
DEEPMOD: DEEP LEARNING FOR MODEL DISCOVERY IN NOISY DATA

Gert-Jan Both
CRI Research
Universite Paris Descartes
Paris, France

Subham Choudhury
CRI Research
Universite Paris Descartes
Paris, France

Pierre Sens
Institut Curie, CNRS UMR168
PSL Research University
Paris, France

Remy Kusters*
CRI Research
Universite Paris Descartes
Paris, France
`remy.kusters@cri-paris.org`

February 25, 2021

ABSTRACT

We introduce DeepMoD, a Deep learning based Model Discovery algorithm. DeepMoD discovers the partial differential equation underlying a spatio-temporal data set using sparse regression on a library of possible functions and their derivatives. A neural network is used as function approximator and its output is used to constructs the function library, allowing to perform the sparse regression *within* the neural network. This construction makes it extremely robust to noise, applicable to small data sets, and, contrary to other deep learning methods, does not require a training set. We benchmark our approach on several physical problems such as the Burgers', Korteweg-de Vries and Keller-Segel equations, and find that it requires as few as $\mathcal{O}(10^2)$ samples and works at noise levels up to 75%. Motivated by these results, we apply DeepMoD directly on noisy experimental time-series data from a gel electrophoresis experiment and find that it discovers the advection-diffusion equation describing this system.

Keywords Model discovery · Deep learning · Sparse regression

1 Introduction

Recently, efforts have been made to combine data-driven science with bottom up physical modelling in a new field known as "theory-guided data science" (1). Integrating first-principle models with data science has already proven successful in material science (2), earth science (3–6) and fluid mechanics (7, 8). This approach has proven useful to infer coefficients of known PDEs from artificial data, the so called Physics-Informed Neural Networks (PINNs) (9–12), and even to directly discover physical models from artificial data, i.e., PDE-NET (13, 14), PDE-Stride (15) and PDE-Find (16, 17).

The problem of data-driven model discovery of PDEs has been approached from several different directions. While information theory provides a rigorous basis for model selection, it becomes computationally infeasible to compare the information criteria of a vast amount of candidate models (18). Alternatively, an approach to discover a PDE from a spatio-temporal data-set is to use sparse regression model selection schemes such as PDE-FIND as proposed by (16, 17). In this approach, the PDE underlying a dataset $u(\{x, t\})$ is discovered by writing the model discovery task as a regression problem,

$$\partial_t u = \Theta \xi, \tag{1}$$

*Corresponding Author.

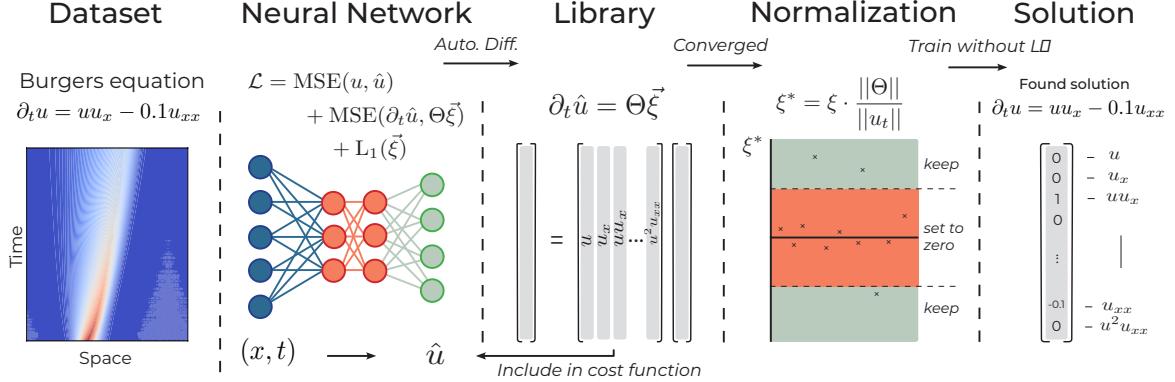


Figure 1: Work-flow of the algorithm: the neural network takes the coordinates of the problem as input and approximates the noisy dataset as output. From this output, the library of candidate terms is constructed and subsequently included in the regression part of cost function of the network, which consists of the MSE-loss, a regression loss and an L_1 regularization. Once the neural network has converged or reached the maximal numbers of allowed iterations, the elements of $\vec{\xi}$ are normalized and thresholded. The resulting sparsity pattern is then used to train the network one final time without L_1 penalty to find the unbiased coefficients.

