Deep Learning and Symbolic Regression for Discovering Parametric Equations

Michael Zhang, Samuel Kim[ID], Peter Y. Lu[ID], and Marin Soljačić[ID]

Abstract—Symbolic regression is a machine learning technique that can learn the equations governing data and thus has the potential to transform scientific discovery. However, symbolic regression is still limited in the complexity and dimensionality of the systems that it can analyze. Deep learning, on the other hand, has transformed machine learning in its ability to analyze extremely complex and high-dimensional datasets. We propose a neural network architecture to extend symbolic regression to parametric systems where some coefficient may vary, but the structure of the underlying governing equation remains constant. We demonstrate our method on various analytic expressions and partial differential equations (PDEs) with varying coefficients and show that it extrapolates well outside of the training domain. The proposed neural-network-based architecture can also be enhanced by integrating with other deep learning architectures such that it can analyze high-dimensional data while being trained end-to-end. To this end, we demonstrate the scalability of our architecture by incorporating a convolutional encoder to analyze 1-D images of varying spring systems.

Index Terms—Deep learning, equation discovery, high-dimensional, partial differential equation (PDE).

I. INTRODUCTION

COMPLEX systems can often be described by relatively simple and interpretable mathematical equations, ranging from Maxwell's equations for electrodynamics [1] to Hooke's law for harmonic oscillators. Thus, the discovery of the governing equations for natural and artificial systems from data is key to many scientific and engineering disciplines. While scientists and engineers have often spent years developing insights to discover such equations, machine learning has become alluring in its potential to tackle and automate extremely complex tasks. In particular, symbolic regression is a machine learning technique that searches for mathematical expressions that best fit the data, ideally resulting in a model that is interpretable and explains the underlying dynamics of the data. In one of the most popular works in this direction, Schmidt and Lipson [2] used symbolic regression to discover Hamiltonians, Lagrangians, and conservation laws for various physical systems, thus demonstrating its potential in scientific discovery.

Symbolic regression is often implemented through genetic programming, which searches through the space of mathematical expressions using evolutionary algorithms [3]. The equations are pieced together through basic building blocks known as *primitives*, which may include constants and simple functions (e.g., addition, multiplication, sine). However, these approaches do not typically scale well to high-dimensional problems and often require numerous hand-built heuristics and rules to ensure that the equation is simple to interpret and viable as a model.

There have been numerous approaches at introducing the power of deep learning into symbolic regression to enable learning equations in more complex settings. For example, AI-Feynman checks for a number of physics-inspired invariances and symmetries using both hand-built rules and neural networks to simplify the data [4]. Sparse identification of nonlinear dynamical systems (SINDy) [5], a system for discovering the governing equations of dynamical systems, has been combined with a neural network to enable discovery on high-dimensional dynamical systems [6]. PDE-Net 2.0 incorporates a symbolic network to discover partial differential equations (PDEs) using convolutional networks with constrained filters [7]. Lu et al. [8] incorporate a symbolic network with an encoder network to discover differential equation systems from partial observations. Cranmer et al. [9] perform traditional symbolic regression on graph neural network weights after training in a two-step process to discover the dynamics of many-body systems.

In particular, a neural network architecture called the equation learner (EQL) was proposed to perform symbolic regression, which takes a fully connected neural network and replaces the activation functions with the primitive

Michael Zhang is with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 USA.

Samuel Kim was with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 USA. He is now with the Department of Research and Exploratory Development, Johns Hopkins University Applied Physics Laboratory, Laurel, MD 20723 USA (e-mail: samkim@mit.edu).

Peter Y. Lu was with the Department of Physics, Massachusetts Institute of Technology, Cambridge, MA 02139 USA. He is now with the Data Science Institute, The University of Chicago, Chicago, IL 60637 USA.

Marin Soljačić is with the Department of Physics, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: soljacic@mit.edu).

functions [10], [11]. Kim et al. [12] expand upon the EQL network by integrating it with other deep learning architectures (including convolutional networks and recurrent networks) such that the entire model can be trained end-to-end through backpropagation, thus enabling symbolic regression on complex datasets including high-dimensional and dynamical systems where the relevant parameters may not be known ahead of time. Costa et al. [13] extend the EQL for recursive programs, implicit functions, and image classification.

One type of complexity we explore in this work are datasets described by parametric equations in which the underlying equation structure may stay the same but coefficients may vary along one or more dimensions. PDEs are ubiquitous in describing the dynamics of many systems, but even the most simple settings can require varying coefficients. For example, solving for electromagnetic modes or electron wave function in a material requires solving Maxwell's equation with spatially varying permittivity or the Schrödinger equation with varying potential [14], respectively. Parameters may be influenced or even controlled by external factors that are not captured in the data [15]. The nonlinear Schrödinger equation with varying coefficients has found applications in describing the Bose–Einstein condensates [16]. In addition, the varying coefficients may change in complex ways that are not easily expressible symbolically, which would result in standard symbolic regression tools failing to discover interpretable equations. Various approaches have been proposed to discover specifically parametric *PDEs*, including group sparsity combined with SINDy [17], genetic algorithms combined with averaging over local windows [18], and linear regression with kernel smoothing over adjacent coefficients [19].

In this work, we propose to discover a more general class of parametric *equations* where the equation structure is constant, but the coefficients may vary in complex ways. We extend the approach from Kim et al. [12] and enable neural-network-based symbolic regression on datasets governed by parametric equations. To this end, we propose two novel architectures: the stacked EQL (SEQL) network and the hyper EQL (HEQL) network. We demonstrate our method on various analytic equations, PDEs, and a dataset consisting of 1-D images of particles. In the last example, we combine the architectures with a convolutional neural network to analyze the images and demonstrate symbolic regression on high-dimensional datasets.

II. BACKGROUND

A. EQL Network

The EQL network is a neural network architecture that can perform symbolic regression by replacing the nonlinear activation functions with primitive functions. The architecture was initially proposed in [10] and [11] and further expanded in [12]. In Section II-A1, we briefly review the base EQL architecture for symbolic regression, while more details can be found in [12]. We also propose several modifications to the EQL network that improve its training behavior.

In Sections III-A and III-B, we propose two variants of the EQL architecture that can discover parametric equations. Note that our discussion and notation below assume that the coefficients are parameterized with respect to *time* as this provides a convenient intuition applicable to many systems. However, the parameterization could also be with respect to other quantities (e.g., space).

1) Base Architecture: The EQL network architecture closely resembles a fully connected neural network in which the output $\mathbf{h}^{(i)}$ of the ith layer can be described as

$$\mathbf{g}^{(i)} = \mathbf{W}^{(i)}\mathbf{h}^{(i-1)} \tag{1}$$

$$\mathbf{h}^{(i)} = f\big(\mathbf{g}^{(i)}\big) \tag{2}$$

where $\mathbf{W}$ is a weight matrix of the ith layer, f is the nonlinear activation function, $\mathbf{g}^{(i)}$ is a vector of the preactivation units, and $\mathbf{h}_0 = \mathbf{x}$ is the input data. A schematic of a single layer is shown in Fig. 1(b). In regression tasks, the activation function for the final layer is typically omitted, so the output of the neural network with L hidden layers is $y = \mathbf{W}^{(L+1)}\mathbf{h}^{(L)}$.

While conventional neural networks typically use activation functions such as ReLU or sigmoid, the EQL network uses a set of functions that correspond to primitive functions in symbolic regression, which represent the building blocks for more complex equations. As shown in Fig. 1(b), each component of $\mathbf{g}$ may go through a different primitive function, and a primitive function may also take multiple inputs (e.g., multiplication). The primitive functions may be duplicated to reduce the sensitivity of training to random initialization. The network is trained using the same techniques as conventional neural networks, i.e., stochastic gradient descent, and once it is trained, the discovered equation can simply be read off of the weights.

2) Sparsity: To ensure the interpretability of symbolic regression, we need to force the system to learn the simplest expression that describes the data. In genetic-programming-based approaches, this is typically done by limiting the number of terms in the expression. For the EQL network, we enforce this through the use of sparsity regularization on the network weights such that as many of the weights are set to 0 as possible. While Kim et al. [12] primarily use a smoothed $L_{0.5}$ regularization, in this work we use a relaxed form of L_0 regularization [20]. We briefly review the details here and refer the reader to [12] and [20] for more details.

