Integration of Neural Network-Based Symbolic Regression in Deep Learning for Scientific Discovery

Samuel Kim, Peter Y. Lu, Srijon Mukherjee, *Student Member, IEEE*, Michael Gilbert, Li Jing,
Vladimir Čeperić, *Member, IEEE*, and Marin Soljačić

***Abstract*—Symbolic regression is a powerful technique to discover analytic equations that describe data, which can lead to explainable models and the ability to predict unseen data. In contrast, neural networks have achieved amazing levels of accuracy on image recognition and natural language processing tasks, but they are often seen as black-box models that are difficult to interpret and typically extrapolate poorly. In this article, we use a neural network-based architecture for symbolic regression called the equation learner (EQL) network and integrate it with other deep learning architectures such that the whole system can be trained end-to-end through backpropagation. To demonstrate the power of such systems, we study their performance on several substantially different tasks. First, we show that the neural network can perform symbolic regression and learn the form of several functions. Next, we present an MNIST arithmetic task where a convolutional network extracts the digits. Finally, we demonstrate the prediction of dynamical systems where an unknown parameter is extracted through an encoder. We find that the EQL-based architecture can extrapolate quite well outside of the training data set compared with a standard neural network-based architecture, paving the way for deep learning to be applied in scientific exploration and discovery.**

***Index Terms*—Discovery, kinematics, neural network, ordinary differential equation (ODE), simple harmonic oscillator (SHO), symbolic regression.**

I. INTRODUCTION

MANY complex phenomena in science and engineering can be reduced to general models that can be described in terms of relatively simple mathematical equations. For example, classical electrodynamics can be described by Maxwell's equations and nonrelativistic quantum mechanics can be described by the Schrödinger equation. These models elucidate the underlying dynamics of a particular system and can provide general predictions over a very wide range of conditions. On the other hand, modern machine learning techniques have become increasingly powerful for many tasks, including image recognition and natural language processing, but the neural network-based architectures in these state-of-the-art techniques are black-box models that often make them difficult for use in scientific exploration. In order for machine learning to be widely applied to science, there needs to be interpretable models that can extract meaningful information from complex data sets and extrapolate outside of the training data set.

Symbolic regression is a type of regression analysis that searches the space of mathematical expressions to find the best model that fits the data and can thus fit a much wider range of data sets than other models such as linear regression. Assuming that the resulting mathematical expression correctly describes the underlying model for the data, it is easier to interpret and can extrapolate better than black-box models such as neural networks. Symbolic regression is typically carried out using techniques such as genetic programming, in which the structure of the mathematical expression is found using evolutionary algorithms to best fit the data [1]. This approach has been used to extract the underlying laws of physical systems from experimental data [2]. However, due to the combinatorial nature of the problem, genetic programming does not scale well to large systems and can be prone to overfitting.

Alternative approaches to finding the underlying laws of data have been explored. For example, sparsity has been combined with regression techniques and numerically evaluated derivatives to find partial differential equations (PDEs) that describe dynamical systems [3]–[5].

There has also been significant work on designing neural network architectures that are either more interpretable or contain inductive biases that make them more suitable for scientific exploration. Neural networks with unique activation functions that correspond to functions common in science and engineering have been used for finding mathematical expressions that describe data sets [6], [7]. A deep learning architecture called the PDE-Net has been proposed to predict the

---

Manuscript received December 10, 2019; revised April 1, 2020 and July 5, 2020; accepted August 8, 2020. Date of publication August 28, 2020; date of current version September 1, 2021. This work was supported in part by the Army Research Office under Cooperative Agreement W911NF-18-2-0048, in part by the Department of Defense through the National Defense Science and Engineering Graduate Fellowship (NDSEG) Program, in part by the MIT–SenseTime Alliance on Artificial Intelligence, in part by the Defense Advanced Research Projects Agency (DARPA) under Agreement HR00111890042, and in part by the United States Air Force Research Laboratory under Cooperative Agreement FA8750-19-2-1000. *(Corresponding author: Samuel Kim.)*

Samuel Kim and Michael Gilbert are with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: samkim@mit.edu).

Peter Y. Lu, Srijon Mukherjee, and Marin Soljačić are with the Department of Physics, Massachusetts Institute of Technology, Cambridge, MA 02139 USA.

Li Jing was with the Department of Physics, Massachusetts Institute of Technology, Cambridge, MA 02139 USA. He is now with the Facebook AI Research, New York, NY USA.

Vladimir Čeperić is with the Faculty of Electrical Engineering and Computing, University of Zagreb, 10000 Zagreb, Croatia.

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2020.3017010

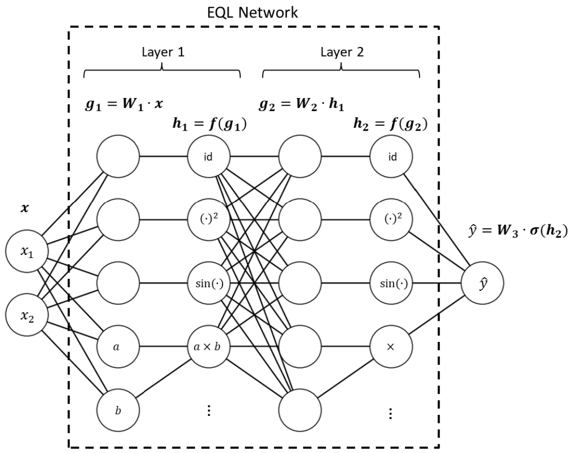


Fig. 1. Example of the EQL network for symbolic regression using a neural network. Here, we show only four activation functions (identity or "id," square, sine, and multiplication) and two hidden layers for visual simplicity, but the network can include more functions or more hidden layers to fit a broader class of functions.

dynamics of spatiotemporal systems and produce interpretable differential operators through constrained convolutional filters [8], [9]. Trask *et al.* [10] proposed a neural network module called the neural arithmetic logic unit (NALU) that introduces inductive biases toward arithmetic operations so that the architecture can extrapolate well on specific tasks. Neural network-based architectures have also been used to extract relevant and interpretable parameters from dynamical systems and use these parameters to predict the propagation of a similar system [11], [12]. In addition, Chari *et al.* [13] used symbolic regression as a separate module to discover kinematic equations using parameters extracted from videos regarding various types of motion.

Here, we present a neural network architecture for symbolic regression that is integrated with other deep learning architectures so that it can take advantage of powerful deep learning techniques while still producing interpretable and generalizable results. Because this symbolic regression method can be trained through backpropagation, the entire system can be trained end-to-end without requiring multiple steps.

Source code is made publicly available.[1]

II. EQL ARCHITECTURE

The symbolic regression neural network we use is similar to the equation learner (EQL) network proposed in [6] and [7]. As shown in Fig. 1, the EQL network is based on a fully connected neural network where the ith layer of the neural network is described by

$$\begin{aligned}\mathbf{g}_i &= \mathbf{W}_i \mathbf{h}_{i-1}\\ \mathbf{h}_i &= f(\mathbf{g}_i)\end{aligned}$$

where $\mathbf{W}_i$ is the weight matrix of the ith layer and $\mathbf{h}_0 = \mathbf{x}$ is the input data. The final layer does not have an activation function, so for a network with L hidden layers, the output of the network is described by

$$y = \mathbf{h}_{L+1} = \mathbf{W}_{L+1}\mathbf{h}_L.$$

The activation function $f(\mathbf{g})$, rather than being the usual choices in neural networks such as ReLU or tanh, may consist of a separate function for each component of $\mathbf{g}$ (such as sine or the square function) and may include functions that take two or more arguments while producing one output (such as the multiplication function)

$$f(\mathbf{g}) = \begin{bmatrix} f_1(g_1) \\ f_2(g_2) \\ \vdots \\ f_{n_h}(g_{n_g-1}, g_{n_g}) \end{bmatrix}. \tag{1}$$

Note that an additive bias term can be absorbed into $f(\mathbf{g})$ for convenience. These activation functions in (1) are analogous to the primitive functions in symbolic regression. Allowing functions to take more than one argument allows for multiplicative operations inside the network.

