Deep Learning and Symbolic Regression for Discovering Parametric Equations

Anonymous Authors[1]

Abstract

Symbolic regression is a machine learning technique that can learn the governing formulas from data and thus has the potential to transform scientific discovery. However, symbolic regression is still limited in the complexity 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. Here we develop a method that uses neural networks to extend symbolic regression to parametric systems where some coefficient may vary as a function of time but the underlying governing equation remains constant. We demonstrate our method on various analytic expressions and PDEs with varying coefficients and show that it extrapolate well outside of the training domain. The neural network-based architecture can also integrate with other deep learning architectures so that it can analyze high-dimensional data while being trained end-to-end in a single step. To this end we integrate our architecture with convolutional neural networks and train the system end-to-end to discover various physical quantities from 1D images of spring systems where the spring constant may vary.

1. Introduction

Discovering the governing equations of nature is key to all scientific disciplines. Many complex systems can be described by mathematical equations which in turn can be used for discovery and design, ranging from Hooke's law for harmonic oscillators to Maxwell's equations for electrodynamics. While scientists typically spend years developing insights to discover these equations, machine learning has become alluring in its potential to tackle and automate extremely complex tasks. For example, deep learning in recent years has been able to create images from captions (Ramesh et al., 2022) and predict a protein's 3D structure (Jumper et al., 2021) far better than humans could with hand-constructed algorithms. However, deep learning models are often black-box, making it difficult to gain scientific insight from these techniques. Thus, in order to make deep learning widely applicable for scientific discovery, we need to develop methods that are interpretable so that scientists can extract meaningful information from complex datasets.

Symbolic regression is a machine learning method that finds a mathematical expression that fits the data, thus resulting in an interpretable model. Symbolic regression is typically implemented through genetic programming, which searches through the space of mathematical expressions while ensuring that the equation is viable through various heuristics (Koza, 1994). The equations are pieced together through basic building blocks known as primitive functions, which include constants and simple functions (e.g. addition, multiplication, sine). Schmidt & Lipson (2009), one of the most popular earlier works in this direction, demonstrated how symbolic regression could discover equations of motions including Hamiltonians and Lagrangians for various physical systems. However, these approaches do not scale well to high-dimensional problems and typically require numerous hand-built heuristics and rules.

There have been numerous approaches into extending symbolic regression to more complex datasets. For example, the Sparse Identification of Nonlinear Dynamical systems (SINDy) method discovers the governing equations for dynamical systems and has been demonstrated on a variety of systems including ODEs, PDEs, conservation laws, and control systems. Additionally, SINDy has been extended to parametric systems in which some coefficient in the equation may vary over time or space such that the equation evolves (Rudy et al., 2019).

There have also been numerous approaches at combining deep learning and symbolic regression to analyze more complex tasks. 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 (Udrescu & Tegmark, 2020). Neural network autoencoders have been combined with SINDy to enable equation discovery on high-dimensional systems (Champion et al., 2019).

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

PDE-Net 2.0 incorporates a symbolic network to discover PDEs using convolutional networks with constrained filters (Long et al., 2019). Lu et al. (2021) incorporates a symbolic network with a neural network encoder to discover ODE and PDE systems from partial observations. Cranmer et al. (2020) performs traditional symbolic regression on a graph neural network weights in a 2-step process to discover the dynamics of many-body systems.

In particular, a neural network architecture called the EQL network was proposed that can perform symbolic regression by replacing the activation functions with primitive functions (Martius & Lampert, 2016; Sahoo et al., 2018). Kim et al. (2020) showed how this architecture can then be integrated into other deep learning architectures including convolutional networks and recurrent networks to perform symbolic regression on high-dimensional and dynamic systems, while allowing the entire architecture to be trained end-to-end through backpropagation. Costa et al. (2020) further extended this for recursive programs, implicit functions, and image classification.

In this work we extend (Kim et al., 2020) and enable neural network-based symbolic regression to parametric equations, where one or multiple of the coefficients may vary along some dimension (such as time) while the underlying equation structure remains the same. We propose a novel architectures, the parametric EQL network, which can each discover parametric equations. We demonstrate our method on various analytic equations, a PDE, and a high-dimensional dataset consisting of images of particles.