The weights of the neural network are reparameterized as

$$\mathbf{W} = \tilde{\mathbf{W}} \odot \mathbf{z}$$

where $\mathbf{z}$ has the same dimensions as $\mathbf{W}$ and can be interpreted as a gate variable, and the multiplication is componentwise. Ideally, each element of $\mathbf{z}$ is a binary "gate" such that $z \in \{0, 1\}$. However, this is not differentiable and so we allow z to be a stochastic variable drawn from the hard concrete distribution [20]

$$
\begin{aligned}
u &\sim \mathcal{U}(0, 1) \\
s &= \text{sigmoid}\big(\big[\log u - \log(1 - u) + \log \alpha\big]/\beta\big) \\
\bar{s} &= s(\zeta - \gamma) + \gamma \\
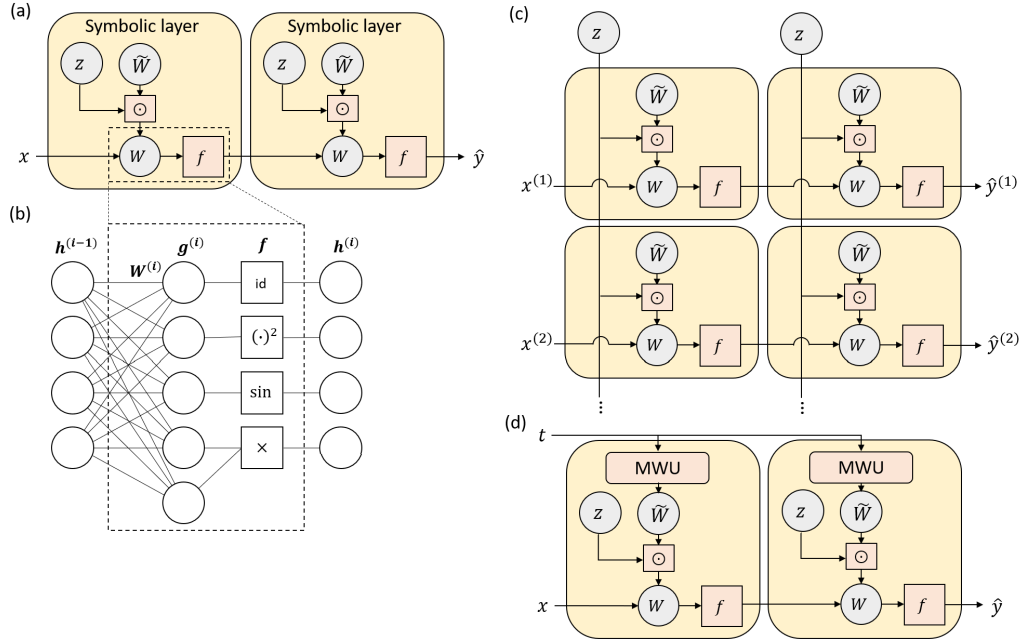z &= \min(1, \max(0, \bar{s}))
\end{aligned}
$$

Fig. 1. EQL architectures and variants for parametric equations. (a) Architecture of the base EQL network with relaxed L_0 regularization. The weights $\mathbf{W}$ are reparameterized as an elementwise product of the gate variables $\mathbf{z}$ and the weight values $\widetilde{\mathbf{W}}$. (b) Core of the symbolic layer, where the activation functions consist of the primitive functions for symbolic regression, where each element may contain a different primitive function and primitive functions may take multiple inputs. (c) Architecture of the SEQL network. Note that the indexing $x^{(j)}$ is for the time step. Each horizontal row represents an EQL network for each time step. The gate $\mathbf{z}$ is shared across time steps. (d) Architecture of the HEQL network. Note that in all the schematics, the final (linear) layer is omitted for visual simplicity.

where u is a random variable drawn from the uniform distribution $\mathcal{U}$, α is a trainable variable that describes the location of the hard concrete distribution, and β, ζ, γ are the hyperparameters that describe the distribution. The random variable s is distributed as a binary concrete distribution, which is a continuous relaxation of a binary random variable [21]. Finally, the distribution is stretched out to the (γ, ζ) interval and "folded" to delta peaks at 0 and 1 to achieve the hard concrete distribution.

In the case of binary gates, the regularization penalty would simply be the sum of $\mathbf{z}$ (i.e., the number of nonzero elements in $\mathbf{W}$). However, in the case of the hard concrete distribution, an analytical form for the expectation of the regularization penalty over the distribution parameters can be calculated [20]. The sparsity regularization loss is then

$$\mathcal{L}_R = \sum_j \text{sigmoid}\left(\log \alpha_j - \beta \log \frac{-\gamma}{\zeta}\right)$$

where j is indexing through all the weight components. While Louizos et al. [20] apply group sparsity to the rows of the weight matrices with the goal of computational efficiency, we apply parameter sparsity (to individual elements) with the goal of simplifying the expression in symbolic regression.

The advantage of L_0 regularization is that it enforces sparsity without placing a penalty on the magnitude of the weights by placing a penalty on the expected number of nonzero weights. In addition, it lends itself to a straightforward definition of group sparsity across time steps as we will see in Section III-A. In our experiments, we use the hyperparameters for the L_0 regularization suggested by Louizos et al. [20].

3) Skip Connections: In this work, we also add skip connections to the EQL network to introduce an inductive bias toward simpler equations while simultaneously enabling the discovery of more complex equations. The most well-known type of skip connections were introduced in ResNets, which take the output of a layer and add it to the layer ahead with the goal of allowing gradient information to efficiently propagate through many layers and enabling extremely deep architectures [22]. While these would be feasible to implement in the EQL network, they would increase the complexity of the equation as information flows through the network. In contrast, we turn to the skip connections introduced by DenseNets which concatenates, rather than sums, the output of the previous layer with that of the next layer [23]. More specifically, we modify (2) as

$$\mathbf{h}^{(i)} = \left[f\left(\mathbf{g}^{(i)}\right); \mathbf{h}^{(i-1)} \right]. \tag{3}$$

Skip connections introduce a slight inductive bias toward learning simpler functions, since functions can route "directly" to the output without needing to go through the identity primitive function of successive layers. In addition, skip connections minimize instabilities during training that can arise as a result of gradients exploding as they pass through the primitive functions. Thus, skip connections allow us to train EQL networks with more layers, which in turn can enable learning more complex equations.

B. Parametric Equations

In this work, we focus specifically on learning equations where the *structure* of the equation (i.e., terms and operators)
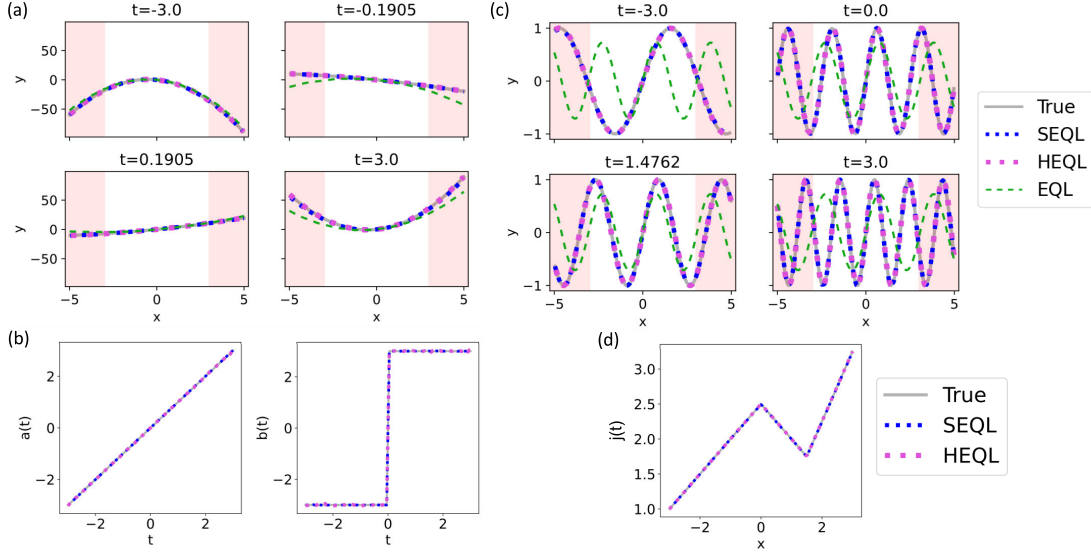
Fig. 2. Learning parametric equations. (a) and (b) Learning the function f_1 which contains a discontinuity at $t = 0$. (c) and (d) Learning the function f_2 which corresponds to a sinusoid with a frequency that varies nonsmoothly as a function of t. (a) and (c) Predictions after training the EQL, SEQL, and HEQL networks in the range $-3 < x < 3$ for various values of t. Values outside of this range (highlighted in red) are extrapolated. (b) and (d) Learned functions for the varying coefficients.

TABLE I

LEARNED EQUATIONS BY THE EQL NETWORK AFTER TRAINING ON SIMPLE PARAMETRIC EQUATIONS

True	EQL
$f_1 = tx^2 + 3\,\mathrm{sgn}(t)x$	$0.02x^2 - 0.14t + 1.86xt + 1.01x^2t$ $+ 0.03t^2 - 0.01x^2t^2 - 0.02t^3 - 0.01xt^3$ $+ (0.09 - 1.80x + 0.01x^2)\sin(1.91t)$ $+ 0.77x\sin(3.71t) + 0.32\sin(3.22xt)$ $- 0.10$
$f_2 = \sin(j(t)x)$	$0.720\sin(2.05x)$

remains constant, but the value of the numerical coefficients or constants may vary over the dataset. Prior works have focused specifically on learning PDEs where the differential terms are fixed but the coefficients vary over space or time, termed *parametric* PDEs [17], [18], [19]. Here, we generalize symbolic regression to parametric *equations*. The ability to learn parametric equations may greatly expand the scope of symbolic regression, especially in cases where the coefficients may vary over the dataset in arbitrarily complex ways that are difficult to express symbolically. Such behavior would greatly impede the performance of traditional symbolic regression approaches that attempt to find the simplest equation describing the dataset. To illustrate this, we train the EQL network on two simple parametric equations listed in Table I.

The first function is a parabolic curve described as

$$f_1(x, t) = tx^2 + 3\,\mathrm{sgn}(t)x$$

where sgn is the *sign* function (also known as the *signum* function)

$$\mathrm{sgn}(t) = \begin{cases} -1, & \text{if } t < 0 \\ 0, & \text{if } t = 0 \\ 1, & \text{if } t > 0. \end{cases}$$

The function notably contains a discontinuity at $t = 0$ and thus cannot be described in terms of smooth functions.