While the schematic in Fig. 1 only shows four activation functions in each hidden layer for visual simplicity, $f(\mathbf{g})$ in 1 can include other functions, including $\exp(g)$ and $\text{sigmoid}(g) = (1/(1+e^{-g}))$. In addition, we allow for activation functions to be duplicated within each layer (i.e., multiple components in g can use the same activation function). This reduces the system's sensitivity to random initializations and creates a smoother optimization landscape so that the network does not get stuck in local minima as easily. This also allows the EQL network to fit a broad range of functions. More details can be found in Appendix A.

By stacking multiple layers (i.e. $L \geq 2$), the EQL architecture can fit complex combinations and compositions of a variety of primitive functions. Note that L is analogous to the maximum tree depth in genetic programming approaches and sets the upper limit on the complexity of the resulting expression. While this model is not as general as conventional symbolic regression, it is powerful enough to represent most functions that are typically seen in science and engineering. More importantly, because the EQL network can be trained by backpropagation, it can be integrated with other neural network-based models for end-to-end training.

A. Sparsity

A key ingredient of making the results of symbolic regression interpretable is enforcing sparsity regularization such that the system finds the simplest possible equation that fits the data. The goal of sparsity is to set as many weight parameters to 0 as possible such that those parameters are inactive and can be removed from the final expression. Enforcing sparsity in neural networks is an active field of research as modern deep learning architectures using millions of parameters start to become computationally prohibitive [14]–[16]. Sahoo *et al.* [17] evaluated several recent developments in neural network sparsity techniques.

A straightforward and popular way of enforcing sparsity is adding to the loss function a regularization term that is a

---

[1] https://github.com/samuelkim314/DeepSymReg

function of the neural network weight matrices

$$L_q = \sum_{i=0}^{L+1} L_q(\mathbf{W}_i) \tag{2}$$

where i indexes the layer. L_q acts elementwise on the matrix as follows:

$$L_q(\mathbf{W}) = \sum_{j,k} L_q\left(w_{j,k}\right) = \sum_{j,k} \left|w_{j,k}\right|^q$$

where j, k indexes the elements in the weight matrix $\mathbf{W}$.

Setting $q = 0$ in (2) results in L_0 regularization, which penalizes weights for being nonzero regardless of the magnitude of the weights and thus drives the solution toward sparsity. However, L_0 regularization is equivalent to a combinatorics problem that is NP-hard and is not compatible with gradient descent methods commonly used for optimizing neural networks [18]. Recent works have explored training sparse neural networks with a relaxed version of L_0 regularization through stochastic gate variables, allowing this regularization to be compatible with backpropagation [14], [19].

A much more popular and well-known sparsity technique is L_1 regularization, which was used in the original EQL network [6]. Although it does not push solutions toward sparsity as strongly as L_0 regularization, L_1 regularization is a convex optimization problem that can be solved using a wide range of optimization techniques, including gradient descent to drive the weights toward 0. However, since L_1 also penalizes the magnitude of the weights, $L_{0.5}$ has been proposed to enforce sparsity more strongly without penalizing the magnitude of the weights as much as L_1 [20], [21]. $L_{0.5}$ regularization is still compatible with gradient descent (although it is no longer convex) and has been applied to neural networks [22], [23]. Experimental studies suggest that $L_{0.5}$ regularization performs no worse than other L_q regularizers for $0 < q < 0.5$, implying that $L_{0.5}$ is optimal for enforcing sparsity [21]. Our experiments with $L_{0.3}$ and $L_{0.7}$ regularizers show no significant overall improvement compared with the $L_{0.5}$ regularizer, in agreement with this study. In addition, our experiments show that $L_{0.5}$ drives the solution toward sparsity more strongly than L_1, producing simpler expressions.

However, $L_{0.5}$ regularization has a singularity in the gradient as the weights go to 0, which can make training difficult for gradient descent-based methods. To avoid this, we use a smoothed version of $L_{0.5}$ proposed in [23], which we label as $L^*_{0.5}$. The $L^*_{0.5}$ regularizer uses a piecewise function to smooth out the function at small magnitudes

$$L^*_{0.5}(w) = \begin{cases} |w|^{1/2} & |w| \geq a \\ \left(-\dfrac{w^4}{8a^3} + \dfrac{3w^2}{4a} + \dfrac{3a}{8}\right)^{1/2} & |w| < a \end{cases} \tag{3}$$

where $a \in \mathbb{R}^+$ is the transition point between the standard $L_{0.5}$ function and the smoothed function.

A plot of the $L_{0.5}$ and $L^*_{0.5}$ regularization is shown in Fig. 2. The smoothed $L^*_{0.5}$ regularization avoids the extreme gradient values to improve training convergence. In our experiments, we set $a = 0.01$. When the EQL network is integrated with other deep learning architectures, the regularization is only applied to the weights of the EQL network.

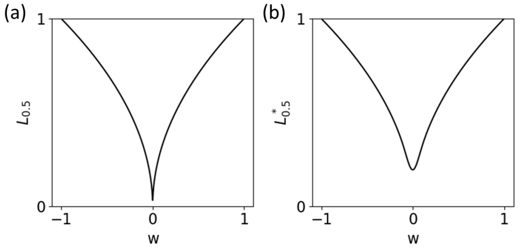


Fig. 2. (a) $L_{0.5}$ and (b) $L^*_{0.5}$ regularization, as described in (2) and (3), respectively. The threshold for the plot of (3) is set to $a = 0.1$ for easy visualization, but we use a threshold of $a = 0.01$ in our experiments.

We have also implemented an EQL network with the relaxed L_0 regularization proposed in [14], the details of which can be found in Appendix B.

III. EXPERIMENTS

A. Symbolic Regression on Analytic Expressions

To validate the EQL network's ability to perform symbolic regression, we first test the EQL network on data generated by analytic expressions, such as $\exp\left(-x^2\right)$ or $x_1^2 + \sin(2\pi x_2)$. The data are generated on the domain $x_i \in [-1, 1]$. Because of the network's sensitivity to random initialization of the weights, we run 20 trials for each function. We then count the number of times that the network has converged to the correct answer ignoring small terms and slight variations in the coefficients from the true value. In addition, equivalent answers (such as $\sin(4\pi + x)$ instead of $\sin(2\pi + x)$) are counted as correct. These results are shown in Appendix A.

The network only needs to be able to find the correct answer at least once over a reasonable number of trials, as one can construct a system that picks out the desired equation from the different trials using a combination of equation simplicity and extrapolation ability. We find that when we measure the extrapolation ability by measuring the equation error evaluated on the domain $x_i \in [-2, 2]$, this extrapolation error of the correct equation tends to be orders of magnitude lower than that of other equations that the network may find, making it simple to pick out the correct answer.

The network is still able to find the correct answer when 10% noise is added to the data. We also test an EQL network with three hidden layers, which still finds the correct expression and is able to find even more complicated expressions such as $(x_1 + x_2 x_3)^3$.

B. MNIST Arithmetic

In the first experiment, we demonstrate the ability to combine symbolic regression and image recognition through an arithmetic task on MNIST digits. MNIST, a popular data set for image recognition, can be notated as $\mathcal{D} = \{\chi, \psi\}$, where χ are 28×28 grayscale images of handwritten digits and $\psi \in \{0, 1, \ldots, 9\}$ is the integer-value label. Here, we wish to learn a simple arithmetic function, $y = \psi_1 + \psi_2$, with the corresponding images $\{\chi_1, \chi_2\}$ as inputs, and train the

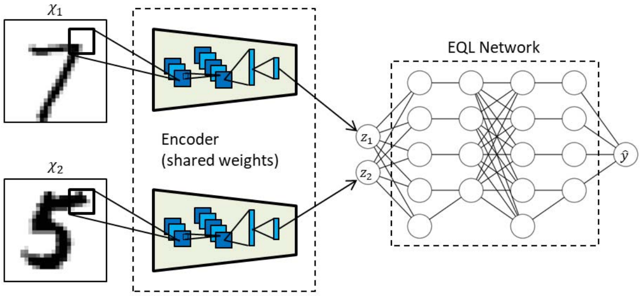


Fig. 3. Schematic of the MNIST addition architecture. An encoder consisting of convolutional layers and fully connected layers operates on each MNIST image and extracts a single-dimensional latent variable. The two encoders share the same weights. The two latent variables are then fed into the EQL network. The entire system is fed end-to-end and without pretraining.

system end-to-end such that the system learns how to "add" two images together.