where Θ is a matrix containing a library of polynomial and spatial derivative functions (e.g. u, u_x, uu_x). Here model discovery turns into finding a sparse representation of the coefficient vector ξ . Rudy et al. (16) introduce the regression algorithm TrainSTridge to solve this task on artificial data such as the Burgers' equation in hydrodynamics, the Schrödinger equation in quantum mechanics and the Kuramoto-Sivashinsky equation in chaos theory (16). Although very promising, this method is sensitive to noise and requires a large number of samples. This can largely be traced back to errors in the numerical differentiation and hence inaccurate derivatives in the library Θ . These traits essentially render the method unfeasible on noisy experimental data. To overcome this, we propose to use automatic differentiation within the neural network to accurately calculate the derivatives in the library function. Indeed, Rudy et al. (16) recognize this possibility to improve the performance of existing model discovery methods such as SINDY (18) and PDE-find (16). A first approach would be to use a neural network to learn the mapping of the data, i.e. $\{x, t\} \rightarrow u$, and then employ automatic differentiation to accurately calculate the derivatives of u with respect to x and t , which can then be used to construct Θ . Unfortunately, this implementation is susceptible to overfitting noisy data, which significantly decreases the accuracy of the library.

The novelty of our work is that we circumvent this ubiquitous issue by implementing Eq. 1 *within the cost function of the neural network*. Consequently, training the network not only adjusts the weights and biases of the network, but also adjusts the components of the sparse vector ξ , corresponding to Eq. 1. An L_1 term on ξ is added to the cost function to ensure its sparsity. Training the neural network yields the underlying PDE and denoises the data set. We show that this approach outperforms state-of-the-art methods of model selection (12, 16) by applying it on artificial data sets of the Burgers', Korteweg-de Vries (KdV), 2D advection diffusion and the Keller-Segel equations (See SI for a comparison). Finally we demonstrate that DeepMoD can discover the PDE underlying an electrophoresis experiments and discover the 2D advection diffusion equation. This shows that this deep learning based selection algorithm can consistently discover the second order advection diffusion equation directly from a simple time-series of images of a diffusing dye.

Methods

Our goal is to develop a fully-automated procedure which discovers the partial differential equation (PDE) underlying a measured data set. Given a data set $u(\{x, t\})$, we can write this problem as,

$$\partial_t u(x, t) = \mathcal{F}(u, u_x, uu_x, u_{xx}, \dots), \quad (2)$$

where we seek the function \mathcal{F} . To find \mathcal{F} , we generate a large set of possible models by considering all permutations of a library of candidate terms. The choice of the library depends on the problem at hand but generally consists of polynomial basis functions and their corresponding spatial derivatives. For example, in the one dimensional examples we present in this paper, the library consists of all polynomials in u up to second order, the derivatives of u with respect

to the coordinates (e.g. $\partial_x u$) up to third order and any possible combinations of these two sets (e.g. $u^2 u_{xx}$), totalling just 12 terms. However, one can construct more than 4000 unique models from this limited set of functions, rendering an information theory approach computationally unfeasible (18).

We circumvent this problem by utilizing a sparse regression approach, in which the model discovery problem is rewritten as

$$\partial_t u = \Theta \xi, \quad (3)$$

where $\partial_t u$ is a column vector of size N containing the time derivative of each sample and Θ contains all M possible terms for each of the samples, so that we can write it as

$$\Theta = \begin{bmatrix} 1 & u(\{x, t\}_0) & u_x(\{x, t\}_0) & \dots & u^2 u_{xx}(\{x, t\}_0) \\ 1 & u(\{x, t\}_1) & u_x(\{x, t\}_1) & \dots & u^2 u_{xx}(\{x, t\}_1) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & u(\{x, t\}_N) & u_x(\{x, t\}_N) & \dots & u^2 u_{xx}(\{x, t\}_N) \end{bmatrix} \quad (4)$$

Since Θ contains significantly more terms than required, most coefficients in ξ will be zero and hence we are looking for a sparse solution of the vector ξ . Note that since we directly discover the governing PDE rather than a closed form of the solution, the differential equation that we discover is independent of the precise boundary conditions of the problem at hand. In the next section, we discuss how this regression task is solved using Lasso, a sparsity promoting regression method, *within a neural network*.