As seen in Table I, the EQL network learns an overly complicated and incorrect equation with over a dozen terms that likely signify its attempt to fit the discontinuity. It is difficult to interpret and thus fails the goal of discovery. In addition, the results of fitting the EQL network to the data are shown in Fig. 2(a). While the EQL network seems to fit reasonably well inside the training regime (in the range $-3 < x < 3$), it fails to extrapolate well since it has not learned the correct equation. (In principle, the function f_1 is simple enough such that an EQL network with sigmoidal activation functions could approximate it with reasonable accuracy, but we choose this example to illustrate some of the difficulties of symbolic regression.)

In the second example, we look at a sinusoidal curve where the frequency varies nonsmoothly as a function of time

$$f_2(x, t) = \sin(j(t)x)$$

where we have defined a "jagged" function

$$j(t) = \begin{cases} 0.5t + 2.5, & \text{if } t < 0 \\ -0.5t + 2.5, & \text{if } 0 \le t < 1.5 \\ t + 0.25, & \text{otherwise.} \end{cases}$$

The jagged function $j(t)$ is illustrated in Fig. 2(d). As seen in Table I, the EQL network is unable to learn the parametric form of the sinusoidal frequency and thus fits poorly.

In contrast, the proposed architectures in Section III are able to learn the correct equations as shown in Table II, as we will discuss next.

III. PARAMETRIC EQL VARIANTS

In this work, we propose two variants of the EQL network to learn parametric equations: the SEQL and the HEQL.

A. Stacked Architecture (SEQL)

The first extension we propose to analyze parametric equations is to train a separate EQL network for each time step, an architecture that we call the SEQL network. Suppose we have a dataset that is indexed by the time step j

$$\mathcal{D} = \left\{ \left\{ x^{(i,j)}, y^{(i,j)} \right\}_{i=1}^{N^{(j)}} \right\}_{j=1}^{N_t} \tag{4}$$

where N_t is the number of time steps and $N^{(j)}$ is the number of data points in the jth time step (note that $N^{(j)}$ does not need to be constant across time steps). For layer i of the SEQL network, we can construct N_t separate weight matrices, $\{\tilde{\mathbf{W}}^{(i,j)}\}_{j=1}^{N_t}$, such that (1) and (2) are modified as

$$\mathbf{g}^{(i,j)} = \mathbf{W}^{(i,j)} \mathbf{h}^{(i-1,j)} \tag{5}$$
$$\mathbf{h}^{(i,j)} = f\left(\mathbf{g}^{(i,j)}\right). \tag{6}$$

In other words, we parameterize a separate EQL network for each time step, as shown in Fig. 1(c).

If we naïvely train N_t separate EQL networks, then it is possible that each network may learn a different equation in each time step. In addition, each network would only see approximately $1/N_t$ of the total data, thus reducing data efficiency. To counteract this, we enforce that the different networks learn the same equation by implementing group sparsity through weight sharing of the gate variable $\mathbf{z}$. For the ith layer of the jth time step, we further modify (6) as

$$\mathbf{h}^{(i,j)} = f\left(\left(\tilde{\mathbf{W}}^{(i,j)} \odot \mathbf{z}^{(i)}\right) \mathbf{h}^{(i-1,j)}\right). \tag{7}$$

Note that $\mathbf{z}$ is not parameterized with respect to the time step j. For an architecture with L hidden layers, there are $(L+1)\cdot N_t$ weight matrices and $L+1$ gate matrices.

Another modification we make to the architecture is weight regularization across time steps to introduce an inductive bias toward smoothness in the coefficients. We use L_2 regularization loss between adjacent time steps. Looking at just a single element $w_{k,l}$ of $\mathbf{W}$ in a single layer for notational simplicity, the interlayer L_2 loss is simply

$$L_{S,k,l} = \sum_{j=1}^{N_t-1} \left(w_{k,l}^{(j+1)} - w_{k,l}^{(j)} \right)^2 \tag{8}$$

and the total interlayer regularization loss is

$$\mathcal{L}_S = \sum_{i,k,l} L_{S,k,l}^{(i)} \tag{9}$$

where i indexes the layer. This regularization pushes coefficients in adjacent time steps closer together and can more effectively counteract noisy datasets.

B. HEQL Architecture

We also propose a second variant of the EQL network, the HEQL network, in which the weights $\tilde{\mathbf{W}}$ are reparameterized as a function of the varying coefficient, i.e., $\tilde{\mathbf{W}}(t)$. While a number of models can be used to parameterize the weights, we use a fully connected neural network as it is a flexible model that can fit arbitrary functions and can be trained with backpropagation, allowing the entire system to be trained end-to-end. We call this fully connected neural network the meta-weight unit (MWU). The architecture is shown in Fig. 1(d).

This idea is similar to that of hypernetworks, in which a neural network is used to generate the weights of another neural network [24]. The general idea of using a network to parameterize or interact with the weights of another network has been most notably leveraged for meta-learning [25], [26], [27], [28] and has also been applied to a variety of other architectures, including the neural ODE [29] and HyperPINN [30].

The HEQL has a separate MWU in each layer (including the linear output layer) which takes the parametric variable t as an input and outputs the weight matrix $\tilde{\mathbf{W}}^{(i)}(t)$ for that layer. The gate variables $\mathbf{z}$ are not modified and are thus not a function of t. As a result, all the "time steps" share the same sparsity regularization, thus avoiding the need for any further modifications to implement group sparsity.

The advantage of this architecture compared with the SEQL is that the HEQL does not replicate the EQL network for each time step, thus saving on computational memory especially for large N_t. The architecture can also make predictions on a continuous domain of t and does not need require the data to align along a fixed grid in time, unlike the prior work on discovering parametric PDEs [17], [18], [19]. More specifically, rather than viewing the dataset as (4), we have greater flexibility and can view the dataset as

$$\mathcal{D} = \left\{ x^{(i)}, y^{(i)}, t^{(i)} \right\}_{i=1}^{N}. \tag{10}$$

Although we do not explicitly regularize the functional space of the parametric coefficients, neural networks tend to generalize well despite typically being overparameterized, which is a topic of significant interest [31], [32], [33], [34]. In practice, this means that the predictions of neural networks for regression tasks tend to be smooth, and so the function of the parametric coefficient will also tend to be smooth.

IV. RESULTS

We now look at several different problem settings with parametric quantities that can be analyzed by our system. For simplicity, we highlight some of the results here, and the remainder can be found in the appendix. Section IV-A demonstrates some simple benchmarks to highlight the aspects of learning parametric equations. Section IV-B shows results on the PDE datasets taken from other works. Finally, Section IV-C presents results on 1-D images of a spring system to demonstrate the ability to perform symbolic regression on higher dimensional systems.

Our datasets and code are made publicly available at https://github.com/samuelkim314/parametric-eql.

A. Analytic Expressions

To verify the ability of the SEQL and HEQL networks to discover parametric equations, we benchmark the networks on the analytical expressions discussed in Section II-B and listed in Table I. While we train the networks on data drawn from

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

6   IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS

TABLE II
RESULTS FOR TRAINING ON PARAMETRIC ANALYTIC EXPRESSIONS. LEARNED EQUATIONS ARE EXTRACTED FOR VARIOUS VALUES OF t

Benchmark	t	True	Learned equations SEQL	Learned equations HEQL
$f_1(t,x)$	-2.619	$-2.619x^2 - 3x$	$-2.621x^2 - 2.995x - 0.045$	$-2.628x^2 - 3.017x - 0.026$
	-1.095	$-1.095x^2 - 3x$	$-1.096x^2 - 3.003x - 0.024$	$-1.106x^2 - 3.005x - 0.011$
	0.381	$0.381x^2 + 3x$	$0.383x^2 + 3.001x + 0.006$	$0.386x^2 + 3.013x + 0.005$
	1.905	$1.905x^2 + 3x$	$1.908x^2 + 3.000x + 0.025$	$1.909x^2 + 3.020x + 0.031$
$f_2(t,x)$	-2.619	$\sin(1.191x)$	$0.999\sin(1.191x)$	$1.000\sin(1.190x)$
	-1.095	$\sin(1.953x)$	$0.999\sin(1.952x)$	$1.000\sin(1.952x)$
	0.381	$\sin(2.310x)$	$0.997\sin(2.310x)$	$1.000\sin(2.309x)$
	1.905	$\sin(2.155x)$	$0.999\sin(2.155x)$	$1.000\sin(2.155x)$

the domain $x \in [-3, 3]$, we test the networks on a wider domain $x \in [-5, 5]$ to evaluate extrapolation performance. In Appendix B, we also provide benchmarks for additional analytic expressions.

Fig. 2 shows the results for learning f_1 and f_2 using the SEQL and HEQL networks. The true function and the predicted function are plotted for various values of t. In all the cases, the SEQL/HEQL predictions are visually indiscernible from the true function in both the training regime and the test regime, demonstrating that the architectures are able to extrapolate. As in the case of the original EQL network, the learned equations can be extracted from the trained network by simply processing the learned weights with software for symbolic mathematics. In particular, we use SymPy, a Python package for symbolic mathematics, to simplify the resulting expression [35]. For clarity, we also omit negligible terms (i.e., those with coefficient magnitudes < 0.01) in the final expression. The extracted equations found by the SEQL and HEQL for the parametric function f_1 and f_2 at various time steps are shown in Table II. The mean squared error (mse) on the test datasets can be found in Appendix B.