The deep learning architecture is shown in Fig. 3. The input consists of two MNIST digits, $x = \{\chi_1, \chi_2\}$. During training, χ_i is randomly drawn from the MNIST training data set. Each of $\{\chi_1, \chi_2\}$ is fed separately into an encoder to produce single-dimensional latent variables $\{z_1, z_2\}$ that are not constrained and can take on any real value, $z_i \in \mathbb{R}$. Alternatively, one can think of the architecture as having a separate encoder for each digit, where the two encoders share the same weights, as shown in Fig. 3. The encoder consists of two convolutional layers with max-pooling layers, followed by two fully connected layers and a batch normalization layer at the output. More details on the encoder can be found in Appendix C. The latent variables $\{z_1, z_2\}$ then feed into the EQL network. The EQL network has a single scalar output $\hat{y}$, which is trained on the true label $y = \psi_1 + \psi_2$.

The entire network is trained end-to-end using a mean-squared error (MSE) loss between the predicted label $\hat{y}$ and the true label y. In other words, the encoder is not trained separately from the EQL network. Note that the encoder closely resembles a simple convolutional neural network used for classifying MNIST digits except that it outputs a scalar value instead of logits that encode the digit. While there is no constraint on the properties of $z_{1,2}$, we expect it to map one-to-one to the true label $\psi_{1,2}$.

C. Dynamical System Analysis

In the next set of experiments, we apply the EQL network to analyzing physical time-varying systems. A potentially powerful application of deep learning in science exploration and discovery is discovering parameters in dynamical systems in an unsupervised setting and using these parameters to predict the propagation of similar systems. For example, Zheng *et al.* [11] used multilayer perceptrons to extract relevant properties from a system of bouncing balls (such as the mass of the balls or the spring constant of a force between the balls) and simultaneously predicted the trajectory of a different set of objects. Lu *et al.* [12] accomplished a similar goal but using a dynamics encoder (DE) with convolutional layers and a propagating decoder (PD) with deconvolutional layers to enable analysis and prediction of spatiotemporal systems, such as those governed by PDEs. This DE-PD architecture is designed to analyze spatiotemporal systems that may have an uncontrolled dynamical parameter that varies among different instances of the data set, such as the diffusion constant in the diffusion equation. The parameters encoded in a latent variable are fed into the PD along with a set of initial conditions, in which the PD propagates forward in time based on the extracted physical parameter and learned dynamics.

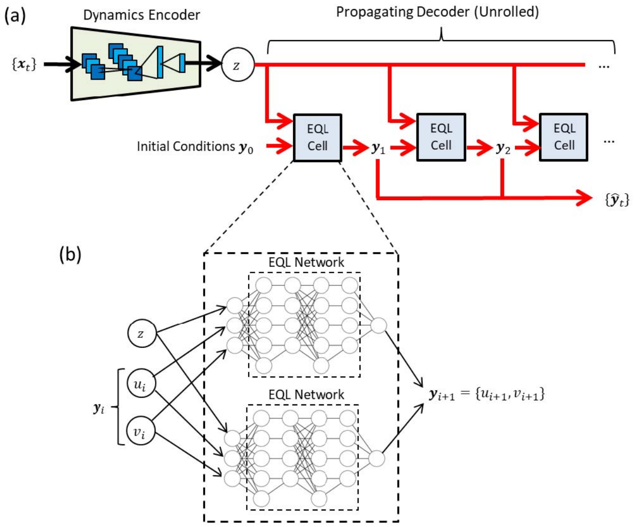


Fig. 4. (a) Architecture to learn the equations that propagate a dynamical system. (b) Each EQL cell in the PD consists of a separate EQL network for each dimension of $\mathbf{y}$ to be predicted. In our case, $\mathbf{y} = \{u, v\}$ where u is the position and v is velocity, so there are two EQL networks in each EQL cell.

Here, we present a deep learning architecture shown in Fig. 4, which is based on the DE-PD architecture. The DE takes in the full input series $\{\mathbf{x}_t\}_{t=0}^{T_x}$ over T_x time steps and outputs a single-dimensional latent variable z. Unlike the original DE-PD architecture presented in [12], the DE here is not a VAE. The DE here consists of several convolutional layers followed by fully connected layers and a batch normalization layer. More details are given in Appendix C. The parameter z and a set of initial conditions $\mathbf{y}_0$ are fed into the PD, which predicts the future time steps $\{\hat{\mathbf{y}}_t\}_{t=1}^{T_y}$ based on the learned dynamics. The PD consists of an "EQL cell" in a recurrent structure such that each step in the recurrent structure predicts a single time step forward. The EQL cell consists of separate EQL networks for each feature, or dimension, in $\hat{\mathbf{y}}_t$.

The full architecture is trained end-to-end using an MSE loss between the predicted dynamics $\{\hat{\mathbf{y}}_t\}_{t=1}^{T_y}$ and the target series $\{\mathbf{y}_t\}_{t=1}^{T_y}$. Similar to the architecture in Section III-B, the DE and PD are not trained separately, and there is no restriction or bias placed on the latent variable z. We explore two different physical systems [kinematics and simple harmonic oscillator (SHO)] as described in the following.

1) Kinematics: Kinematics describes the motion of objects and is used in physics to study how objects move under an applied force. A schematic of a physical scenario described by 1-D kinematics is shown in Fig. 5(a) in which an object on a frictionless surface has a force applied to it where the

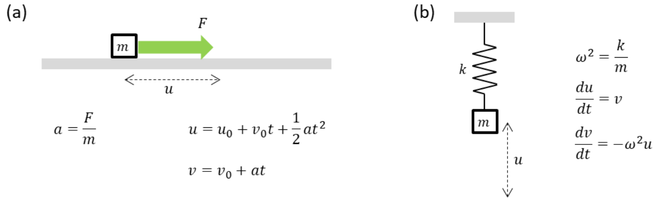


Fig. 5. (a) Kinematics describes the dynamics of an object where a force F is applied to a mass m. (b) SHO describes a mass m on a spring with spring constant k. In both cases, u is the displacement of the mass and v is the velocity.

direction of the force is parallel to the surface. The relevant parameter to describe the object's motion can be reduced to $a = (F/m)$ for a constant force F and object mass m. Given position u_i and velocity v_i at time step i, the object's state at time step $i+1$ is given by

$$\begin{aligned} u_{i+1} &= u_i + v_i \Delta t + \frac{1}{2}\Delta t^2 \\ v_{i+1} &= v_i + a\Delta t \end{aligned} \tag{4}$$

where Δt is the time step.

Acceleration a varies across different time series in the data set. In our simulated data set, we draw initial state and acceleration from uniform distributions

$$u_0, v_0, a \ \sim \mathcal{U}(-1, 1).$$

We set $\Delta t = 1$. The initial parameters u_0 and v_0 are fed into the PD, and z is expected to correlate with a.

2) Simple Harmonic Oscillator: The second physical system we analyze is the SHO, a ubiquitous model in physics that can describe a wide range of physical systems, including springs, pendulums, quantum potentials, and electric circuits. In general, the dynamics of the SHO can be given by the coupled first-order ordinary differential equation (ODE)

$$\begin{aligned} \frac{du}{dt} &= v \\ \frac{dv}{dt} &= -\omega^2 u \end{aligned} \tag{5}$$

where u is the position, v is the velocity, and ω is the resonant frequency of the system. In the case of a spring as shown in Fig. 5(b), $\omega = (k/m)^{1/2}$, where k is the spring constant and m is the mass of the object on the end of the spring.

The SHO system can be numerically solved using a finite-difference approximation for the time derivatives. For example, the Euler method for integrating ODEs gives

$$\begin{aligned} u_{i+1} &= u_i + v_i \Delta t \\ v_{i+1} &= v_i - \omega^2 u_i \Delta t. \end{aligned} \tag{6}$$

In our experiments, we generate data with parameters drawn from uniform distributions

$$\begin{aligned} u_0, v_0 &\sim \mathcal{U}(-1, 1) \\ \omega^2 &\sim \mathcal{U}(0.1, 1). \end{aligned}$$

The state variables u and v are measured at a time step of $\Delta t = 0.1$ to allow the system to find the finite-difference solution. Because of this small time step, we also need to propagate the solution for more time steps to find the right equation (otherwise, the system learns the identity function). To avoid the recurrent architecture predictions exploding toward $\pm\infty$, we start the training by propagating only one time step and add more time steps as the training continues. A similar strategy is used in [9] except that we are not restarting the training.