2. EQL Network

The EQL network is a neural network architecture that can perform symbolic regression by replacing the nonlinear activation functions with primitive functions. In Section 2.1 we briefly introduce the base EQL architecture for symbolic regression, and more details can be found in (Kim et al., 2020). We also propose several modifications to the EQL network that improve its training behavior. In Section 2.4 we propose a variant of the EQL architecture that can discover parametric equations. Note that in our discussion and notation, we assume that the coefficients are parameterized with respect to *time* as this provides a convenient intuition applicable to many systems. However, the parameterization can also be with respect to other quantities (e.g. space).

2.1. Base Architecture

The output of the ith layer of a fully-connected neural network can be described by

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

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

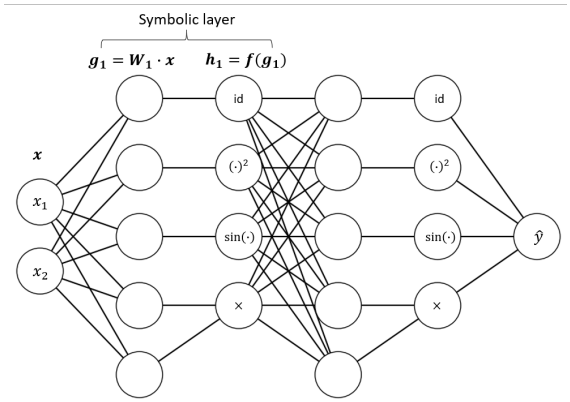

Figure 1. EQL network architecture for symbolic regression

where $\mathbf{W}$ is a weight matrix, f is the activation function, and $\mathbf{h}_0 = \mathbf{x}$ is the input data. The activation function for the final layer is typically linear, 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 functions such as ReLU or sigmoid for the activation function, the EQL network uses a vector of primitive functions, where each component may be a different primitive function (e.g. identity, square, sine) and where a primitive function may take multiple inputs (e.g. multiplication). 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.

2.2. Sparsity

To ensure the interpretability of symbolic regression, we need 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 symbols in the expression. For the EQL network, we use sparsity regularization on the network weights such that as many of the weights are set to 0 as possible. While (Kim et al., 2020) primarily uses a smoothed $L_{0.5}$ regularization, in this work we use a relaxed form of L_0 regularization (Louizos et al., 2017). We briefly review the details here, and refer the reader to the above references for more details.

The weights of the neural network are reparameterized as

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

where $\mathbf{z}$ can be interpreted as a gate variable. 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:

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

where α is a trainable variable that describes the location of the hard concrete distribution, and β, ζ, γ are hyperparameters that describe the distribution. In the case of binary gates, the regularization penalty would simply be the element-wise sum of $\mathbf{z}$ (i.e., the number of non-zero elements in $\mathbf{W}$). In the case of the hard concrete distribution, we can calculate an analytical form for the expectation of the sparsity regularization penalty over the distribution parameters:

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

where j is indexing over all of the elements of the weights. This is differentiable and so it can be applied to neural networks.

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 non-zero weights. Additionally, it lends itself to a straightforward definition of group sparsity across time-steps as we will see in Section 2.4. In our experiments, we use the hyperparameters for the L_0 regularization suggested by (Louizos et al., 2017).

2.3. Skip Connections

In this work, we add skip connections to the EQL network to introduce an inductive bias towards simpler equations while simultaneously enabling the learning of more complex equations. In particular, we turn to the skip connections introduced by DenseNets which concatenates the output of the previous layer with that of the next layer (Huang et al., 2017). More specifically, we modify Equation 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 towards learning simpler functions, since functions can route "directly" to the output without needing to go through the identity primitive function of successive layers.

