Journal of Machine Learning Research 25 (2024) 1-26 Submitted 3/23; Revised 12/23; Published 1/24

Improving physics-informed neural networks with meta-learned optimization

Alex Bihlo ABIHLO@MUN.CA
Department of Mathematics and Statistics
Memorial University of Newfoundland,
St. John's (NL) A1C 5S7, Canada

Editor: Samy Bengio

Abstract

We show that the error achievable using physics-informed neural networks for solving differential equations can be substantially reduced when these networks are trained using meta-learned optimization methods rather than using fixed, hand-crafted optimizers as traditionally done. We choose a learnable optimization method based on a shallow multilayer perceptron that is meta-trained for specific classes of differential equations. We illustrate meta-trained optimizers for several equations of practical relevance in mathematical physics, including the linear advection equation, Poisson's equation, the Korteweg–de Vries equation and Burgers' equation. We also illustrate that meta-learned optimizers exhibit transfer learning abilities, in that a meta-trained optimizer on one differential equation can also be successfully deployed on another differential equation.

Keywords: Scientific machine learning, Physics-informed neural networks, Learnable optimization, Meta-learning, Transfer learning

1. Introduction

Physics-informed neural networks are a class of methods for solving systems of differential equations. Originally proposed in the 1990s by Lagaris et al. (1998) and popularized through the work of Raissi et al. (2019), physics-informed neural networks have seen an immense raise in popularity in the past several years. This is in part due to the overall rise in interest in all things related to deep neural networks, see LeCun et al. (2015), but also due to some practical advantages of this method compared to traditional numerical approaches such as finite difference, finite elements or finite volume methods. These advantages include the evaluation of derivatives using automatic differentiation, see Baydin et al. (2018), their mesh-free nature and an overall ease of implementation through modern deep-learning frameworks such as `JAX` (Bradbury et al., 2018), `TensorFlow` (Abadi et al., 2015) or `PyTorch` (Paszke et al., 2019). Given the expressive power of deep neural networks, cf. Cybenko (1989), neural networks are also a well-suited class of function approximation for the solution of systems of differential equations.

A main downside of physics-informed neural networks is that a complicated optimization problem involving a rather involved composite loss function has to be solved Raissi et al. (2019). The difficulty in solving such so-called multi-task problems is well-documented in the deep learning literature, see e.g. Yu et al. (2020). Moreover, since essentially all methods

of optimization for deep neural network used to solve differential equations today are at most of first-order, such as stochastic gradient descent, and its momentum-based flavours such as Adam, Kingma and Ba (2014), the level of error that can typically be achieved with vanilla physics-informed neural networks as proposed by Lagaris et al. (1998); Raissi et al. (2019) is often subpar compared to their traditional counterparts used in numerical analysis. While lower numerical error can be achieved using more involved strategies, such as domain decomposition approaches, see Jagtap et al. (2020); modified loss functions, see Jin et al. (2021); Wang et al. (2022); operator-based approaches, see Wang and Perdikaris (2023); or higher-order optimizers such as L-BFGS, see Nocedal and Wright (1999), all of these approaches either sacrifice some of the simplicity of vanilla physics-informed neural networks or substantially increase their training times.

Since a main culprit in the overall unsatisfactory error levels achievable with vanilla physics-informed neural networks is the optimization method used, it is natural to aim to find better optimizers. More broadly, optimization is a topic extensively studied in the field of machine learning, with many new optimizers being proposed that aim to overcome some of the (performance or memory) shortcomings of the de-facto standard Adam, see e.g. Lucas et al. (2019); Shazeer and Stern (2018). There has also been growing interest in the field of learnable optimization, referred to as *learning to learn* (Chen et al., 2022), which aims to develop optimization methods parameterized by neural networks, that are then meta-learned on a suitably narrow class of tasks, on which they typically outperform generic (non-learnable) optimization methods.

The aim of this paper is to explore the use of learnable optimization for training physics-informed neural networks. We show that meta-trained learnable optimizers with very few parameters can substantially outperform standard optimizer in this field. Moreover, once meta-trained, these optimizers can be used to train physics-informed neural networks with minimal computational overhead compared to traditional optimizers.

The further organization of this paper is as follows. In Section 2 we present a more formalized review on how neural networks can be used to solve differential equations. Section 3 presents a short overview of the relevant previous work on both physics-informed neural networks and learnable optimization. The main Section 4 introduces the class of learnable optimizers used in this work. Section 5 contains the numerical results obtained by using these meta-trained optimizers for solving a variety of differential equations using physics-informed neural networks. The transfer learning capabilities of the proposed learnable optimizers are investigated in Section 6. A summary with a discussion on further possible research directions can be found in the final Section 7.

2. Solving differential equations with neural networks

The numerical solution of differential equations with neural networks was first proposed in Lagaris et al. (1998). In this algorithm, the trial solution is brought into a form that accounts for initial and/or boundary conditions (as hard constraints), with the actual solution being found upon minimizing the mean-squared error that is defined as the residual of the given differential equations evaluated over a finite number of collocation points which are distributed over the domain of the problem. This method was recently popularized by Raissi et al. (2019), coining the term *physics-informed neural networks*, and extended to

also allow for the identification of differential equations from data. A recent review on this subject can be found in Cuomo et al. (2022).

More formally, consider the following initial–boundary value problem for a general system of L partial differential equations of order n,

$$\begin{aligned}
&\Delta^l(t,\mathbf{x},\mathbf{u}_{(n)})=0, \qquad l=1,\dots,L, \quad t\in[0,t_\mathrm{f}],\ \mathbf{x}\in\Omega,\\
&\mathsf{I}^{l_\mathrm{i}}(\mathbf{x},\mathbf{u}_{(n_\mathrm{i})|t=0})=0, \qquad l_\mathrm{i}=1,\dots,L_\mathrm{i}, \quad \mathbf{x}\in\Omega,\\
&\mathsf{B}^{l_\mathrm{b}}(t,\mathbf{x},\mathbf{u}_{(n_\mathrm{b})})=0, \quad l_\mathrm{b}=1,\dots,L_\mathrm{b}, \qquad t\in[0,t_\mathrm{f}],\ \mathbf{x}\in\partial\Omega,
\end{aligned} \tag{1}$$

where $t\in[0,t_\mathrm{f}]$ is the time variable, $\mathbf{x}=(x_1,\dots,x_d)\in\Omega$ is the tuple of spatial independent variables, $\mathbf{u}=(u^1,\dots,u^q)$ is the tuple of dependent variables, and $\mathbf{u}_{(n)}$ is the tuple of all derivatives of the dependent variables with respect to the independent variables of order not greater than n. The initial value operator is denoted by $\mathsf{I}=(\mathsf{I}^1,\dots\mathsf{I}^{L_\mathrm{i}})$ and $\mathsf{B}=(\mathsf{B}^1,\dots,\mathsf{B}^{L_\mathrm{b}})$ denotes the boundary value operator. The spatial domain is Ω and the final time is t_f.

In the following, we consider evolution equations for which the initial value operator reduces to

$$\mathsf{I}=\mathbf{u}(0,\mathbf{x})-\mathbf{f}(\mathbf{x}),$$

where $\mathbf{f}(\mathbf{x})=(f^1(\mathbf{x}),\dots,f^q(\mathbf{x}))$ is a fixed vector-valued function. We also consider Dirichlet boundary conditions of the form

$$\mathsf{B}=\mathbf{u}(t,\mathbf{x})-\mathbf{g}(t,\mathbf{x}),$$

where $\mathbf{g}(t,\mathbf{x})=(g^1(\mathbf{t},\mathbf{x}),\dots,g^q(t,\mathbf{x}))$ is another fixed vector-valued function.

Solving system (1) with a neural network $\mathcal{N}^{\boldsymbol{\theta}}$ requires the parameterization of the solution of this system in the form $\mathbf{u}^{\boldsymbol{\theta}}=\mathcal{N}^{\boldsymbol{\theta}}(t,\mathbf{x})$, where the weights $\boldsymbol{\theta}$ of the neural network are found upon minimizing the loss function

$$\mathcal{L}(\boldsymbol{\theta})=\mathcal{L}_\Delta(\boldsymbol{\theta})+\gamma_\mathrm{i}\mathcal{L}_\mathrm{i}(\boldsymbol{\theta})+\gamma_\mathrm{b}\mathcal{L}_\mathrm{b}(\boldsymbol{\theta}). \tag{2a}$$

Here

$$\begin{aligned}
\mathcal{L}_\Delta(\boldsymbol{\theta})&=\frac{1}{N_\Delta}\sum_{i=1}^{N_\Delta}\sum_{l=1}^{L}\left|\Delta^l\big(t_\Delta^i,\mathbf{x}_\Delta^i,\mathbf{u}^{\boldsymbol{\theta}}_{(n)}(t_\Delta^i,\mathbf{x}_\Delta^i)\big)\right|^2,\\
\mathcal{L}_\mathrm{i}(\boldsymbol{\theta})&=\frac{1}{N_\mathrm{i}}\sum_{i=1}^{N_\mathrm{i}}\sum_{l_\mathrm{i}=1}^{L_\mathrm{i}}\left|\mathsf{I}^{l_\mathrm{i}}\big(\mathbf{x}_\mathrm{i}^i,\mathbf{u}^{\boldsymbol{\theta}}_{(n_\mathrm{i})}(0,\mathbf{x}_\mathrm{i}^i)\big)\right|^2,\\
\mathcal{L}_\mathrm{b}(\boldsymbol{\theta})&=\frac{1}{N_\mathrm{b}}\sum_{i=1}^{N_\mathrm{b}}\sum_{l_\mathrm{b}=1}^{L_\mathrm{b}}\left|\mathsf{B}^{l_\mathrm{i}}\big(t_\mathrm{b}^i,\mathbf{x}_\mathrm{b}^i,\mathbf{u}^{\boldsymbol{\theta}}_{(n_\mathrm{b})}(t_\mathrm{b}^i,\mathbf{x}_\mathrm{b}^i)\big)\right|^2,
\end{aligned} \tag{2b}$$

are the mean squared error losses corresponding to the differential equation, the initial condition and the boundary value residuals, respectively, and γ_i and γ_b are positive scaling constants. These losses are evaluated over the collection of collocation points $\big\{(t_\Delta^i,\mathbf{x}_\Delta^i)\big\}_{i=1}^{N_\Delta}$ for the system Δ, $\big\{(0,\mathbf{x}_\mathrm{i}^i)\big\}_{i=1}^{N_\mathrm{i}}$ for the initial data, and $\big\{(t_\mathrm{b}^i,\mathbf{x}_\mathrm{b}^i)\big\}_{i=1}^{N_\mathrm{b}}$ for the boundary data, respectively. Upon successful minimization, the neural network $\mathcal{N}^{\boldsymbol{\theta}}$ provides a numerical parameterization of the solution of the given initial–boundary value problem.

