathrm{T}} + b, \tag{3}$$

where $\{w\}$ represents weights and b denotes the bias term. To promote the model simplicity, one can minimize the quadratic function

$$E = \frac{1}{2}\{w\}\{w\}^{\mathrm{T}} . \tag{4}$$

To ensure bounded prediction errors by ε, we require

$$\left| y_i - \left(\{w\}\{x_i\}^{\mathrm{T}} + b \right) \right| \le \varepsilon, \ \forall\, i. \tag{5}$$

Together, we arrive at a convex optimization problem

$$\begin{aligned} \text{minimize}_{\{w\},b} \quad & \frac{1}{2}\{w\}\{w\}^{\mathrm{T}} \\ \text{subject to} \quad & y_i - \{w\}\{x_i\}^{\mathrm{T}} - b \le \varepsilon,\ \forall\, i, \\ & \{w\}\{x_i\}^{\mathrm{T}} + b - y_i \le \varepsilon,\ \forall\, i. \end{aligned} \tag{6}$$

For a pre-specified ε, there may not be a feasible linear function that obeys the constraints in (6). Slack variables $\{\xi\}$ and $\{\xi^*\}$ are hence introduced to add in flexibility; such flexibility is meanwhile penalized in the objective, resulting in the notable ε-SVR formulation [12]:

$$\begin{aligned} \underset{\{w\},b,\{\xi\},\{\xi^*\}}{\text{minimize}} \quad & \frac{1}{2}\{w\}\{w\}^{\mathrm{T}} + C\sum_{i=1}^{N}(\xi_i + \xi_i^*) \\ \text{subject to} \quad & y_i - \{w\}\{x_i\}^{\mathrm{T}} - b \le \varepsilon + \xi_i,\ \forall\, i, \\ & \{w\}\{x_i\}^{\mathrm{T}} + b - y_i \le \varepsilon + \xi_i^*,\ \forall\, i, \\ & \xi_i \ge 0,\ \xi_i^* \ge 0,\ \forall\, i, \end{aligned} \tag{7}$$

where C is a tunable parameter to the optimization.

For the above constrained convex optimization problem, by introducing Lagrange multipliers to handle the inequality constraints and after simplification [11], we arrive at the dual problem:

$$\begin{aligned} & \text{maximize}\ g(\{\alpha\},\{\beta\}) \\ & \text{subject to}\ \sum_{i=1}^{N}\alpha_i = \sum_{i=1}^{N}\beta_i,\ \alpha_i,\beta_i \in [0,C]\ \forall\, i, \end{aligned} \tag{8}$$

where the dual objective function is

$$\begin{aligned} g(\{\alpha\},\{\beta\}) = & -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i-\beta_i)(\alpha_j-\beta_j)\{x_i\}\{x_j\}^{\mathrm{T}} \\ & -\varepsilon\sum_{i=1}^{N}(\alpha_i+\beta_i) + \sum_{i=1}^{N} y_i(\alpha_i-\beta_i). \end{aligned} \tag{9}$$

The prediction linear function in (3) can be written in terms of $\{\alpha\}$ and $\{\beta\}$:

$$y = \sum_{i=1}^{N}(\alpha_i-\beta_i)\{x_i\}\{x\}^{\mathrm{T}} + b. \tag{10}$$

Maximization of the dual objective function in Equation (9) depends on the input vectors only through inner products of the form $\{x_i\}\{x_j\}^{\mathrm{T}}$. Similarly, the final prediction in Equation (10) also depends on the input data only through inner products $\{x_i\}\{x\}^{\mathrm{T}}$. To empower SVR to handle nonlinearity, a natural idea is to map the input vectors $\{x_i\}$ to a high-dimensional feature space through a nonlinear mapping, denoted as $\Phi(\{x_i\})$, and then feed these mapped vectors $\Phi(\{x_i\})$ into the linear model in Equation (3). This consequently changes the inner product terms in Equations (9) and (10) as:

$$\{x_i\}\{x_j\}^{\mathrm{T}} \mapsto \Phi(\{x_i\})\,\Phi(\{x_j\})^{\mathrm{T}},\ \forall\, i,j \tag{11}$$

$$\{x_i\}\{x\}^{\mathrm{T}} \mapsto \Phi(\{x_i\})\,\Phi(\{x\})^{\mathrm{T}},\ \forall\, i. \tag{12}$$