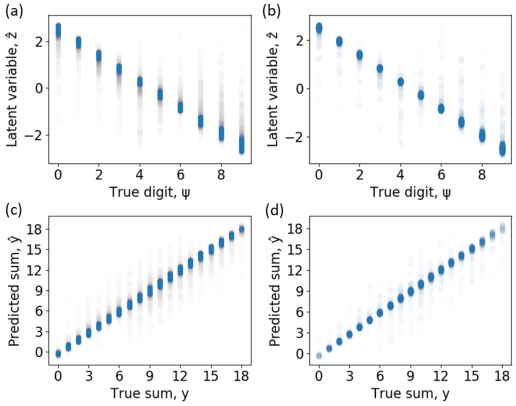


Fig. 6. Ability of the encoder to differentiate between digits as measured by the latent variable z versus the true digit ψ for digits χ drawn from the MNIST (a) training data set and (b) test data set. The correlation coefficients are −0.985 and −0.988, respectively. The ability of the entire architecture to fit the label y as measured by the predicted sum $\hat{y}$ versus the true sum y for digits χ drawn from the MNIST (c) training data set and (d) test data set.

The initial parameters u_0 and v_0 are fed into the PD, and z is expected to correlate with ω^2.

D. Training

The neural network is implemented in TensorFlow [24]. The network is trained using backpropagation with the RMSProp optimizer [25] and the following loss function:

$$\mathcal{L} = \frac{1}{N}\sum (y_i - \hat{y}_i)^2 + \lambda L^*_{0.5}$$

where N is the size of the training data set and λ is a hyperparameter that balances the regularization versus the MSE.

Similar to [6], we introduce a multiphase training schedule. In an optional first phase, we train with a small value of λ, allowing for the parts of the network apart from the EQL to evolve freely and extract the latent parameters during training. In the second phase, λ is increased to a point where it forces the EQL network to become sparse. After this second phase, weights in the EQL network below a certain threshold α are set to 0 and frozen such that they stay 0, equivalent to fixing the L_0 norm. In the final phase of training, the system continues training without $L^*_{0.5}$ regularization (i.e. $\lambda = 0$) and with a reduced maximum learning rate in order to fine-tune the weights.

Specific details for each experiment are listed in Appendix C.

IV. RESULTS

A. MNIST Arithmetic

Fig. 6(b) shows the latent variable z versus the true label ψ for each digit after the entire network has been trained.

TABLE I
MNIST ARITHMETIC EXPECTED AND EXTRACTED EQUATIONS

True	$y = \psi_1 + \psi_2$
Encoder	$\hat{y} = -1.788z_1 - 1.788z_2 + 9.04$
EQL	$\hat{y} = -1.809z_1 - 1.802z_2 + 9$

Note that while the system is trained on digits drawn from the MNIST training data set, we also evaluate the trained network's performance on digits drawn from the MNIST test data set to confirm the encoder's generalizability. We see a strong linear correlation for both data sets, showing that the encoder has successfully learned a linear relation between z and ψ despite not having access to the digit label ψ. Also, note that there is a scaling factor between z and ψ due to the lack of constraint on z. A simple linear regression shows that the relation is

$$\psi = -1.788z + 4.519. \tag{7}$$

The extracted equation from the EQL network for this result is shown in Table I. The "encoder" equation is what we expect based on the encoder result in (7). From these results, we conclude that the EQL network has successfully extracted the additive nature of the function. Fig. 6(c) and (d) shows the predicted sums $\hat{y}$ versus the true sums y. The mean absolute errors of prediction for the model drawing digits from the MNIST training and test data sets are 0.307 and 0.315, respectively.

While the architecture is trained as a regression problem using an MSE loss, we can still report accuracies as if it is a classification task since the labels y are integers. To calculate accuracy, we first round the predicted sum $\hat{y}$ to the nearest integer and then compare it to the label y. The trained system achieves the accuracies of 89.7% and 90.2% for digits drawn from the MNIST training and test data sets, respectively.

To demonstrate the generalization of this architecture to data outside of the training data set, we train the system using a scheme where MNIST digit pairs χ_1, χ_2 are randomly sampled from the MNIST training data set and used as a training data point if they follow the condition $\psi_1 + \psi_2 < 15$. Otherwise, the pair is discarded. In the test phase, MNIST digit pairs χ_1, χ_2 are randomly sampled from the MNIST training data set and kept in the evaluation data set if $\psi_1 + \psi_2 \geq 15$. Otherwise, the pair is discarded. For comparison, we also test the generalization of the encoder by following the abovementioned procedures but drawing MNIST digit pairs χ_1, χ_2 from the MNIST test data set. Generalization results of the network are shown in Table II. In this case, the EQL network has learned the equation $\hat{y} = -1.56z_1 - 1.56z_2 + 8.66$.

First, note the difference between the accuracy evaluated on pairs $y < 15$ and pairs $y \geq 15$. For the architecture with the EQL network, the accuracy drops by a few percentage points. However, for the architecture where the EQL network is replaced by the commonly used fully connected network with ReLU activation functions (which we label as "ReLU"), the accuracy drops to below 1% showing that the results of the EQL are able to generalize reasonably well in a regime where the ReLU cannot generalize at all. Note that this is not a result of the encoder since the system sees all digits 0–9.

TABLE II
MNIST ARITHMETIC GENERALIZATION RESULTS

Source of w_i to form $x = \{w_1, w_2\}$	Network after the encoder	Accuracy [%]	
		$y < 15$	$y \geq 15$
MNIST training dataset	EQL	92	87
	ReLU	93	0.8
MNIST test dataset	EQL	91	83
	ReLU	92	0.6

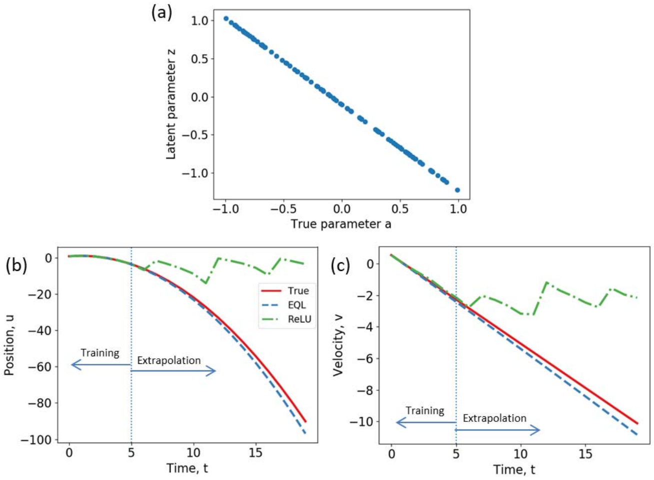


Fig. 7. (a) Latent parameter z of the dynamic encoder architecture after training plotted as a function of the true parameter a. We see a strong linear correlation. (b) and (c) Predicted propagation $\{\hat{\mathbf{y}}_i\} = \{\hat{u}_i, \hat{v}_i\}$ with the EQL cell and a conventional network using ReLU activations. "True" refers to the true propagation $\{\mathbf{y}_i\}$.

Second, the accuracy drops slightly when digits are drawn from the MNIST test data set versus when the digits are drawn from the MNIST training data set, as expected. We did not optimize the hyperparameters of the digit extraction network since the drop in accuracy is small, so the architecture could be optimized further if needed.

Finally, the accuracy drops slightly for pairs $y < 15$ when using the EQL versus the ReLU network. This is unsurprising since the larger size and symmetric activation functions of the ReLU network constrain the network less than the EQL and may make the optimization landscape smoother.

