PROCEEDINGS A

royalsocietypublishing.org/journal/rspa

Research

Cite this article: Lagergren JH, Nardini JT, Lavigne GM, Rutter EM, Flores KB. 2020 Learning partial differential equations for biological transport models from noisy spatio-temporal data. *Proc. R. Soc. A* **476**: 20190800.
http://dx.doi.org/10.1098/rspa.2019.0800

Received: 18 November 2019
Accepted: 17 January 2020

Subject Areas:
mathematical modelling, differential equations, computational mathematics

Keywords:
numerical differentiation, equation learning, sparse regression, partial differential equations, parameter estimation, biological transport

Author for correspondence:
Kevin B. Flores
e-mail: kbflores@ncsu.edu

†These authors contributed equally to this study.

Electronic supplementary material is available online at https://doi.org/10.6084/m9.figshare.c.4860342.

THE ROYAL SOCIETY PUBLISHING

Learning partial differential equations for biological transport models from noisy spatio-temporal data

John H. Lagergren[1,2,†], John T. Nardini[1,3,†], G. Michael Lavigne[1,2], Erica M. Rutter[1,2,4] and Kevin B. Flores[1,2]

[1]Department of Mathematics, and [2]Center for Research in Scientific Computation, North Carolina State University, Raleigh, NC, USA
[3]The Statistical and Applied Mathematical Sciences Institute, Durham, NC, USA
[4]Department of Applied Mathematics, University of California, Merced, CA, USA

KBF, 0000-0002-4000-6971

We investigate methods for learning partial differential equation (PDE) models from spatio-temporal data under biologically realistic levels and forms of noise. Recent progress in learning PDEs from data have used sparse regression to select candidate terms from a denoised set of data, including approximated partial derivatives. We analyse the performance in using previous methods to denoise data for the task of discovering the governing system of PDEs. We also develop a novel methodology that uses artificial neural networks (ANNs) to denoise data and approximate partial derivatives. We test the methodology on three PDE models for biological transport, i.e. the advection–diffusion, classical Fisher–Kolmogorov–Petrovsky–Piskunov (Fisher–KPP) and nonlinear Fisher–KPP equations. We show that the ANN methodology outperforms previous denoising methods, including finite differences and both local and global polynomial regression splines, in the ability to accurately approximate partial derivatives and learn the correct PDE model.

1. Introduction

Recent research has investigated methods for discovering systems of differential equations that describe the underlying dynamics of spatio-temporal data. There are

key advantages to learning and then using mathematical models for prediction instead of using a purely machine learning-based method, e.g. neural networks. First, if the learned mathematical model is an accurate description of the processes governing the observed data, it has the ability to generalize from the set of training data to data outside of the training domain. Second, the learned mathematical model is interpretable, making it informative for scientists to hypothesize the underlying physical or biological laws governing the observed data. Examples of recent methods for inferring the underlying governing equations include the sparse identification of nonlinear dynamics (SINDy) algorithm [1] and the equation learner (EQL) neural network [2,3], both of which are used for discovering systems of ordinary differential equations (ODEs), and the partial differential equation functional identification of nonlinear dynamics (PDE–FIND) algorithm [4], which is used to identify PDE systems. Boninsegna *et al*. [5] recently extended the SINDy algorithm to recover stochastic dynamical systems. Model selection criteria (such as Akaike information criteria and Bayesian information criteria) have been combined with the SINDy algorithm to increase robustness to errors, although incorrect models were still selected at noise levels we consider here [6]. The discovery methods mentioned above assume that the measured data arise from a parametrized n-dimensional dynamical system of the form

$$u_t(x,t) = F\left(x,t,u,u_x,u_{xx},\ldots,\theta\right), \quad x \in [x_0,x_f], \quad t \in [t_0,t_f] \tag{1.1a}$$

and

$$u(x,t_0) = u_0(x), \quad x \in [x_0,x_f] \tag{1.1b}$$

with parameter vector $\theta \in \mathbb{R}^k$ and appropriate boundary conditions.

Discovering a general function, F, from noisy data for u is an active area of research. The PDE–FIND approach from [4] builds a large library of candidate terms

$$\Theta = [1 \; u \; \cdots \; u^p \; u_x \; \cdots \; u^p \odot u_x \; u_{xx} \; \cdots \; u^p \odot u_{xx} \; u_x^2 \; u_x \odot u_{xx} \; u_{xx}^2], \tag{1.2}$$

whose columns represent potential terms comprising F and $\odot$ represents element-wise multiplication. Identifying F can now be re-cast as a sparse regression problem for the following linear system:

$$u_t = \Theta\xi, \tag{1.3}$$

where the unknown vector ξ is a sparse vector whose non-zero entries correspond to the terms predicted to constitute F. In this work, we will assume that the true dynamical system, F, consists of only a few simple terms (e.g. for the diffusion–advection equation, $F = Du_{xx} - cu_x, D, c \in \mathbb{R}$), so that a simple library with $p = 2$ and up to second-order derivatives is sufficient to discover F. Even in this limited setting, a practical challenge arises because one typically does not have access to the noiseless values of $u(x,t)$ or its partial derivatives. Instead, one must approximate these values from noisy experimental data. Here, our goal is to investigate the performance of existing denoising methods (to limit the amount of noise in u_t and Θ) that are used in conjunction with PDE–FIND and to present a novel denoising methodology relying on artificial neural networks (ANNs). We note that in this work we assume that the denoising methods considered sufficiently reduce the noise levels in order to neglect the so-called 'error-in-variables' problem, i.e. where the covariates on the right-hand side of the sparse regression equation (1.3) are inaccurately observed or contain noise. The error-in-variables problem may lead to biased estimates in high-dimensional sparse regression settings when the covariate noise is large [7]. In future work, we will explore the use of alternative sparse regression methods that are able to construct unbiased estimates when there is additive noise in the observed covariates [8–10].

Several methods have been used for denoising data to approximate $u(x,t)$ and its partial derivatives (u_t, u_x, u_{xx}, etc.). The most prevalent methods that have been proposed are finite difference approximations or the use of cubic splines for interpolation, followed by partial differentiation of the fitted splines. However, both of these methods have been found to be prone to inaccuracies in the presence of noise [4]. Recent work has considered recovery of dynamical systems with high amounts of noise added to the time derivative measurement (u_t) by transforming the data into a spectral domain [11]. Zhang & Lin [12] proposed using sparse

Bayesian regression, which allows for error bars for each candidate term in the discovered equation. However, although their method was robust, the noise in this study was also added only to the time derivative term (u_t) instead of the observed data ($u(x,t)$). Importantly, it has been noted that introducing noise to the observed data itself ($u(x,t)$) hinders the recovery of the correct PDE; thus, developing a method of denoising data for $u(x,t)$ has been identified as a current challenge for learning PDEs [11]. To the best of our knowledge, the two methods that, in practice, yield the most accurate approximations for the library terms involved in PDE learning are using finite differences or using splines. The primary challenge involved with using these methods for numerical differentiation, which we further test in this work, is that they are sensitive to noise levels and can amplify noise as the order of the derivative increases. This challenge inhibits learning PDEs for practical biological applications where data may have large noise levels due to many sources of error, including the data collection process, imprecise measurement tools and the inherent stochastic nature of biological processes [13,14]. For example, for ecological measurements of population abundance, typical datasets can have noise levels of the order of a coefficient of variation equal to 0.2 [15,16]. Notably, adding this biologically relevant level of noise to the observation $u(x,t)$ has not been considered in previous PDE learning work.

In this work, we are concerned with the performance of various denoising methods in recovering u and its derivatives from noisy data. There is much theory on approximating noiseless continuous functions, e.g. the Stone–Weierstrass theorem ensures that any compactly supported and continuous function can be uniformly approximated arbitrarily well with polynomial functions. Similarly, Hornik [17] proved that a single hidden layer ANN can approximate a continuous function and its derivatives arbitrarily well under some reasonable assumptions. How such methods perform in the presence of noise is not as well understood. Evaluating the performance of these methods in the presence of noise is a crucial task for equation learning methods, as poor derivative estimation hinders these methods' ability to uncover the correct underlying equations. We thus perform a systematic investigation in this work on the accuracy of these different denoising strategies in (i) estimating u and its derivatives and (ii) learning equations from these computations in the presence of varying amounts of noise. We include finite difference and local spline computations in this study because of their successful use in recent equation learning work which used lower amounts of noise than is considered here [4]. We investigate the use of ANNs as an alternative approach to these methods since they have not been considered previously. Because ANNs must be fitted to an entire set of spatio-temporal data (affording a global context that may help to decrease overfitting), we also consider a global spline method for a comparison between different global methods.

An additional, yet realistic, complication that has not been considered is the presence of non-constant error noise in the spatio-temporal data used for PDE learning. For example, proportional error noise can occur when the variance of the data is proportional to the size of the measurement, e.g. population size or density [18]. Non-constant error noise may also occur when the observed processes occur on different time scales [19]. In the scenario that one has a mathematical model for the biological process generating the data, e.g. $u(x,t)$, the non-constant error noise can be accounted for with a statistical model used in conjunction with the mathematical model [20]. For example, for a set of observed data at space points x_i, $i=1,\ldots,M$ and time points t_j, $j=1,\ldots,N$, a general statistical model is given by

$$U_{i,j} = u(x_i,t_j) + w_{i,j} \odot \mathcal{E}_{i,j}, \tag{1.4}$$

where the noiseless observations are corrupted by noise modelled by the random variable $w_{i,j} \odot \mathcal{E}_{i,j}$. Finite difference methods assume $w_{i,j} \odot \mathcal{E}_{i,j} = 0$ while regression methods using splines often assume that the variance of $w_{i,j} \odot \mathcal{E}_{i,j}$ is constant. More generally, the error term $\mathcal{E}_{i,j}$ may instead be generated by a probability distribution that is weighted by

$$w_{i,j} = \left(\beta_1 u_1^{\gamma}(x_i,t_j), \ldots, \beta_n u_n^{\gamma}(x_i,t_j)\right)^{\mathrm{T}}, \tag{1.5}$$

for $\gamma \geq 0$ and $\beta_1, \ldots, \beta_n \in \mathbb{R}$. Constant error noise is modelled by assuming $\gamma = 0$ and $\beta_1, \ldots, \beta_n = 1$. Proportional error noise is modelled by assuming $\gamma > 0$, $\beta_1, \ldots, \beta_n \neq 0$ [21]. We hypothesize that the assumption of constant variance error leads finite difference and spline approximations to yield poor estimates of the noiseless data $u(x,t)$ and its partial derivatives when the data contain proportional error noise. In this work, we investigate this hypothesis and develop a methodology using ANNs as a model for $u(x,t)$ in conjunction with an appropriate statistical error model that accounts for the presence of proportional error when denoising spatio-temporal data.

The denoising methods present in this work focus on spatio-temporal data for learning PDEs; however, the methods we describe can be readily applied to learning ODEs. We choose to focus our study on a specific set of diffusive PDE models, which have provided a wealth of insight into many biological transport phenomena, including ecological migration and invasion [22], neuronal transport [23], cancer progression [24–27] and wound healing [28,29]. We demonstrate that an ANN can be used with a non-constant error statistical model to accurately approximate $u(x,t)$ from noisy proportional error data better than finite difference and spline methods. We further demonstrate that the PDE–FIND algorithm can more accurately infer the governing PDE equations from data when its library of terms is constructed using an ANN-based method than when using these other methods.