Due to the alluded sole dependency of Equations (9) and (10) on inner products, the kernel trick in SVR works by directly defining an explicit *kernel function* K such that

$$K(\{x\},\{x'\}) = \Phi_K(\{x\})\,\Phi_K(\{x'\})^{\mathrm{T}}\ \forall\,\{x\},\{x'\} \tag{13}$$

for a certain implicit nonlinear feature mapping Φ_K. Applying the kernel trick, the dual objective function in Equation (9) becomes

$$\begin{aligned} -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i-\beta_i)(\alpha_j-\beta_j)K(\{x_i\},\{x_j\}) \\ -\varepsilon\sum_{i=1}^{N}(\alpha_i+\beta_i) + \sum_{i=1}^{N} y_i(\alpha_i-\beta_i) \end{aligned} \tag{14}$$

and the prediction model in Equation (3) becomes

$$y = \sum_{i=1}^{N}(\alpha_i-\beta_i)K(\{x_i\},\{x\}) + b. \tag{15}$$

The kernel trick avoids explicit calculation of the feature mapping and works by direct mapping to inner products. This often results in computational saving and allows implicit mapping into very high- or even infinite-dimensional spaces [17]. The resulting problem is convex and can be solved up to considerably large scales with specialized solvers; see, e.g., [18].

We study three kernels in this paper, namely, linear, polynomial, and Gaussian kernels. The linear kernel mapping is simply the identity mapping. An inhomogeneous polynomial kernel of degree d can be written as

$$K(\{x_i\},\{x_j\}) = \left(1 + \{x_i\}^{\mathrm{T}}\{x_j\}\right)^d. \tag{16}$$

A Gaussian kernel is in the following form

$$K(\{x_i\},\{x_j\}) = \exp\left(-\frac{||\{x_i\}-\{x_j\}||^2}{2\sigma^2}\right), \tag{17}$$

where σ determines the bandwidth of the kernel that is tunable. The Gaussian kernel induces an implicit mapping that maps the input vectors into an infinite-dimensional space.

C. DNN Regression

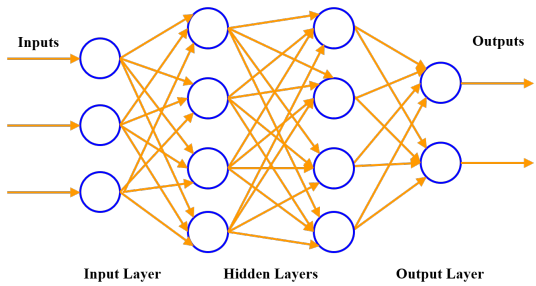
As shown in Fig. 4(a), a feedforward neural network consists of many connected nodes in multiple layers. The number of nodes belonging to individual layers can be lumped into a vector $\{L\} = \left(L^{\text{in}}, L^1, \ldots, L^h, \ldots, L^n, L^{\text{out}}\right)$ where L^{in}, L^h, L^{out} denote the number of nodes in the input layer, the h^{th} hidden layer, and the output layer, respectively. The input to the h^{th} hidden layer can be found through