3. Related work

Physics-informed neural networks were proposed by Lagaris et al. (1998), and popularized through the work of Raissi et al. (2019), and have since been used extensively for solving differential equations in science and engineering. While the general algorithm for training neural networks to solve differential equations is straightforward, several complications arise in practice. Firstly, balancing the individual loss contributions in (2b) so that all the initial values, the boundary values, and the differential equations are adequately enforced simultaneously constitutes a multi-task learning problem which may not be properly solved by minimizing the composite loss function (2a), see Sener and Koltun (2018); Yu et al. (2020) for some work on multi-task learning problems. Secondly, it is well-known that training neural networks using gradient descent methods leads to a spectral bias in the form of low frequencies being learned first and high-frequencies requiring longer training times, see Rahaman et al. (2019). Correspondingly, oscillatory solutions or stiff problems may not be accurately learned using standard physics-informed neural networks. Lastly, the general setup (2) requires proportionally more collocation points the larger the spatio-temporal domain of the differential equation being solved is. Training neural networks for solving differential equations over large spatio-temporal domains can destabilize training, which is frequently encountered in practice. In most cases, physics-informed neural networks for such problems incorrectly converge to a trivial constant solution of the given differential equation, see e.g. Bihlo and Popovych (2022); Penwarden et al. (2023); Wang et al. (2022). One straightforward solution for this problem is to break the entire domain into multiple subdomains, and solve a sequence of smaller problems with multiple neural networks instead. This multi-model approach has recently been used for solving the shallow-water equations on a rotating sphere by Bihlo and Popovych (2022).

Learnable optimization has been the topic of research since the works by Bengio et al. (1990, 1995), with Andrychowicz et al. (2016) popularizing the use of neural network based *learning to learn* optimization. The latter paper specifically introduced an LSTM-type neural network optimizer that is being trained using gradient descent. Subsequent work focussed on improving the performance of learnable neural network based optimizers by improving their training strategies, see e.g. Lv et al. (2017); Vicol et al. (2021), improving the LSTM architecture of the optimizer (Wichrowska et al., 2017), or replacing the LSTM-based architecture in favour of a simpler MLP-based one, cf. Harrison et al. (2022); Metz et al. (2022). Also, exploratory work has been done that aims to understand what exactly these learnable optimizers are learning (Maheswaranathan et al., 2021). Below, we will use the optimizer proposed in Harrison et al. (2022), as this optimizer was found to be both stable and fast to meta-train, and able to generalize to problems that are different from those the optimizer was trained on, all of which are properties desirable for physics-informed neural networks. For a more comprehensive review on learnable optimization consult the recent review paper by Chen et al. (2022).

To the best of our knowledge, the use of learnable optimization for physics-informed neural networks has not been pursued so far. The related field of using meta-learning to accelerating the training of physics-informed neural networks has been investigated in Liu et al. (2022) and Psaros et al. (2022) recently. Specifically, in these works the authors used meta-learning to discover suitable initialization methods and physics-informed neural net-

work loss functions that generalize across relevant task distributions, respectively, thereby speeding up training of individual physics-informed neural networks from these task distributions.

4. Meta-learnable optimization for physics-informed neural networks

A main goal of meta-learned optimization it to improve hand-designed optimization rules such as the Adam optimizer introduced by Kingma and Ba (2014) for updating the weight vector $\boldsymbol{\theta}$ of a neural network with loss function $L(\boldsymbol{\theta})$. Recall that the Adam update rule is given by

$$\begin{aligned}
&\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_{t-1}), \quad \mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)(\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_{t-1}))^2, \\
&\hat{\mathbf{m}}_t = \mathbf{m}_t/(1-\beta_1^t), \quad \hat{\mathbf{v}}_t = \mathbf{v}_t/(1-\beta_2^t), \\
&\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta \mathbf{w}_{\text{adam}} = \boldsymbol{\theta}_t - \eta \hat{\mathbf{m}}_t/(\sqrt{\hat{\mathbf{v}}_t} + \varepsilon),
\end{aligned}$$

where $t = 1, \ldots,$ is the optimization time step, $\mathbf{m}$ and $\mathbf{v}$ are the first and second moment vectors, with $\beta_1, \beta_2 \in [0, 1)$ being the exponential decay rates for the moment estimates, ε being a regularization constant, and η being the learning rate.

Similarly, the parameter updates of a meta-learned optimizer is structured as

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \mathbf{f}(\mathbf{z}_t; \boldsymbol{\vartheta}), \tag{3}$$

where $\mathbf{f}$ is the parametric update function with $\mathbf{z}_t$ referring to the input features of the learnable optimizer, and $\boldsymbol{\vartheta}$ are the trainable meta-parameters of the optimizer, usually the weights of a neural network. To allow for the learnable optimizer to be transferable to neural networks of different sizes it is customary to have the parameter update rule (3) act component-wise, with each weight θ_i of the weight vector $\boldsymbol{\theta}$ being updated in the same way. Thus, in the following we describe the parameteric update formula in terms of scalar variables, rather than vectorial quantities.

While there are several optimizer architectures that have been proposed in the literature, cf. Chen et al. (2022), here we use a relatively simple multi-layer perceptron for the optimizer architecture. Notably, we follow the work by Harrison et al. (2022) and structure the parametric update formula for each weight θ_i as

$$f = \lambda_1 \exp(\lambda_2 s_{\boldsymbol{\vartheta}}^{\text{adam}}) w_{\text{adam}} + \frac{\lambda_3}{\sqrt{v_t} + \varepsilon} d_{\boldsymbol{\vartheta}}^{\text{bb}} \exp(\lambda_4 s_{\boldsymbol{\vartheta}}^{\text{bb}}), \tag{4}$$

where λ_i, $i = 1, \ldots, 4$ are positive constants, w_{adam} corresponds to the Adam update step and $s_{\boldsymbol{\vartheta}}^{\text{adam}}$, $s_{\boldsymbol{\vartheta}}^{\text{bb}}$ and $d_{\boldsymbol{\vartheta}}^{\text{bb}}$ are to the output heads of the meta-learned optimizer with neural network weights $\boldsymbol{\vartheta}$.

On a high level, the first term in the learnable update formula (4) can be seen as a nominal term derived from the Adam update formula with scalable learning rate $\lambda_1 \exp(\lambda_2 s_{\boldsymbol{\vartheta}}^{\text{adam}}))$, which guarantees an update step in a descent direction, and the second term corresponds to a blackbox update term structured as the product of a directional and magnitudinal term, $d_{\boldsymbol{\vartheta}}^{\text{bb}}$ and $\exp(\lambda_4 s_{\boldsymbol{\vartheta}}^{\text{bb}})$, respectively, with the denominator $\sqrt{v_t} + \varepsilon$ acting as a pre-conditioner that should guarantee that the overall update formula leads corresponds to a descending on

the loss surface. For more details on the rationale behind the update rule (4), consult (Harrison et al., 2022).

The inputs $\mathbf{z}_t$ at optimization step t to the multi-layer perceptron optimizer with output heads $s_{\boldsymbol{\vartheta}}^{\text{adam}}$, $s_{\boldsymbol{\vartheta}}^{\text{bb}}$ and $d_{\boldsymbol{\vartheta}}^{\text{bb}}$ are chosen as follows:

1. The weights $\boldsymbol{\theta}_t$;
2. The gradients $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t)$;
3. The second momentum accumulators $\mathbf{v}_t$ with decay rates $\beta_2 \in \{0.5, 0.9, 0.99, 0.999\}$;
4. One over the square root of the above four second momentum accumulators;
5. The time step t.

Here, we build upon the extensive study carried out in Metz et al. (2022), with the above input parameters heuristically being found to perform well for the physics-informed neural networks that were trained in this work.

All input features (except the time step) were normalized to have a second moment of one. The time step is converted into a total of 11 features by computing $\tanh(t/x)$ where $x \in \{1, 3, 10, 30, 100, 300, 1000, 3000, 10k, 30k, 100k\}$. All features were then concatenated and passed through a standard multi-layer perceptron to yield the above three output heads.

It is also interesting to point out that the work by Choi et al. (2019) has shown that upon the right choice of hyper-parameters, a more general optimizer including another optimizer as a specific case, should never perform worse than this specific optimizer. Since for the choice of $\lambda_1 = 1$, $\lambda_2 = \lambda_3 = 0$ the learnable optimizer reduces to standard Adam as a specific case, upon the right tuning of parameters of the learnable optimizer (which is done with a persistent evolutionary strategy here, see the following Section 5 for a further discussion), the learnable optimizer should always outperform Adam. We show below that this is indeed the case.