2. Methods

The process of learning a system of equations from noisy data can be divided into two main components: (i) the data denoising and library construction component, in which the underlying dynamical system $u(x,t)$ and its partial derivatives are approximated from the noisy realizations of $\{U_{i,j}\}_{i=1,j=1}^{M,N}$ in (1.4), and (ii) the equation learning component, in which, given approximations for u, u_t, u_x, u_{xx}, etc., one employs an algorithm that can effectively uncover the mechanistic form of F in (1.1*a*) (figure 1). Below, we describe the mathematical models used for data generation, the method of constructing a library from noisy data and equation learning. All of the denoising methods were implemented in Python 2.7 using the Scipy package for polynomial splines and the Keras machine learning library for ANNs. All code and accompanying animations are available at https://github.com/biomathlab/PDElearning/.

(a) Data generation

We consider three diffusive PDE models for biological transport in this work, each of which has been used previously to interpret biological data [28,30–32]. These models include the diffusion–advection equation

$$u_t = -cu_x + Du_{xx}, \quad D, c \in \mathbb{R}, \tag{2.1}$$

the classical Fisher–Kolmogorov–Petrovsky–Piskunov (Fisher–KPP) equation

$$u_t = Du_{xx} + ru - ru^2, \quad D, r \in \mathbb{R} \tag{2.2}$$

and the nonlinear Fisher–KPP equation

$$u_t = Duu_{xx} + Du_x^2 + ru - ru^2, \quad D, r \in \mathbb{R}, \tag{2.3}$$

where D is the diffusion coefficient, r is the intrinsic population growth rate and c is the advection rate.

Assume $u(x,t)$ denotes the solution to one of the above mathematical models. We generate noisy data by using equation (1.4) with $w_{i,j} = \sigma u(x_i, t_j)$ (i.e. $\beta = \sigma$ and $\gamma = 1$), in which all $\mathcal{E}_{i,j}$ terms are simulated as i.i.d. normal random variables with mean 0 and variance 1. We generate six datasets for each mathematical model, setting $\sigma = 0$, 0.01, 0.05, 0.10, 0.25 and 0.50. For equation (2.1), we use its analytical solution to compute $u(x,t)$. For equations (2.2) and (2.3), we use finite difference computations to numerically approximate $u(x,t)$. The numerical step sizes in these computations were chosen small enough to not introduce significant noise into the solution.

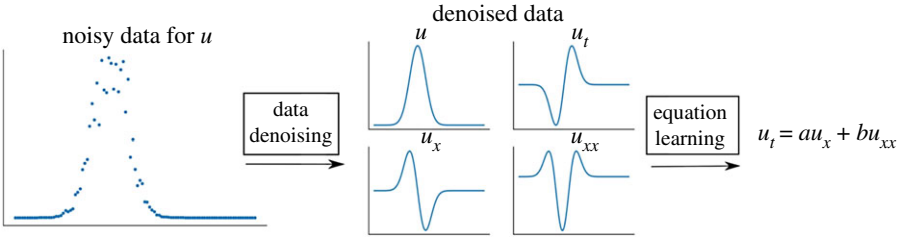

Figure 1. The two components to learning PDEs from data. The first component is to approximate u, u_t, u_x, u_{xx}, etc. from noisy data. The second component uses the output from the first component as an input for the PDE–FIND algorithm to learn a PDE. We also employ a pruning algorithm after the PDE–FIND step, not depicted here. (Online version in colour.)

We use $M = 101$ spatial locations and $N = 300$ time points to generate data for the diffusion–advection equation and $M = 199$ spatial locations and $N = 99$ time points for the Fisher–KPP and nonlinear Fisher–KPP equations. As a preprocessing step, each dataset is scaled to $[0, 1]$ in order to consistently measure errors across the various datasets and noise levels.

(b) Data denoising and library construction

In *data denoising*, we are interested in approximating the noisy dataset, $\{U_{i,j}\}_{i=1,j=1}^{M,N}$, with some representation, $f(x, t|\theta)$. Computation of f depends on parameters, θ, such as knots for polynomial splines or weights and biases for an ANN. This representation should match the dominant pattern of the data, so θ is chosen by finding the values that minimize the generalized least-squares (GLS) cost function[1]

$$\mathcal{J}_f(\theta) = \frac{1}{MN} \sum_{i=1,j=1}^{M,N} \left(\frac{f(x_i, t_j|\theta) - u_{i,j}}{|f(x_i, t_j|\theta)|^\gamma} \right)^2. \tag{2.4}$$

Note that this cost function accounts for the statistical error model in (1.4) and also reduces to the ordinary least-squares (OLS) estimator when $\gamma = 0$. In the case when the statistical error model is incorrectly specified, the resulting residuals can exhibit non-i.i.d. behaviour [13], violating our assumptions on equation (1.4). When the appropriate error model and cost function are not known *a priori*, residual computations and difference-based methods can provide insight into how to select these [33]. Some methodology for choosing an appropriate error model is explained in electronic supplementary material, §S1. Thus, $\hat{\theta} = \arg\min_\theta \mathcal{J}_f(\theta)$, so that $f(x, t|\hat{\theta}) \approx u(x, t)$. From this representation, we can then take partial derivatives of f with respect to x and t to estimate the partial derivatives of u.

(i) Finite differences and spline approximations

Finite differences. We use central difference formulae on interior points and forward differences at the boundaries to obtain the first-order derivative approximations. For higher order derivatives (e.g. u_{xx}), first-order finite difference rules are repeated on the corresponding previous order derivative approximations.

Finite difference approximations can be obtained accurately, efficiently and directly from noiseless data; however, their accuracy quickly deteriorates in the presence of observation error. Following [4], we also employ polynomial spline regression. To thoroughly investigate the performance of spline computations for data denoising, we will consider three separate methods of spline computation in this work: local splines with a constant variance (CV) error model (i.e. $\gamma = 0$), local splines with a non-constant variance error (NCV) model (i.e. $\gamma = 1$) and global splines with an NCV error model.

[1]Note that function values $f(x_i, t_j|\theta)$ less than 1×10^{-4} in absolute value are set equal to 1 in the denominator $|f(x_i, t_j|\theta)|^\gamma$ during all training and evaluation for practical implementation.

Local splines. For a given data point $u_{i,j}$ in the set of observations, we fit a cubic bi-spline on a small two-dimensional neighbourhood of size 11×11 centred at $u_{i,j}$ by minimizing equation (2.4). Denoised function and derivative approximations are then obtained using evaluations and the analytic derivatives of the fitted polynomial at the centre point $u_{i,j}$. Note that since we only approximate derivatives up to second order, higher order splines are not considered. For values of $u_{i,j}$ close to the boundary, we evaluate along the spline approximation that is nearest to the boundary. We found that cubic bi-spline approximations were generally more robust than the one-dimensional cubic splines used in [4]; see electronic supplementary material, §S2 and figures S2–S4.

Global splines. We perform global spline approximation of $\{U_i, j\}_{i=1,j=1}^{M,N}$ by minimizing equation (2.4) for $\gamma = 1$ with

$$S(x,t) = \sum_{k=1,l=1}^{K,L} c_{k,l} S_{k,l}(x,t), \tag{2.5}$$

where $S_{k,l}(x,t)$ are normalized bivariate cubic bi-splines defined on the knot locations $(x,t)_{k,l} = \{\tilde{x}_k, \ldots, \tilde{x}_{k+4}\} \times \{\tilde{t}_l, \ldots, \tilde{t}_{l+4}\}$ [34]. To implement this approximation, we estimate a smoothness coefficient, the knot locations and the spline coefficients by splitting the data into training and validation sets (50%/50%). The optimal values were chosen to be those that minimized the error (2.4) on the validation set. Details on implementation of this procedure are provided in the electronic supplementary material, §S3. Derivative estimates are obtained analytically from the final form of $S(x,t)$ by evaluating the analytic derivatives of each $S_{k,l}(x,t)$ term.

(ii) Artificial neural network approximations

An ANN, denoted $h(\boldsymbol{x}|\theta)$, was used to approximate $u(x,t)$, with one hidden layer of the form

$$h(\boldsymbol{x}|\theta) = a_2\Big(W_2\big(a_1(W_1\boldsymbol{x} + b_1)\big) + b_2\Big), \tag{2.6}$$

where $\boldsymbol{x} = [x \;\; t]^{\mathrm{T}}$ and $a_i(\cdot)$ represents the continuous nonlinear activation function for the ith layer. The matrices W_i and vectors b_i (typically called weights and biases) constitute the total set of trainable parameters $\theta = \{W_1, b_1, W_2, b_2\}$ of the network. We note that the use of one hidden layer in a neural network is a sufficient condition to make it a universal function approximator under the assumption that the activation function is bounded and non-constant [17]. This result extends to ANNs with multiple hidden layers; however, we found that, while training multi-layer ANNs resulted in faster convergence, the derivative approximations were worse. The task is therefore to find the optimal parameters θ^* such that $h(x,t|\theta^*) \approx u(x,t)$. The fitted surface function $h(x,t|\theta^*)$ and the computation of analytic derivatives of this function are used to approximate $u(x,t)$ in (1.4) and its partial derivatives for library construction in the PDE learning task.

We found that the choice of activation function, $a_i(\cdot)$, in the ANN plays an important role in the accuracy of the partial derivative approximations. Typical activations such as sigmoid and hyperbolic tangent yield oscillations in higher order derivative terms (see electronic supplementary material, movie S1.). To mitigate this, we chose to use the 'softplus' activation function, which takes the form $\log(1 + \mathrm{e}^z)$. This function has many desirable properties for approximating $u(x,t)$ (e.g. smoothness and infinitely many derivatives), which help ensure that the ANN can approximate the true function and its partial derivatives sufficiently well [17]. However, the softplus function is unbounded, which violates an assumption of ANNs as universal approximators. While the assumptions on activation functions in [17] are sufficient conditions and not necessary, one can address the unboundedness of the softplus function by including an ℓ_2-regularization penalty on the activations a_i in (2.6), but we found that no regularization was needed for the 18 datasets considered in this paper.

It becomes necessary to include an additional squared error term in the loss function to penalize function values outside $[0,1]$. Without this term, the function values can blow up during training since $\mathcal{J}_h(\theta) \to 1$ as $h(\boldsymbol{x}|\theta) \to +\infty$ when $\gamma = 1$. Thus, the complete loss function used for

training the ANN is

$$\mathcal{L}(\theta) = \frac{1}{MN} \sum_{i=1,j=1}^{M,N} \left(\frac{h(x_i, t_j|\theta) - u_{i,j}}{|h(x_i, t_j|\theta)|^\gamma} \right)^2 + \frac{1}{MN} \sum_{h \notin [0,1]} h^2, \tag{2.7}$$

where the first term corresponds to the generalized least-squares cost function (2.4) and the second term corresponds to the additional squared error term to penalize function values outside $[0, 1]$.