$$\{z^h\} = \{x^{h-1}\}\left[W^h\right], \tag{18}$$

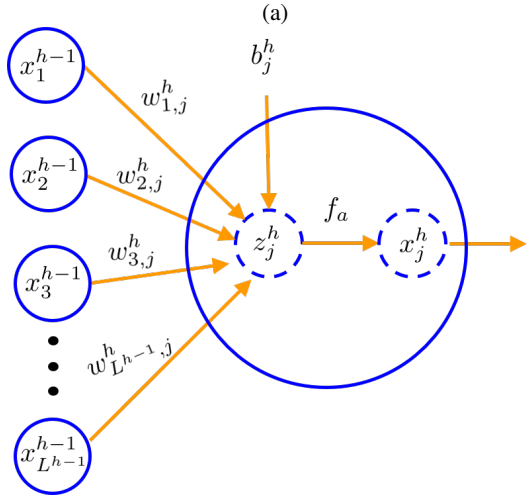
where the $L^{h-1} \times L^h$ matrix $\left[W^h\right]$ contains the weights mapping output of the $(h-1)^{\text{th}}$ layer to input of the h^{th} layer. The output vector $\{x^h\}$ of the h^{th} hidden layer is obtained as

$$\{x^h\} = f_a\left(\{z^h\} + \{b^h\}\right), \tag{19}$$

where f_a is a predefined scalar-to-scalar function that acts element-wise on the vector $\{z^h\}$. Such f_a is called the



(a)



(b)

Fig. 4: (a) A feedforward neural network typically consists of one input layer, several hidden layers, and one output layer; (b) a node connects the $(h-1)^{\text{th}}$ layer to the h^{th} layer. The w's and b are the weights, and f_a is the activation function.

activation function. The output from the final layer, i.e., $\{x^{\text{out}}\}$, is the prediction $\{\widehat{y}\}$. It is possible to return a vector directly; in this paper, for simplicity, we focus on returning a scalar value, i.e., when the output layer only consists of a single node.

The forms of nonlinear activation function considered in this work include the rectified linear unit or ReLU [19]

$$f_a(z) = \max\{0, z\}, \tag{20}$$

the hyperbolic tangent

$$f_a(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \tag{21}$$

and the sigmoid function

$$f_a(z) = \frac{1}{1 + e^{-z}}. \tag{22}$$

Based on the input vector $\{x_i\}$, the described feedforward mechanism predicts the eye height or width $\{\widehat{y}_i\}$, which differs from the true $\{y_i\}$ by

$$\{e_i\} = \{y_i\} - \{\widehat{y}_i\}. \tag{23}$$

To make $\{\widehat{y}_i\}$ a good approximation of $\{y_i\}$ uniformly over the training examples, one minimizes the cost function

$$E = \frac{1}{2}\sum_{i=1}^{N}\{e_i\}\{e_i\}^{\text{T}}. \tag{24}$$

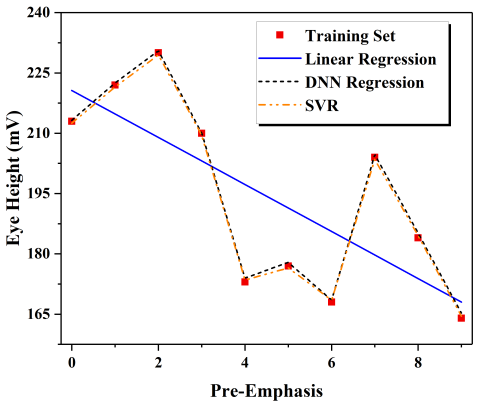


Fig. 5: Comparison of the three different regression methods in handling the nonlinearity imposed by the transmitter model.

The weights stored in matrices $[W^h]$ and vectors $\{b^h\}$ are the optimization variables. To find a local minimum of the cost function, the stochastic gradient descent (SGD) method is used, i.e.,

$$[W^h] = [W^h] - \gamma\widehat{\nabla}_{[W^h],\{b^h\}}E, \ \forall\ h = 1, \ldots, n, \& \text{ out} \tag{25}$$

where $\widehat{\nabla}_{[W^h],\{b^h\}}E$ is a stochastic approximation to the true gradient to $\nabla_{[W^h],\{b^h\}}E$, and γ is called the learning rate, or the step size parameter. The stochastic gradients $\widehat{\nabla}_{[W^h],\{b^h\}}E$ are computed efficiently by the back-propagation method.