B. Kinematics

Fig. 7(a) shows the extracted latent parameter z plotted as a function of the true parameter a. We see a linear correlation with correlation coefficient close to -1, showing that the DE has extracted the relevant parameter of the system. Again, there is a scaling relation between z and a, which we can extract through linear regression

$$a = -0.884z - 0.091. \tag{8}$$

An example of the equations found by the EQL cell after training is shown in Table III. The "DE" equations are what

TABLE III

KINEMATICS EXPECTED AND EXTRACTED EQUATIONS

True	$u_{i+1} = u_i + v_i + \frac{1}{2}a$ $v_{i+1} = v_i + a$
DE	$\hat{u}_{i+1} = u_i + v_i - 0.442z - 0.045$ $\hat{v}_{i+1} = v_i - 0.884z - 0.091$
EQL	$\hat{u}_{i+1} = 1.002u_i + 1.002v_i - 0.475z$ $\hat{v}_{i+1} = 1.002v_i - 0.918z - 0.102$

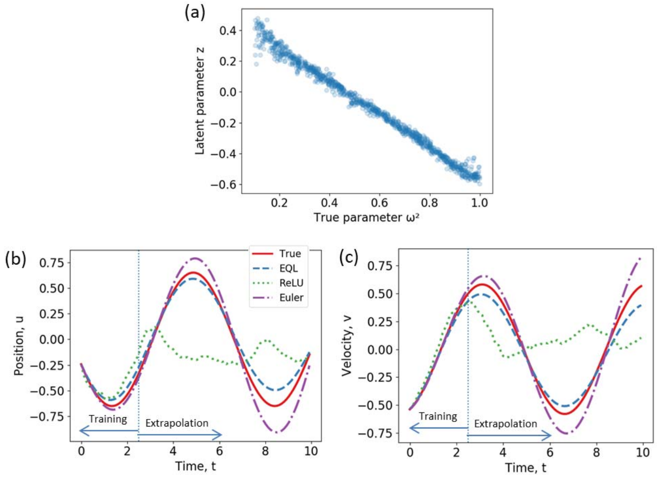


Fig. 8. Results of training on the SHO system. (a) Latent parameter z of the dynamic encoder architecture after training plotted as a function of the true parameter ω^2. We see a good linear correlation. (b) Position u and (c) velocity v as a function of time for various models. "True" refers to the analytical solution. "EQL" refers to the propagation equation discovered by the EQL network. "ReLU" refers to propagation by a conventional neural network that uses ReLU activation functions. "Euler" refers to the finite-difference solution using the Euler method.

we expect based on the true equation and the extract relation in (8). We can see that the EQL equations match closely with what we expect.

The predicted propagation $\{\hat{\mathbf{y}}_i\}$ is plotted in Fig. 7(b) and (c). "True" is the true solution that we want to fit, and "EQL" is the solution propagated by the EQL network. For comparison, we also train a neural network with a similar architecture to the one shown in Fig. 4 but where the EQL cell is replaced by a standard fully-connected neural network with two hidden layers of 50 neurons each and ReLU activation functions (which we label as "ReLU"). While both networks match the true solution very closely in the training regime (left of the dotted line), the ReLU network quickly diverges from the true solution outside of the training regime. The EQL cell is able to match the solution reasonably well for several more time steps, showing how it can extrapolate beyond the training data.

TABLE IV

SHO EXPECTED AND EXTRACTED EQUATIONS

True	$u_{i+1} = u_i + 0.1v_i$ $v_{i+1} = v_i - 0.1\omega^2 u_i$
DE	$\hat{u}_{i+1} = u_i + 0.1v_i$ $\hat{v}_{i+1} = v_i - 0.0464u_i + 0.0927u_i z$
DE, 2nd Order	$\hat{u}_{i+1} = u_i + 0.1v_i$ $\hat{v}_{i+1} = 0.998v_i - 0.0464u_i + 0.0927u_i z$ $+ 0.0046v_i z$
EQL	$\hat{u}_{i+1} = 0.994u_i + 0.0992v_i - 0.0031$ $\hat{v}_{i+1} = 0.995v_i - 0.0492d + 0.084u_i z$ $+ 0.0037v_i z + 0.0133z^2$

C. SHO

The plot of the latent variable z as a function of the true parameter ω^2 after training on the SHO system is shown in Fig. 8(a). Note that there is a strong linear correlation between z and ω^2 as opposed to between z and ω. This reflects the fact that using ω^2 requires fewer operations in the propagating equations than ω, the latter of which would require a squaring function. In addition, the system was able to find that ω^2 is the simplest parameter to describe the system due to the sparsity regularization on the EQL cell. We see a strong linear correlation with a correlation coefficient of -0.995, showing that the DE has successfully extracted the relevant parameter of the SHO system. A linear regression shows that the relation is

$$\omega^2 = -0.927z + 0.464. \tag{9}$$

The equations extracted by the EQL cell are shown in Table IV. The "DE" equation is what we expect based on the DE result in 9. Immediately, we see that the expression for $\hat{u}_{i+1}$ and the first three terms of $\hat{v}_{i+1}$ match closely with the Euler method approximation using the latent variable relation extracted by the DE.

An interesting point is that while we normally use the first-order approximation of the Euler method for integrating ODEs

$$v_{i+1} = v_i + \Delta t \left.\frac{dv}{dt}\right|_{t=i} + \mathcal{O}(\Delta t^2)$$

it is possible to expand the approximation to find higher order terms. If we expand the Euler method to its second-order approximation, we get

$$\begin{aligned} v_{i+1} &= v_i + \Delta t \left.\frac{dv}{dt}\right|_{t=i} + \frac{1}{2}\Delta t^2 \left.\frac{d^2 v}{dt^2}\right|_{t=i} + \mathcal{O}(\Delta t^3) \\ &\approx v_i - \Delta t \omega^2 \ u_i - \frac{1}{2}\Delta t^2 \omega^2 \ v_i. \end{aligned}$$

The expected equation based on the DE result and assuming the second-order expansion is labeled as "DE, 2nd Order" in Table IV. It appears that the EQL network, in this case, has not only found the first-order Euler finite-difference method but it has also added on another small term that corresponds to the second-order term in the Taylor expansion of v_{i+1}. The last term found by the EQL network $0.0133z^2$ is likely from either cross terms inside the network or a lack of convergence to exactly 0 and would likely disappear with another thresholding process.

The solution propagated through time is shown in Fig. 8(b) and (c). As before, "ReLU" is the solution propagated by an architecture where the EQL network is replaced by a conventional neural network with 4 hidden layers of 50 units each and ReLU activation functions. For an additional comparison, we have also calculated the finite-difference solution using Euler's method to integrate the true ODEs which is labeled as "Euler."

Within the training regime, all of the methods fit the true solution reasonably well. However, the conventional neural network with ReLU activation functions completely fails to extrapolate beyond the training regime. The Euler method and the EQL network are both able to extrapolate reasonably well beyond the training regime, although they both start to diverge from the true solution due to the large time step and the accumulated errors from numerical integration. A more accurate method, such as the Runge–Kutta method, almost exactly fits the analytical solution, which is not surprising due to its small error bound. However, it is more complex than the Euler method and would likely require a larger EQL network to find an expression similar to the Runge–Kutta method. Interestingly, the EQL network solution has a smaller error than the Euler solution, demonstrating that the EQL network was able to learn higher order corrections to the first-order Euler method. This could possibly lead to discovery of more efficient integration schemes for differential equations that are difficult to solve through finite-difference methods.

V. CONCLUSION

We have shown how we can integrate symbolic regression with deep learning architectures and train the entire system end-to-end to take advantage of the powerful deep learning training techniques that have been developed in recent years. Namely, we show that we can learn arithmetic on MNIST digits where the system must learn to identify the images in an image recognition task while simultaneously extracting the mathematical expression that relates the digits to the answer. In addition, we show that we can simultaneously extract an unknown parameter from a dynamical system and extract the propagation equations. In the SHO system, the results suggest that we can discover new techniques for integrating ODEs, potentially paving the way for improved integrators, such as integrators for stiff ODEs that may be difficult to solve with numerical methods.

One direction for future work is to study the role of random initializations and make the system less sensitive to random initializations. As seen by the benchmark results of the EQL network in Appendix A, the EQL network is not always able to find the correct mathematical expression. This is because there are a number of local minima in the EQL network that the network can get stuck in, and gradient-based optimization methods are only guaranteed to find local minima rather than global minima. Local minima are not typically a concern for neural networks because the local minima are typically close enough in performance to the global minimum [26]. However, for the EQL network, we often want to find the true global minimum. In this work, we have alleviated this issue by increasing stochasticity through large learning rates and by decreasing the sensitivity to random initializations by duplicating activation functions. In addition, we run multiple trials and find the best results, either manually or through an automated system [6], [7]. In future work, it may be possible to find the true global minimum without resorting to multiple trials as it has been shown that overparameterized neural networks with certain types of activation functions are able to reach the global minimum through gradient descent in linear time regardless of the random initialization [27].