We used 1000 neurons in the hidden layer of the ANN. This choice was large enough to have maximal capacity to fit the data, while still allowing the optimization of θ to be computationally feasible on a desktop computer (3.4 GHz Intel Core i5 processor, 8 Gb RAM) without the need for GPU processing. The network parameters θ are optimized using the first-order gradient-based 'Adam' optimizer [35] with default parameters and a batch size of 10. We note that a small batch size paired with the adaptive moment estimation in Adam helps the ANN escape local minima during training, which stabilizes convergence across various datasets. In order to prevent overfitting, the data were randomly split into training and validation sets (90%/10%) when training the ANN. The optimal network parameters were chosen to be those that minimized the error (2.4) on the validation set. We did not train the ANNs for some fixed number of epochs since (i) the parameters of each network are randomly initialized and (ii) the networks are trained on different datasets, which can lead to faster or slower convergence rates. Instead, early stopping of 50 (i.e. stopping training once validation error had not decreased for 50 consecutive epochs) was used to ensure convergence regardless of the dataset or initial parameter values.

Representative examples of results from the bi-spline and ANN methods are shown in electronic supplementary material, movies S2 and S3, respectively. All movies for all methods and noise levels considered in this work can be found at https://github.com/biomathlab/PDElearning/animations/.

(c) Equation learning

We use the PDE–FIND algorithm [4] to discover the form of F in equation (1.1*a*) using computations of u, u_x, u_{xx} and u_t from the ANN, spline and finite difference methods. Prior to implementing PDE–FIND, the numerical approximations are scaled from $[0, 1]$ back into their original scales. We discuss the PDE–FIND implementation in §2c(i) and an additional pruning method in §2c(ii) that is used to remove extra terms from the final learned equation. We further discuss how we analyse our results in §2c(iii).

(i) PDE–FIND implementation

Once $u(x, t)$ and its partial derivatives have been computed, a large library of potential PDE terms is formed column-wise in the matrix, Θ, given by

$$\Theta = [1 \; u \; \cdots \; u^p \; u_x \; \cdots \; u^p \odot u_x \; u_{xx} \; \cdots \; u^p \odot u_{xx} \; u_x^2 \; u_x \odot u_{xx} \; u_{xx}^2], \tag{2.8}$$

where each column of Θ is some vectorization of the written term. All spline methods and the ANN have difficulty capturing the early dynamics of the diffusion–advection equation, so we skip the first 20 time points from the denoised data when building Θ for all datasets and denoising strategies. To reduce the computational time, only every fifth remaining time point is included in Θ. Hence, while the datasets for the diffusion–advection equation begin with $N = 300$ time points, only $(300 - 20)/5 = 56$ time points are used in constructing Θ. We set $p = 2$, resulting in $d = 12$ columns in Θ. Each column of Θ thus represents a candidate term comprising F, so we assume

$$u_t \approx \Theta \xi, \tag{2.9}$$

where ξ is a vector whose non-zero entries correspond to the true terms of F. The vector ξ is estimated using methods from sparse regression [36]. Sequential threshold ridge regression was

found to be a suitable method for estimating ξ for PDE–FIND in a previous study [4]. However, we found that a greedy algorithm performed well for the data and models we considered in this work. The greedy algorithm computes

$$\hat{\xi} = \arg\min_{\xi \in \mathbb{R}^d} \frac{1}{MN} \|u_t - \Theta \xi\|_2^2, \quad \text{subject to } \|\xi\|_0 \leq k, \tag{2.10}$$

for some sparsity parameter, k [37].

The tolerance for which we solve equation (2.10) for a given dataset is treated as a hyperparameter that is found by splitting the library data into separate training and validation sets, and then optimizing over the validation set. In this training–validation procedure, we randomly divide our data points for u_t into 5×5 tiles of adjacent spatio-temporal points and then randomly assign 50% of these tiles to a training dataset, u_t^{train}, and the remaining 50% to a validation set, u_t^{validate}. We split the corresponding rows of Θ into Θ^{train} and Θ^{validate}. We perform our hyperparameter search over 51 tolerance values, k, between 0 and 10^3. For each value, we estimate $\hat{\xi}$ from the training set. For each estimate, we then compute its mean-squared error (MSE) over the validation set. We choose the hyperparameter corresponding to the $\hat{\xi}$ estimate with the smallest MSE on the validation data. The equation that results from sparse regression with this hyperparameter is our final equation from the PDE–FIND algorithm. We refer to the validation MSE from the final equation 'val_0' in the remaining text.

(ii) Pruning method

We chose a 50–50 training and validation split for the data to avoid overfitting to the training data with a large validation set. Even so, we will demonstrate in §3b that PDE–FIND is able to learn small but systematic biases from the ANN's fit to u and its derivatives by incorporating extra terms into the final equations. Pruning methods have previously been developed that remove extra terms that do not significantly increase an algorithm's performance (e.g. [38,39]). Accordingly, we implement the following pruning method after the PDE–FIND implementation described in §2c(i) for all methods in order to delete the extra terms from the final equation.

The pruning procedure starts with a reduced library of candidate terms, $\tilde{\Theta}$, for the right-hand side of equation (1.1*a*) that correspond to the non-zero entries of $\hat{\xi}$ that resulted from our training–validation procedure. We then perform a sensitivity test for the remaining terms as follows. Suppose $\tilde{d}$ terms remain in $\tilde{\Theta}$, and let $\tilde{\Theta}_i$, $i = 1, \ldots, \tilde{d}$ denote the further-reduced library where the ith column of $\tilde{\Theta}$ has been removed. For each value of i, we find the least-squares solution (without regularization) on the training data to the equation

$$u_t^{\text{train}} = \tilde{\Theta}_i^{\text{train}} \xi_i. \tag{2.11}$$

We then use our $\hat{\xi}_i$ estimate and compute the MSE over the validation data when the ith term has been removed and call this computation val_i. We then remove any candidate terms for our library that result in $\text{val}_i/\text{val}_0 < 1 + \alpha$ for some $\alpha > 0$. After this pruning step, we perform one final round of training without regularization over the fully reduced library to find the final form of our underlying equation.

It is important to note that choice of the α pruning threshold value warrants careful decision. If this value is chosen too high, then too few terms will be selected and the learned equation will be incomplete. If the chosen value is too small, then the final equation will admit extra terms arising from the systematic errors in derivative estimation. We will demonstrate below that the arbitrary choice of $\alpha = 0.25$ provides promising results for the diffusion–advection and Fisher–KPP equations, while $\alpha = 0.05$ is suitable for the nonlinear Fisher–KPP equation.

(iii) Accuracy metrics

To quantitatively assess the accuracy in recovering the correct PDE that generated the data, i.e. using the combined PDE–FIND with the pruning methodology described above, we introduce

the *true positive ratio* (TPR) for a given vector ξ as

$$\mathrm{TPR}(\xi) = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN} + \mathrm{FP}}, \tag{2.12}$$

where TP (true positive) denotes the number of correctly specified non-zero coefficients in ξ, FN (false negative) denotes the number of coefficients in ξ that are incorrectly specified as zero and FP (false positive) denotes the number of coefficients in ξ that are incorrectly specified as non-zero. Recall that the non-zero entries of ξ correspond to the relevant terms in an equation (i.e. for a library of $\Theta = [1\ u\ u_x\ u_{xx}]$, $\xi = [0\ 1\ 2\ 0]^{\mathrm{T}}$ corresponds to the equation $u_t = u + 2u_x$). For example, when trying to learn equation (2.1), an equation of the form $u_t = u_{xx} + uu_x$ would have $\mathrm{TP} = 1$ (the non-zero coefficient for u_{xx} is correct), $\mathrm{FN} = 1$ (the missing u_x term is incorrect), $\mathrm{FP} = 1$ (the non-zero uu_x term is incorrect), resulting in a final score of $\mathrm{TPR} = 1/3$. Note that the TPR value is similar in nature to the Jaccard index: larger TPR values suggest that the true equation form has been better approximated, and $\mathrm{TPR} = 1$ signifies that the correct equation form has been recovered.

We note that the learned equation from the PDE–FIND with pruning method was often found to be sensitive to the random split of u_t and Θ into training and validation data. Therefore, we performed PDE learning for 1000 different random training–validation data splits of u_t and Θ for each dataset and for each computational method (finite differences, splines and ANN). We then consider the distribution of $\mathrm{TPR}(\hat{\xi})$ scores to assess the overall performance of the methodology. We declare the most commonly learned equation among the 1000 data splits as the final learned equation for each dataset and computational method.

3. Results

In this section, we detail our results using the ANN to denoise data for $u(x,t)$ and compute partial derivatives. In addition, we test the accuracy of using the ANN method in conjunction with PDE–FIND to learn PDEs. Analogous results are presented for finite differences and splines. We begin by demonstrating the accuracy of the partial derivative calculations in §3a, we explain why PDE–FIND finds small systematic bias terms in §3b, and then we detail the accuracy in learning of the diffusion–advection, Fisher–KPP and nonlinear Fisher–KPP equations in §3c–e.

(a) Derivative calculations

We found that the finite difference method most accurately approximates u and its derivatives for the advection–diffusion equation for $\sigma = 0$ (table 1). This result is not surprising, as finite difference computations assume that there is no error in the data. For all other values of σ, we observe that the ANN produces the most accurate derivative calculations, although either the local or global NCV splines outperform the ANN at inferring $u(x,t)$ when the data are very noisy ($\sigma > 0.25$). It is important to note that the ANN's derivative calculations are often several orders of magnitude more accurate than the spline and finite difference approximations, and this disparity between the computations appears to increase with σ (table 1). For example, at $\sigma = 0.01$, the ANN's relative mean-squared error (RMSE) for u_t is four orders of magnitude smaller than the RMSE for finite differences and at least two orders of magnitude smaller than the RMSE for all of the spline methods. At $\sigma = 0.50$, the ANN's RMSE for u_t has become six orders of magnitude smaller than the RMSE for finite differences and at least four orders of magnitude smaller than the RMSE for all spline methods. The other derivative computations show similar results.

Similarly, we find that the ANN is most accurate for computing derivatives from noisy data from the Fisher–KPP equation (table 2) and the nonlinear Fisher–KPP equation (table 3). Recall that we do not have analytical solutions to these equations, so we used finite difference computations on the noiseless data ($\sigma = 0$) as an estimate for the analytical derivative values for the Fisher–KPP and nonlinear Fisher–KPP equations. Again, in both cases we observe that the

Table 1. The relative mean-squared error (RMSE) between the noiseless data or true derivative values and our denoised data or derivative computations using finite differences, local splines with constant or non-constant variance, global splines with non-constant variance and the ANN for the diffusion–advection equation. FD, finite differences; LCVSP, local splines with constant variance; LNCVSP, local splines with non-constant variance; GNCVSP, global splines with non-constant variance; and ANN, artificial neural network. Bold text denotes the lowest errors of the methods.