III. Numerical Examples

In this section, we first evaluate the three regression methods in terms of their capability of handling the nonlinearities present in the high-speed channels. We then focus on SVR and DNN regression and investigate their performances in eye height and width prediction. As both SVR and DNN regression have many tunable components and parameters, we also empirically study the impact of kernel selection on SVR, and likewise, activation function and optimization methods on DNN regression. Representation and quality of data are the key determining factors on the effectiveness of machine learning methods for practical problems [20]. Our experiments demonstrate that data preprocessing affects the convergence speeds of numerical optimization methods on both SVR and DNN regression—when maximum number of iterations is enforced, this translates into different prediction accuracies at the last iteration. Our implementation of SVR is based on the scikit-learn [21] package, and DNN regression on the TensorFlow [22] package.

A. Comparison on Capability of Handling Nonlinearities

To generate the training set, we sweep the pre-emphasis level on the transmitter side while fixing the remaining input parameters and record the eye height. This simplifies the multivariate regression problem into a univariate one. The training

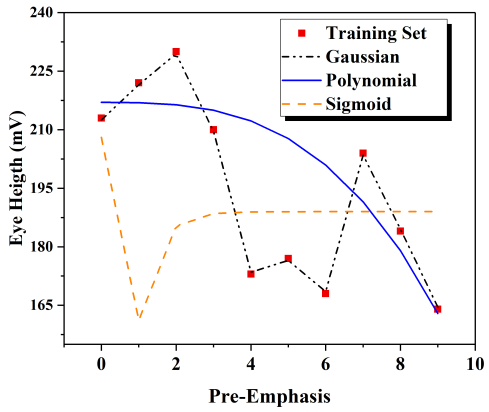


Fig. 6: Performance of SVR in eye-height prediction with different kernels. The polynomial kernel has degree three.

set thus contains only 10 examples, corresponding to all admissible pre-emphasis levels (Fig. 5). Linear regression cannot handle nonlinearities, which is evident from its formulation and the demonstration in Fig. 5—the predicted eye heights by linear regression deviate significantly from the truth. Both SVR and DNN regression achieve excellent accuracy in eye-height prediction in the presence of nonlinearities. The DNN regression uses two hidden layers, each with 100 nodes. SVR with different kernels can handle nonlinearities of different forms and scales. As shown in Fig. 6, only Gaussian-kernel SVR successfully handles the nonlinearity in this numerical example. Therefore, it is critical to select an appropriate kernel in order for SVR to accurately model the high-speed channel.

B. Performance of the DNN Regression

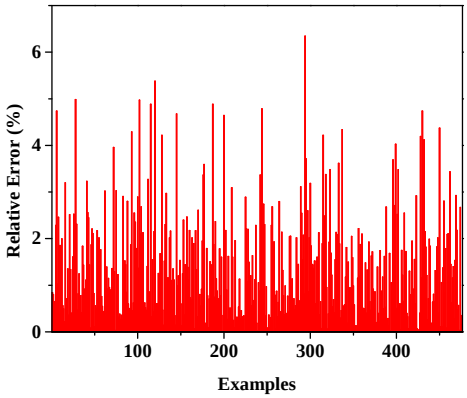


Fig. 7: Relative errors of the predicted eye heights from the DNN regression on the test set.

In this section, we consider all the input parameters tabulated in Fig. 1(b) and investigate the performance of DNN regression for the complete multivariate regression.