5. Numerical results

In this section we showcase the use of meta-learned optimization for solving some well-known differential equations from mathematical physics, that have been extensively studied using physics-informed neural networks. In all of the following examples we use the vanilla version of physics-informed neural networks as laid out in Lagaris et al. (1998); Raissi et al. (2019). As discussed in Section 3, it is well-understood by now that this formulation can suffer from several drawbacks which to remedy is currently an active research field. As such, the goal of this section is not to obtain the best possible numerical solution for each given model, but to show how meta-learned optimization can improve the results obtainable using vanilla physics-informed neural network when compared to using standard optimization. Our base optimizer we compare against is the Adam optimizer, the de-facto standard being used in the field of physics-informed neural networks today. In Section 6, when investigating the transfer learning capabilities of learnable optimizers, we then will show a more sophisticated training strategy for physics-informed neural networks.

In all the examples below, the output heads of the meta-learned optimizer were initialized using a normal distribution with zero mean and variance of 10^{-3}, to guarantee that the neural network output is close to zero at the beginning of meta-training of the optimizer.

Due to the form of the meta-learned optimizer (4), this means that before meta-training starts, the meta-learned optimizer is very close to the standard Adam optimizer.

For all examples, the multi-layer perceptron being used for the meta-learned optimizer has two hidden layers with 32 units each, using the *swish* activation function. This architecture was found using hyperparameter tuning to give a good balance between computational overhead of meta-training the optimizer and error level of the resulting optimizer. We should like to note here that in contrast to the application of meta-learned optimization in areas of modern deep learning, such as computer vision or natural language processing, which work with neural networks with up to hundreds of hidden layers and billions of weights, the neural networks arising in physics-informed neural networks are typically relatively small. In fact, all of the architectures considered in this paper have less than 10,000 trainable parameters. This allows for larger neural networks being used for the meta-learned optimizer, without incurring computationally infeasible costs. Still, the underlying multi-layer perceptron of the meta-learned optimizer is relatively small, having only 2,115 trainable parameters.

We train this optimizer using the *persistent evolutionary strategy*, a zeroth-order stochastic optimization method described in Vicol et al. (2021). This algorithm has several hyperparameters, including the total number of particles N used for gradient computation, the partial unroll length K of the inner optimization problem before a meta-gradient update is computed, the standard deviation of perturbations σ and the learning rate α for the meta-learned weight update. Using hyperparamter tuning, we determined $N = 2$ (antithetic) particles, $K = 1$ epochs and a learning rate of $\alpha = 10^{-4}$ to be the best hyperparameters for our problem, but it would be interesting to carry out a more in-depth study on the hyperparameters of this algorithm. For more details, see Algorithm 2 in Vicol et al. (2021).

For each problem, unless otherwise specified, we then sample a total of 20 different tasks and meta-train the learnable optimizer for a total of 50 epochs on the associated tasks. Each task corresponds to a new instantiation of the particular neural network architecture, where we also slightly perturb the parameters for each equation during meta-training, which we found useful in aiding the generalization capabilities of the learned optimizers. Specifically, for each task we sample uniformly randomly $c \in [0.9, 1.1]$ for the advection velocity in the linear advection equation $\nu \in [0, 0.1]$ for the dispersion coefficient in the KdV equation and $\mu \in [0, 0.01]$ for the diffusion coefficient for Burgers' equation. We found empirically that training the meta-learned optimizer for relatively few epochs (50 epochs compared to using the learned optimizer for more than 1,000 epochs at testing stage) provided a good balance between performance and meta-training cost. We also note that due to the stochastic nature of the training algorithm and task sampling strategies, the learnable optimizer may at times converge to a suboptimal solution. If this occurs, we simply re-train the optimizer again to be able to show the best possible results obtainable with the learnable optimizer under the chosen parameter and training regime. To guarantee a fair comparison at testing time, the initial weights and biases of all neural networks being trained with the respective optimizers are the same for any experiment.

In Table 1 we summarize the parameters of the physics-informed neural networks trained in this section. We use hyperbolic tangents as activation function for all hidden layers. We use mini-batch gradient computation with a total of 10 batches per epoch.

We report both the time series of the loss for the standard Adam optimizer and the meta-learned optimizer, and the point-wise error $e = u_{\text{nn}} - u_{\text{ref}}$, where u_{ref} is either the

	Linear adv.	Poisson	KdV	Burgers	Shallow-water adv.
# hidden layers	2	4	6	6	4
# units	20	20	20	20	20
# PDE points	10,000	10,000	10,000	10,000	100,000
# IC/BC points	100	400	100	100	10,000
# epochs	6,000	2,000	1,500	3,000	3,000

Table 1: Parameters of the physics-informed neural networks trained below.

analytical solution (if available), or a high-resolution numerical reference solution obtained from using a pseudo-spectral method for the spatial discretization and an adaptive Runge–Kutta method for time stepping using the method of lines approach, see Durran (2010).

The algorithm described here has been implemented using `TensorFlow` 2.11 and the codes will be made available on `GitHub`[1].

5.1 One-dimensional linear advection equation

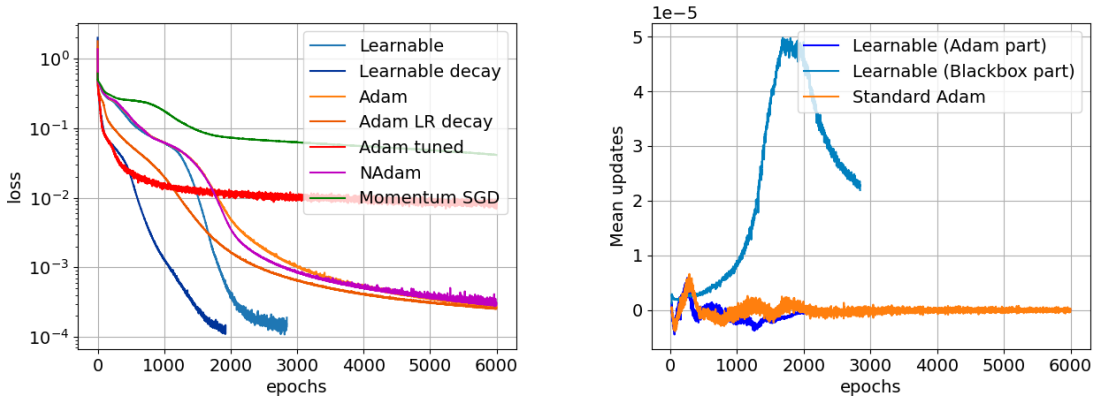
As a first example, consider the one-dimensional linear advection equation

$$u_t + cu_x = 0,$$

where we consider $t \in [0, 3]$ and $x \in [-1, 1]$ with $c = 1$ being the advection velocity. We set $u(0, x) = u_0(x) = \sin \pi x$ and use periodic boundary conditions. We enforce the periodic boundary conditions as hard constraint in the physics-informed neural networks, using the strategy introduced by Bihlo and Popovych (2022). We set $\gamma_{\mathrm{i}} = 1$ in the loss function (2a).

To establish standard Adam as a strong baseline for all subsequent problems, we test a total of 7 different optimizers for this equation, namely momentum SGD (with $\beta = 0.9$), standard Adam, Adam with square root learning rate decay, a hyper-parameter tuned version of Adam, NAdam, and the proposed learnable optimizer as well as the learnable optimizer using square root learning rate decay. The learning rate for all standard optimizers was set to $\eta = 5 \cdot 10^{-4}$. For the hyper-parameter tuned version of Adam we were using a tree-structured Parzen Estimator algorithm implemented in `optuna`, see Akiba et al. (2019). The search space for the parameters were the intervals $\beta_1, \beta_2 \in [0.1, 0.99999]$ and $\eta \in [10^{-5}, 10^{-2}]$. To keep the computational cost between the hyper-parameter tuned version of Adam and the learnable optimizer comparable, we optimize the hyper-parameter tuned version of Adam for 50 epochs over a total of 50 trials. The optimal parameters found were then $\beta_1 = 0.347$, $\beta_2 = 0.424$ and $\eta = 7.65 \cdot 10^{-3}$. The constants of the learnable optimizer were chosen as $\lambda_1 = 5 \cdot 10^{-4}$ and $\lambda_i = 10^{-3}, i = 2, \dots, 4$. We then use each optimizer to train a physics-informed neural network for the linear advection equation until the learnable optimizers have converged, which is after about 1,900 and 2,800 epochs for the learnable optimizer with and without learning rate decay, respectively. To provide a comparison for the best obtainable numerical solutions for all optimizers, we in addition continue training the physics-informed neural networks by all other optimizers for a total of 6,000 epochs, whence they have converged to the point that any further improvements would require an excessive amount of additional training steps.

1. `https://github.com/abihlo/LearnableOptimizationPinns`



(a) Training loss for all optimizers.

(b) Mean optimizer step updates.

Figure 1: Time series of losses for the standard and meta-learned optimizers (*left*), and mean update step sizes for the meta-learned optimizer and the standard Adam optimizer with fixed learning rates (*right*), as used for the linear advection equation. See also Table 2.

The numerical results for this example are depicted in Figures 1 and 2. For this particular example, the meta-learned optimizers considerably outperform all baseline standard optimizers, resulting in a training loss and point-wise error that is more than 10 times smaller by the time they have converged to their final solutions. Both learnable optimizers still outperform all standard optimizers by the time they have obtained their final solutions. Adding in a learning-rate decay also shows that the performance of the learnable optimizer can be further boosted, to the point that it achieves an error of the same level as the learnable optimizer with fixed learning rate after less than 2,000 epochs. This situation somewhat parallels the case of Adam with learning rate decay, which does converge slightly faster than standard Adam with fixed learning rate.