σ	method	U RMSE	U_t RMSE	U_x RMSE	U_{xx} RMSE
0.00	FD	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{5.39 \times 10^{-5}}$	$\mathbf{5.77 \times 10^{-4}}$	$\mathbf{3.69 \times 10^{-2}}$
0.00	LCVSP	1.22×10^{-2}	1.12×10^{0}	3.33×10^{-2}	4.86×10^{1}
0.00	LNCVSP	8.08×10^{-4}	1.41×10^{2}	2.83×10^{0}	5.56×10^{1}
0.00	GNCVSP	7.47×10^{-3}	1.76×10^{2}	1.51×10^{4}	2.88×10^{7}
0.00	ANN	2.86×10^{-4}	3.96×10^{-1}	8.47×10^{-3}	3.75×10^{-1}
0.01	FD	$\mathbf{1.02 \times 10^{-4}}$	2.08×10^{2}	4.34×10^{-1}	3.52×10^{1}
0.01	LCVSP	1.11×10^{-2}	7.34×10^{0}	4.93×10^{-2}	4.87×10^{1}
0.01	LNCVSP	7.83×10^{-4}	4.39×10^{2}	3.29×10^{0}	5.72×10^{1}
0.01	GNCVSP	2.63×10^{-3}	1.05×10^{1}	1.36×10^{2}	1.19×10^{5}
0.01	ANN	8.40×10^{-4}	$\mathbf{7.71 \times 10^{-2}}$	$\mathbf{1.15 \times 10^{-2}}$	$\mathbf{6.93 \times 10^{-1}}$
0.05	FD	2.51×10^{-3}	2.42×10^{3}	9.97×10^{0}	1.01×10^{3}
0.05	LCVSP	1.19×10^{-2}	7.47×10^{1}	2.62×10^{-1}	5.01×10^{1}
0.05	LNCVSP	1.10×10^{-3}	1.95×10^{4}	3.04×10^{1}	2.49×10^{2}
0.05	GNCVSP	1.64×10^{-2}	2.82×10^{6}	4.34×10^{4}	1.50×10^{9}
0.05	ANN	$\mathbf{5.61 \times 10^{-4}}$	$\mathbf{1.95 \times 10^{-1}}$	$\mathbf{7.71 \times 10^{-3}}$	$\mathbf{7.90 \times 10^{-1}}$
0.10	FD	1.00×10^{-2}	4.04×10^{3}	7.59×10^{1}	3.78×10^{3}
0.10	LCVSP	1.04×10^{-2}	2.32×10^{2}	1.38×10^{0}	5.45×10^{1}
0.10	LNCVSP	1.97×10^{-3}	1.44×10^{4}	2.13×10^{1}	2.77×10^{2}
0.10	GNCVSP	1.81×10^{-1}	1.76×10^{2}	1.59×10^{3}	7.96×10^{6}
0.10	ANN	$\mathbf{9.51 \times 10^{-4}}$	$\mathbf{1.23 \times 10^{-1}}$	$\mathbf{1.44 \times 10^{-2}}$	$\mathbf{7.69 \times 10^{-1}}$
0.25	FD	6.28×10^{-2}	9.04×10^{4}	2.20×10^{2}	3.65×10^{4}
0.25	LCVSP	2.51×10^{-2}	6.21×10^{3}	5.83×10^{0}	2.42×10^{2}
0.25	LNCVSP	$\mathbf{6.76 \times 10^{-3}}$	1.64×10^{5}	3.98×10^{2}	3.42×10^{3}
0.25	GNCVSP	3.73×10^{-1}	1.31×10^{3}	1.42×10^{2}	1.93×10^{6}
0.25	ANN	7.29×10^{-3}	$\mathbf{1.53 \times 10^{0}}$	$\mathbf{4.49 \times 10^{-2}}$	$\mathbf{7.21 \times 10^{-1}}$
0.50	FD	2.41×10^{-1}	2.58×10^{6}	1.39×10^{3}	1.05×10^{5}
0.50	LCVSP	3.71×10^{-2}	3.78×10^{4}	6.68×10^{1}	7.95×10^{2}
0.50	LNCVSP	3.49×10^{-2}	3.86×10^{5}	1.32×10^{3}	7.24×10^{3}
0.50	GNCVSP	$\mathbf{2.08 \times 10^{-2}}$	3.51×10^{2}	2.96×10^{4}	2.58×10^{10}
0.50	ANN	6.34×10^{-2}	$\mathbf{3.43 \times 10^{0}}$	$\mathbf{1.05 \times 10^{-1}}$	$\mathbf{1.44 \times 10^{0}}$

finite difference calculations perform best in computing the RMSE for $\sigma = 0$, but, on average, the ANN provides the best calculations for the derivatives for larger values of σ. One of the local spline methods is consistently the most accurate at inferring u from noisy data. The disparity between the RMSE derivative calculations for the ANN as compared with the splines or finite differences again appears to increase with σ for these two equations.

Table 2. The relative mean-squared error (RMSE) between the noiseless data or true derivative values and our denoised data or derivative computations using finite differences, local splines with constant or non-constant variance, global splines with non-constant variance and the ANN for the Fisher–KPP equation. Keys and abbreviation as given in table 1.

σ	method	U RMSE	U_t RMSE	U_x RMSE	U_{xx} RMSE
0.00	FD	$\mathbf{0.00 \times 10^{0}}$	$\mathbf{3.16 \times 10^{-5}}$	$\mathbf{4.46 \times 10^{-4}}$	$\mathbf{4.67 \times 10^{-3}}$
0.00	LCVSP	6.28×10^{-5}	2.83×10^{-4}	2.83×10^{-3}	2.01×10^{-1}
0.00	LNCVSP	3.80×10^{-6}	9.43×10^{-2}	5.54×10^{-1}	5.17×10^{-1}
0.00	GNCVSP	2.56×10^{-2}	3.36×10^{0}	6.51×10^{1}	2.96×10^{3}
0.00	ANN	4.86×10^{-4}	6.98×10^{-2}	1.18×10^{-1}	2.66×10^{0}
0.01	FD	9.82×10^{-5}	4.84×10^{0}	6.47×10^{1}	1.19×10^{3}
0.01	LCVSP	6.91×10^{-5}	1.89×10^{-1}	1.56×10^{0}	$\mathbf{2.19 \times 10^{0}}$
0.01	LNCVSP	$\mathbf{1.00 \times 10^{-5}}$	1.49×10^{1}	6.29×10^{0}	9.63×10^{0}
0.01	GNCVSP	1.01×10^{-1}	7.29×10^{0}	2.91×10^{2}	6.24×10^{3}
0.01	ANN	3.90×10^{-4}	$\mathbf{3.85 \times 10^{-2}}$	$\mathbf{1.19 \times 10^{-2}}$	2.55×10^{0}
0.05	FD	2.52×10^{-3}	9.49×10^{1}	9.43×10^{2}	6.78×10^{4}
0.05	LCVSP	2.20×10^{-4}	8.85×10^{0}	5.83×10^{1}	2.39×10^{2}
0.05	LNCVSP	$\mathbf{1.71 \times 10^{-4}}$	6.06×10^{2}	2.20×10^{3}	6.22×10^{2}
0.05	GNCVSP	8.33×10^{-3}	1.90×10^{1}	3.96×10^{3}	3.39×10^{3}
0.05	ANN	4.67×10^{-4}	$\mathbf{1.03 \times 10^{-2}}$	$\mathbf{1.63 \times 10^{-2}}$	**1.57**
0.10	FD	9.95×10^{-3}	4.05×10^{2}	2.81×10^{3}	2.02×10^{5}
0.10	LCVSP	6.80×10^{-4}	1.64×10^{1}	6.66×10^{1}	9.35×10^{2}
0.10	LNCVSP	$\mathbf{6.17 \times 10^{-4}}$	1.22×10^{3}	1.73×10^{3}	2.65×10^{3}
0.10	GNCVSP	3.90×10^{-2}	1.13×10^{1}	3.20×10^{3}	5.94×10^{3}
0.10	ANN	8.46×10^{-4}	$\mathbf{4.26 \times 10^{-2}}$	$\mathbf{5.29 \times 10^{-2}}$	$\mathbf{1.23 \times 10^{0}}$
0.25	FD	6.30×10^{-2}	2.92×10^{3}	3.29×10^{4}	3.92×10^{5}
0.25	LCVSP	4.14×10^{-3}	2.26×10^{2}	5.65×10^{2}	1.44×10^{4}
0.25	LNCVSP	$\mathbf{4.07 \times 10^{-3}}$	5.17×10^{3}	9.49×10^{3}	2.05×10^{4}
0.25	GNCVSP	1.81×10^{-2}	4.53×10^{2}	1.28×10^{5}	3.27×10^{3}
0.25	ANN	6.41×10^{-3}	$\mathbf{6.94 \times 10^{-2}}$	$\mathbf{1.04 \times 10^{-1}}$	$\mathbf{5.90 \times 10^{0}}$
0.50	FD	2.38×10^{-1}	9.22×10^{3}	1.01×10^{5}	1.04×10^{6}
0.50	LCVSP	$\mathbf{1.52 \times 10^{-2}}$	5.44×10^{2}	1.32×10^{3}	3.22×10^{4}
0.50	LNCVSP	3.29×10^{-2}	2.36×10^{4}	2.19×10^{4}	7.33×10^{4}
0.50	GNCVSP	1.87×10^{-1}	2.82×10^{1}	1.14×10^{3}	2.33×10^{4}
0.50	ANN	6.60×10^{-2}	$\mathbf{5.48 \times 10^{-1}}$	$\mathbf{8.98 \times 10^{-1}}$	$\mathbf{3.52 \times 10^{1}}$

(b) PDE–FIND without pruning learns the wrong equation

We found that, in general, the PDE–FIND method learns the wrong equation, even when no noise is added to the data (electronic supplementary material, §S4 and figures S5–S7). Each denoising method resulted in accurate estimates for $u(x, t)$ and its partial derivatives in this case, however. For example, the residuals between the ANN model and the analytical values for u, u_t, u_x and u_{xx} were small when $\sigma = 0$ (figure 2). We observed that, while small, the ANN residuals include

Table 3. The relative mean-squared error (RMSE) between the noiseless data or true derivative values and our denoised data or derivative computations using finite differences, local splines with constant or non-constant variance, global splines with non-constant variance and the ANN for the nonlinear Fisher–KPP equation. Keys and abbreviation as given in table 1.