2.4. Parameterized Architecture

To modify the EQL network to learn parametric systems, we parameterize the weights $\mathbf{W}$ themselves so that they are a function of time, $\mathbf{W}(t)$. While a number of models can be used to parameterize the weights, we use what we

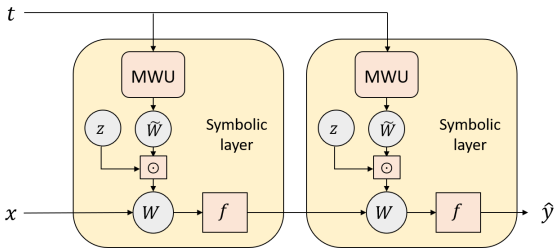

Figure 2. Architecture of the parameterized EQL network.

call the meta-weight unit (MWU), which consists of a fully-connected network that time t as an input and outputs a weight matrix for a single layer. The gate variables $\mathbf{z}$ are not modified from the original EQL network and are thus not a function of the parametric variable. As a result, all of the "time steps" share the same sparsity regularization allowing us to forego any further modifications to implement group sparsity.

The MWU can handle arbitrary functions, including those with discontinuities, and can be trained with backpropagation, allowing the entire system including the EQL network to be trained end-to-end. We call the overall architecture the parameterized EQL (PEQL) network, and is shown in Figure 2.

The advantage of this architecture is that it can make predictions on a continuous domain of t and does not need to restrict the data to fixed points in time. More specifically, we can view the dataset as

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

2.5. Training

All neural network architectures are implemented in Tensorflow (Abadi et al., 2015). We use a sum of the MSE and the sparsity regularization for the loss function, and RMSprop to minimize the loss. For both learning rate and regularization weight schedules, we use a one cycle policy.

3. Results

3.1. Analytic Expressions

To verify the ability of the parameterized and stacked EQL architectures to perform symbolic regressions on parametric systems, we benchmark the networks on data generated from the analytical expressions listed in Table 1, namely $t \cdot x^2 + 3\,\text{sgn}(t) \cdot x$ and $\sin\left(\frac{5+t}{2} \cdot x\right)$, where sgn is the *sign* function (also known as the *signum* function).

For all tests, 512 training data points with $x \in [-3, 3]$ are sampled for each of 128 fixed values of $t \in [-3, 3]$ for a total of $512 \cdot 128 = 65\,536$ training examples. To test

Table 1. Learned equations of the parameterized network on select t values for various analytical equations.

$t \cdot x^2 + 3\,\mathrm{sgn}(t) \cdot x$

t	TRUE	LEARNED
-2.619	$-2.62x^2 - 3.00x$	$-2.63x^2 - 3.02x - 0.06$
-1.095	$-1.10x^2 - 3.00x$	$-1.10x^2 - 3.00x + 0.01$
0.381	$0.38x^2 + 3.00x$	$0.38x^2 + 3.00x - 0.01$
1.905	$1.91x^2 + 3.00x$	$1.90x^2 + 2.99x + 0.03$

$\sin\left(\frac{5+t}{2} \cdot x\right)$

t	TRUE	LEARNED
-2.619	$\sin(1.190x)$	$\sin(1.190x)$
-1.095	$\sin(1.952x)$	$\sin(1.952x)$
0.381	$\sin(2.690x)$	$\sin(2.691x)$
1.905	$\sin(3.452x)$	$\sin(3.452x)$

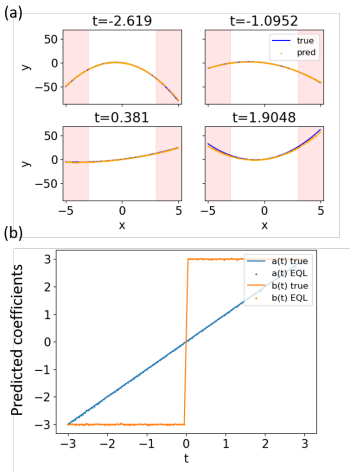

Figure 3. Results of the PEQL network for the function $f(t, x) = t \cdot x^2 + 3\,\mathrm{sgn}(t) \cdot x$. (a) Predictions for select values of t. Outputs with $|x| > 3$ (highlighted in red) are extrapolated. (b) Learned coefficient functions.

generalization, the EQL architectures are evaluated on test data points with $x \in [-5, 5]$.

Due to sensitivity of the 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 paper, 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.

Table 1 shows the learned equations for our benchmarks. Note that the PEQL does not learn a functional form for t, and so we list the learned equations at chosen points in t. Thus, the PEQL is able to learn arbitrary functions of the parametric coefficient, although we choose simple functional forms here for convenience.