Other directions for future work include expanding the types of deep learning architectures that the EQL network can integrate with. For example, supporting spatiotemporal systems can lead to PDE discovery. The spatial derivatives could be calculated using known finite-difference approximations or learnable kernels [8]. Another possible extension is to introduce parametric dependence in which unknown parameters have a time dependence, which has been studied in PDE discovery using group sparsity [28]. In addition, the encoder can be expanded to capture a wider variety of data, such as videos [13], audio signals, and text. These capabilities will allow deep learning to be applied in scientific exploration and discovery.

APPENDIX A
EQL NETWORK DETAILS

The activation functions in each hidden layer consist of

$$[1(\times 2), g(\times 4), g^2(\times 4), \sin(2\pi g)(\times 2),\\ e^g(\times 2), \mathrm{sigmoid}(20g)(\times 2), g_1 * g_2(\times 2)]$$

where the sigmoid function is defined as

$$\mathrm{sigmoid}(g) = \frac{1}{1+e^{-g}}$$

and the $(\times i)$ indicated the number of times each activation function is duplicated. The sin and sigmoid functions have multipliers inside so that the functions more accurately represent their respective shapes inside the input domain of $x \in [-1, 1]$. Unless otherwise stated, these are the activation functions used for the other experiments as well. The exact number of duplications is arbitrary and does not have a significant impact on the system's performance. Future work may include experimenting with a larger number of duplications.

We use two phases of training, where the first phase has a learning rate of 10^{-2} and a regularization weight of 5×10^{-3} for 2000 iterations. Small weights are frozen and set to 0 after the first phase. The second phase of training has a learning rate of 10^{-3} for 10 000 iterations.

To benchmark our symbolic regression system, we choose a range of trial functions that our architecture can feasibly construct, train the network through 20 trials, and count how many times it reaches the correct answer. Benchmarking results are shown in Table V. As mentioned in Section III-A, we only need the network to find the correct equation at least once since we can construct a system that automatically picks out the correct solution based on equation simplicity and test error.

A. Computational Efficiency

With respect to the task of symbolic regression, we should note that this algorithm does not offer an asymptotic speedup

TABLE V

BENCHMARK RESULTS FOR THE EQL NETWORK

Function	Success Rate	
	$L_{0.5}$	L_0
x	1	1
x^2	0.6	0.75
x^3	0.3	0.05
$\sin(2\pi x)$	0.45	0.85
xy	0.8	1
$\frac{1}{1+e^{-10x}}$	0.3	0.55
$\frac{xy}{2}+\frac{z}{2}$	0.05	0.95
$\exp(-x^2)$	0.05	0.15
$x^2+\sin(2\pi y)$	0.2	0.8
x^2+y-2z	0.6	0.9

over conventional symbolic regression algorithms, as the problem of finding the correct expression requires a combinatorial search over the space of possible expressions and is NP-hard. Rather, the advantage here is that by solving symbolic regression problems through gradient descent, we can integrate symbolic regression with deep learning architectures.

Experiments are run on an Nvidia GTX 1080 Ti. Training the EQL network with two hidden layers ($L = 2$) for 20 000 epochs takes 37 s, and training the EQL network with three hidden layers ($L = 3$) takes 51 s.

In general, the computational complexity of the EQL network itself is the same as that of a conventional fully connected neural network. The only difference is the activation functions that are applied by iterating over $\mathbf{g}$ and thus takes $\mathcal{O}(n)$ time, where n is the number of nodes in each layer. However, the computational complexity of a neural network is dominated by the weight matrix multiplication, which takes $\mathcal{O}(n^2)$ time for both the EQL network and the conventional fully connected neural network.

APPENDIX B
RELAXED L_0 REGULARIZATION

We have also implemented an EQL network that uses a relaxed form of L_0 regularization for neural networks introduced in [14]. We briefly review the details here, but refer the reader to [14] for more details.

The weights $\mathbf{W}$ of the neural network are reparameterized as

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

where ideally each element of $\mathbf{z}$, $z_{j,k}$, is a binary "gate," $z_{j,k} \in \{0, 1\}$. However, this is not differentiable and so we allow $z_{j,k}$ to be a stochastic variable drawn from the hard concrete distribution

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

where $\alpha_{j,k}$ is a trainable variable that describes the location of the hard concrete distribution, and β, ζ, and γ are hyperparameters that describe the 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, we can calculate an analytical form for the expectation of the regularization penalty over the distribution parameters. The total loss function is then

$$\mathcal{L} = \frac{1}{N}\sum(y_i - \hat{y}_i)^2 + \sum_{j,k}\text{sigmoid}\left(\log\alpha_{j,k} - \beta\log\frac{-\gamma}{\zeta}\right).$$

The advantage of L_0 regularization is that it enforces sparsity without placing a penalty on the magnitude of the weights. This also allows us to train the system without needing a final stage where small weights are set to 0 and frozen. While the reparameterization in [14] requires us to double the number of trainable parameters in the neural network, the regularization is only applied to the EQL network, which is small compared to the rest of the architecture.

In our experiments, we use the hyperparameters for the L_0 regularization suggested in [14], although these can be optimized in future work. In addition, while Louizos *et al.* [14] applied group sparsity to the rows of the weight matrices with the goal of computational efficiency, we apply parameter sparsity with the goal of simplifying the symbolic expression. We benchmark the EQL network using L_0 regularization with the aforementioned trial functions and list the results in Table V. The success rates appear to be as good or better than the network using $L_{0.5}$ regularization for most of the trial functions that we have picked. We have also integrated the EQL network using L_0 regularization into the MNIST arithmetic and kinematics architectures and have found similar results as the EQL network using the $L_{0.5}$ regularization.

APPENDIX C
EXPERIMENT DETAILS

A. MNIST Arithmetic

The encoder network consists of a convolutional layer with 32 5×5 filters followed by a max-pooling layer, another convolutional layer with 64 5×5 filters followed by a max-pooling layer, and two fully connected layers with 128 and 16 units each with ReLU activation units. The max-pooling layers have a pool size of 2 and a stride length of 2. The fully connected layers are followed by a one-unit layer with batch normalization. The output of the batch normalization layer is divided by 2 such that the standard deviation of the output is 0.5. This decreases the range of the inputs to the EQL network since the EQL network was constructed assuming an input domain of $x \in [-1, 1]$. In addition, the output of the EQL network, $\hat{y}^*$, is scaled as $\hat{y} = 9\hat{y}^* + 9$ before being fed into the loss function so as the normalize the output against the range of expected y (this is equivalent to normalizing y to the range $[-1, 1]$).

The ReLU network that is trained in place of the EQL network for comparison consists of two hidden layers with 50 units each and ReLU activation.

We use two phases of training, where the first phase uses a learning rate of 10^{-2} and a regularization weight $\lambda = 0.05$. The second phase uses a learning rate of 10^{-4} and no regularization. The small weights are frozen between the first and second phases with a threshold of $\alpha = 0.01$. Each phase is trained for 10 000 iterations.

B. Kinematics

To generate the kinematics data set, we sample 100 values for a and generate a time series $\{\mathbf{x}_t\}_{t=0}^{T_x-1}$ and $\{\mathbf{y}_t\}_{t=0}^{T_y}$ for each a. The input series is propagated for $T_x = 100$ time steps.

The DE consists of 2 1-D convolutional layers with 16 filters of length 5 in each layer. These are followed by a hidden layer with 16 nodes and ReLU activation function, an output layer with one unit, and a batch normalization layer with a standard deviation of 0.5. The ReLU network that is trained in place of the EQL network for comparison is the same as that of the MNIST task.

We use two phases of training, where the first phase uses a learning rate of 10^{-2} and a regularization weight of $\lambda = 10^{-3}$ for a total of 5000 iterations. The system is trained on $T_y = 1$ time step for the first 1000 iterations and then $T_y = 5$ time steps for the remainder of the training. The small weights are frozen between the first and second phases with a threshold of $\alpha = 0.1$. The second phase uses a base learning rate of 10^{-3} and no regularization for 10 000 iterations.