From Fig. 1a it is also interesting to note that the hyper-parameter tuned version of Adam has a lower loss than all other optimizers for the first several hundred epochs, but flattens out at a significantly higher loss level than the learnable optimizers. This is noteworthy, as both the hyper-parameter tuned version of Adam and the learnable optimizer have been trained/optimized for the same number of epochs. This result illustrates that the learnable optimizer used here is more than just an optimally hyper-parameter tuned version of the standard Adam optimizer.

Fig. 2 and Table 2 then illustrate that these substantially different loss values also result in quite different numerical solutions. Notably, even after 6,000 epochs the numerical errors obtained by the Adam optimizer family is still higher than the numerical error obtained by the learnable optimizers in less than half the number of training steps. Momentum SGD stalls at a loss of around 10^{-1}, much worse than all other optimizers.

Fig. 1b depicts the mean gradient step updates corresponding to the standard Adam optimizer, the scaled Adam part of the learnable optimizer (the first term in Eqn. (4)) and the blackbox term of the learnable optimizer (the second term in Eqn. (4)). For the first

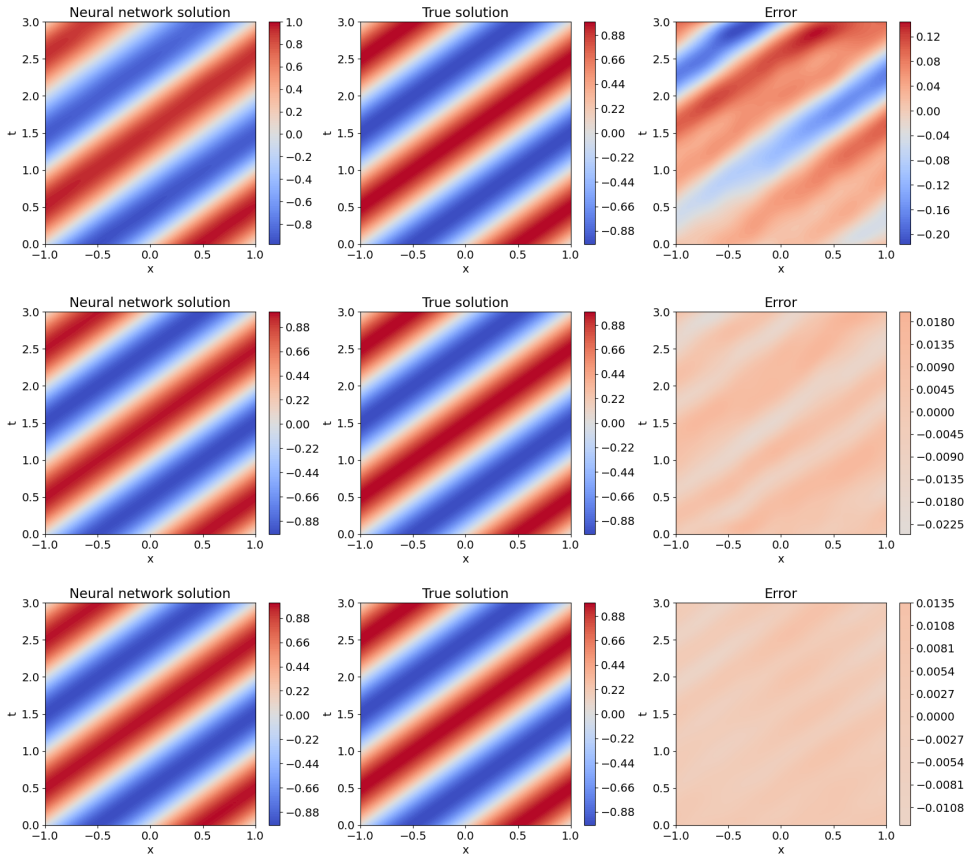


Figure 2: Numerical results for the linear advection equation. *Top row:* Standard Adam optimizer after 2,845 epochs, the number of training steps required for the learnable optimizer to converge. *Middle row:* Adam after 6,000 epochs, the number of training steps required for Adam to converge. *Bottom row:* Meta-learned optimizer. Left to right shows the numerical solution obtained from the physics-informed neural networks, the exact solution, and the difference between the numerical solution and the exact solution.

500 epochs, the Adam part of the learnable optimizer closely follows the standard Adam optimizer, but once the blackbox term starts to dominate the overall gradient step update, also the Adam part of the learnable optimizer behaves differently from the standard Adam optimizer, indicating that these optimizers indeed follow different paths to their respective minima. It is also noteworthy that the rather steep increase in the blackbox part after 1,000 epochs followed by an equally steep decrease after 2,000 epochs corresponds directly to the substantially better performance of the learnable optimizer compared to the Adam optimizer in this period of the training regime.

A quantitative evaluation of the numerical solutions themselves is given in Table 2. This table shows that the two learnable optimizers with fixed and learning rate decay, respectively, give the best numerical solutions among all optimizers tested. This table also

Epochs	L2O	L2O decay	Adam	Adam decay	Adam tuned	NAdam	SGDM
1,925	$1.33 \cdot 10^{-4}$	$\mathbf{1.33 \cdot 10^{-5}}$	$1.16 \cdot 10^{-1}$	$1.98 \cdot 10^{-3}$	$5.79 \cdot 10^{-3}$	$1.60 \cdot 10^{-1}$	$3.09 \cdot 10^{-1}$
2,845	$3.58 \cdot 10^{-5}$	—	$3.54 \cdot 10^{-4}$	$2.69 \cdot 10^{-4}$	$3.91 \cdot 10^{-3}$	$2.94 \cdot 10^{-4}$	$2.48 \cdot 10^{-1}$
6,000	—	—	$7.80 \cdot 10^{-5}$	$7.68 \cdot 10^{-5}$	$3.41 \cdot 10^{-3}$	$1.57 \cdot 10^{-4}$	$1.48 \cdot 10^{-1}$

Table 2: Numerical results for the linear advection equation. Shown are the l_2-errors of the obtained solutions for three experiments: i) After training with all optimizers for 1,925 epochs, i.e. when the learnable optimizer with learning rate decay (L2O decay) has converged. ii) After training with all but the L2O decay optimizer for 2,845 epochs, i.e. when the learnable optimizer with fixed learning rate (L2O) has converged. iii) After training with all standard optimizers for 6,000 epochs, i.e. when the comparison optimizers have converged as well.

shows that the standard Adam optimizers with or without learning rate decay give the best numerical results among all standard optimizers tested, provided they have been given more than twice the number of optimization steps than the learnable optimizers.

In view of these results, in the following examples we will only show the comparison between standard Adam, the optimizer of choice for virtually all physics-informed neural networks today, and the learnable optimizer proposed above although the results obtained for the linear advection equation do hint at further improvements that are possible with suitable learning rate decay strategies.

5.2 Poisson equation

As an example for a boundary-value problem, consider the two-dimensional Poisson equation

$$u_{xx} + u_{yy} = f(x, y),$$

over the domain $\Omega = [-1, 1] \times [-1, 1]$ for the exact solution

$$u_{\text{exact}}(x, y) = (0.1 \sin 2\pi x + \tanh 10x) \sin 2\pi y,$$

with the associated right-hand side using Dirichlet boundary conditions. This problem was considered in Kharazmi et al. (2021). Since this is a boundary value problem, there is no initial loss in the loss function (2a) and we use $\gamma_{\text{b}} = 1000$. This value was chosen heuristically to balance the differential equation and boundary value losses. The learning rate for Adam for this example was set to $\eta = 5 \cdot 10^{-4}$ and the constants of the meta-learned optimizer, were $\lambda_1 = 5 \cdot 10^{-4}$ and $\lambda_i = 10^{-3}$, $i = 2, \ldots, 4$.

The training loss for this example is shown in Fig. 3, the numerical results as compared to the exact solution with the associated point-wise error are depicted in Fig. 4. As with the linear advection equation from the previous example, also for the Poisson equation the meta-learned optimization method leads to better results both in terms of a lower training loss and smaller point-wise errors compared to the standard Adam optimizer.

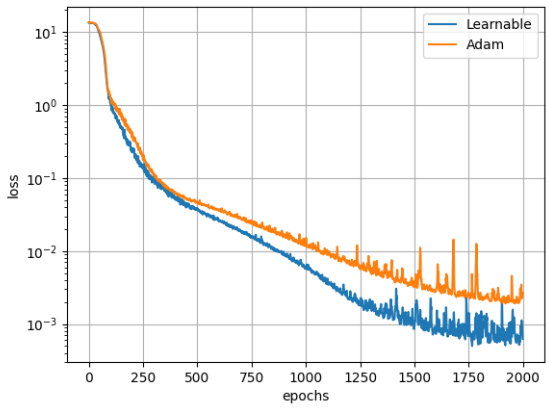


Figure 3: Training loss for the Adam and meta-learned optimizers for the two-dimensional Poisson equation.