Upon inspection of the extracted equations over multiple time steps for learning f_1, we see that the architectures have successfully discovered the function $\hat{f}_1 = a(t)x^2 + b(t)x + \epsilon(t)$ where $a(t)$ and $b(t)$ are the varying coefficients and ϵ is a small number that can be either eliminated with further training or ignored upon inspection. The predicted parametric coefficients $a(t)$ and $b(t)$ match the true coefficients extremely closely, as seen in Fig. 2(b). Note that the SEQL/HEQL networks are able to learn the discontinuous sgn function without any apparent smoothing at $t = 0$. Discontinuous coefficients would be difficult to learn using other methods for parametric equations that rely on local averaging [18] or smoothing [19]. This also contrasts with the original EQL network which was unable to learn the parametric equation as shown in Table I.

For learning the sinusoidal function f_2, both the SEQL and HEQL networks have learned the equation $\hat{f}_2 = \sin(a(t)x)$ as seen in Table II where $a(x)$ is plotted in Fig. 2(d). Again, the predictions match the true function extremely well across time steps and outside of the training regime. Although sinusoidal functions are typically difficult to learn through linear regression techniques, the SEQL and HEQL networks are able to learn this function across multiple spatial frequencies.
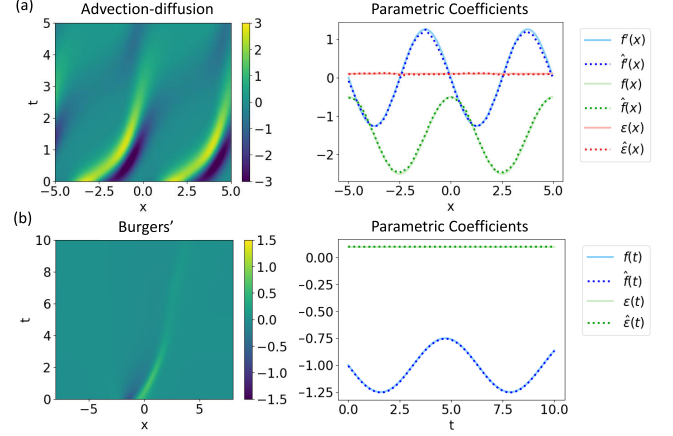

Fig. 3. Results for learning (a) advection–diffusion equation using the HEQL network and (b) Burgers' equation using the SEQL network. Left-hand plots show the predicted values of u_t, and the right-hand plots illustrate the varying coefficient functions.

Note that because the varying coefficient is inside the sgn and sin functions for f_1 and f_2, respectively, other methods for learning parametric equations such as those proposed in [5] or [19] that rely on linear regression techniques would not be able to discover these types of equations. In contrast, the multilayer architecture of the SEQL and HEQL networks allows for the varying coefficients to be inside nested functions, enabling discovery of much more complex parametric equations.

The results for additional benchmark equations are listed in Appendix B. Interestingly, there is no clear trend on whether the SEQL or HEQL tends to perform better.

B. Partial Differential Equations

Next, we investigate learning PDEs with varying coefficients from data. In this setting, a quantity of interest $u(x, t)$ can be defined by a function of its partial derivatives (e.g., u_t, u_x, u_xx) and a parametric dependence on time, $\mu(t)$

$$u_t = N(u, u_x, u_xx, \ldots, \mu(t))$$

where N is the evolution function that we wish to learn. For notational convenience, we drop the explicit dependence of u on x and t.

Prior works in discovering parametric equations have focused on the setting of PDEs [17], [18], [19], as PDEs

TABLE III
MSE After Training on the PDE Datasets

Benchmark	u_t MSE			Coefficient MSE		
	SINDy	SEQL	HEQL	SINDy	SEQL	HEQL
Advection-diffusion, with cross terms	$\mathbf{2.34 \times 10^{-7}}$	2.87×10^{-5}	1.99×10^{-5}	3.99×10^{-2}	2.99×10^{-3}	$\mathbf{1.44 \times 10^{-3}}$
Advection-diffusion, no cross terms	$\mathbf{2.34 \times 10^{-7}}$	2.12×10^{-5}	1.57×10^{-5}	1.73×10^{-2}	2.30×10^{-3}	$\mathbf{1.37 \times 10^{-3}}$
Burgers', with cross terms	$\mathbf{5.11 \times 10^{-8}}$	5.66×10^{-6}	5.54×10^{-6}	$\mathbf{3.95 \times 10^{-7}}$	8.81×10^{-6}	9.83×10^{-6}
Burgers', no cross terms	2.30×10^{-4}	$\mathbf{2.55 \times 10^{-7}}$	8.02×10^{-6}	1.73×10^{-3}	$\mathbf{8.41 \times 10^{-6}}$	4.44×10^{-5}

are ubiquitous in describing dynamics in a variety of fields. For ease of comparison, we benchmark our architectures on two of the datasets provided by Rudy et al. [17]: the advection–diffusion equation and Burgers' equation with varying coefficients. The partial differential terms (e.g., u_x, u_{xx}) are precomputed from the dataset and concatenated with the input u.

We note that SINDy is not able to automatically calculate cross terms (e.g., uu_x) and so the cross terms were also precomputed and fed into SINDy in the original work [17]. We label this approach as "with cross terms" in Table III. In contrast, the SEQL and HEQL architectures are able to automatically discover cross terms as necessary, and so we also carry out experiments that omit the cross terms in the input, labeled "no cross terms."

1) Advection–Diffusion Equation: The advection–diffusion equation describes numerous physical transport systems and has been applied to describe the movement of pollutants, reservoir flow, heat, and semiconductors. We use an adaptation of the equation that includes a spatially dependent velocity field, as in [17]

$$u_t = f'(x)u + f(x)u_x + \epsilon u_{xx} \tag{11}$$

where $f(x) = -1.5 + \cos(2\pi x/5)$ and $\epsilon = 0.1$. Note that the parametric quantities vary with respect to space rather than time. Thus, for our experiments, we modify the SEQL and HEQL architectures to parameterize the varying coefficient with respect to space (in practice this simply involves relabeling the dataset).

Table III lists the mse for both the predicted u_t and the learned coefficient functions. Unsurprisingly, SINDy achieves the lowest mse on u_t since it reduces the problem to a linear system that can be solved efficiently. Interestingly, SEQL and HEQL achieve better predictions on the coefficient functions, which is perhaps due to the implicit regularization in the architectures. Without cross terms, the SEQL and HEQL achieve an even lower error, which may be due to the reduced dimensionality of the input. Table IV shows the equations that the SEQL and HEQL have learned after training for select values of x. Both the networks have learned an equation of the form $\hat{u}_t = \hat{f}'(x)u + \hat{f}(x)u_x + \hat{\epsilon}(x)u_{xx}$ and have thus successfully discovered the equation structure. The predicted $\hat{u}_t$ along with the learned parametric coefficients [i.e., $\hat{f}'(x), \hat{f}(x), \hat{\epsilon}(x)$] are shown in Fig. 3(a) for the HEQL network, which match the actual values very closely. The results for the SEQL (not shown) are visually very similar.

2) Burgers' Equation: Burgers' equation is an important differential equation originally proposed to model turbulent flow that has also been applied to other processes such as traffic flow and boundary layer behavior. Here, we analyze Burgers' equation with an oscillating coefficient for the nonlinear term, as in [17]

$$u_t = f(t)uu_x + \epsilon u_{xx} \tag{12}$$

where $f(t) = -(1 + (\sin(t)/4))$ and $\epsilon = 0.1$.

Note that this equation contains a cross term, uu_x. When including the cross term in the input, SINDy correctly learns the equation and achieves a low error, as shown in Table III. However, when the cross term is omitted from the input, SINDy is unable to learn the correct equation and adds incorrect terms (i.e., u_x, u_{xxx}) to compensate. In contrast, the SEQL and HEQL are able to achieve low errors in both the cases and achieve the correct equation form as shown in Table IV. Fig. 3(b) shows that the SEQL network is able to accurately predict the function and the parametric coefficients. Thus, our system is able to automatically learn these cross terms using the multiplication primitive and, more generally, can learn the form of a nonlinear PDE.

C. Spring System

Finally, we demonstrate the ability of the parametric EQL networks to perform symbolic regression on structured, high-dimensional data by integrating our architectures with other deep learning architectures and training the entire model end-to-end.

We consider a dataset that consists of pairs of 1-D images of point particles that interact through a spring-like force. The input data are a 1-D grayscale image with 64 pixels which represents a 1-D spatial domain $\psi \in [-4, 4]$. Each image contains a single particle, represented by a Gaussian with mean centered at its position ψ_i and a fixed variance of 0.1. We look at two different targets for symbolic regression: the spring force

$$F = -k(t)(\psi_2 - \psi_1) \tag{13}$$

and the spring energy

$$E = \frac{k(t)}{2}(\psi_2 - \psi_1)^2 \tag{14}$$

where $k(t) = (5 - t)/2$. These are interpretable equations in that we know that the spring force and potential only depend on the spring constant, $k(t)$, and the distance between the two particles. The spring constant decreases over time, which we can imagine represents a spring degrading with use. The manner in which the spring degrades may or may not be analytical, and so we treat this as a parametric quantity.