σ	method	U RMSE	U_t RMSE	U_x RMSE	U_{xx} RMSE
0.00	FD	$\mathbf{9.71 \times 10^{-36}}$	$\mathbf{2.21 \times 10^{-5}}$	$\mathbf{1.52 \times 10^{-4}}$	$\mathbf{1.34 \times 10^{-1}}$
0.00	LCVSP	1.53×10^{-5}	4.19×10^{-5}	1.09×10^{-3}	5.71×10^{0}
0.00	LNCVSP	2.88×10^{-6}	3.14×10^{-2}	2.12×10^{-2}	5.59×10^{0}
0.00	GNCVSP	1.61×10^{-2}	3.91×10^{1}	1.87×10^{2}	1.42×10^{2}
0.00	ANN	8.73×10^{-4}	1.41×10^{1}	6.16×10^{0}	1.40×10^{2}
0.01	FD	1.02×10^{-4}	2.21×10^{2}	9.16×10^{3}	8.26×10^{2}
0.01	LCVSP	2.01×10^{-5}	1.02×10^{1}	6.20×10^{2}	$\mathbf{3.32 \times 10^{0}}$
0.01	LNCVSP	$\mathbf{7.98 \times 10^{-6}}$	2.99×10^{1}	6.26×10^{2}	1.77×10^{1}
0.01	GNCVSP	2.95×10^{-3}	5.76×10^{0}	1.05×10^{2}	1.30×10^{3}
0.01	ANN	6.87×10^{-4}	$\mathbf{5.55 \times 10^{0}}$	$\mathbf{6.58 \times 10^{1}}$	1.90×10^{2}
0.05	FD	2.43×10^{-3}	5.45×10^{3}	2.65×10^{5}	3.35×10^{4}
0.05	LCVSP	1.57×10^{-4}	2.74×10^{2}	6.84×10^{3}	$\mathbf{6.69 \times 10^{1}}$
0.05	LNCVSP	$\mathbf{1.35 \times 10^{-4}}$	8.13×10^{2}	6.56×10^{3}	5.10×10^{2}
0.05	GNCVSP	2.73×10^{-3}	$\mathbf{6.13 \times 10^{0}}$	1.55×10^{2}	1.26×10^{3}
0.05	ANN	1.08×10^{-3}	8.26×10^{0}	$\mathbf{1.68 \times 10^{0}}$	1.70×10^{2}
0.10	FD	1.01×10^{-2}	2.30×10^{4}	1.16×10^{6}	5.69×10^{4}
0.10	LCVSP	6.20×10^{-4}	1.32×10^{3}	1.73×10^{4}	7.23×10^{1}
0.10	LNCVSP	$\mathbf{5.37 \times 10^{-4}}$	1.12×10^{4}	1.77×10^{4}	4.97×10^{3}
0.10	GNCVSP	8.57×10^{-4}	3.94×10^{3}	5.23×10^{3}	$\mathbf{6.72 \times 10^{1}}$
0.10	ANN	1.84×10^{-3}	$\mathbf{1.93 \times 10^{1}}$	$\mathbf{1.10 \times 10^{1}}$	2.04×10^{2}
0.25	FD	6.25×10^{-2}	1.47×10^{5}	4.95×10^{6}	9.86×10^{5}
0.25	LCVSP	4.00×10^{-3}	5.69×10^{3}	1.24×10^{5}	1.54×10^{3}
0.25	LNCVSP	$\mathbf{3.88 \times 10^{-3}}$	2.89×10^{4}	1.39×10^{5}	4.75×10^{3}
0.25	GNCVSP	5.79×10^{-3}	3.15×10^{3}	3.09×10^{3}	8.96×10^{2}
0.25	ANN	6.34×10^{-3}	$\mathbf{2.58 \times 10^{1}}$	$\mathbf{2.34 \times 10^{1}}$	$\mathbf{2.59 \times 10^{2}}$
0.50	FD	2.43×10^{-1}	5.73×10^{5}	1.92×10^{7}	2.46×10^{6}
0.50	LCVSP	$\mathbf{1.38 \times 10^{-2}}$	2.87×10^{4}	4.16×10^{5}	1.13×10^{4}
0.50	LNCVSP	1.49×10^{-2}	3.94×10^{4}	4.45×10^{5}	3.38×10^{4}
0.50	GNCVSP	3.57×10^{-2}	1.42×10^{4}	1.11×10^{4}	7.11×10^{2}
0.50	ANN	6.89×10^{-2}	$\mathbf{6.49 \times 10^{1}}$	$\mathbf{1.97 \times 10^{2}}$	$\mathbf{4.88 \times 10^{2}}$

systematic biases that comprise regions of over- and under-prediction. For example, all points near $(x, t) = (0.6, 0.4)$ for the ANN's calculation for u_x appear to over-predict the true value for u_x in this region (figure 2). This contradicts the assumption of independence in $\{\epsilon_{i,j}\}_{i=1,j=1}^{M,N}$ for the statistical model in equation (1.4). As we will now demonstrate, these small, systematic error terms from the ANN cause PDE–FIND to learn the incorrect equation.

We illustrate here that PDE–FIND learns the incorrect equation when training data for u_t consisting of u_t at all spatial points for the first half of the given time points and the validation

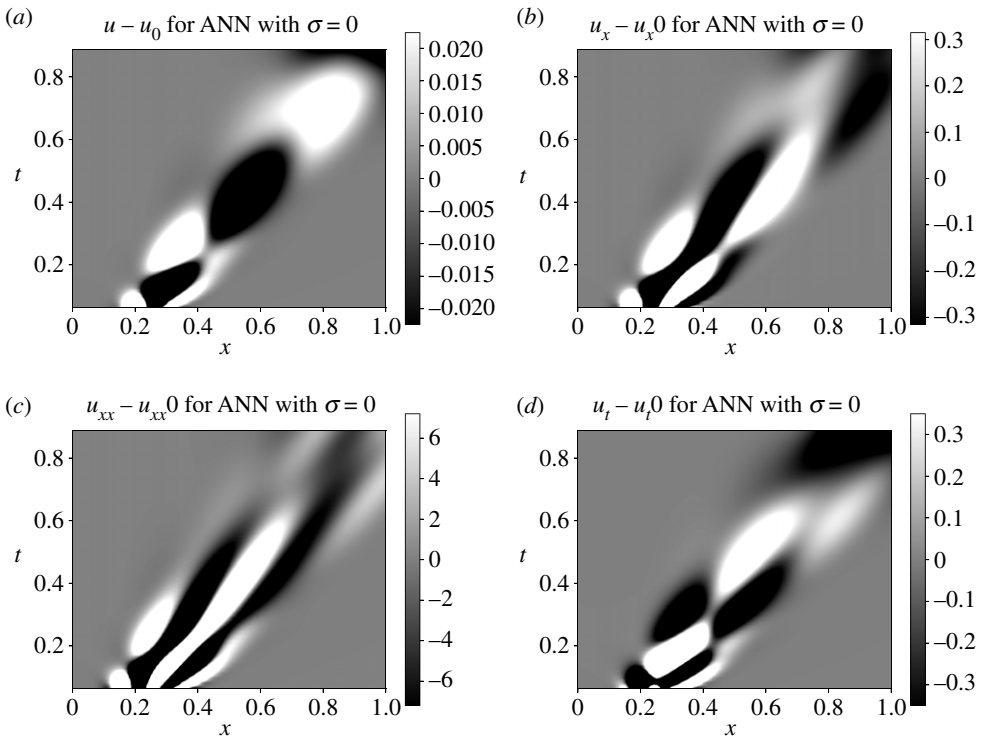

Figure 2. Contours of the residual values between the ANN model (u, u_x, etc.) and analytical solutions (u_0, u_x0, etc.) for the diffusion–advection data with $\sigma = 0$. (*a*) Residuals for u, (*b*) residuals for u_x, (*c*) residuals for u_{xx}, (*d*) residuals for u_t.

data comprise all spatial points for the second half of all time points. Recall that, in our actual implementation discussed below, we randomly split the training and validation data into 5×5 bins of adjacent spatio-temporal points. Using denoised values for $u(x, t)$ and its partial derivatives from the ANN in the case where $\sigma = 0$ in the data, our training–validation procedure without pruning learns an equation of the form

$$u_t = a + bu_x + cu_{xx} + du^2 + eu + fu^2u_x + gu^2u_{xx}, \quad a, \ldots, g \in \mathbb{R}. \tag{3.1}$$

Similarly, the learned equations using finite difference and spline computations are

$$u_t = au_x + bu_{xx} + cu^2u_x, \quad a, b, c \in \mathbb{R} \tag{3.2}$$

and

$$u_t = au_x + bu_{xx} + cu^2u_x + du^2u_{xx} + eu_x^2, \quad a, \ldots, e \in \mathbb{R}, \tag{3.3}$$

respectively.

Each of these equations is incorrect and has extra terms on the right-hand side of the learned PDE for the diffusion–advection equation. In figure 3, we depict illustrative portions of the training and validation sets comparing the analytical values of u_t against the computed values of u_t and PDE–FIND's selected equation using ANN approximations. We found that PDE–FIND selects equation (3.1) in place of the true diffusion–advection equation because it recovers the ANN's incorrect computations of u_t in both the training and validation data. In doing so, PDE–FIND fits the erroneous u_t computations from the ANN approximation by including extra terms in the learned PDE.

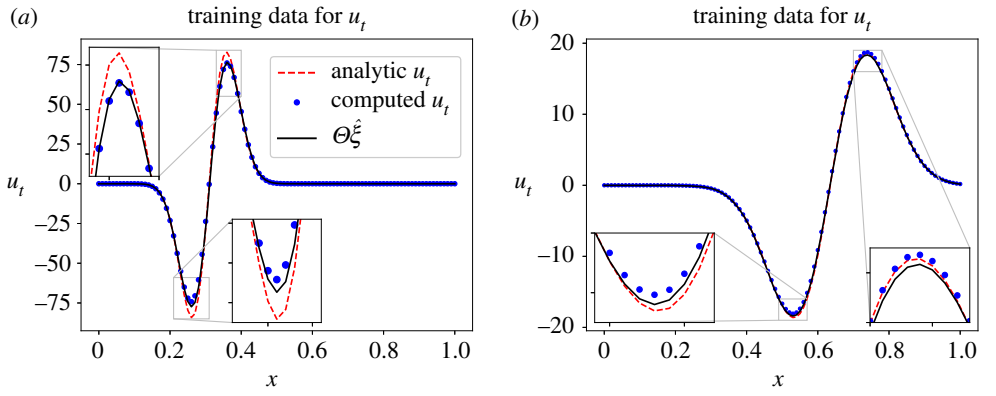

Figure 3. (*a*) Results from the training and (*b*) validation procedures in PD–FIND. The red dashed line denotes the analytical value of u_t, the blue dots denote the computed u_t values from the ANN and the black line denotes the equation for u_t that has been computed with PDE–FIND. (Online version in colour.)