A DNN is first trained to predict the eye height. The training, validation, and test sets contain 717, 48, and 476 examples, respectively. The DNN has three hidden layers of 100, 300, and 200 nodes, respectively. The learning rate is chosen as 0.01 and the batch size is 25. The maximum number of iterations is set to 4000. The eye height in the data set varies from 148 to 253 mV. Table I shows the prediction performance measured by the root-mean-square error (RMSE) and the maximum relative error. The RMSE is defined as

$$\text{RMSE} = \sqrt{\frac{1}{N}\text{RSS}}, \tag{26}$$

where RSS is the least-squares objective function defined in Equation (2) and N is the number of examples. Several variants of the SGD method are available; they often have different convergence speeds. With our pre-specified maximum number of iterations, the different convergence speeds may translate into different prediction accuracies. We test three variants: the plain SGD (`GradientDescentOptimizer` in TensorFlow), the momentum variant (`MomentumOptimizer` in TensorFlow), and the RMSProp variant (`RMSPropOptimizer` in TensorFlow) [23]. The RMSE on the training set with plain SGD is 3.1 mV and 3.4 mV on the test set. When the momentum variant is used, the RMSE on the training set is reduced to 1.9 mV and to 2.7 mV on the test set. Figure 7 depicts the relative errors of the predicted eye heights on the test set, and the majority are below 3%.

Another DNN is trained to predict the eye width. The eye width is measured in unit intervals (UI)—one UI is defined as one data bit-width. The training, validation, and test sets contain 509, 35, and 203 examples, respectively. This DNN has seven hidden layers of 10, 20, 20, 30, 20, 20, and 10 nodes, respectively. The batch size is changed to 15. The eye width varies from 0.21 to 0.37 UI in the data set. From Table II, the RMSE with the momentum method is 0.006 UI on the training set and 0.008 UI on the test set, and the maximum relative errors in both training and test sets are less than 10%.

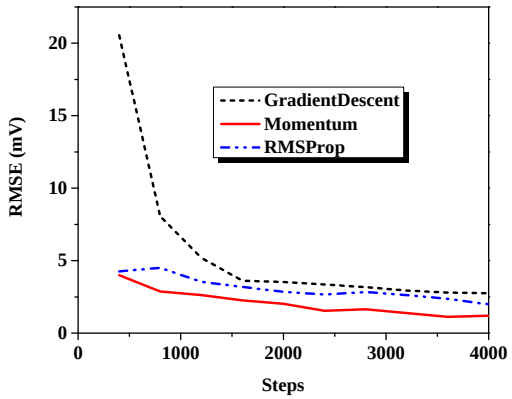


Fig. 8: Evolution of the RMSE of the predicted eye height with respect to the iteration index with the DNN regression. The three variants of SGD used in this paper are compared.

Figure 8 compares the convergence behaviors of the three variants of SGD. The network architecture and common

TABLE I: Accuracy of predicted eye heights from the DNN regression.

		Gradient Descent	Momentum	RMSProp
On Training Set	RMSE (mV)	3.1	1.9	2.6
	Maximum Relative Error (%)	6.2	4.1	5.2
On Test Set	RMSE (mV)	3.4	2.7	3.1
	Maximum Relative Error (%)	5.9	6.1	6.3

TABLE II: Accuracy of predicted eye widths from the DNN regression. A unit interval (UI) is defined as one data bit-width, regardless of data rate. Eye width is measured in the UI.

		Gradient Descent	Momentum	RMSProp
On Training Set	RMSE (UI)	0.006	0.006	0.008
	Maximum Relative Error (%)	7.9	8.1	8.7
On Test Set	RMSE (UI)	0.008	0.008	0.01
	Maximum Relative Error (%)	10.6	9.3	9.5

TABLE III: Accuracy of predicted eye heights from the SVR. Three different kernels are compared and the polynomial kernel has degree three. The results shown here are after standardization is applied to the data set.