As an example of the prediction fit and extrapolation ability of the PEQL, we look at the results for learning the function $f(t, x) = t \cdot x^2 + 3\,\mathrm{sgn}(t) \cdot x$ in Figure 3. We see that the

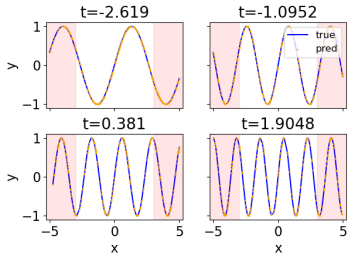

Figure 4. PEQL network predictions on select t values for the function $f(t, x) = \sin\left(\frac{5+t}{2} \cdot x\right)$. Outputs with $|x| > 3$ (shown in red) are extrapolated.

PEQL network prediction not only matches the training data extremely well, it also extrapolates outside of the training regime, $|x| > 3$. This is only possible because the PEQL has learned the underlying governing equation of the system. Figure 3(b) shows that the PEQL also learns the parametric coefficient, and demonstrates that it can learn arbitrary functions of t that may have discontinuities.

We also show the prediction results of the stacked network for the function $f(t, x) = \sin\left(\frac{5+t}{2} \cdot x\right)$ in Figure 4. Again, the predictions match the true data extremely well across time steps and outside of the training regime. Although sinusoidal functions are typically difficult to learn through linear regression techniques, our method is able to learn this function across multiple spatial frequencies. Additionally, our method integrates t into symbolic discovery and does not merely perform a regression to compute the t-dependent coefficients.

3.2. Differential Equation Datasets

We now look at a PDE datasets investigated in Rudy et al. (2019). In Rudy et al. (2019), the partial differential terms (e.g. $u_x, u_x x$ and their combinations (e.g. uu_x) were pre-computed and fed into SINDy to discover the governing PDE. Here we pre-compute the individual partial differential terms, but we do not explicitly pre-compute the combinations.

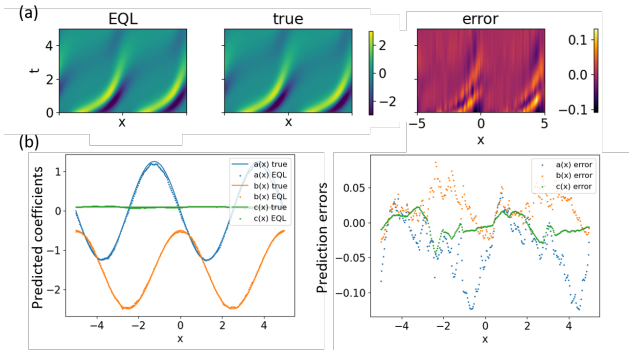

Figure 5. Results for learning the advection-diffusion equation using the PEQL network. (a) Predicted vs. actual values of u_t. (b) Predicted coefficient functions and prediction errors in ($u_t = a(x) \cdot u + b(x) \cdot u_x + c(x) \cdot u_{xx}$) with $a(x) = -\frac{2\pi}{5} \sin\left(\frac{2\pi}{5} \cdot x\right)$, $b(x) = -1.5 + \cos\left(\frac{2\pi}{5} \cdot x\right)$, and $c(x) = 0.1$.

3.2.1. 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 (Rudy et al., 2019):

$$u_t = f'(x)u + f(x)u_x + \epsilon u_{xx}. \tag{5}$$

Note that the parametric quantities vary with respect to space rather than time. The PDE is solved numerically using a spectral method on the domain $x \in [-5, 5]$ and $t \in [0, 5]$ with $f(x) = -1.5 + \cos\left(\frac{2\pi x}{5}\right)$ and $\epsilon = 0.1$ using code from (Rudy et al., 2019). Data was sampled from 256 different points in the x-domain and 512 different points in the t-domain, for a total of $256 \cdot 512 = 131\,072$ examples.

The predicted value of u_t and the predicted coefficients $\hat{f}'(x)$ and $\hat{f}(x)$ after training are shown in Figure 5. The predicted values match the actual values very closely and the EQL network is able to extract the correct equation. Again, note that the predicted coefficients by the fully-connected neural network are smooth as a function of x despite the lack of explicit regularization.