TABLE IV

LEARNED EQUATIONS ON SELECT x OR t VALUES (DEPENDING ON THE VARYING PARAMETER) FOR THE PDE DATASETS WITHOUT CROSS TERMS

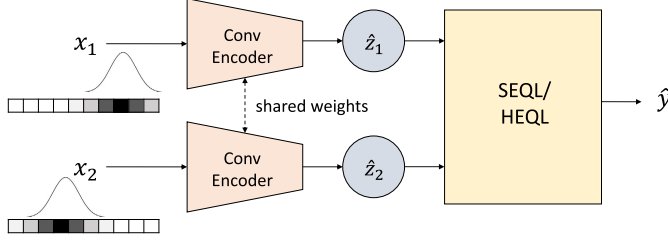| Benchmark	x or t	True	SEQL	HEQL
Advection-diffusion equation	-4.375	$-0.89u - 0.79u_x + 0.10u_{xx}$	$-0.86u - 0.77u_x + 0.11u_{xx}$	$-0.86u - 0.76u_x + 0.11u_{xx}$
	-1.875	$0.89u - 2.21u_x + 0.10u_{xx}$	$0.83u - 2.14u_x + 0.07u_{xx}$	$0.86u - 2.16u_x + 0.09u_{xx}$
	0.625	$-0.89u - 0.79u_x + 0.10u_{xx}$	$-0.86u - 0.77u_x + 0.11u_{xx}$	$-0.86u - 0.77u_x + 0.11u_{xx}$
	3.125	$0.89u - 2.21u_x + 0.10u_{xx}$	$0.83u - 2.14u_x + 0.07u_{xx}$	$0.88u - 2.17u_x + 0.10u_{xx}$
Burgers' equation	0.627	$-1.15uu_x + 0.10u_{xx}$	$-1.15uu_x + 0.10u_{xx}$	$-1.16uu_x + 0.10u_{xx}$
	3.137	$-1.00uu_x + 0.10u_{xx}$	$-1.00uu_x + 0.10u_{xx}$	$-1.01uu_x + 0.10u_{xx}$
	5.647	$-0.85uu_x + 0.10u_{xx}$	$-0.86uu_x + 0.10u_{xx}$	$-0.85uu_x + 0.10u_{xx}$
	8.157	$-1.24uu_x + 0.10u_{xx}$	$-1.24uu_x + 0.10u_{xx}$	$-1.25uu_x + 0.10u_{xx}$

Fig. 4. Combined architecture used for high-dimensional system tasks involving a convolutional encoder followed by an EQL network.

Fig. 5. Results for learning the spring force F. (a) Predictions for select values of t. Outputs with $|\psi_2 - \psi_1| > 4$ (highlighted in red) are extrapolated. (b) Coefficient functions in the equation $\hat{F}(t, \hat{z}_1, \hat{z}_2) = \hat{k}_1(t) \cdot \hat{z}_1 - \hat{k}_2(t) \cdot \hat{z}_2$ learned by the SEQL network. (c) Latent variable encodings for the force function F learned by (left) convolutional SEQL network and (right) ReLU network.

To approach this problem, we use the architecture shown in Fig. 4. Each image is fed into a separate encoder, where the two encoders share the same weights. The encoder consists of two convolutional layers followed by three fully connected layers and a batch normalization layer. Each encoder outputs a single-dimensional latent variable (either $\hat{z}_1$ or $\hat{z}_2$) which is then fed into the parametric EQL network (which can be either the SEQL or the HEQL). The batch normalization layer serves to constrain the range of the latent variable so that the SEQL/HEQL network does not need to scale to arbitrarily sized inputs when training end-to-end. The SEQL/HEQL network has a single scalar output, which is trained to match either the spring force or the spring energy. The entire network is trained end-to-end and is only shown the inputs and the output, but must learn an appropriate representation $\hat{z}_i$. While there are no constraints on the latent representation $\hat{z}_i$, we expect it to have a one-to-one mapping to the true position of the particle, ψ_i.

For all the tests, 512 training data points with $\psi_1, \psi_2 \in [-3, 3]$ were sampled for each of 128 fixed values of $t \in [-3, 3]$. To evaluate the extrapolation ability of these architectures, training data points were restricted to pairs with $|\psi_2 - \psi_1| \leq 4$, while no such restriction was imposed on testing data. In addition, we compare against a baseline test of a model consisting of the same encoder architecture with a dense ReLU network replacing EQL network. We call this baseline the **ReLU network**.

The results for learning the spring force are shown in Fig. 5. All three of the SEQL, HEQL, and ReLU architectures accurately predict the force inside the training domain, but only the SEQL and HEQL networks are able to extrapolate outside of the training regime, whereas the ReLU network fails to extrapolate. In addition, the SEQL network learns the governing equation as shown in Table V, with the learned parametric coefficient plotted in Fig. 5(b) (results for the
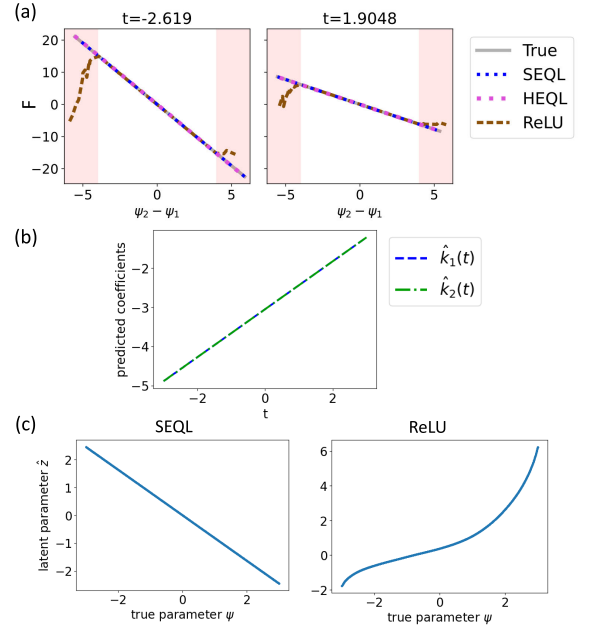
TABLE V

LEARNED EQUATIONS OF THE SEQL ON SELECT t VALUES FOR THE SPRING FORCE FUNCTION $F(t, \psi_1, \psi_2) = -(5 - t)/2 \cdot (\psi_2 - \psi_1)$ IN THE LATENT SPACE AND TRANSFORMED TO THE ORIGINAL PARAMETER SPACE

t	True	Learned Latent	Learned Transformed
-2.619	$-3.81(\psi_2 - \psi_1)$	$-4.66\hat{z}_1 + 4.66\hat{z}_2$	$3.82\hat{\psi}_1 - 3.82\hat{\psi}_2$
-1.095	$-3.05(\psi_2 - \psi_1)$	$-3.72\hat{z}_1 + 3.72\hat{z}_2$	$3.05\hat{\psi}_1 - 3.05\hat{\psi}_2$
0.381	$-2.31(\psi_2 - \psi_1)$	$-2.82\hat{z}_1 + 2.82\hat{z}_2$	$2.31\hat{\psi}_1 - 2.31\hat{\psi}_2$
1.905	$-1.55(\psi_2 - \psi_1)$	$-1.89\hat{z}_1 + 1.89\hat{z}_2$	$1.55\hat{\psi}_1 - 1.55\hat{\psi}_2$

HEQL are similar). The equations that the SEQL network learns are fairly simple and interpretable and can be written as $\hat{F} = \hat{k}_1(t)\hat{z}_1 - \hat{k}_2(t)\hat{z}_2$. Upon inspection, we see that $\hat{k}_1(t) \approx \hat{k}_2(t)$ and so we can simplify the learned expression to $\hat{F} = \hat{k}(t)\hat{z}_1 - \hat{k}(t)\hat{z}_2$. Thus, the SEQL network has discovered the true force equation underlying the system.

In addition, while the SEQL network discovers an equation in terms of $\hat{z}_{1,2}$, it also learns a linear mapping of the latent variable to the true position as shown in Fig. 5(c). While there is no explicit constraint or regularization placed on the

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

ZHANG et al.: DEEP LEARNING AND SYMBOLIC REGRESSION FOR DISCOVERING PARAMETRIC EQUATIONS 9

TABLE VI

LEARNED EQUATIONS OF THE HEQL NETWORK ON SELECT t VALUES FOR THE FUNCTION $E(t, \psi_1, \psi_2) = (5 - t)/4 \cdot (\psi_2 - \psi_1)^2$ IN THE LATENT SPACE AND TRANSFORMED TO THE ORIGINAL PARAMETER SPACE