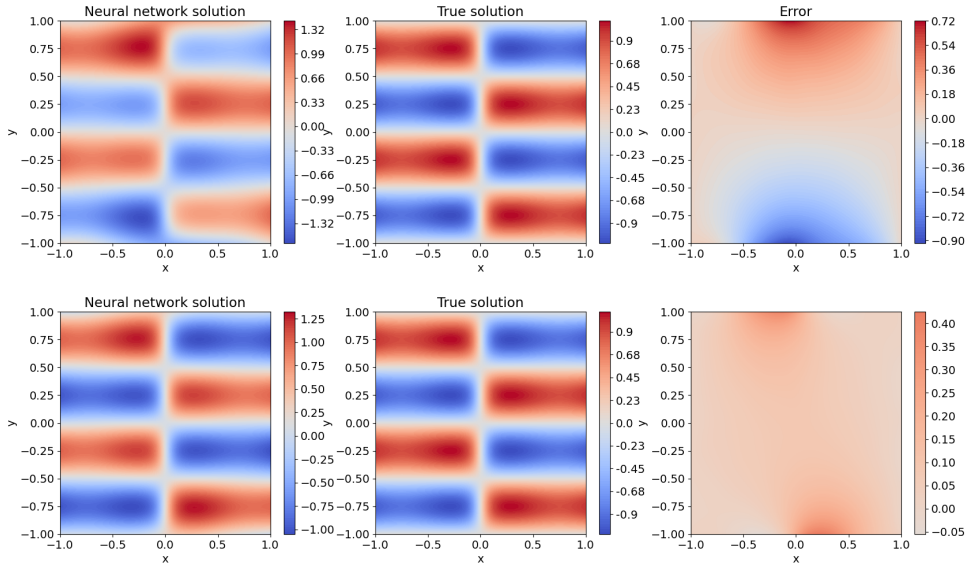


Figure 4: Numerical results for the Poisson equation. *Top row:* Standard Adam optimizer. *Bottom row:* Meta-learned optimizer. Left to right shows the numerical solution obtained from the physics-informed neural networks, the exact solution, and the difference between the numerical solution and the exact solution.

5.3 Korteweg–de Vries equation

We next consider the Korteweg–de Vries equation

$$u_t + uu_x - \nu u_{xxx} = 0,$$

with initial condition $u(0, x) = \cos \pi x$ using periodic boundary conditions over the domain $x \in [-1, 1]$ and $t \in [0, 1]$, setting $\nu = 0.0025$. This equation has been extensively studied using physics-informed neural networks, see e.g. Jagtap et al. (2020); Raissi et al. (2019). Again, we enforce the periodic boundary conditions as hard constraint and set $\gamma_{\mathrm{i}} = 1$ in the loss function (2a). The learning rate of the Adam optimizer was chosen as $\eta = 5 \cdot 10^{-4}$, and the constants of the meta-learned optimizer were set to $\lambda_1 = 5 \cdot 10^{-4}$ and $\lambda_i = 10^{-3}$, $i = 2, \ldots, 4$.

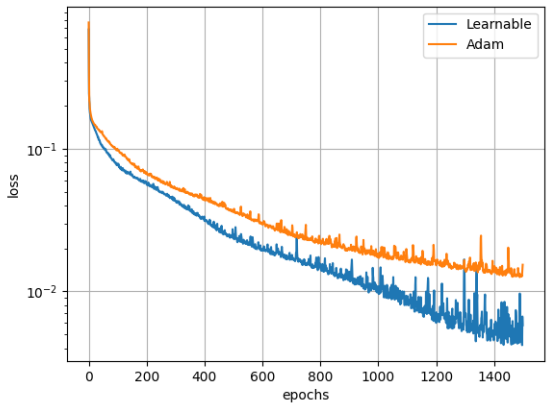


Figure 5: Training loss for the Adam and meta-learned optimizers for the Korteweg–de Vries equation.

Figure 5 contains the respective training losses of the Adam and meta-learned optimizers. The numerical solutions for the associated trained physics-informed neural networks as compared against the numerical solution obtained from a pseudo-spectral numerical integration method are featured in Figure 6. These plots again illustrate that the meta-learned optimizer reduces the training loss considerably faster than the standard Adam optimizer, which also improves upon the point-wise error of the numerical solution compared to the reference solution. In fact, the training loss after about 1,100 epochs is lower for the meta-learned optimizer than what the Adam optimizer achieves at the end of training.

5.4 Burgers' equation

As our next example we consider Burgers' equation

$$u_t + uu_x - \mu u_{xx} = 0,$$

over the temporal-spatial domain $[0, 1] \times [-1, 1]$ with initial condition $u(0, x) = -\sin \pi x$ and periodic boundary conditions in x-direction. The diffusion parameter was set as $\mu = 0.01/\pi$. Burgers equation is also one of the most prominent examples considered using physics-informed neural networks, see Raissi et al. (2019) for some results.

As for the Korteweg–de Vries equation, we enforce the periodic boundary conditions as hard constraints, use $\gamma_{\mathrm{i}} = 1$ in the loss function (2a), and set the learning rate of

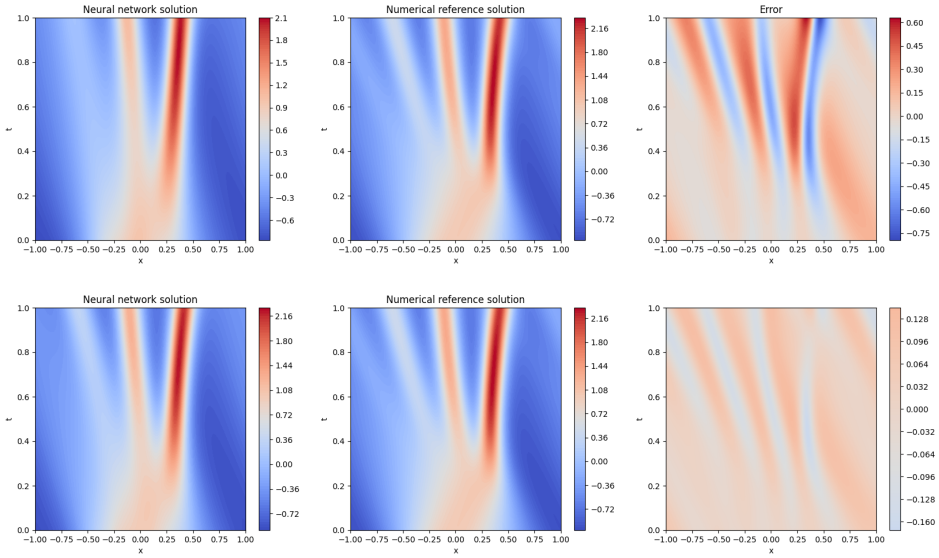


Figure 6: Numerical results for the Kortweg–de Vries equation. *Top row:* Standard Adam optimizer. *Bottom row:* Meta-learned optimizer. Left to right shows the numerical solution obtained from the physics-informed neural networks, the numerical reference solution, and the difference between the numerical solution and the reference solution.

the Adam optimizer to $\eta = 10^{-4}$, and the constants of the meta-learned optimizer to $\lambda_1 = 10^{-4}$ and $\lambda_i = 10^{-3}$, $i = 2, \ldots, 4$. The meta-learned optimizer is being trained as for the previous examples, i.e. using Burgers' equation on 20 tasks, which each task being a newly instantiated neural network with different random initial weights.

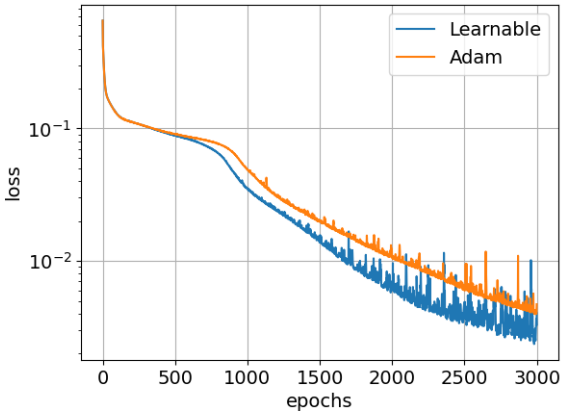


Figure 7: Training loss for the Adam and meta-learned optimizers for Burgers' equation.

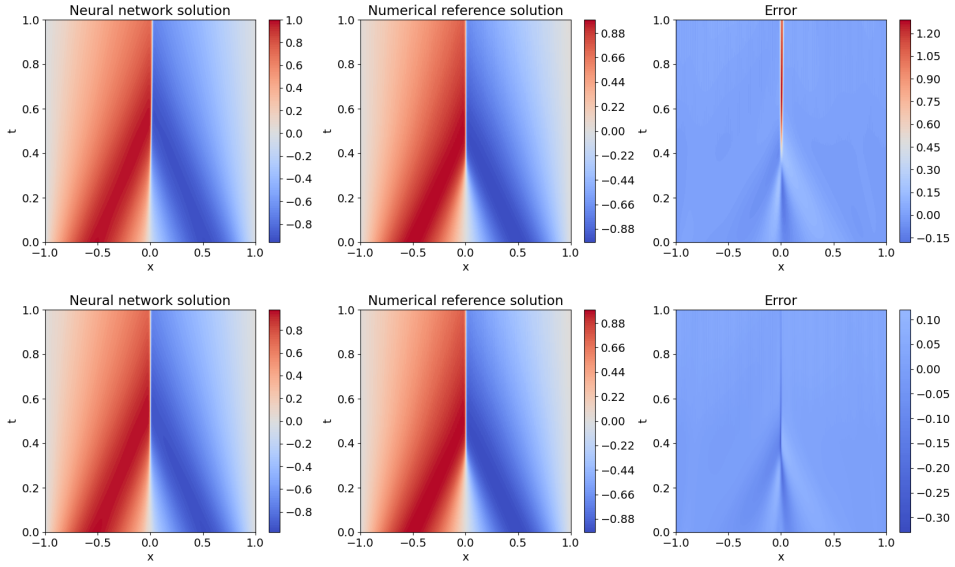


Figure 8: Numerical results for Burgers' equation. *Top row:* Standard Adam optimizer. *Bottom row:* Meta-learned optimizer using Burgers equation. Left to right shows the numerical solution obtained from the physics-informed neural networks, a numerical reference solution, and the difference between the numerical solution and the reference solution.