Lasso in neural network

In order to solve the regression task of the previous section we need to construct the function library, Θ . Here we employ a densely-connected feed-forward neural network which takes the spatial and temporal coordinates of the problem, i.e. $\{x, t\}$ as input, and outputs \hat{u} , an approximation of u at $\{x, t\}$ (9, 10). In other words, the neural network approximates the function $u(x, t)$ and employs this approximation to construct the library function, Θ . Using feed-forward neural networks as function approximators has three major advantages, i) they naturally accommodate non-linear constraints, without the need to linearize any operators, ii) they do not require any time-stepping scheme and iii) they allow the use of automatic differentiation to accurately differentiate the output of the neural network with respect to the input coordinates.

The neural network we consider here is trained by optimizing the cost function,

$$\mathcal{L} = \mathcal{L}_{MSE} + \mathcal{L}_{Reg} + \mathcal{L}_{L_1}. \quad (5)$$

Here, \mathcal{L}_{MSE} , is the mean squared error (MSE) of the output of the neural network \hat{u} with respect to the dataset $u(\{x, t\})$,

$$\mathcal{L}_{MSE} = \frac{1}{N} \sum_{i=1}^N |u(\{x, t\}_i) - \hat{u}_i|^2. \quad (6)$$

The last two terms of Eq. 5 correspond to the Lasso regularization: \mathcal{L}_{Reg} performs regression to find the coefficient vector ξ and \mathcal{L}_{L_1} is an L_1 regularizer on ξ . In order to implement the regression problem (Eq. 3) within the neural network, we introduce the regression based cost function,

$$\mathcal{L}_{Reg} = \frac{1}{N} \sum_{i=1}^N |\Theta_{ij} \xi_j - \partial_t \hat{u}_i|^2. \quad (7)$$

Note that the coefficient vector ξ is updated alongside the weights and biases of the neural network, while the terms in Θ are computed from the output of the neural network (i.e. \hat{u}). Automatic differentiation is used to calculate all the spatial and temporal derivatives in Θ , returning machine-precision derivatives. This approach is considerably more accurate than any form of numerical differentiation. Moreover, \mathcal{L}_{Reg} acts as a regularizer on \hat{u} , preventing overfitting of the noisy data set, even though our library contains a large amount of terms (See Results and Fig. 2c).