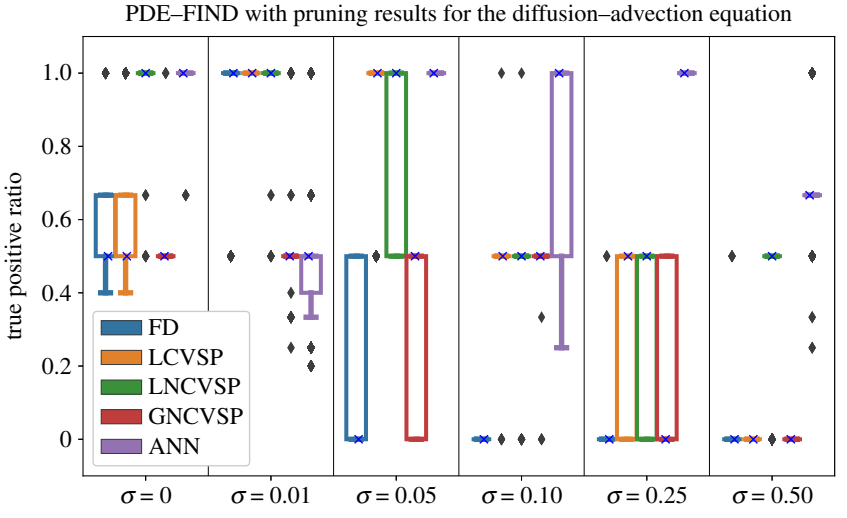

Figure 4. TPR values for the diffusion–advection equation. We calculated the TPR (see equation (2.12)), for 1000 different training–validation splits. These plots demonstrate the range of TPR values for each case. In each plot, the lower line in the coloured box portion provides the 25% quartile of the data and the upper line denotes the 75% quartile. The '×' on each box plot denotes the median TPR value for that scenario. The lengths of the upper and lower whiskers are 1.5 times the interquartile range of the distribution, and diamonds denote outlier points. Any plot depicted as a solid horizontal line (e.g. the neural net computations for $\sigma = 0$) denotes that that this value is the majority of the range of the distribution. (Online version in colour.)

(c) PDE–FIND with pruning for the diffusion–advection equation

We tested whether implementing an additional pruning step with PDE–FIND could remove the extra terms resulting from the biases discussed in §2b. For the diffusion–advection equation, we found that, for all values of σ except $\sigma = 0.01$, PDE–FIND with pruning achieves the highest median TPR when using ANN approximations (figure 4). The ANN's median value is TPR = 1 (meaning that over half of the simulations yielded the correct equation form) for $\sigma = 0$, 0.05, 0.10 and 0.25. The ANN resulted in a median TPR = 0.667 at $\sigma = 0.50$. By contrast, the spline methods only achieve a median TPR = 1 at the lower noise levels $\sigma = 0$, 0.01 and 0.05 for the local methods, and never achieve a median TPR = 1 for the global method. For $\sigma \geq 0.10$, the medians for all of the spline methods were TPR $\leq$0.5. The finite difference method resulted in a median TPR = 1 at $\sigma = 0.01$, but the median TPR = 0 for larger values of σ.

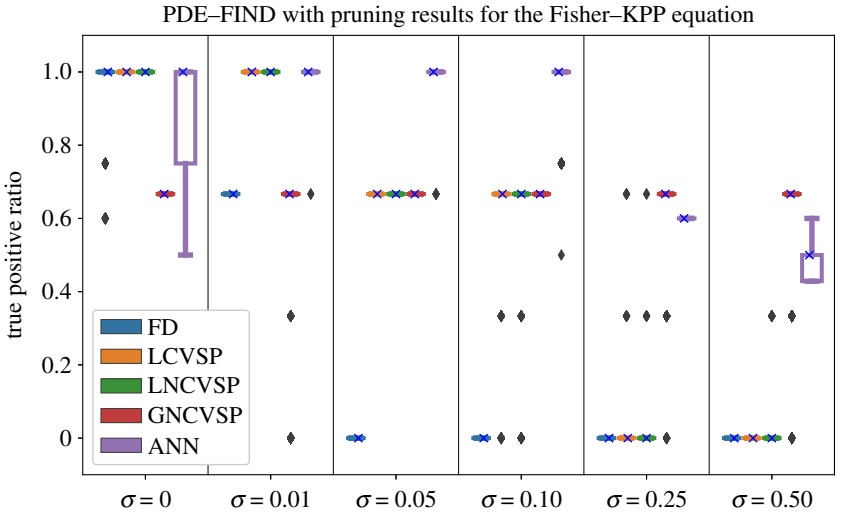

Figure 5. TPR values for the Fisher–KPP equation. We calculated the TPR (see equation (2.12)), for 1000 different training–validation splits. These plots demonstrate the range of TPR values for each case. In each plot, the lower line in the coloured box portion provides the 25% quartile of the data and the upper line denotes the 75% quartile. The '×' on each box plot denotes the median TPR value for that scenario. The lengths of the upper and lower whiskers are 1.5 times the interquartile range, and diamonds denote outlier points. Any plot depicted as a solid horizontal line (e.g. the finite difference computations for $\sigma = 0$) denotes that this value is the majority of the distribution. (Online version in colour.)

Electronic supplementary material, table S1 shows the most commonly learned PDEs for each denoising method at each noise level. We found that the ANN method, used in conjunction with PDE–FIND with pruning, resulted in the correct PDE for $\sigma = 0, 0.05, 0.10, 0.25$. The ANN specifies the incorrect equation for $\sigma = 0.01$ and $\sigma = 0.50$. However, in both of these cases, the extra terms have small parameter values (e.g. 0.001) that a scientist with an understanding of the system under consideration may manually neglect. On the other hand, PDE–FIND cannot discover the correct equation with finite difference or spline computations for $\sigma \geq 0.10$. These results suggest that the ANN method enables PDE–FIND with pruning to learn the diffusion–advection equation accurately at biologically realistic noise levels, e.g. $\sigma = 0.05, 0.10$ and 0.25.

(d) PDE–FIND with pruning for the Fisher–KPP equation

We tested the PDE–FIND with pruning method in conjuction with several denoising strategies using data from the Fisher–KPP equation. We found that the ANN method had a median TPR = 1 (meaning that the correct equation is specified for at least half of the training–validation data splits) for $\sigma = 0, 0.01, 0.05$ and 0.10 (figure 5). By contrast, the finite difference calculations only had a median TPR = 1 at $\sigma = 0$, and the local spline methods only have median TPR = 1 at $\sigma = 0, 0.01$ while the global spline method never achieves a median TPR = 1. The accuracy in using PDE–FIND with the spline and finite difference methods quickly deteriorates for high noise levels. The finite difference method resulted in a median TPR = 0 for $\sigma \geq 0.05$ and all of the spline methods result in a median TPR of less than 0.667 at $\sigma = 0.05, 0.10$. At $\sigma = 0.25, 0.50$, the local spline methods have median TPR = 0 while the global spline method maintains a median TPR = 0.667. The ANN had a median TPR = 0.6 and 0.5 for $\sigma = 0.25$ and 0.5, respectively.

Electronic supplementary material, table S2 shows the most commonly chosen PDEs resulting from the PDE–FIND with pruning method. We found that PDE–FIND with pruning is able to discover the correct equation form for $\sigma = 0, 0.01, 0.05$ and 0.10 when using the ANN approximations. While PDE–FIND is unable to specify the correct equations with ANN data for $\sigma = 0.25$ and 0.50, all of the terms in the Fisher–KPP equation were included in the learned PDEs. By contrast, using the local spline methods for denoising resulted in only learning the

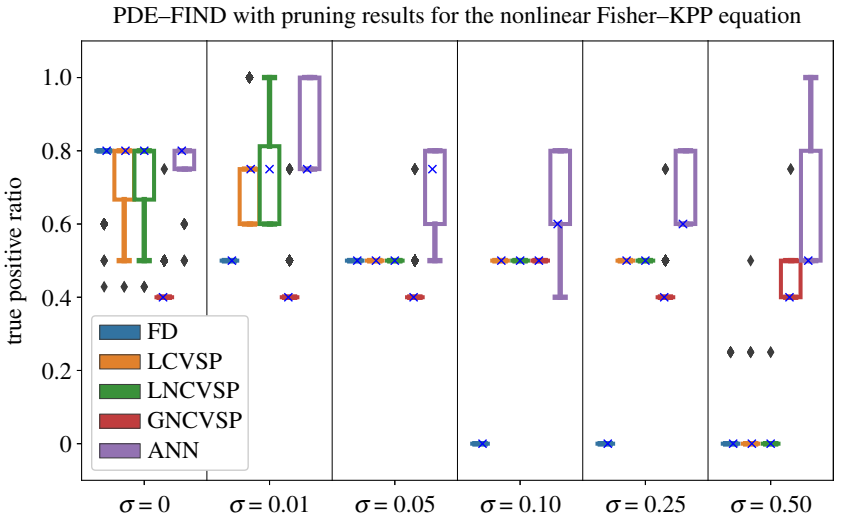

Figure 6. TPR values for the nonlinear Fisher–KPP equation. We calculated the TPR (see equation (2.12)), for 1000 different training–validation splits. These plots demonstrate the range of TPR values for each case. In each plot, the lower line in the coloured box portion provides the 25% quartile of the data and the upper line denotes the 75% quartile. The 'x' on each box plot denotes the median TPR value for that scenario. The lengths of the upper and lower whiskers are 1.5 times the interquartile range, and diamonds denote outlier points. Any plot depicted as a lone solid horizontal line (e.g. the finite difference computations for $\sigma = 0$) denotes that this value is the whole range of the data. (Online version in colour.)

correct PDE for $\sigma \leq 0.01$. For σ, the spline methods resulted in large errors in the derivative approximations and did not recover the u_{xx} terms for $\sigma = 0.05, 0.10$ and did not yield any terms on the right-hand side of the learned PDE for $\sigma = 0.25, 0.50$. The global spline method never recovers the u_{xx} term in its final recovered equation, but does recover the u and u^2 terms. Similarly, the finite difference method resulted in only learning the true equation form for $\sigma = 0$. These results suggest that only the ANN method enables PDE–FIND with pruning to learn the Fisher–KPP equation for reasonably high noise levels of $\sigma = 0.05, 0.10$, whereas the spline methods cannot for $\sigma > 0.05$.

(e) PDE–FIND with pruning for the nonlinear Fisher–KPP equation

We found that the PDE–FIND with pruning method was not able to recover the correct PDE from data that have been generated by the nonlinear Fisher–KPP equation for all denoising strategies considered. PDE–FIND could not achieve a median TPR = 1 for any of these methods, meaning that the correct equation was never specified for over half of the training–validation data splits (figure 6). All methods have median TPR $\leq$ 0.8 at $\sigma = 0$. When using ANN approximations, the PDE–FIND with pruning method has median TPR = 0.8 at $\sigma = 0.01$ and 0.05, TPR = 0.6 at $\sigma = 0.10$ and 0.25, and TPR = 0.5 at $\sigma = 0.50$. When using local spline computations, PDE–FIND with pruning has median TPR = 0.8 at $\sigma = 0.01$, TPR = 0.5 at $\sigma = 0.05, 0.10$ and 0.25, and TPR = 0 at $\sigma = 0.50$. When using the global spline computation, PDE–FIND with pruning has median TPR = 0.4 at $\sigma = 0, 0.01, 0.05, 0.25$ and 0.50 and TPR = 0.5 at $\sigma = 0.10$. When using finite difference computations, PDE–FIND with pruning has median = 0.50 at $\sigma = 0.01$ and 0.05 and TPR = 0 for $\sigma \geq 0.10$.