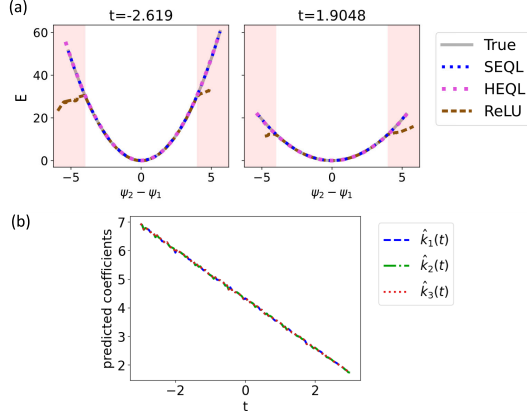| t	True	Learned Latent	Learned Transformed
-2.619	$-1.90(\psi_2 - \psi_1)^2$	$6.59\hat{z}_1^2 + 6.59\hat{z}_2^2 - 13.18\hat{z}_1\hat{z}_2 + 0.02$	$1.91\hat{\psi}_1^2 + 1.91\hat{\psi}_2^2 - 3.82\hat{\psi}_1\hat{\psi}_2 + 0.02$
-1.095	$-1.52(\psi_2 - \psi_1)^2$	$5.27\hat{z}_1^2 + 5.27\hat{z}_2^2 - 10.55\hat{z}_1\hat{z}_2 + 0.01$	$1.53\hat{\psi}_1^2 + 1.53\hat{\psi}_2^2 - 3.06\hat{\psi}_1\hat{\psi}_2 + 0.01$
0.381	$-1.16(\psi_2 - \psi_1)^2$	$4.01\hat{z}_1^2 + 4.01\hat{z}_2^2 - 8.02\hat{z}_1\hat{z}_2 + 0.01$	$1.16\hat{\psi}_1^2 + 1.16\hat{\psi}_2^2 - 2.33\hat{\psi}_1\hat{\psi}_2 + 0.01$
1.905	$-0.77(\psi_2 - \psi_1)^2$	$2.64\hat{z}_1^2 + 2.64\hat{z}_2^2 - 5.28\hat{z}_1\hat{z}_2 + 0.01$	$0.77\hat{\psi}_1^2 + 0.77\hat{\psi}_2^2 - 1.53\hat{\psi}_1\hat{\psi}_2 + 0.01$

Fig. 6. Results for learning the spring energy E. (a) Predictions for select values of t. Outputs with $|\psi_2 - \psi_1| > 4$ (highlighted in red) are extrapolated. (b) Coefficient functions in $f(t, \hat{z}_1, \hat{z}_2) = \hat{k}_1(t) \cdot \hat{z}_1^2 + \hat{k}_2(t) \cdot \hat{z}_2^2 - 2\hat{k}_3(t) \cdot \hat{z}_1\hat{z}_2$ learned by the HEQL network.

latent space, because the EQL network must learn to use the latent variable to form the equation, the end-to-end training of the architecture forces the mapping to be an analytical transformation of the original variable, which in this case is a linear mapping. In contrast, while it is one-to-one, the latent variable mapping for the ReLU network is not linear since there is no bias to make the mapping linear. Using this linear mapping, we can perform a linear regression to find the approximate relationship between $\hat{z}$ and $\hat{\psi}$ and reconstruct the discovered equation in terms of $\hat{\psi}$, which is shown in the right-most column of Table V.

We see similar results for the spring potential data, this time using the HEQL network, in Fig. 6 and Table VI. Again, the SEQL and HEQL architectures are able to extrapolate outside of the training regime, whereas the ReLU network fails to extrapolate. Note that in this case, the HEQL learns $\hat{E}(t, \hat{z}_1, \hat{z}_2) = \hat{k}_1(t)\hat{z}_1^2 + \hat{k}_2(t)\hat{z}_2^2 - 2\hat{k}_3(t)\hat{z}_1\hat{z}_2 + \epsilon(t)$ where $\hat{k}_1 \approx \hat{k}_2 \approx \hat{k}_3$ and ϵ is small. Thus, the HEQL network has discovered the correct equation.

V. DISCUSSION

All the results for both the architectures can be found in the appendix. Comparing the two architectures, for a moderate number of time steps (e.g., $N_t < 512$) the SEQL has fewer parameters than the HEQL; despite this, however, the HEQL trains on each minibatch $3.7\times$ faster than the SEQL on the analytic equations for our settings of hyperparameters and network sizes. This is likely because the limiting factor is the computation of the activation functions, which must be processed separately for each component of the layer output h (whereas in a conventional neural network the use of a single activation function is able to take advantage of vectorization optimizations). For a larger number of time steps, (e.g., $N_t > 512$), the HEQL is more memory-efficient as well since the SEQL parameters scale linearly with the number of time steps. Thus, the HEQL is able to scale to larger datasets. Future work can include reducing the memory requirements of the MWU inside the HEQL, perhaps by parameterizing the EQL network using lower rank matrices so that the dimensionality of the MWU output can be reduced.

In terms of the data format, prior methods rely on gridded data [17], [18] while both the SEQL and the HEQL allow a variable grid along the varying dimension. The HEQL architecture takes this flexibility a step further in that it is able to interpolate in time and make predictions at arbitrary time points, whereas the stacked architecture is fixed to certain time points. On the other hand, we find that the stacked architecture is less sensitive to the random initialization and converges more quickly to the solution. Thus, the two architectures trade off between performance and flexibility. One possible direction for future work to bridge this gap is to introduce different learning rate schedules for the EQL network and the MWU in the HEQL architecture, as the EQL network typically requires large learning rates to escape local minima and converge, whereas large learning rates may be detrimental to the MWU.

As mentioned in Section IV, the SEQL and HEQL architectures are also more flexible than previous approaches in the types of equations that can be discovered. For example, the previous approaches rely on variants of linear regression and are thus not able to discover varying coefficients that are inside other functions such as $\sin(f(t)x)$. In addition, our approach is able to automatically discover cross terms, whereas the SINDy framework relies on these terms being precomputed.

VI. CONCLUSION

We have proposed two different variants of the EQL network—the SEQL architecture and the HEQL architecture—to enable neural-network-based symbolic regression of parametric equations where coefficients may vary. We have demonstrated our system on simple analytic equations, PDEs, and a dataset encoded as images and have found that we are able to discover interpretable equations that can extrapolate outside of the training regime. Our method has the potential to combine the power of deep learning and symbolic regression to enable scientific discovery on complex and high-dimensional datasets.

We note that in our experiments we used simple functions for the varying coefficients for simplicity. However, our method is not constrained to these types of expressions, and the parametric coefficient can more generally be any arbitrary function. Thus, our method can be applied to systems that we know are partially governed by an analytic equation, but partially governed by some other mechanism that may be too complex or noisy to capture. This is similar in spirit to methods for solving PDEs that replace part of the equation with a neural network, often to correct for discretization errors [36], [37].

The HEQL architecture can be viewed as implementing functional regularization. Functional regularization, which imposes regularization on the learned function rather than on the parameters, is attractive as it is much more intuitive and can lead to more natural methods for tasks such as continual learning [38], [39]. It has been explored in neural networks through regularizing the predictions on batches of data [38] or through defining the prior over functions rather than weights in the case of Bayesian neural networks [40], [41]. In the case of the EQL network, the coefficients of the resulting equation are typically very simple functions (oftentimes the identity function) of the weights themselves. This means that in practice, the L_2 smoothing regularization in the SEQL network architecture often implicitly applies to the *function space*, even though we are explicitly applying the regularization in the *weight space*. In the case of the HEQL architecture, the output of fully connected neural networks will tend to be smooth due to modern training methods such as stochastic gradient descent (which is a topic of great interest in itself), and so the MWU itself acts as a regularization on the function space of the EQL network. Given this inherent regularization, another interesting direction for future work would be to characterize the data efficiency of our proposed architectures, especially for sparse datasets, as large datasets are not always available in science and engineering applications.

While the proposed architectures in this work aim to address the challenge of discovering parametric equations, they still share some of the limitations of the original EQL network proposed by Kim et al. [12] including sensitivity to random initializations and difficulties with converging when using nonconventional activation functions. The skip connections and L_0 regularization in this work improve the convergence behavior compared with the original EQL network, but there is still further room for improvement. For example, the Snake function, defined as $x + (1/a)\sin^2(ax)$ where a is a learnable parameter, could be used to learn periodic functions while maintaining monotonicity and thus improve convergence [42]. Padé activation units (PAUs) [43] or the neural arithmetic logic unit (NALU) [44], [45] could be used to learn rational functions, since the discontinuity of the division operator makes it difficult to straightforwardly incorporate as an activation function. These limitations have not hampered most of the existing proposed models for discovering differential equations, as known differential equations rarely include such terms. However, since such functions are widely prevalent in science and engineering equations, a future direction should explore a more robust way to learn these types of functions.
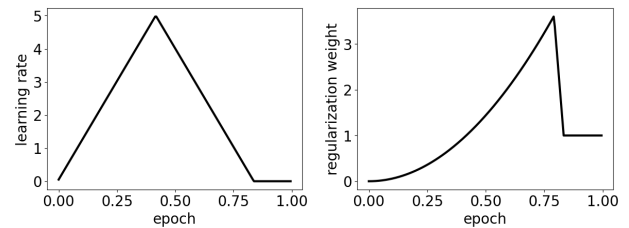

Fig. 7. (Left) Learning rate and (right) regularization weight schedules during training relative to `base_lr` and `base_rw`.

APPENDIX A
ARCHITECTURE DETAILS

Each of the SEQL and HEQL consists of two hidden layers. The activation functions in each hidden layer consist of

$$[1(\times 2), g(\times 4), g^2(\times 4), \sin(2\pi g)(\times 2), g_1 * g_2(\times 2)]$$

where $(\times i)$ indicates the number of times each activation function is duplicated. The sin function has a multiplier inside so that the functions more accurately represent their respective shapes inside the input domain of $x \in [-1, 1]$. The exact number of duplications is arbitrary and does not have a significant impact on the system's performance.

For the HEQL, the MWU consists of a fully connected neural network with three hidden layers of 64, 64, and 256 hidden units, respectively. The hidden layers in the MWU use the ReLU function as the activation.