		Gaussian	Polynomial	Linear
On Training Set	RMSE (mV)	2.5	17.2	17.9
	Maximum Relative Error (%)	6.2	20.7	19.3
On Test Set	RMSE (mV)	5.7	16.7	19.3
	Maximum Relative Error (%)	10.2	19.5	17.9

TABLE IV: Accuracy of predicted eye widths from the SVR with three different kernels. A unit interval (UI) is defined as one data bit-width, regardless of data rate. Eye width is measured in the UI. The results shown here are after standardization is applied to the data set.

		Gaussian	Polynomial	Linear
On Training Set	RMSE (UI)	0.008	0.02	0.029
	Maximum Relative Error (%)	12.8	41.3	32.9
On Test Set	RMSE (UI)	0.04	0.038	0.081
	Maximum Relative Error (%)	23.4	22.8	33.3

optimization parameters are tuned based on the hyperbolic tangent activation and plain SGD. For the comparison, these parameters are fixed and only the optimization methods are replaced each time. The momentum variant works by accumulating gradient information across iterations, and the RMSRrop variant works by modifying the gradient and adaptively tuning the learning rates. Our experiment suggests these heuristics indeed lead to faster convergence.

Data preprocessing also influences the convergence speed. In the above experiments, individual input parameters were "standardized" to have zero mean and unit variance:

$$x_{\text{new}} = \frac{x - \mu}{\sigma}, \tag{27}$$

where μ and σ are the empirical mean and standard deviation of the individual parameters, respectively. We compare this type of preprocessing with data normalization

$$x_{\text{new}} = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}} (\alpha_{\text{max}} - \alpha_{\text{min}}) + \alpha_{\text{min}}, \tag{28}$$

which rescales the individual parameters into the interval $[\alpha_{\text{min}}, \alpha_{\text{max}}]$. Here x_{min} and x_{max} denote the minimum and maximum values of the original data, respectively. Figure 9 compares the impacts of data standardization, data normalization, and primitive input on the convergence speed. Both data standardization and normalization significantly speed up the convergence.

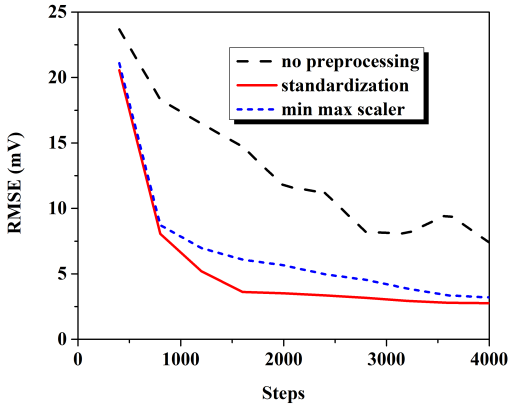


Fig. 9: Evolution of the RMSE of the predicted eye height with respect to the iteration index with the DNN regression. Three data preprocessing methods are compared.

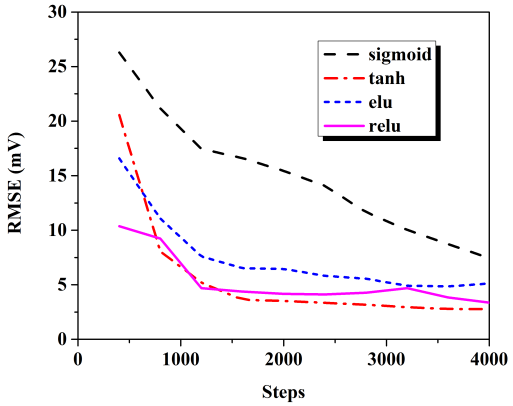


Fig. 10: Evolution of the RMSE of the predicted eye height with respect to the iteration index with the DNN regression. Four different activation functions are compared.