While all of the denoising strategies lead to incorrect equations, the ANN strategy recovers the most relevant terms in its final equations. Electronic supplementary material, table S3 shows the most commonly chosen PDEs resulting from the PDE–FIND with pruning method. We found that all methods except the global splines predict the true equation with an extra Fickian diffusion term, u_{xx}, for $\sigma = 0$. The global splines do not recover the uu_x and u_x^2 terms. When using the ANN approximations, the PDE–FIND with pruning algorithm recovers the true equation with

Table 4. Inferred parameters for the nonlinear Fisher–KPP equation data when performing an inverse problem on equation (3.4).

	true equation
	$u_t = 0.02uu_{xx} + 0.02u_x^2 + 10u - 10u^2$
σ	revised equation
0.00	$u_t = 2.3 \times 10^{-12}u_{xx} + 0.017uu_{xx} + 0.021u_x^2 + 10.0u - 10.0u^2 - 1.60 \times 10^{-8}$
0.01	$u_t = 1.4 \times 10^{-5}u_{xx} + 0.021uu_{xx} + 0.019u_x^2 + 10.0u - 10.0u^2 - 1.38 \times 10^{-8}$
0.05	$u_t = 2.5 \times 10^{-4}u_{xx} + 0.0034uu_{xx} + 0.029u_x^2 + 9.9u - 10.0u^2 - 2.86 \times 10^{-8}$
0.10	$u_t = 6.1 \times 10^{-4}u_{xx} + 8.42 \times 10^{-4}uu_{xx} + 0.023u_x^2 + 9.38u - 9.52u^2 - 2.44 \times 10^{-4}$
0.25	$u_t = 7.2 \times 10^{-4}u_{xx} + 0.015uu_{xx} + 0.024u_x^2 + 9.8u - 9.14u^2 + 3.53 \times 10^{-7}$
0.50	$u_t = 2.1 \times 10^{-3}u_{xx} - 3.72 \times 10^{-3}uu_{xx} + 0.032u_x^2 + 9.7u - 7.5u^2 + 1.38 \times 10^{-6}$

added Fickian diffusion at $\sigma = 0.05$, and it recovers three of the correct terms but excludes uu_{xx} at $\sigma = 0.01$. For larger values of σ with the ANN approximations, PDE–FIND with pruning recovers three correct terms, excludes the uu_{xx} term and includes an extra Fickian diffusion term (as well as an additional constant term at $\sigma = 0.50$). When using local spline computations, the PDE–FIND with pruning algorithm recovers three correct terms, excludes the uu_{xx} term and adds an extra Fickian diffusion term at $\sigma = 0.01$. For $\sigma = 0.05 - 0.25$, the final equation recovers two correct terms but excludes the uu_{xx} and u_x^2 terms. At $\sigma = 0.50$, all terms are deleted when using local spline computations. When using global spline computations, PDE–FIND with pruning recovers only the reaction terms when $\sigma = 0.10$. For all other noise levels, it replaces the nonlinear diffusion terms with Fickian diffusion. When using finite difference computations, the PDE–FIND with pruning algorithm correctly recovers two terms but excludes the uu_{xx} and u_x^2 terms at $\sigma = 0.01$ and 0.05. For larger values of σ, no correct terms are included in the final equation form.

We investigated if the recovered terms from the PDE–FIND with pruning algorithm using ANN approximations can be used as the specified mathematical model in an inverse problem methodology (cf. [21]) to recover the final parameter estimate values from the nonlinear Fisher–KPP equation. If we take the union of all terms that are included in the final equations in electronic supplementary material, table S3 for the ANN method using noisy data ($\sigma > 0$), we have an equation of the form

$$u_t = au_{xx} + buu_{xx} + cu_x^2 + du + eu^2 + f; \quad a, \ldots, f \in \mathbb{R}. \tag{3.4}$$

We estimated the parameters $a, \ldots, f$ in equation (3.4) for each value of σ by simulating the solution to this PDE using the method of lines and minimizing equation (2.4) using the Nelder–Mead algorithm. We input the equations from electronic supplementary material, table S3 for the ANN method as the initial guess for each dataset. We found that performing this inverse problem leads to accurate parameter estimates for the true terms in the nonlinear Fisher–KPP equation and small coefficient values for the incorrect terms (u_{xx} and 1) for $\sigma = 0, 0.01, 0.05$ and 0.25 (table 4). At $\sigma = 0.10$ and 0.50, this inverse problem methodology leads to small coefficient estimates for uu_{xx} in addition to u_{xx} and 1. Note that this same process would not lead to ultimately recovering the true equation and parameter estimates from the spline or finite difference approximations because their final equations never included the correct uu_{xx} term in the final equation for noisy data ($\sigma > 0$).

4. Conclusion and future work

The novel use of the ANN method presented here is a significant step towards making PDE learning more achievable in realistic scenarios with noisy biological data. Because an ANN is a fully differentiable function, it can be used to approximate derivative computations to build the library of terms needed for PDE learning. The current practice to build a library

of terms for learning PDEs when noise is present in the observed data $u(x,t)$ is to use finite difference or local spline approximations for small amounts of noise [4,12]. Finite difference and spline-based techniques were reported as state-of-the-art denoising methods for equation learning in a study by Rudy *et al.* [4]. We note that Rudy *et al.* also considered several other numerical methods, such as Gaussian kernel smoothing, Tikhonov differentiation and spectral differentiation with high-frequency term thresholding, and found that the most reliable and robust method was polynomial interpolation. We expanded on their use of local uni-variate polynomial splines by implementing local bi-variate splines, which we found to be more robust for function and derivative approximation, and also implemented global spline approximation as a closer comparison to the performance of the ANN as a global approximation method. Our findings suggest that finite difference and spline methods are highly sensitive to the amount of noise in the data in the range of less than 5% noise. We showed that the ANN method outperforms spline and finite difference approximations of the partial derivatives of $u(x,t)$, even when some spline methods better approximated $u(x,t)$ than the ANN, in the presence of significant levels of non-constant error noise. It is important to note that this level of noise and non-constant variance are typical phenomena encountered in biological data [18].

We compared polynomial spline approximations and ANNs as global methods to denoise data because there exist previous theoretical results regarding the ability of polynomials and ANNs to approximate continuous functions up to arbitrary precision on compact domains [17,40]. This theory is only relevant in the case of noiseless data and in the limit of a large number of neurons or right-hand-side terms. In practice, neither of these assumptions is realistic when dealing with biological data, which tend to have a large noise-to-signal ratio, and where one is required to optimize hyperparameters (e.g. number of neurons, smoothness, knots) for a given dataset. To the best of our knowledge, there has not been a direct comparison between the practical performance of polynomial splines and ANNs in approximating a continuous function and its partial derivatives from noisy non-constant variance data. We also note that derivative estimation is often ignored in the optimization for various methods, e.g. splines and ANNS, used to approximate a function $f(x,t|\theta)$ for $u(x,t)$. For example, equation (2.4) does not consider the ability of $f_x(x,t|\theta)$ to approximate $u_x(x,t)$ or $f_t(x,t|\theta)$ to approximate $u_t(x,t)$. The numerical results presented here are the first such comparison between various local and global approximation methods in their ability to estimate both $u(x,t)$ and its derivatives. We found that the ANN is comparable to spline methods in approximating $u(x,t)$ and is more accurate in approximating its spatial and temporal derivatives. The disparity between the accuracy of ANNs and other methods in estimating derivatives sharply increased with higher noise in the data. These results suggest the need for further theoretical investigation into the fidelity of derivative estimation from the function approximation methods considered here in the presence of realistic forms and levels of noise. Since a number of heuristics were used in the development of our proposed method, this suggests that future modifications to the network architecture may increase denoising and derivative approximation accuracy. We postulate that one reason for the superior performance of our proposed ANN for derivative approximation is that the chosen activation function (i.e. softplus) is sufficiently smooth. This satisfies conditions in [17] that ensure the capability of ANNs to converge sufficiently close to the true partial derivatives in addition to the underlying dynamical system. Our results suggest that ANNs outperform all other methods considered here in this respect for the biological transport models we considered (tables 1–3).

The importance of our denoising methodology results is underscored by the need for accurate derivative estimates in equation learning techniques. We note that while methods such as Bayesian inference and Kalman filtering have been commonly used in the mathematical modelling literature [41] for denoising data, we did not compare these approaches in this work as they require specifying a mathematical model for the data *a priori*. By contrast, the two-step procedure we performed is used to learn a mathematical model from the data; this requires a model-free method to first denoise the data and numerically approximate derivatives. Indeed, we found that the PDE–FIND algorithm can successfully recover the true equations underlying $u(x,t)$ when $u(x,t)$ and its derivatives have been accurately recovered. For example, the ANN outperforms all spline methods in computing the derivatives of u for the advection–diffusion

equation when $\sigma > 0$ (table 1), and in turn the ANN computations allow PDE–FIND to learn the correct underlying equation form for $\sigma = 0.05, 0.10$ and 0.25, whereas all finite difference and spline computations fail in identifying the underlying equation for $\sigma > 0.05$ (figure 4). Using the ANN estimates, we found that PDE–FIND could be used to learn the Fisher–KPP equation for up to 10% noise levels ($\sigma = 0.1$), whereas the other methods fail at 5% ($\sigma = 0.05$). None of the methods we considered were able to learn the nonlinear Fisher–KPP equation, but the ANN computations lead to an equation that can then be used with an inverse problem methodology [21] to infer which terms are meaningful. We found in this study that identifying the correct underlying equation under the high levels of observation noise we considered was a challenging problem for current state-of-the-art equation learning methods, even under the restriction of setting $p = 2$ in equation (1.2). Future work will investigate correctly inferring model equations from much larger libraries for the current equation learning method.

We focused on three common transport models in this work. These models are broadly relevant across several fields in biology and have been used previously to describe the movement and growth of a varying array of spatio-temporal processes, e.g. wound healing [19,28,30,31], cancer progression [24,26,27], and animal development and herd migration [15,22,32]. In future work, we aim to extend the results of this study to more complex PDE models in biology, such as systems of PDEs used to model several subpopulations, processes in higher spatial dimensions and more complex nonlinearities used to describe migration.

There are several reasonable explanations for the difficulties in learning the nonlinear Fisher–KPP equation. Three of the terms in equation (2.3) are the product of two terms including u or its derivatives (u^2, u_x^2 and uu_{xx}). In practice, these terms may be inaccurate approximations from noisy data (as demonstrated in table 3). Multiplying two inaccurate terms may lead to an even larger amount of uncertainty associated with these estimates. We postulate that the high level of uncertainty in these types of terms resulting from the product of inaccurate estimates likely increases the difficulty of learning to include them in the process of PDE learning. Furthermore, it must be noted that the data for this equation were generated numerically by the finite difference method. Though some analytical solutions to the Fisher–KPP equations are known, they come either in series form, which is beyond the scope of this article as it would require learning an infinite number of polynomial terms ($p = \infty$), or in travelling wave form, which would be indistinguishable for PDE–FIND from the advection equation [42]. The finite difference methods used to approximate the spatial derivatives in the nonlinear Fisher–KPP equation introduce second- and fourth-order error terms, which lead to numerical dispersion and diffusion effects that may account for the recovery of some unexpected terms.