C. SHO

To generate the SHO data set, we sample 1000 data points values for ω^2 and generate time series $\{\mathbf{x}_t\}_{t=0}^{T_x-1}$ and $\{\mathbf{y}_t\}_{t=0}^{T_y}$ for each ω^2. The input series is propagated for $T_x = 500$ time steps with a time step of $\Delta t = 0.1$. The output series is propagated for $T_y = 25$ time steps with the same time step.

The DE is the same architecture as used in the kinematics experiment. Due to the greater number of time steps that the system needs to propagate, the EQL network does not duplicate the activation functions for all functions. The functions in each hidden layer consist of

$$[1(\times 2), g(\times 2), g^2, \sin(2\pi g), e^g, 10 g_1 * g_2(\times 2)].$$

The ReLU network that is trained in place of the EQL network for comparison consists of four hidden layers with 50 units each and ReLU activation functions.

We use three phases of training, where the first phase uses a learning rate of 10^{-2} and a regularization weight of $\lambda = 4 \times 10^{-5}$ for a total of 2000 iterations. The system starts training on $T_y = 1$ time steps for the first 500 time steps and then add 2 more time steps every 500 iterations for a total of $T_y = 7$ time steps. In the second phase of training, we increase the number of time steps to $T_y = 25$, decrease the base learning rate to 2×10^{-3}, and increase the regularization weight to $\lambda = 2 \times 10^{-4}$. The small weights are frozen between the second and third phases with a threshold of $\alpha = 0.01$. The third and final phases of training use a base learning rate of 10^{-3} and no regularization.

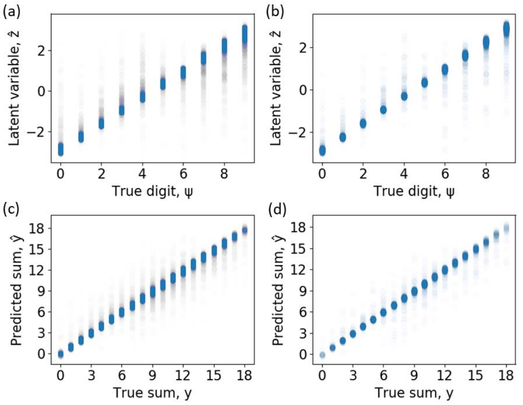


Fig. 9. Ability of the encoder to differentiate between digits as measured by the latent variable z versus the true digit ψ for digits χ drawn from the MNIST (a) training data set and (b) test data set. The ability of the entire architecture to fit the label y as measured by the predicted sum $\hat{y}$ versus the true sum y for digits χ drawn from the MNIST (c) training data set and (d) MNIST test data set.

APPENDIX D
ADDITIONAL MNIST ARITHMETIC DATA

The results presented in Fig. 6 and Table I are drawn from one of several trials, where each in each trial the network is trained from a different random initialization of the network weights. Due to the random initialization, the EQL does not reach the same equation every time. Here, we present results from additional trials to demonstrate the variability in the system's behavior as well as the system's robustness to the random initializations.

The experimental details are described in Section III-B where digits $\chi_{1,2}$ are drawn from the entire MNIST training data set. We refer to the results shown in Fig. 6 and Table I as Trial 1.

The results for Trial 2 are shown in Fig. 9. Similar to Trial 1, Trial 2 produces a linear relationship between the true digit ϕ and the latent variable z, although there is a positive instead of negative correlation. As previously mentioned, there is no bias placed on the latent variable z so whether there is a positive or negative correlation is arbitrary and depends on the random initialization of the weights. The trained architecture produced the following expression from the EQL network:

$$\hat{y} = 1.565 z_1 + 1.558 z_2 + 9. \tag{10}$$

Note the positive coefficients in (10), which reflects the positive correlation shown in Fig. 9(a) and (b). As shown in Fig. 9(c) and (d), the network is still able to accurately predict the sum y.

The results for Trial 3 are shown in Fig. 10. Note that in this case, the relationship between ϕ and z is no longer linear. However, the encoder still finds a one-to-one mapping between ϕ and z, and the EQL network is still able to extract the information from z such that it can predict the correct sum, as shown in Fig. 10(c) and (d).

The equation found by the EQL network is

$$\hat{y} = -4.64 \sin(2.22 z_1) - 4.63 \sin(2.21 z_2) + 9. \tag{11}$$

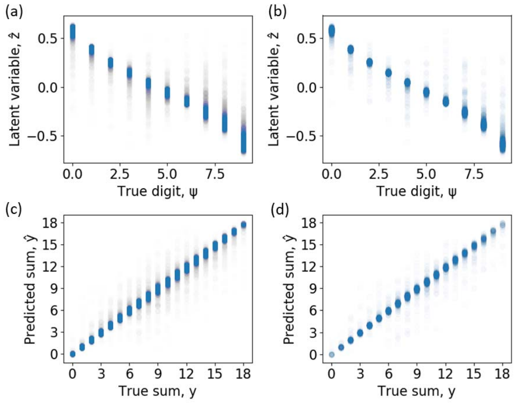


Fig. 10. Ability of the encoder to differentiate between digits as measured by the latent variable z versus the true digit ψ for digits χ drawn from the MNIST (a) training data set and (b) test data set. The ability of the entire architecture to fit the label y as measured by the predicted sum $\hat{y}$ versus the true sum y for digits χ drawn from the MNIST (c) training data set and (d) MNIST test data set.

This is consistent with the insight that the curve in Fig. 10(a) and (b) represents an inverse sine function. Thus, (11) is first inverting the transformation from ϕ to z to produce a linear mapping and then adding the two digits together. While the EQL network does not always give the exact equation we expect, we can still gain insight into the system from analyzing the latent variable and the resulting equation.

ACKNOWLEDGMENT

The authors would like to acknowledge Joshua Tenenbaum, Max Tegmark, Jason Fleischer, Alexander Alemi, Jasper Snoek, Stjepan Picek, Rumen Dangovski, and Ilan Mitnikov for fruitful conversations. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

REFERENCES

[1] J. Koza, “Genetic programming as a means for programming computers by natural selection,” *Statist. Comput.*, vol. 4, no. 2, pp. 87–112, Jun. 1994. [Online]. Available: http://link.springer.com/10.1007/BF00175355

[2] M. Schmidt and H. Lipson, “Distilling free-form natural laws from experimental data,” *science*, vol. 324, no. 5923, pp. 5–81, Apr. 2009. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/19342586

[3] 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. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/27035946 and http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC4839439

[4] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz, “Data-driven discovery of partial differential equations,” *Sci. Adv.*, vol. 3, no. 4, Apr. 2017, Art. no. e1602614.

[5] H. Schaeffer, “Learning partial differential equations via data discovery and sparse optimization,” *Proc. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 473, no. 2197, Jan. 2017, Art. no. 20160446. [Online]. Available: http://rspa.royalsocietypublishing.org/content/royprsa/473/2197/20160446.full.pdf

[6] G. Martius and C. H. Lampert, “Extrapolation and learning equations,” 2016, *arXiv:1610.02995*. [Online]. Available: http://arxiv.org/abs/1610.02995

[7] S. S. Sahoo, C. H. Lampert, and G. Martius, “Learning equations for extrapolation and control,” 2018, *arXiv:1806.07259*. [Online]. Available: http://arxiv.org/abs/1806.07259

[8] Z. Long, Y. Lu, X. Ma, and B. Dong, “PDE-net: Learning PDEs from data,” in *Proc. Mach. Learn. Res.*, Jul. 2018, pp. 3208–3216. [Online]. Available: http://proceedings.mlr.press/v80/long18a.html

[9] Z. Long, Y. Lu, and B. Dong, “PDE-net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network,” 2018, *arXiv:1812.04426*. [Online]. Available: http://arxiv.org/abs/1812.04426

[10] A. Trask, F. Hill, S. E. Reed, J. Rae, C. Dyer, and P. Blunsom, “Neural arithmetic logic units,” in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. New York, NY, USA: Curran Associates, Inc., 2018, pp. 8035–8044. [Online]. Available: http://papers.nips.cc/paper/8027-neural-arithmetic-logic-units.pdf

[11] D. Zheng, V. Luo, J. Wu, and J. B. Tenenbaum, “Unsupervised learning of latent physical properties using perception-prediction networks,” in *Proc. 34th Conf. Uncertainty Artif. Intell. (UAI)*, vol. 1, 2018, pp. 497–507.