3.3. Spring System

Finally, we demonstrate the ability of the PEQL network to performing symbolic regression on structured, high-dimensional data by integrating with other deep learning architectures and training end-to-end.

We consider a dataset that consists of pairs of 1D images of point particles that interact through a spring-like force. The input data is a 1D grayscale image with 64 pixels which represents a 1D spatial domain $z \in [-4, 4]$. Each image

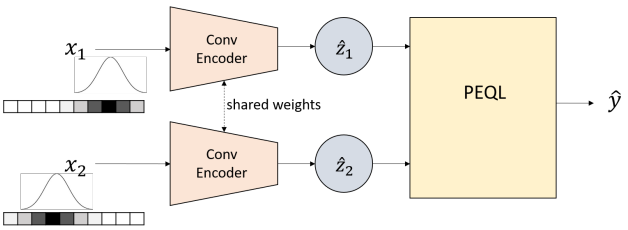

Figure 6. The combined architecture used for high-dimensional system tasks involving a convolutional encoder followed by an EQL network.

contains a single particle, represented by a Gaussian with mean centered at its position z_i and a fixed variance of 0.1. We look at two different target for symbolic regression: the spring force $F = -k(t)(z_2 - z_1)$ with $k(t) = \frac{5-t}{2}$.

To approach this problem, we use the architecture shown in Figure 6. Each image is fed into a separate encoder, where the two encoders share the same weights. The encoder consists of 2 convolutional layers followed by 3 fully-connected layers and a batch normalization layer. The encoders each output a single-dimensional latent variable $\hat{z}_1, \hat{z}_2$, which are then fed into the PEQL network. The batch normalization layer serves to constrain the variance (and thus, range) of the latent variable so that the PEQL network does not need to scale to arbitrarily-sized inputs. The PEQL network has a single scalar output, which is 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, z_i.

For all tests, 512 training data points with $z_1, z_2 \in [-3, 3]^2$ 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 $|z_2 - z_1| \leq 4$, 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**. 20 trials were run for each experiment and the trial with the lowest generalization error was selected.

Results for learning the spring force are shown in Figure **??**. We see that both the EQL network and the ReLU architectures are able to train on the data inside the training domain, but only the EQL network is able to extrapolate outside of the training regime whereas the ReLU network completely fails to extrapolate. Additionally, the EQL network learns the governing equation, with the learned parametric coefficient plotted in Figure 7 (left). Note that while the data is generated by the equation $F = -k(t)(z_2 - z_1)$, the EQL

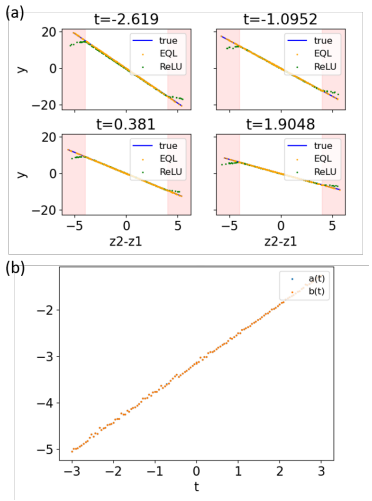

Figure 7. Results for learning the spring force function. (a) Predictions for select t values. Outputs with $|z_2 - z_1| > 4$ (shown in red) are extrapolated. (b) Learned coefficient functions in the equation $f(t, z_1, z_2) = -k(t) \cdot (z_2 - z_1) = -a(t) \cdot z_1 + b(t) \cdot z_2$ with $k(t) = \frac{5-t}{2}$ using the convolutional PEQL.

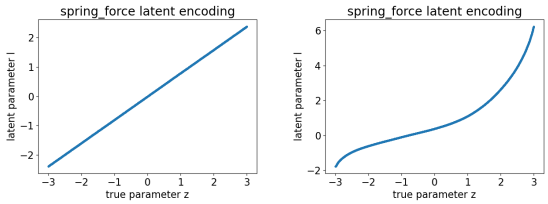

Figure 8. Learned latent representation using the (left) convolutional EQL network and (right) convolutional ReLU network for the spring force problem.

network learns the expression $\hat{F} = -a(t)\hat{z}_1 + b(t)\hat{z}_2$. Upon inspection, we see that $a(t) \approx b(t)$ and so the EQL network has discovered an approximately equal expression to what we expect.