Figures 7 and 8 contain the associated numerical results for this example, showing the training loss of the respective optimizers and the actual numerical results for solving Burgers' equation using the trained neural networks. Figures 7 illustrates that the meta-learned optimizer again outperforms the Adam optimizer with the final loss value of the Adam optimizer being reached already after 2,200 epochs by the meta-learned optimizer.

The loss values are also consistent with the numerical results shown in Fig. 8, illustrating that the meta-learned optimizer using Burgers' equation leads to a much more accurate solution in the vicinity of the developing shock, which is failed to be captured by the standard Adam optimizer.

5.5 Linear advection on the sphere

The last example we consider here is the shallow-water advection on a sphere. This is one of the standard test cases from Williamson et al. (1992) for numerical methods for the shallow-water equations on the sphere. Specifically, the equation to solve reads:

$$h_t + \frac{u}{a\cos\theta} h_\lambda + \frac{v}{a} h_\theta = 0,$$

where $h = h(t, \lambda, \theta)$ is the height field in spherical geometry with longitude $\lambda \in [-\pi, \pi]$ and latitude $\theta \in [-\pi/2, \pi/2]$, u and v are prescribed velocity fields of the form

$$u = U(\cos\theta\cos\alpha + \sin\theta\cos\lambda\sin\alpha),$$
$$v = -U\sin\lambda\sin\alpha,$$

and a is the radius of Earth. The initial condition for h is chosen as a cosine bell of the form

$$h(0, \lambda, \theta) = \begin{cases} H(1 + \cos(\pi r/R))/2 & \text{if} \quad r < R \\ 0 & \text{if} \quad r \geqslant R, \end{cases}$$

where $r = a\arccos(\sin\theta_c\sin\theta + \cos\theta_c\cos\theta\cos(\lambda - \lambda_c))$. We choose the constants as $U = 2\pi a/12$ m $\cdot$ days^{-1}, $H = 1000$ m and $R = a/3$. The centre of the bell is placed at $(\lambda_c, \theta_c) = (\pi/2, 0)$. We set $\alpha = 0$ making the bell traverse alone the equator. The boundary conditions on the sphere are enforced using hard constraints, as discussed in Bihlo and Popovych (2022), with the equations being solved for $t \in [0, 2]$ days. The learning rate for the Adam optimizer was set to $\eta = 10^{-3}$ and the constants of the meta-learned optimizer are all $\lambda_i = 10^{-3}, i = 1, \dots, 4$. The meta-learned optimizer is trained for 100 tasks for a total of 20 steps, with each task being a newly instantiated neural network.

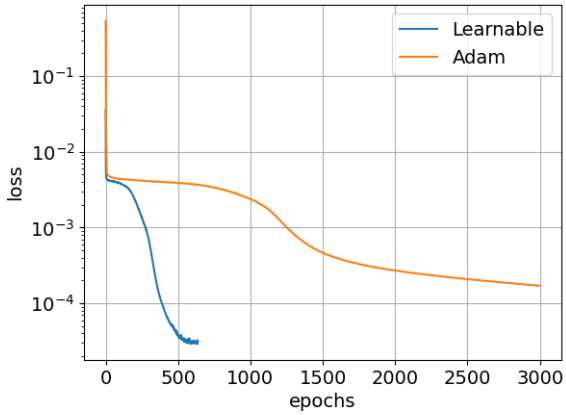


Figure 9: Training loss for the Adam and meta-learned optimizers for the shallow-water advection equation on the sphere.

The numerical results for this test case are summarized in Figs. 9 and 10. As in the previous examples, the meta-learned optimizer again drastically outperforms the standard Adam optimizer. Here, the minimum is reached after about 600 epochs, by which time the Adam optimizer is still stuck in a plateau on the loss surface. Thus, to provide a more reasonable comparison we train the comparison physics-informed neural network with the Adam optimizer for 3,000 epochs instead, by which time the loss is starting to level out. Still, it is evident from Fig. 10 that the meta-learned optimizer still outperforms Adam despite the latter having had roughly 5 time more compute on the evaluation problem.

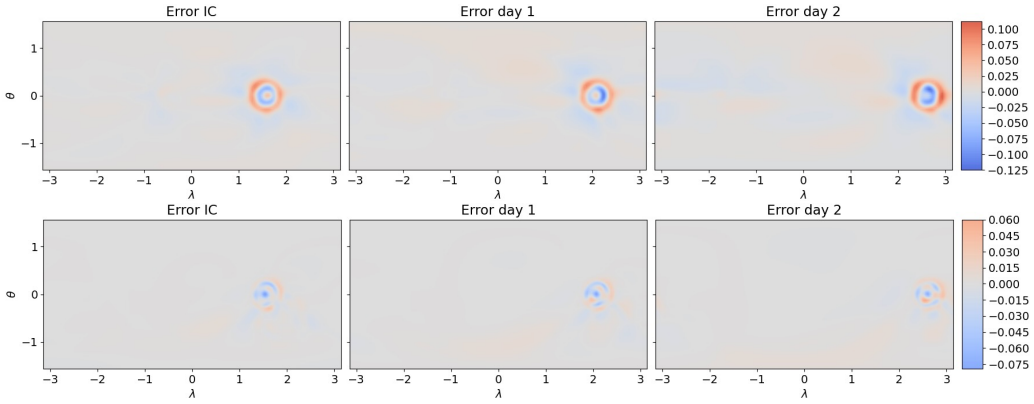


Figure 10: Numerical results for the shallow-water advection equation on the sphere. *Top row:* Standard Adam optimizer after 3,000 epochs of training. *Bottom row:* Meta-learned optimizer after 600 epochs of training. Left to right shows the error of the numerical solution obtained from the physics-informed neural networks at days zero (initial condition), one and two, when compared to the exact reference solution.

6. Transfer learning

The previous section illustrated the potential of meta-learnable optimization to outperform standard optimizers for training physics-informed neural networks. We should like to remind here that physics-informed neural networks follow a different ideology from traditional machine learning, in that these networks are not trained with generalization capabilities in mind. Physics-informed neural networks are meant to be solution interpolants, whereas in traditional machine learning, extrapolation (from the training to the testing dataset) is the main goal of learning.

Still, having accomplished this goal, in this section we ask the following wider question: Do meta-learnable optimizers for physics-informed neural networks generalize across similar tasks or do they only overfit the single task they were trained on? While it is non-trivial to properly define the notion of similarity for partial differential equations in this context, in this section we provide a first investigation into the transfer learning capabilities of meta-learned optimization for physics-informed neural networks.

6.1 Transfer learning across similar tasks: Different initial conditions

A common task in the numerical solution of differential equations is to change the initial condition of a given system of equations. For standard physics-informed neural networks this requires re-training of the network, which is computationally costly. We thus investigate here the transfer learning abilities of meta-learnable optimizers that have been trained on an ensemble of initial conditions for one fixed differential equation.

For the sake of illustration, we choose the Korteweg–de Vries equation here. More specifically, we sample our task distribution for meta-training the optimizer from an ensemble of initial conditions here, where for the sake of simplicity we consider initial conditions of the

form

$$u(0,x) = \cos(kx + \phi),$$

where k is sampled from integers between 1 and 3 and ϕ is sampled uniformly from $[-\pi/2, \pi/2]$. We choose a relatively narrow task distribution to speed up meta-learning. Once trained, we evaluate the optimizer on the unseen test problem with $k = 2$ and $\phi = -\pi/4$. Since this is a harder problem than using the meta-learned optimizer on the same problem (i.e. same initial condition and same differential equation), we meta-train the optimizer on a total of 50 tasks here instead of the 20 tasks used so far.

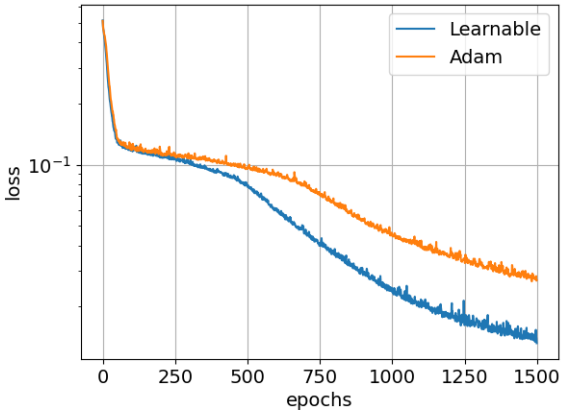


Figure 11: Training loss for the Adam and meta-learned optimizers for the Korteweg–de Vries equation using transfer learning.

The results of this experiment are depicted in Figures 11 and 12. These figures again show improvement of the meta-learned optimizer when compared to the results obtained using Adam. The meta-learned optimizer is able to generalize to the unseen test problem, by reaching both lower loss values and a smaller overall point-wise error in the solution of the KdV equation. This example demonstrates that transfer learning across the same equation class, i.e. choosing different initial values but keeping the equation the same, is indeed feasible.