We found that the use of pruning following our implementation of the PDE–FIND algorithm increased our ability to recover the correct equation in terms of the TPR. It may be argued that these additional terms learned from the PDE–FIND implementation without pruning would be removed from the final equation if more regularization (i.e. a larger value of k in implementation of the greedy algorithm) were used. However, when faced with the issue of learning the governing equation from actual data in practice, one will not have the ability to know when the specified equation is correct or not. We thus need to identify the correct hyperparameters without any *a priori* knowledge. We observed that the systematic biases in our ANN (depicted in figure 2) make it difficult to choose a hyperparameter value that leads to the correct equation because these biases are present in the training and validation data. The pruning algorithm is a way to correct for when the incorrect hyperparameter has been chosen by ensuring that all terms in the learned PDE are sufficiently sensitive to constitute a strong signal in the data, instead of resulting from a bias in our approximation methods.

Our use of pruning to remove terms from learned PDEs could be improved in future work. While effective, our implementation is somewhat crude, in which we pre-specify a threshold level to prune parameters based on out-of-sample MSE values on the validation dataset. Previous studies have discussed F-statistics as one way to infer the increase in variance that pruning a variable will lead to, but there are still many different interpretations of these results, which makes a definitive statistical pruning method challenging to ascertain [43].

Data accessibility. All code, data and accompanying animations are available at https://github.com/biomathlab/PDElearning/.

Authors' contributions. J.H.L. implemented the methods on denoising/derivative approximation; J.T.N. implemented the methods on equation learning and parameter estimation; J.T.N., J.H.L., G.M.L., E.M.R. and K.B.F. interpreted the results; J.T.N. created the simulated datasets; J.H.L., E.M.R. and K.B.F. conceived the ANN methodology; J.T.N. and G.M.L. conceived and implemented the pruning methodology; J.T.N., J.H.L., G.M.L., E.M.R. and K.B.F. wrote the paper. J.T.N. and J.H.L. contributed equally to this work.

Competing interests. We declare we have no competing interest.

Funding. This research was supported in part by National Science Foundation grant nos. DMS-1514929 and IOS-1838314 and in part by National Institute of Aging grant no. R21AG059099.

References

1. Brunton SL, Proctor JL, Kutz JN. 2016 Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proc. Natl Acad. Sci. USA* **113**, 3932–3937. (doi:10.1073/pnas.1517384113)
2. Martius G, Lampert CH. 2016 Extrapolation and learning equations. (http://arxiv.org/abs/1610.02995)
3. Sahoo S, Lampert C, Martius G. 2018 Learning equations for extrapolation and control. In *Proc. of the 35th Int. Conf. on Machine Learning, Stockholm, Sweden, 10–15 July 2018* (eds J Dy, A Krause), vol. 80, pp. 4442–4450. Proceedings of Machine Learning Research.
4. Rudy SH, Brunton SL, Proctor JL, Kutz JN. 2017 Data-driven discovery of partial differential equations. *Sci. Adv.* **3**, e1602614. (doi:10.1126/sciadv.1602614)
5. Boninsegna L, Nüske F, Clementi C. 2018 Sparse learning of stochastic dynamical equations. *J. Chem. Phys.* **148**, 241723. (doi:10.1063/1.5018409)
6. Mangan NM, Kutz JN, Brunton SL, Proctor JL. 2017 Model selection for dynamical systems via sparse regression and information criteria. *Proc. R. Soc. A* **473**, 20170009. (doi:10.1098/rspa.2017.0009)
7. Rosenbaum M, Tsybakov AB. 2010 Sparse recovery under matrix uncertainty. *Ann. Stat.* **38**, 2620–2651. (doi:10.1214/10-AOS793)
8. Chen Y, Caramanis C, Mannor S. 2013 Robust sparse regression under adversarial corruption. In *Proc. of the 30th Int. Conf. on Machine Learning, Atlanta, GA, 17–19 June 2013* (eds S. Dasgupta, D. McAllester), vol. 28, pp. 774–782. Proceedings of Machine Learning Research.
9. Datta A, Zou H. 2017 Cocolasso for high-dimensional error-in-variables regression. *Ann. Stat.* **45**, 2400–2426. (doi:10.1214/16-AOS1527)
10. Loh P-L, Wainwright MJ. 2012 High-dimensional regression with noisy and missing data: provable guarantees with nonconvexity. *Ann. Stat.* **40**, 1637–1664. (doi:10.1214/12-AOS1018)
11. Schaeffer H. 2017 Learning partial differential equations via data discovery and sparse optimization. *Proc. R. Soc. A* **473**, 20160446. (doi:10.1098/rspa.2016.0446)
12. Zhang S, Lin G. 2018 Robust data-driven discovery of governing physical laws with error bars. *Proc. R. Soc. A* **474**, 20180305. (doi:10.1098/rspa.2018.0305)
13. Banks H, Thompson WC, Hu S. 2014 *Modeling and inverse problems in the presence of uncertainty*. Boca Raton, FL: CRC Press.
14. Codling EA, Plank MJ, Benhamou S. 2008 Random walk models in biology. *J. R. Soc. Interface* **5**, 813–834. (doi:10.1098/rsif.2008.0014)
15. Francis CRIC, Hurst RJ, Renwick JA. 2003 Quantifying annual variation in catchability for commercial and research fishing. *Fish. Bull.* **101**, 293–304.
16. Perretti C, Munch S, Sugihara G. 2013 Model-free forecasting outperforms the correct mechanistic model for simulated and experimental data. *Proc. Natl Acad. Sci. USA* **110**, 5253–5257. (doi:10.1073/pnas.1216076110)
17. Hornik K. 1991 Approximation capabilities of multilayer feedforward networks. *Neural Netw.* **4**, 251–257. (doi:10.1016/0893-6080(91)90009-T)
18. Banks HT, Sutton KL, Clayton Thompson W, Bocharov G, Roose D, Schenkel T, Meyerhans A. 2011 Estimation of cell proliferation dynamics using CFSE data. *Bull. Math. Biol.* **73**, 116–150. (doi:10.1007/s11538-010-9524-5)
19. Johnston ST, Simpson MJ, McElwain DLS. 2014 How much information can be obtained from tracking the position of the leading edge in a scratch assay? *J. R. Soc. Interface* **11**, 20140325. (doi:10.1098/rsif.2014.0325)

20. Anderssen RS, Bloomfield P. 1974 Numerical differentiation procedures for non-exact data. *Numerische Math.* **22**, 157–182. (doi:10.1007/BF01436965)
21. Banks HT, Tran HT. 2009 *Mathematical and experimental modeling of physical and biological processes*. Boca Raton, FL: CRC Press.
22. Hastings A *et al.* 2005 The spatial spread of invasions: new developments in theory and evidence. *Ecol. Lett.* **8**, 91–101. (doi:10.1111/j.1461-0248.2004.00687.x)
23. Jones CKRT. 1984 Stability of the travelling wave solution of the FitzHugh-Nagumo system. *Trans. Am. Math. Soc.* **286**, 431–431. (doi:10.1090/S0002-9947-1984-0760971-6)
24. Baldock AL *et al.* 2014 Patient-specific metrics of invasiveness reveal significant prognostic benefit of resection in a predictable subset of gliomas. *PLoS ONE* **9**, e99057. (doi:10.1371/journal.pone.0099057)
25. Rockne RC *et al.* 2015 A patient-specific computational model of hypoxia-modulated radiation resistance in glioblastoma using ^{18}F-FMISO-PET. *J. R. Soc. Interface* **12**, 20141174. (doi:10.1098/rsif.2014.1174)
26. Rutter EM *et al.* 2017 Mathematical analysis of glioma growth in a murine model. *Sci. Rep.* **7**, 2508. (doi:10.1038/s41598-017-02462-0)
27. Stepien TL, Rutter EM, Kuang Y. 2015 A data-motivated density-dependent diffusion model of in vitro glioblastoma growth. *Math. Biosci. Eng.* **12**, 1157–1172. (doi:10.3934/mbe.2015.12.1157)
28. Maini PK, McElwain DS, Leavesley DI. 2004 Travelling waves in a wound healing assay. *Appl. Math. Lett.* **17**, 575–580. (doi:10.1016/S0893-9659(04)90128-0)
29. Nardini JT, Chapnick DA, Liu X, Bortz DM. 2016 Modeling keratinocyte wound healing: cell-cell adhesions promote sustained migration. *J. Theor. Biol.* **400**, 103–117. (doi:10.1016/j.jtbi.2016.04.015)
30. Jin W, Shah ET, Penington CJ, McCue SW, Chopin LK, Simpson MJ. 2016 Reproducibility of scratch assays is affected by the initial degree of confluence: experiments, modelling and model selection. *J. Theor. Biol.* **390**, 136–145. (doi:10.1016/j.jtbi.2015.10.040)
31. Sherratt JA, Murray JD. 1990 Models of epidermal wound healing. *Proc. R. Soc. Lond. B* **241**, 29–36. (doi:10.1098/rspb.1990.0061)
32. Sibert JR, Hampton J, Fournier DA, Bills PJ. 1999 An advection-diffusion-reaction model for the estimation of fish movement parameters from tagging data, with application to skipjack tuna (*Katsuwonus pelamis*). *Can. J. Fish. Aquat. Sci.* **56**, 925–938. (doi:10.1139/f99-017)
33. Banks H, Catenacci J, Hu S. 2016 Use of difference-based methods to explore statistical and mathematical model discrepancy in inverse problems. *J. Inverse Ill-posed Probl.* **24**, 413–433. (doi:10.1515/jiip-2015-0090)
34. Dierckx P. 1981 An algorithm for surface-fitting with spline functions. *IMA J. Numer. Anal.* **1**, 267–283. (doi:10.1093/imanum/1.3.267)
35. Kingma DP, Ba JL. 2014 Adam: a method for stochastic optimization. (https://arxiv.org/abs/1412.6980)
36. Rish I, Grabarnik GY. 2015 *Sparse modeling: theory, algorithms, and applications*. Boca Raton, FL: CRC Press.
37. Zhang T. 2009 Adaptive forward-backward greedy algorithm for sparse learning with linear models. In *NIPS'08: Proc. of the 21st Int. Conf. on Neural Information Processing Systems, Vancouver, Canada, 8–11 December 2008*, pp. 1921–1928. Red Hook, NY: Curran Associates, Inc.
38. Anders U, Korn O. 1999 Model selection in neural networks. *Neural Netw.* **12**, 309–323. (doi:10.1016/S0893-6080(98)00117-8)
39. Mackay DJC. 1995 Probable networks and plausible predictions—a review of practical Bayesian methods for supervised neural networks. *Network Comput. Neural Syst.* **6**, 469–505. (doi:10.1088/0954-898X_6_3_011)
40. Stone MH. 1948 The generalized Weierstrass approximation theorem. *Math. Mag.* **21**, 237. (doi:10.2307/3029337)
41. Smith RC. 2013 *Uncertainty quantification: theory, implementation, and applications*. Philadelphia, PA: Society for Industrial and Applied Mathematics.
42. Murray JD. 2002 *Mathematical biology I. An introduction*, vol. 17. Interdisciplinary Applied Mathematics, 3rd edn. New York, NY: Springer New York.
43. Burgess AN. 1995 Non-linear model identification and statistical significance tests and their application to financial modelling. In *Proc. 4th Int. Conf. on Artificial Neural Networks, Cambridge, UK, 26–28 June1995*, pp. 312–317. London, UK: Institution of Engineering and Technology.