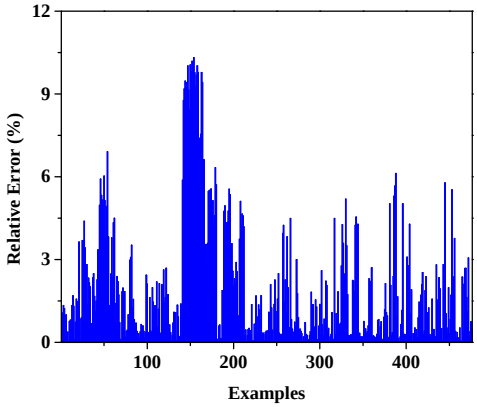


Fig. 11: Relative errors of the predicted eye heights from Gaussian-kernel SVR on the test set.

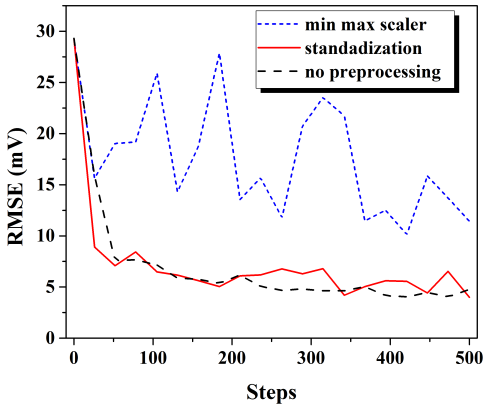


Fig. 12: Evolution of the RMSE of the predicted eye height with respect to the iteration index. Convergence speed of the Gaussian-kernel SVR is not sensitive to data standardization.

Finally, we study the impact of different activation functions. Again, the network architecture and optimization parameters are tuned based on the hyperbolic tangent activation and plain SGD, with the activation functions being replaced each time. Figure 10 shows that the hyperbolic tangent and ReLU enjoy faster convergence than the other two activation functions.

C. Performance of the SVR

We adopt the same training, validation, and test sets in the previous section to analyze the prediction accuracy of SVR. The maximum number of iterations is set to 4000. As shown in Table III, the SVR with Gaussian kernel achieves a maximum relative error of 10.2% on the test set in eye-height prediction, which is higher than that of the DNN regression. The instance-wise relative errors of the predicted eye heights with the SVR is depicted in Fig. 11. The relative errors for the majority of examples in the test set are below 6%. Similarly, for eye-width prediction as shown in Table IV, the Gaussian-kernel SVR has a maximum relative error of 23.4%, in contrast to the 10.6% by the DNN regression.

As for the impact of different kernels, the polynomial-kernel SVR does not make satisfactory predictions on eye height and the maximum relative error is as high as 19.5% on test set. The linear SVR cannot handle the nonlinearities imposed by the transmitter and receiver model, which is also shown in Tables III and IV.

We also study the sensitivity of the convergence speed on SVR with respect to data preprocessing. We focus on the Gaussian-kernel SVR due to its superior prediction accuracy as compared to other kernel choices. Figure 12 depicts the evolution of the RMSE of the predicted eye height with respect to the iteration index. The Gaussian-kernel SVR appears to be insensitive to data standardization in our experiment, as the evolution curves with and without the data standardization roughly align with each other. However, data normalization substantially slows down the convergence—in fact, empirically the Gaussian-kernel SVR seems to take forever to converge after the normalization.

IV. Conclusion and Discussion

In this work, we propose using machine learning methods to predict eye-diagram metrics of high-speed channels for signal integrity analysis. Regression models are learned based on the large amount of prior simulation data. Once the learning is completed, the learned models can be used to predict the eye height and the eye width in future designs. This learning-based approach requires no substantial domain knowledge and meanwhile saves complex and expensive circuit simulations. Through numerical examples, we studied three regression methods including linear, support vector, and DNN regressions on their capability of handling nonlinearities in the high-speed channels. We also studied the impacts of various tunable parameters including kernels in SVR, activation functions and optimization methods in DNN regression, and two data preprocessing schemes on the convergence speed at training and the prediction accuracy at testing.