6.2 Transfer learning across similar tasks: Longer time integrations

In this paper we have focussed exclusively on improving vanilla physics-informed neural networks using learnable optimization. However, as was reviewed in Section 3, numerous modified training methodologies were put forth in the literature that aim to improve some of the shortcomings of vanilla physics-informed neural networks. Among these strategies, we focus on the multi-model approach here: Rather than training a single neural network for the entire spatio-temporal domain, we split the domain into smaller sub-domains and train one neural network for each sub-domain, taking into account the interface conditions between sub-domains in a suitable manner. As we focus exclusively on evolution equations here, we consider the splitting into temporal slices only, and do not consider spatial domain

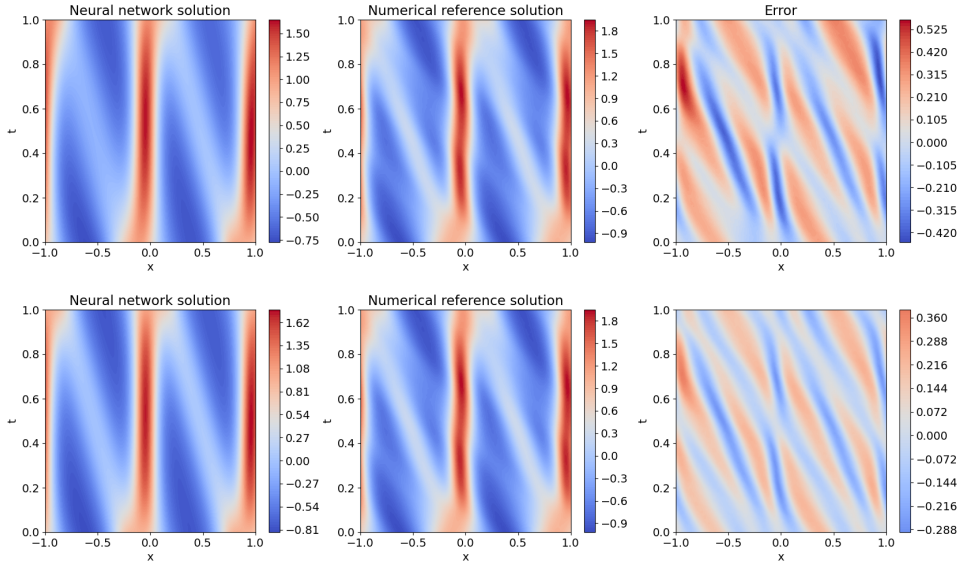


Figure 12: Numerical results for the Kortweg–de Vries equation using transfer learning. *Top row:* Standard Adam optimizer. *Bottom row:* Meta-learned optimizer. Left to right shows the numerical solution obtained from the physics-informed neural networks, the numerical reference solution, and the difference between the numerical solution and the reference solution.

decomposition here. The initial conditions for later neural networks are then derived from the final solutions of earlier neural networks, with training proceeding in a sequential fashion. For more details on this approach, see e.g. Bihlo and Popovych (2022).

In particular, we consider solving the linear advection equation for twice the size of the temporal domain considered in Section 5.1, i.e. here $t \in [0, 6]$, and we use two sub-models trained for $t \in [0, 3]$ and $t \in [3, 6]$, respectively, to patch together the solution over the entire domain. Notably, the meta-learned optimizer is trained only for the first sub-domain $t \in [0, 3]$, and then applied for both sub-domains $t \in [0, 3]$ and $t \in [3, 6]$, i.e. the optimizer has never seen any solution on the temporal interval $t \in [3, 6]$.

The numerical results of this investigation are depicted in Figures 13 and 14. These results illustrate that the optimizer trained for one particular solution (or task), the solution of the linear advection equation over the interval $t \in [0, 3]$, generalizes to another particular solution (or task), the solution of the same equation over the interval $t \in [3, 6]$. This is especially useful as the multi-model approach is currently one of the best possibilities of improving numerical solutions obtainable using physics-informed neural networks, as training many neural networks over shorter time intervals generally leads to a more accurate solution than training a single neural network over a longer time interval.

In Fig. 13 it is also interesting to note that the Adam optimizer has lower loss for the second model during the first 1,200 epochs of training. This is, however, not a reflection of the standard optimizer outperforming the learnable one in this first part of training the second model; rather, it is a reflection that failure to learn the solution over the initial

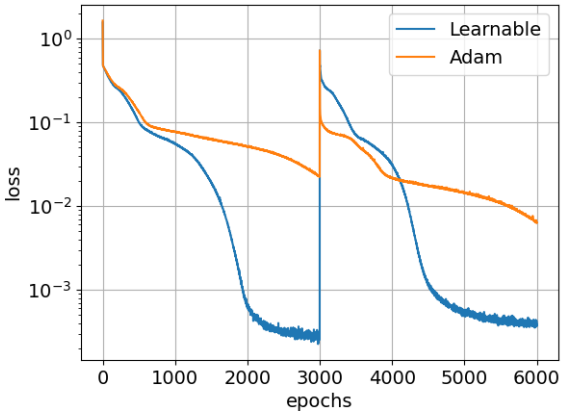


Figure 13: Training loss for the Adam and meta-learned optimizers for solving the linear advection equation over the temporal domain $t \in [0, 6]$ using the multi-model approach with two sub-models. We train the meta-learned optimizer for the temporal domain $t \in [0, 3]$ and then apply the learned optimizer for training a physics-informed neural network for the initial temporal domain $t \in [0, 3]$ and a second network for the subsequent temporal domain $t \in [3, 6]$.

interval adequately results in a simpler solution that will be used as an initial condition at time $t = 3$ for the second model.

6.3 Transfer learning across different tasks

As the last investigation into transfer learning capabilities of learnable optimizers, we investigate the capabilities of meta-learned optimizers across different tasks. Specifically, we investigate the abilities of a meta-learned optimizer trained on one particular class of partial differential equations to generalize across different partial differential equations.

For this, we meta-train a new learnable optimizer on each of the evolutionary equations from Section 5 and then apply it to all the evolutionary equations presented in that section. We exclude the Poisson equation from this study as this is a boundary-value problem and as such requires a different training setup from the other equations from Section 5. Specifically, the loss function to be minimized for the initial value problems is $\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_\Delta(\boldsymbol{\theta}) + \gamma_\mathrm{i}\mathcal{L}_\mathrm{i}(\boldsymbol{\theta})$ while for the Poisson equation it is $\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_\Delta(\boldsymbol{\theta}) + \gamma_\mathrm{b}\mathcal{L}_\mathrm{b}(\boldsymbol{\theta})$.

For all experiments, we use neural networks with 6 hidden layers and 20 units per layer. A learning rate of $\eta = 2 \cdot 10^{-4}$ was used, and the constants of the meta-learned optimizer were set to $\lambda_1 = 2 \cdot 10^{-4}$ and $\lambda_i = 10^{-3}$, $i = 2, \dots, 4$. Each optimizer is trained and applied five times, with mean values and standard deviations of the errors reported in Table 3. These errors are the final l_2-errors of the numerical solution obtained after 1,000 epochs of training, as compared to the reference solutions for these problems, which were the same as in Section 5.

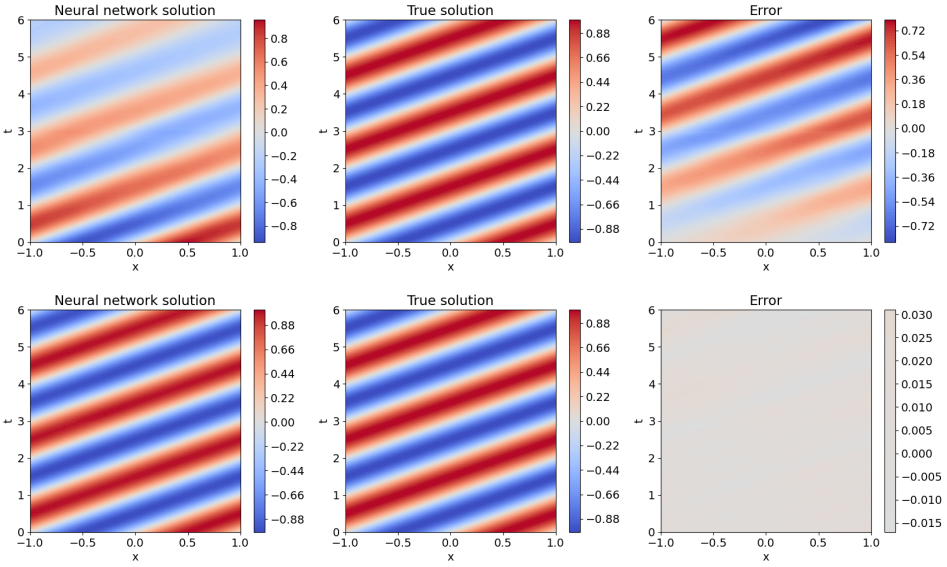


Figure 14: Numerical results for the Adam and meta-learned optimizers for solving the linear advection equation over the temporal domain $t \in [0, 6]$ using the multi-model approach with two sub-models.

Evaluated for → Trained on ↓	LA	KdV	Burgers
LA	$\mathbf{4.01 \cdot 10^{-3} \pm 4.52 \cdot 10^{-3}}$	$5.52 \cdot 10^{-2} \pm 1.52 \cdot 10^{-2}$	$2.48 \cdot 10^{-3} \pm 1.88 \cdot 10^{-3}$
KdV	$7.36 \cdot 10^{-2} \pm 6.34 \cdot 10^{-2}$	$\mathbf{5.48 \cdot 10^{-2} \pm 3.14 \cdot 10^{-2}}$	$7.45 \cdot 10^{-3} \pm 4.67 \cdot 10^{-3}$
Burgers	$5.86 \cdot 10^{-2} \pm 5.28 \cdot 10^{-2}$	$5.81 \cdot 10^{-2} \pm 3.65 \cdot 10^{-2}$	$\mathbf{2.26 \cdot 10^{-3} \pm 9.71 \cdot 10^{-4}}$
Adam	$1.39 \cdot 10^{-1} \pm 3.48 \cdot 10^{-2}$	$1.14 \cdot 10^{-1} \pm 1.66 \cdot 10^{-2}$	$5.91 \cdot 10^{-3} \pm 2.55 \cdot 10^{-3}$

Table 3: Transfer learning abilities for the meta-learned optimizers across different tasks. Shown are the l_2-errors for meta-learned optimizers that have been trained on one particular class of evolution equations from Section 5 and then being used for all the evolution equations from Section 5.