An additional feature of the EQL network is the linear mapping of the latent variable to the true position in Figure 7 (right). While there is no explicit constraint or regularization placed on the 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, the latent variable mapping for the ReLU network is shown in Figure 8. While it is one-to-one, it is not linear since there is no bias to make the mapping linear.

4. Discussion

We have proposed a variant of the EQL network—the parameterized architecture—to enable neural network-based symbolic regression of parametric systems. We have demonstrated our system on parametric analytic equations, a PDE, as well as a dataset encoded as images. 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 we used analytic expressions for the parameterizations of the coefficients for simplicity of analysis. However, this is unnecessary and the parametric coefficient can be an arbitrary function without an analytic form. Thus, our system is useful to analyze 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 (Pathak et al., 2020; Kochkov et al., 2021).

The PEQL is much more flexible than other methods for parametric systems such as (Rudy et al., 2019), as the PEQL is able to interpolate in time and make predictions at arbitrary time points. We note that while the PEQL does not always perfectly converge, once the base equation has been found, further fine-tuning can be done to more accurately extract the parametric coefficients. In addition, another direction for future work to bridge this gap is to introduce different learning rate schedules for the EQL network and the MWU in the parameterized 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.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Champion, K., Lusch, B., Kutz, J. N., and Brunton, S. L. Data-driven discovery of coordinates and governing equations. *Proceedings of the National Academy of Sciences*, 116(45):22445–22451, 2019.

Costa, A., Dangovski, R., Dugan, O., Kim, S., Goyal, P., Soljačić, M., and Jacobson, J. Fast neural models for symbolic regression at scale. *arXiv preprint arXiv:2007.10784*, 2020.

Cranmer, M., Sanchez Gonzalez, A., Battaglia, P., Xu, R., Cranmer, K., Spergel, D., and Ho, S. Discovering symbolic models from deep learning with inductive biases. *Advances in Neural Information Processing Systems*, 33: 17429–17442, 2020.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

Kim, S., Lu, P. Y., Mukherjee, S., Gilbert, M., Jing, L., Čeperić, V., and Soljačić, M. Integration of neural network-based symbolic regression in deep learning for scientific discovery. *IEEE Transactions on Neural Networks and Learning Systems*, 32(9):4166–4177, 2020.

Kochkov, D., Smith, J. A., Alieva, A., Wang, Q., Brenner, M. P., and Hoyer, S. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21), 2021.

Koza, J. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 4(2):87–112, jun 1994. ISSN 0960-3174. doi: 10.1007/BF00175355. URL http://link.springer.com/10.1007/BF00175355.

Long, Z., Lu, Y., and Dong, B. Pde-net 2.0: Learning pdes from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, 399:108925, 2019.

Louizos, C., Welling, M., and Kingma, D. P. Learning Sparse Neural Networks through L_0 Regularization. *arXiv preprint arXiv:1712.01312*, dec 2017. URL https://arxiv.org/abs/1712.01312.

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

Martius, G. and Lampert, C. H. Extrapolation and learning equations. *arXiv preprint arXiv:1610.02995*, oct 2016. URL http://arxiv.org/abs/1610.02995.

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

Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., and Chen, M. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

Rudy, S., Alla, A., Brunton, S. L., and Kutz, J. N. Data-driven identification of parametric partial differential equations. *SIAM Journal on Applied Dynamical Systems*, 18(2):643–660, 2019. doi: 10.1137/18M1191944. URL https://doi.org/10.1137/18M1191944.

Sahoo, S., Lampert, C., and Martius, G. Learning equations for extrapolation and control. In *International Conference on Machine Learning*, pp. 4442–4450. PMLR, 2018.

Schmidt, M. and Lipson, H. Distilling free-form natural laws from experimental data. *Science (New York, N.Y.)*, 324(5923):81–5, apr 2009. ISSN 1095-9203. doi: 10.1126/science.1165893. URL http://www.ncbi.nlm.nih.gov/pubmed/19342586.

Udrescu, S.-M. and Tegmark, M. Ai feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, 2020.