[12] P. Y. Lu, S. Kim, and M. Soljačić, “Extracting interpretable physical parameters from spatiotemporal systems using unsupervised learning,” 2019, *arXiv:1907.06011*. [Online]. Available: http://arxiv.org/abs/1907.06011

[13] P. Chari, C. Talegaonkar, Y. Ba, and A. Kadambi, “Visual physics: Discovering physical laws from videos,” 2019, *arXiv:1911.11893*. [Online]. Available: http://arxiv.org/abs/1911.11893

[14] C. Louizos, M. Welling, and D. P. Kingma, “Learning sparse neural networks through L_0 regularization,” 2017, *arXiv:1712.01312*. [Online]. Available: http://arxiv.org/abs/1712.01312

[15] D. Molchanov, A. Ashukha, and D. Vetrov, “Variational dropout sparsifies deep neural networks,” in *Proc. 34th Int. Conf. Mach. Learn. (ICML)*, vol. 5, 2017, pp. 3854–3863.

[16] M. Zhu and S. Gupta, “To prune, or not to prune: Exploring the efficacy of pruning for model compression,” 2017, *arXiv:1710.01878*. [Online]. Available: http://arxiv.org/abs/1710.01878

[17] T. Gale, E. Elsen, and S. Hooker, “The state of sparsity in deep neural networks,” 2019, *arXiv:1902.09574*. [Online]. Available: http://arxiv.org/abs/1902.09574

[18] B. K. Natarajan, “Sparse approximate solutions to linear systems,” *SIAM J. Comput.*, vol. 24, no. 2, pp. 227–234, Apr. 1995. [Online]. Available: http://epubs.siam.org/doi/10.1137/S0097539792240406

[19] S. Srinivas, A. Subramanya, and R. V. Babu, “Training sparse neural networks,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jul. 2017, pp. 455–462. [Online]. Available: http://ieeexplore.ieee.org/document/8014795/

[20] Z. Xu, H. Zhang, Y. Wang, X. Chang, and Y. Liang, “$\mathrm{L}_{1/2}$ regularization,” *Sci. China Inf. Sci.*, vol. 53, no. 6, pp. 1159–1169, Jun. 2010. [Online]. Available: http://link.springer.com/10.1007/s11432-010-0090-0

[21] Z.-B. Xu, H.-L. Guo, Y. Wang, and H. Zhang, “Representative of $\mathrm{L}_{1/2}$ Regularization among L_q $(0 < \mathrm{q} \leq 1)$ regularizations: An experimental study based on phase diagram,” *Acta Automatica Sinica*, vol. 38, no. 7, pp. 1225–1228, Jul. 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1874102911602930

[22] Q. Fan, J. M. Zurada, and W. Wu, “Convergence of online gradient method for feedforward neural networks with smoothing $\mathrm{L}_{1/2}$ regularization penalty,” *Neurocomputing*, vol. 131, pp. 208–216, May 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231213010825

[23] W. Wu, Q. Fan, J. M. Zurada, J. Wang, D. Yang, and Y. Liu, “Batch gradient method with smoothing $L_{1/2}$ regularization for training of feedforward neural networks,” *Neural Netw.*, vol. 50, pp. 72–78, Feb. 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608013002700

[24] M. Abadi *et al.* (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. [Online]. Available: http://tensorflow.org/

[25] T. Tieleman and G. Hinton, “Lecture 6.5-RMSPROP: Divide the gradient by a running average of its recent magnitude,” *COURSERA, Neural Netw. Mach. Learn.*, vol. 4, no. 2, pp. 26–31, 2012.

[26] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, “The loss surfaces of multilayer networks,” *J. Mach. Learn. Res.*, vol. 38, pp. 192–204, 2015.

[27] S. S. Du, J. D. Lee, H. Li, L. Wang, and X. Zhai, “Gradient descent finds global minima of deep neural networks,” 2018, *arXiv:1811.03804*. [Online]. Available: http://arxiv.org/abs/1811.03804

[28] 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, Apr. 2019.

Samuel Kim received the A.B. degree in physics from Harvard University, Cambridge, MA, USA, in 2015, and the M.S. degree in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, in 2019, where he is currently pursuing the Ph.D. degree.

From 2013 to 2015, he was a Research Assistant with the Laboratory for Nanoscale Optics, Harvard University. From 2015 to 2017, he was a Research Scientist with the Johns Hopkins University Applied Physics Laboratory, Laurel, MD, USA. His research interests include computational electromagnetics, and the intersection between physics and machine learning.

Mr. Kim was a recipient of the National Defense Science and Engineering Graduate Fellowship in 2019.

Peter Y. Lu received the A.B. degree in physics and mathematics from Harvard University, Cambridge, MA, USA, in 2016. He is currently pursuing the Ph.D. degree in physics with the Massachusetts Institute of Technology, Cambridge.

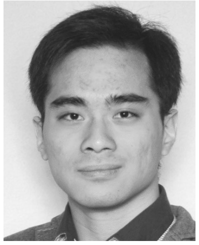
His research interest includes computational methods for physics with an emphasis on machine learning and statistical methods.

Mr. Lu is a member of the American Physical Society. He was a recipient of the National Defense Science and Engineering Graduate Fellowship in 2018.

Srijon Mukherjee (Student Member, IEEE) was born in Mountain View, CA, USA, in 1999. He is currently pursuing the B.S. degree in physics and computer science with the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA.

He worked as a Software Development Intern at Amazon, Seattle, WA, USA, in 2019, and an Quantitative Analyst Intern at D. E. Shaw and Company New York, NY, USA, in 2020. He is currently a Teaching Assistant with the Department of Electrical Engineering and Computer Science, MIT. His research interests include security, high-performance computing, and theoretical computer science.

Mr. Mukherjee is a member of the IEEE-HKN. He was a recipient of the MIT EECS Undergraduate Teaching Assistant Award in 2020.



Michael Gilbert was born in Jakarta, Indonesia, in 2000. He is currently pursuing the bachelor's degree in computer science with the Massachusetts Institute of Technology, Cambridge, MA, USA.

From 2018 to 2019, he worked part time under the Undergraduate Research Opportunities Program (UROP) at MIT Quest for Intelligence. In the following summer, he was an Intern at International Business Machines Research, Yorktown Heights, NY, USA. He was an undergraduate Researcher with MIT Media Lab from 2019 to 2020 and the MIT Computer Science and Artificial Intelligence Laboratory in the summer of 2020.

Li Jing received the B.S. degree in physics from Peking University, Beijing, China, in 2014, and the Ph.D. degree in physics from the Massachusetts Institute of Technology, Cambridge, MA, USA, in 2019.

He has been a Post-Doctoral Researcher at Facebook AI Research, New York, NY, USA, since 2020. His research interests include self-supervised learning, recurrent neural networks, and AI for scientific applications.

Vladimir Čeperić (Member, IEEE) received the dual Ph.D. degree in electrical engineering from KU Leuven, Leuven, Belgium, and the University of Zagreb, Zagreb, Croatia, in 2013.

He is currently an Assistant Professor with the Department of Electronics, Microelectronics, Computer and Intelligent Systems, University of Zagreb, Zagreb, Croatia. His research interests include microelectronics and machine learning.

Marin Soljačić received the bachelor's degree in computer science from the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, in 1996, and the Ph.D. degree in physics from Princeton University, Princeton, NJ, USA, in 2000.

He is currently a Professor of physics with MIT. He was the Founder of WiTricity Corporation, Watertown, MA, USA, in 2007, Lux Labs, in 2017, and Lightelligence in 2017. He is a coauthor of more than 200 scientific articles and more than 100 issued U.S. patents. He has been invited to give more than 100 invited talks at conferences and universities around the world.

Dr. Soljačić 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. In 2008, he was awarded a MacArthur fellowship "genius" grant. He is an international member of the Croatian Academy of Engineering since 2009. In 2011, he became a Young Global Leader (YGL) of the World Economic Forum. In 2014, he received the Blavatnik National Award, as well as Invented Here! (Boston Patent Law Association). In 2017, he was awarded "The Order of the Croatian Daystar, with the image of Ruder Bošković," the Croatian President's top medal for Science. In 2017, the Croatian President also awarded him with "The Order of the Croatian Interlace" medal.