Similar as shown in Section 5, Table 3 again shows that each meta-learned optimizer outperforms Adam if it was trained and evaluated on the same equation. Interestingly, with the exception of the KdV-trained learnable optimizer evaluated for Burgers' equation, all meta-learned optimizers trained on one equation but evaluated for a different equation outperform the standard Adam optimizer for that different equation, hinting at these optimizers being able to learn transferable features beyond the class of equations they have been trained on. Note also that the optimizer trained for the linear advection equation performs close to optimal for all models. This suggest that meta-learning an optimizer on easier equations (such as the linear advection equation), and then applying them to more complicated equations is preferable compared to training them on more complicated equa-

tions (such as Burgers' or the KdV equation) and then applying them to simpler equations. This is physically reasonable, as both Burgers' equation and the KdV equation have some characteristics of the simple advection equation, which the meta-learned optimizer trained on the linear advection equation appears to be able to leverage when applied to these more complicated models. Conversely, the optimizers trained on the more complicated equations seem to learn the specifics of those models, which then do not readily generalize to other models.

7. Conclusion

We have investigated the use of meta-learned optimization for improving the training of physics-informed neural networks in this work. Meta-learned optimization, or learning-to-learn, has become an increasingly popular topic in deep learning and thus it is natural to investigate its applicability to scientific machine learning as well. We have done so here by illustrating that meta-learned optimization can be used to improve the numerical results obtainable using physics-informed neural networks, which is a popular machine learning-based method for solving differential equations. We have also shown that meta-learned optimization exhibits transfer learning capabilities that could be leveraged to further speed up training of physics-informed neural networks by allowing learned optimizers to be reused for different classes of equations.

The goal of this paper was to illustrate that meta-learned optimization alone can substantially improve the vanilla form of physics-informed neural networks, which was laid out in the seminal works by Lagaris et al. (1998) and Raissi et al. (2019). This form has been extensively studied, and we have shown here that meta-learned optimization can give (sometimes substantially) better numerical results compared to standard hand-crafted optimization rules. This means that meta-learned optimizers are able to reach a particular error level quicker than standard optimizers, resulting in either shorter training times (for a given target computational error) or better numerical accuracy (for the same number of training epochs). It is also possible that these learnable optimizers reach a minimum not achievable with a standard optimizer.

There are several avenues for future research that would provide natural extensions to the present work. Firstly, one could investigate the use of meta-learned optimization for other formulations of physics-informed neural networks. Here we have shown that meta-learned optimization improves vanilla physics-informed neural networks, and also works for temporal domain decomposition approaches. The latter is a main remedy for training issues encountered in physics-informed neural networks. However, there is also a zoo of other variations to the main physics-informed neural network approach that are applicable to different classes of differential equations. This list of methods includes, to name a few, variational formulations, see e.g. Kharazmi et al. (2021), formulations based on spatio-temporal domain decompositions, see e.g. Jagtap et al. (2020), techniques based on improved loss functions, such as shown by Psaros et al. (2022); Wang et al. (2022), re-sampling strategies as introduced by Wu et al. (2023), and operator-based formulations as described in Wang and Perdikaris (2023). It should also be stressed that not all of these methods substantially outperform vanilla physics-informed networks, and the latter are still being used extensively in the literature today, see e.g. Cuomo et al. (2022) for a recent review.

Secondly, there is a multitude of other meta-learned optimization algorithms based on neural networks that have been proposed in the literature, see the review paper by Chen et al. (2022) for an extensive list of such optimizers. There are also several training strategies available for meta-learned optimization, including gradient descent, evolutionary strategies and reinforcement learning based ones, see again Chen et al. (2022). Together, this provides a rich set of training strategies, meta-learnable optimizer architectures and physics-informed model formulations that could be explored together to possibly find more accurate solutions of differential equations using physics-informed neural networks. We plan to explore some of these possibilities in the near future.

Acknowledgments

This research was undertaken thanks to funding from the Canada Research Chairs program and the NSERC Discovery Grant program. The author thanks the three anonymous referees for their valuable suggestions in the course of the review process.

References

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

T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29, 2016.

A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *J. Mach. Learn. Res.*, 18:Paper No. 153, 2018.

S. Bengio, Y. Bengio, and J. Cloutier. On the search for new learning rules for ANNs. *Neural Process. Lett.*, 2(4):26–30, 1995.

Y. Bengio, S. Bengio, and J. Cloutier. *Learning a synaptic learning rule*. Université de Montréal, 1990.

A. Bihlo and R. O. Popovych. Physics-informed neural networks for the shallow-water equations on the sphere. *J. Comput. Phys.*, 456:111024, 2022.

J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

T. Chen, X. Chen, W. Chen, H. Heaton, J. Liu, Z. Wang, and W. Yin. Learning to optimize: A primer and a benchmark. *J. Mach. Learn. Res.*, 23(189):1–59, 2022.

D. Choi, C. J. Shallue, Z. Nado, J. Lee, C. J. Maddison, and G. E. Dahl. On empirical comparisons of optimizers for deep learning. arXiv:1910.05446, 2019.

S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli. Scientific machine learning through physics–informed neural networks: where we are and what's next. *J. Sci. Comput.*, 92(3):88, 2022.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Math. Control Signals Syst.*, 2(4):303–314, 1989.

D. R. Durran. *Numerical methods for fluid dynamics: With applications to geophysics*, volume 32. Springer Science & Business Media, 2010.

J. Harrison, L. Metz, and J. Sohl-Dickstein. A closer look at learned optimization: Stability, robustness, and inductive biases. *Advances in neural information processing systems*, 35: 3758–3773, 2022.

A. D. Jagtap, E. Kharazmi, and G. E. Karniadakis. Conservative physics-informed neural networks on discrete domains for conservation laws: applications to forward and inverse problems. *Comput. Methods Appl. Mech. Eng.*, 365:113028, 2020.

X. Jin, S. Cai, H. Li, and G. E. Karniadakis. NSFnets (Navier–Stokes flow nets): Physics-informed neural networks for the incompressible Navier–Stokes equations. *J. Comput. Phys.*, 426:109951, 2021.

E. Kharazmi, Z. Zhang, and G. E. Karniadakis. hp-VPINNs: Variational physics-informed neural networks with domain decomposition. *Comput. Methods Appl. Mech. Eng.*, 374: 113547, 2021.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. arXiv:1412.6980, 2014.

I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Trans. Neural Netw.*, 9(5):987–1000, 1998.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

X. Liu, X. Zhang, W. Peng, W. Zhou, and W. Yao. A novel meta-learning initialization method for physics-informed neural networks. *Neural. Comput. Appl.*, 34(17):14511–14534, 2022.

J. Lucas, S. Sun, R. Zemel, and R. Grosse. Aggregated momentum: Stability through passive damping. In *International Conference on Learning Representations*, 2019.

K. Lv, S. Jiang, and J. Li. Learning gradient descent: Better generalization and longer horizons. In *International Conference on Machine Learning*, pages 2247–2255. PMLR, 2017.

N. Maheswaranathan, D. Sussillo, L. Metz, R. Sun, and J. Sohl-Dickstein. Reverse engineering learned optimizers reveals known and novel mechanisms. *Advances in neural information processing systems*, 34:19910–19922, 2021.

L. Metz, C. D. Freeman, J. Harrison, N. Maheswaranathan, and J. Sohl-Dickstein. Practical tradeoffs between memory, compute, and performance in learned optimizers. In *Conference on Lifelong Learning Agents*, pages 142–164. PMLR, 2022.

J. Nocedal and S. J. Wright. *Numerical optimization*. Springer, 1999.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems* 32:8024–8035, 2019.

M. Penwarden, A. D. Jagtap, S. Zhe, G. E. Karniadakis, and R. M. Kirby. A unified scalable framework for causal sweeping strategies for physics-informed neural networks (PINNs) and their temporal decompositions. arXiv:2302.14227, 2023.

A. F. Psaros, K. Kawaguchi, and G. E. Karniadakis. Meta-learning PINN loss functions. *J. Comput. Phys.*, 458:111121, 2022.

N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville. On the spectral bias of neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages 5301–5310. PMLR, 2019.

M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.*, 378:686–707, 2019.

O. Sener and V. Koltun. Multi-task learning as multi-objective optimization. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 525–536, 2018. arXiv:1810.04650.

N. Shazeer and M. Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR, 2018.

P. Vicol, L. Metz, and J. Sohl-Dickstein. Unbiased gradient estimation in unrolled computation graphs with persistent evolution strategies. In *International Conference on Machine Learning*, pages 10553–10563. PMLR, 2021.

S. Wang and P. Perdikaris. Long-time integration of parametric evolution equations with physics-informed deeponets. *J. Comput. Phys.*, 475:111855, 2023.

S. Wang, S. Sankaran, and P. Perdikaris. Respecting causality is all you need for training physics-informed neural networks. arXiv:2203.07404, 2022.

O. Wichrowska, N. Maheswaranathan, M. W. Hoffman, S. G. Colmenarejo, M. Denil, N. Freitas, and J. Sohl-Dickstein. Learned optimizers that scale and generalize. In *International conference on machine learning*, pages 3751–3760. PMLR, 2017.

D. L. Williamson, J. B. Drake, J. J. Hack, J. Rüdiger, and P. N. Swarztrauber. A standard test set for numerical approximations to the shallow water equations in spherical geometry. *J. Comput. Phys.*, 102:211–224, 1992.

C. Wu, M. Zhu, Q. Tan, Y. Kartha, and L. Lu. A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks. *Comput. Methods Appl. Mech. Eng.*, 403:115671, 2023.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. In *Advances in Neural Information Processing Systems*, 33:5824–5836, 2020