Machine learning methods open an array of opportunities for signal integrity analysis. As an immediate extension, the regression problem considered in this paper can be directly formed as a classification problem, because eye masks are readily available in many standard high-speed interfaces and hence the pass or fail labels are known. Another possibility is to extend the current learning framework to analyze the four-level pulse amplitude modulation (PAM-4) signaling, which is the leading contender for implementing high-speed channels with 56 Gbps throughput and higher. In practice, simulation data are gathered over time as the design cycle progresses. Therefore, it may be more appropriate to adopt an online learning setting in which the channel models are being gradually improved as new data become available. Finally, provided that adequate design data could be gathered, it is plausible to consider an "inverse design" setting, in which the target channel performance is specified and possible design parameter combinations are suggested by learning models.

References

[1] S. H. Hall and H. L. Heck, *Advanced signal integrity for high-speed digital designs*. John Wiley & Sons, 2011.

[2] J.-M. Jin, *The finite element method in electromagnetics*. John Wiley & Sons, 2015.

[3] Y. Li and J.-M. Jin, "A vector dual-primal finite element tearing and interconnecting method for solving 3-D large-scale electromagnetic problems," *IEEE Transactions on Antennas and Propagation*, vol. 54, no. 10, pp. 3000–3009, 2006.

[4] S. V. Polstyanko, R. Dyczij-Edlinger, and J.-F. Lee, "Fast frequency sweep technique for the efficient analysis of dielectric waveguides," *IEEE transactions on microwave theory and techniques*, vol. 45, no. 7, pp. 1118–1126, 1997.

[5] S.-H. Lee, T.-Y. Huang, and R.-B. Wu, "Fast waveguide eigenanalysis by wide-band finite-element model-order reduction," *IEEE transactions on microwave theory and techniques*, vol. 53, no. 8, pp. 2552–2558, 2005.

[6] W.-D. Guo, J.-H. Lin, C.-M. Lin, T.-W. Huang, and R.-B. Wu, "Fast methodology for determining eye diagram characteristics of lossy transmission lines," *IEEE Transactions on Advanced Packaging*, vol. 32, no. 1, pp. 175–183, 2009.

[7] B. K. Casper, M. Haycock, and R. Mooney, "An accurate and efficient analysis method for multi-gb/s chip-to-chip signaling schemes," in *VLSI Circuits Digest of Technical Papers, 2002. Symposium on*. IEEE, 2002, pp. 54–57.

[8] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[9] H. Drucker, C. J. Burges, L. Kaufman, A. J. Smola, and V. Vapnik, "Support vector regression machines," in *Advances in neural information processing systems*, 1997, pp. 155–161.

[10] J. Schmidhuber, "Deep learning in neural networks: an overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[11] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and computing*, vol. 14, no. 3, pp. 199–222, 2004.

[12] V. Vapnik, *The nature of statistical learning theory*. Springer science & business media, 2013.

[13] H. Borchani, G. Varando, C. Bielza, and P. Larrañaga, "A survey on multi-output regression," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 5, no. 5, pp. 216–233, 2015.

[14] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[15] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao, "Why and when can deep-but not shallow-networks avoid the curse of dimensionality: A review," *International Journal of Automation and Computing*, pp. 1–17, 2017.

[16] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[17] A. J. Smola and B. Schölkopf, *Learning with kernels*. GMD-Forschungszentrum Informationstechnik, 1998.

[18] R.-E. Fan, P.-H. Chen, and C.-J. Lin, "Working set selection using second order information for training support vector machines," *Journal of machine learning research*, vol. 6, no. Dec, pp. 1889–1918, 2005.

[19] K. Jarrett, K. Kavukcuoglu, Y. LeCun *et al.*, "What is the best multi-stage architecture for object recognition?" in *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2009, pp. 2146–2153.

[20] S. Kotsiantis, D. Kanellopoulos, and P. Pintelas, "Data preprocessing for supervised leaning," *International Journal of Computer Science*, vol. 1, no. 2, pp. 111–117, 2006.

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[22] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[23] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.