The network is trained using the RMSProp optimizer and a sum of the mse loss and regularization. For the HEQL architecture, the regularization is simply the L_0 regularization, whereas the SEQL has an additional regularization across time steps to induce smooth functions as described in Section III-A

For both learning rate and regularization weight schedules, we use a one cycle policy, as shown in Fig. 7. We start off with a small learning rate and regularization to ensure the EQL network settles into a stable configuration containing many different terms such that the network weights do not explode. The learning rate is ramped up to allow the EQL network escape local minima in search of global minima, and the regularization is likewise increased to pare down the number of terms. Finally, we expect the EQL network to have learned the correct equation structure partway through training, and so we decrease the learning rate and regularization to fine-tune the weights and optimize primarily for mse.

To extract the learned equation from the trained EQL network, we can simply multiply the weights by the primitive functions using symbolic mathematics. We implement this using SymPy, which can automatically simplify the expression [35]. In addition, we use a threshholding procedure in the final expression where we drop terms where the coefficient is smaller than a threshold, which we set to 0.01.

All the experiments were run on an NVIDIA GeForce RTX 2080 Ti graphics card. On average, the SEQL took 1490 s while the HEQL took 413 s to perform 34 800 mini-batch gradient steps on the analytic tasks for our settings of the architecture and training details. As mentioned in Section V, the HEQL is able to scale more efficiently than the SEQL to

TABLE VII
LEARNED EQUATIONS FOR ADDITIONAL ANALYTIC EXPRESSIONS

Benchmark	t	True	Learned equations SEQL	Learned equations HEQL
$f_3 = t \cdot x$	-2.619	$-2.62x$	$-2.62x$	$-2.62x - 0.02$
	-1.095	$-1.10x$	$-1.10x$	$-1.10x$
	0.381	$0.38x$	$0.38x$	$0.38x$
	1.905	$1.90x$	$1.91x$	$1.91x$
$f_4 = t \cdot x^2 + 3\sin(t) \cdot x$	-2.619	$-2.62x^2 - 1.50x$	$-2.62x^2 - 1.51x - 0.08$	$-2.62x^2 - 1.50x + 0.01$
	-1.095	$-1.10x^2 - 2.67x$	$-1.09x^2 - 2.68x - 0.08$	$-1.10x^2 - 2.66x + 0.02$
	0.381	$0.38x^2 + 1.12x$	$0.38x^2 + 1.12x + 0.01$	$0.38x^2 + 1.12x$
	1.905	$1.90x^2 + 2.83x$	$1.90x^2 + 2.85x + 0.06$	$1.90x^2 + 2.83x$
$f_5 = \sin\left(\frac{5+t}{2} \cdot x\right)$	-2.619	$\sin(1.19x)$	$\sin(1.19x)$	$\sin(1.19x)$
	-1.095	$\sin(1.95x)$	$\sin(1.95x)$	$\sin(1.95x)$
	0.381	$\sin(2.69x)$	$\sin(2.69x)$	$\sin(2.69x)$
	1.905	$\sin(3.45x)$	$\sin(3.45x)$	$\sin(3.45x)$

TABLE VIII
RESULTS FOR ANALYTIC EXPRESSION BENCHMARKS

Benchmark	Test MSE of the best trial SEQL	Test MSE of the best trial HEQL	Mean (Standard Deviation) Test MSE over all trials SEQL	Mean (Standard Deviation) Test MSE over all trials HEQL
f_1	$\mathbf{6.97 \times 10^{-6}}$	2.04×10^{-5}	$\mathbf{1.84 \times 10^{-5}}$ $(\mathbf{1.97 \times 10^{-5}})$	2.24×10^{-5} (2.53×10^{-6})
f_2	8.98×10^{-7}	$\mathbf{1.22 \times 10^{-7}}$	1.32×10^{-4} (4.14×10^{-4})	$\mathbf{4.82 \times 10^{-8}}$ $(\mathbf{1.26 \times 10^{-8}})$
f_3	$\mathbf{5.28 \times 10^{-15}}$	4.43×10^{-6}	$\mathbf{5.72 \times 10^{-15}}$ $(\mathbf{3.83 \times 10^{-16}})$	6.60×10^{-6} $(1.45 \times 10^{-6}$
f_4	$\mathbf{9.47 \times 10^{-6}}$	3.17×10^{-5}	$\mathbf{8.32 \times 10^{-5}}$ $(\mathbf{4.74 \times 10^{-4}})$	1.04×10^{-3} (6.04×10^{-3})
f_5	1.21×10^{-7}	$\mathbf{3.63 \times 10^{-8}}$	2.53×10^{-5} (1.51×10^{-4})	$\mathbf{4.87 \times 10^{-8}}$ $(\mathbf{2.29 \times 10^{-8}})$

larger number of time steps, and thus, larger datasets, since the SEQL scales linearly with the number of time steps.

APPENDIX B
ADDITIONAL RESULTS

A. Analytic Expression

For all the tests, 512 training data points with $x \in [-3, 3]$ are randomly sampled for each of 128 fixed, equally spaced, values of $t \in [-3, 3]$ for a total of $512 \times 128 = 65\,536$ training examples. To test generalization, the parametric EQL architectures are evaluated on 256 test data points with $x \in [-5, 5]$ across the same 128 fixed values of t.

Due to sensitivity of the parametric EQL architectures to the random initialization of network weights, 40 trials were run for each function. In practice, the networks only need to learn the correct equation once over a reasonable number of trials, since it is possible to construct a validation method that selects the best equation from a set of learned equations. For all the results in this article, we simply select the trial with the lowest generalization error. Other considerations that can be integrated in the validation process are equation simplicity and prior beliefs about the equation form, for example.

Additional analytical expression benchmarks and the discovered equations for f_3, f_4, and f_5 by the SEQL and HEQL architectures are listed in Table VII. Both the SEQL and HEQL match the true equations very closely. We also list various quantitative metrics for these benchmarks in Table VIII, including the mse on the training and test datasets, as well as the mean and standard deviation of the test mse over all the trials. The aggregate metrics over all the trials tend to be similar in magnitude to the metric of the best trial for many of

the benchmarks, which signifies that the model has learned the correct equation is a large majority of the trials. When a model fails to learn the correct equation, the mse on the test dataset tends to be several orders of magnitude larger than that of the best trial, which would skew the mean and standard deviation of the mse. Interestingly, there is no clear trend on whether the SEQL or the HEQL performs better.

For simple benchmarks such as $f_3 = tx$, both the SEQL and HEQL architectures are able to find the correct equation structure nearly 100% of the time, even if the accuracy of the varying coefficients may vary slightly. However, in other cases such as f_4, the HEQL will sometimes learn the equation

$$\hat{f}_{4,\text{HEQL}} = a(t)x^2 + b(t)x + c(t)\sin(d(t)x + e(t))$$

where $d(t)$ is small. This is likely because the architecture is using the approximately linear region of the low-frequency sinusoid, and adding it to the $b(t)x$ term. We also note that the SEQL is able to find the correct equation more often. Another interesting failure mode is in the case of the sinusoid functions (i.e., f_2 and f_5) where the HEQL will sometimes learn the equation

$$\hat{f}_{5,\text{HEQL}} = a(t)\sin(b(t)x) + c(t)\sin(d(t)x)$$

where $b(t) \approx d(t)$ and $a(t) + c(t) \approx 1$. The symbolic manipulation is unable to combine the two terms, but one can see upon inspection that the HEQL has learned the correct form of the varying parameters.

B. Partial Differential Equations

For the advection–diffusion equation, data were sampled from 256 different points in the x-domain and 512 different points in the t-domain, for a total of $256 \times 512 = 131\,072$

examples. The equation is solved numerically using a spectral method on the domain $x \in [-5, 5]$ and $t \in [0, 5]$ with $f(x) = -1.5 + \cos(2\pi x/5)$ and $\epsilon = 0.1$ using code from [17].

For Burgers' equation, data were sampled from 512 different points in the x-domain and 256 different points in the t-domain, for a total of $512 \times 256 = 131\,072$ examples. The equation was solved numerically using a spectral method on the domain $x \in [-8, 8]$ and $t \in [0, 10]$ using code from [17]. Similar to the analytic expression experiments, 80 trials were run for each equation, and the trial with the lowest training error was selected.

ACKNOWLEDGMENT

REFERENCES

[1] D. J. Griffiths, *Introduction to Electrodynamics*, 4th ed. Boston, MA, USA: Pearson, 2013. [Online]. Available: https://cds.cern.ch/record/1492149

[2] M. Schmidt and H. Lipson, "Distilling free-form natural laws from experimental data," *Science*, vol. 324, no. 5923, pp. 81–85, Apr. 2009. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/19342586

[3] J. Koza, "Genetic programming as a means for programming computers by natural selection," *Statist. Comput.*, vol. 4, no. 2, pp. 87–112, Jun. 1994, doi: 10.1007/BF00175355.

[4] S.-M. Udrescu and M. Tegmark, "AI Feynman: A physics-inspired method for symbolic regression," *Sci. Adv.*, vol. 6, no. 16, Apr. 2020, Art. no. eaay2631.

[5] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proc. Nat. Acad. Sci. USA*, vol. 113, no. 15, pp. 3932–3937, Apr. 2016.

[6] K. Champion, B. Lusch, J. N. Kutz, and S. L. Brunton, "Data-driven discovery of coordinates and governing equations," *Proc. Nat. Acad. Sci. USA*, vol. 116, no. 45, pp. 22445–22451, Nov. 2019.

[7] Z. Long, Y. Lu, and B. Dong, "PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network," *J. Comput. Phys.*, vol. 399, Dec. 2019, Art. no. 108925.

[8] P. Y. Lu, J. Ariño, and M. Soljačić, "Discovering sparse interpretable dynamics from partial observations," 2021, *arXiv:2107.10879*.

[9] M. Cranmer et al., "Discovering symbolic models from deep learning with inductive biases," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 17429–17442.

[10] G. Martius and C. H. Lampert, "Extrapolation and learning equations," 2016, *arXiv:1610.02995*.

[11] S. Sahoo, C. Lampert, and G. Martius, "Learning equations for extrapolation and control," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4442–4450.

[12] S. Kim et al., "Integration of neural network-based symbolic regression in deep learning for scientific discovery," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 9, pp. 4166–4177, Sep. 2021.

[13] A. Costa et al., "Fast neural models for symbolic regression at scale," 2020, *arXiv:2007.10784*.

[14] D. J. Griffiths and D. F. Schroeter, *Introduction to Quantum Mechanics*, 3rd ed. New York, NY, USA: Cambridge Univ. Press, 2018.

[15] Y. Zhang, D. Jiang, and J. Wang, "A recurrent neural network for solving Sylvester equation with time-varying coefficients," *IEEE Trans. Neural Netw.*, vol. 13, no. 5, pp. 1053–1063, Sep. 2002.

[16] Z. Yan and V. Konotop, "Exact solutions to three-dimensional generalized nonlinear Schrodinger equations with varying potential and nonlinearities," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdisc. Top.*, vol. 80, no. 3, 2009, Art. no. 036607.

[17] S. Rudy, A. Alla, S. L. Brunton, and J. N. Kutz, "Data-driven identification of parametric partial differential equations," *SIAM J. Appl. Dyn. Syst.*, vol. 18, no. 2, pp. 643–660, Jan. 2019, doi: 10.1137/18M1191944.

[18] H. Xu, D. Zhang, and J. Zeng, "Deep-learning of parametric partial differential equations from sparse and noisy data," *Phys. Fluids*, vol. 33, no. 3, Mar. 2021, Art. no. 037132.

[19] Y. Luo, Q. Liu, Y. Chen, W. Hu, and J. Zhu, "KO-PDE: Kernel optimized discovery of partial differential equations with varying coefficients," 2021, *arXiv:2106.01078*.

[20] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through L_0 regularization," 2017, *arXiv:1712.01312*.

[21] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," 2016, *arXiv:1611.00712*.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[23] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 2261–2269.

[24] D. Ha, A. Dai, and Q. V. Le, "HyperNetworks," 2016, *arXiv:1609.09106*.

[25] M. Andrychowicz et al., "Learning to learn by gradient descent by gradient descent," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, 2016, pp. 3981–3989.

[26] T. Munkhdalai and H. Yu, "Meta networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 2554–2563.

[27] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *Proc. Int. Conf. Learn. Represent.*, 2017. [Online]. Available: https://openreview.net/forum?id=rJY0-Kcll

[28] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-learning in neural networks: A survey," 2020, *arXiv:2004.05439*.

[29] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 6571–6583.

[30] F. de Avila Belbute-Peres, Y.-f. Chen, and F. Sha, "HyperPINN: Learning parameterized differential equations with physics-informed hypernetworks," in *The Symbiosis of Deep Learning and Differential Equations*, 2021.

[31] P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak, and I. Sutskever, "Deep double descent: Where bigger models and more data hurt," *J. Stat. Mech., Theory Exp.*, vol. 2021, no. 12, Dec. 2021, Art. no. 124003.

[32] J. Liu, G. Jiang, Y. Bai, T. Chen, and H. Wang, "Understanding why neural networks generalize well through GSNR of parameters," 2020, *arXiv:2001.07384*.

[33] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A ConvNet for the 2020s," 2022, *arXiv:2201.03545*.

[34] D. Jakubovitz, R. Giryes, and M. R. Rodrigues, "Generalization error in deep learning," in *Compressed Sensing and Its Applications*. Cham, Switzerland: Springer, 2019, pp. 153–193.

[35] A. Meurer et al., "SymPy: Symbolic computing in Python," *PeerJ Comput. Sci.*, vol. 3, p. e103, Jan. 2017, doi: 10.7717/peerj-cs.103.

[36] J. Pathak, M. Mustafa, K. Kashinath, E. Motheau, T. Kurth, and M. Day, "Using machine learning to augment coarse-grid computational fluid dynamics simulations," 2020, *arXiv:2010.00072*.

[37] D. Kochkov, J. A. Smith, A. Alieva, Q. Wang, M. P. Brenner, and S. Hoyer, "Machine learning–accelerated computational fluid dynamics," *Proc. Nat. Acad. Sci. USA*, vol. 118, no. 21, 2021, Art. no. e2101784118.

[38] A. S. Benjamin, D. Rolnick, and K. Kording, "Measuring and regularizing networks in function space," 2018, *arXiv:1805.08289*.

[39] P. Pan, S. Swaroop, A. Immer, R. Eschenhagen, R. Turner, and M. E. E. Khan, "Continual deep learning by functional regularisation of memorable past," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 4453–4464.

[40] S. Sun, G. Zhang, J. Shi, and R. Grosse, "Functional variational Bayesian neural networks," 2019, *arXiv:1903.05779*.

[41] T. G. Rudner, Z. Chen, and Y. Gal, "Rethinking function-space variational inference in Bayesian neural networks," in *Proc. 3rd Symp. Adv. Approx. Bayesian Inference*, 2020.

[42] L. Ziyin, T. Hartwig, and M. Ueda, "Neural networks fail to learn periodic functions and how to fix it," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 1583–1594.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

ZHANG et al.: DEEP LEARNING AND SYMBOLIC REGRESSION FOR DISCOVERING PARAMETRIC EQUATIONS 13

[43] A. Molina, P. Schramowski, and K. Kersting, "Padé activation units: End-to-end learning of flexible activation functions in deep networks," 2019, *arXiv:1907.06732*.

[44] A. Trask, F. Hill, S. Reed, J. Rae, C. Dyer, and P. Blunsom, "Neural arithmetic logic units," in *Proc. Adv. neural Inf. Process. Syst.*, vol. 31, 2018, pp. 8035–8044.

[45] D. Schlör, M. Ring, and A. Hotho, "INALU: Improved neural arithmetic logic unit," *Frontiers Artif. Intell.*, vol. 3, p. 71, Sep. 2020.

Michael Zhang received the B.S. degree in computer science and mathematics (minor in physics) from the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, in 2023, where he is currently pursuing the Master of Engineering degree.

He is currently a Teaching Assistant with the Department of Electrical Engineering and Computer Science (EECS), MIT. His research interests include machine learning and physics.

Samuel Kim received the A.B. degree in physics from Harvard University, Cambridge, MA, USA, in 2015, and the Ph.D. degree in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, in 2023.

In 2021, he co-founded Kyber Photonics, Cambridge, a spinoff from his research on silicon photonics devices for optical beam steering. He is currently the Research Scientist of the Applied Physics Laboratory, Johns Hopkins University, Laurel, MD, USA. His research interests include photonics, computational electromagnetics, optimization, deep learning, and scientific machine learning.

Dr. Kim was a recipient of the National Defense Science and Engineering Graduate (NDSEG) Fellowship in 2019.

Peter Y. Lu received the A.B. degree in physics and mathematics from Harvard University, Cambridge, MA, USA, in 2016, and the Ph.D. degree in physics from the Massachusetts Institute of Technology (MIT), Cambridge, in 2022.

He was the National Defense Science and Engineering Graduate (NDSEG) Fellow of MIT. He is currently an Eric and Wendy Schmidt AI in Science Post-Doctoral Fellow with The University of Chicago, Chicago, IL, USA, working at the intersection of physics and machine learning. He aims to develop new computational methods for modeling and understanding physical systems with an emphasis on incorporating physics-informed priors and identifying relevant and interpretable latent representations. His research interests include physics-informed machine learning and interpretable representational learning with applications in nonlinear dynamics, condensed matter physics, photonics, fluid dynamics, and biophysics.

Marin Soljačić was the Founder of WiTricity Corporation, Watertown, MA, USA, in 2007, LuxLabs, Boston, MA, USA, in 2017, and Lightelligence, Boston, in 2017. He is currently a Professor of physics with the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA. His main research interests include artificial intelligence and electromagnetic phenomena, focusing on nanophotonics, nonlinear optics, and wireless power transfer.

Prof. Soljačić has been an International Member of the Croatian Academy of Engineering since 2009. He was a recipient of the Adolph Lomb Medal from the Optical Society of America in 2005 and the TR35 Award of the Technology Review Magazine in 2006. He has received the MacArthur Fellowship "Genius" Grant in 2008; the Blavatnik National Award as well as Invented Here! (Boston Patent Law Association) in 2014; and "The Order of the Croatian Daystar, with the image of Ruđer Bošković," the Croatian President's Top Medal for science, in 2017. In 2011, he became the Young Global Leader (YGL) of the World Economic Forum. In 2017, the Croatian President also awarded him with "The Order of the Croatian Interlace" medal. He was a Highly Cited Researcher according to WoS from 2019 to 2022. In 2023, he has received the Max Born Award from Optica.