Finally, an L_1 regularization on the vector ξ is added to ensure its sparsity,

$$\mathcal{L}_{L_1} = \lambda \sum_{i=2}^M |\xi^i|. \quad (8)$$

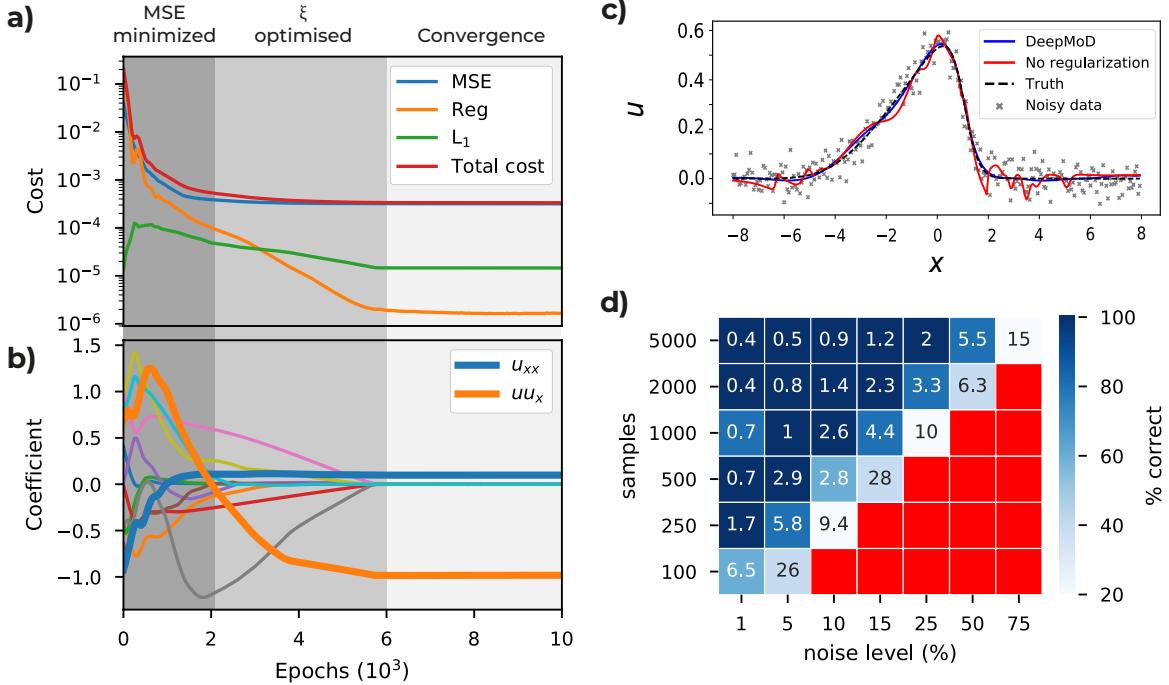


Figure 2: (a) Cost functions and (b) coefficient values, ξ , as function of the number of epochs for the Burgers' dataset consisting of 2000 points and 10% white noise. Initially, the neural network optimizes the MSE and only *after* the MSE is converged the coefficient vector is optimized by the network. (c) Velocity field u for $t = 5$ obtained after training *with* (no overfitting) and *without* (overfitting) the regression regularization \mathcal{L}_{Reg} . (d) The values in the grid indicate the accuracy of the algorithm tested on the Burgers' equation, defined as the mean relative error over the coefficients, as function of the sample size of the data set and level of noise. The coloring represents the fraction of correct runs (Red indicates that in none of the five iterations the correct PDE is discovered).

Here λ is a constant setting the strength of the regularization (further discussed in the SI).

The total cost of the neural network is then minimized using the Adam optimizer. The combination of the MSE term and the regression term in the cost function constrain the network in such a way that it converges to the right solution. To determine if the network has converged, we introduce a convergence criterion. As we show in Fig. 2(a,b), the MSE converges before ξ does, so that our criterion is based on the convergence of ξ :

$$\max \left(\frac{\partial \mathcal{L}}{\partial \xi_i} \frac{\|\partial_t u\|}{\|\Theta_i\|} \right) < \text{tol.} \quad (9)$$

This criterion states that the maximum value of the gradient of the loss function with respect to the coefficients must be smaller than a given tolerance. Note here that we have scaled the gradients as we discuss in the next paragraph. Since it is not guaranteed the network will reach this tolerance, we train the network until the convergence criterion is satisfied, or for a maximum amount of iterations.

Normalization and thresholding

When the neural network has finished training, we obtain the sparse vector ξ . Despite the L_1 regularization, most terms will be non-zero and hence we need to threshold the small coefficients to obtain the true sparse representation. Since each term has different dimensions, Eq. 2 is rendered dimensionless,

$$\partial_t u \rightarrow \frac{\partial_t u}{\|\partial_t u\|}, \Theta \rightarrow \frac{\Theta}{\|\Theta\|}, \xi \rightarrow \xi \frac{\|\Theta\|}{\|\partial_t u\|}, \quad (10)$$

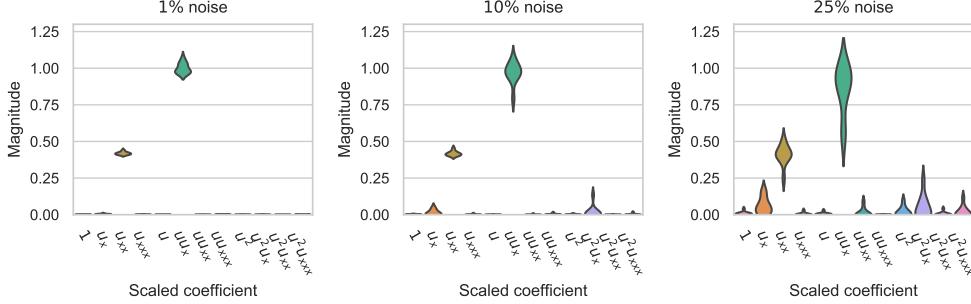


Figure 3: Scaled coefficients before thresholding for the Burgers equation with 1% noise (a), 10% noise (b) and 25% (c). This indicates that there is a considerable difference between the terms that do and do not feature in the PDE.

where $\|\Theta\|$ is the norm of each column of Θ and $\|\partial_t u\|$ the norm of the time-derivative vector. As a result of this transformation, components of ξ will typically be $\mathcal{O}(1)$.

Thresholding prunes components with a negligible impact on the data set by setting all their values to zero. Figure 3 shows the distribution of the scaled coefficients before thresholding for the 1%, 10% and 25% noise runs with sample sizes of 2000 and 5000 (corresponding to the data in figure 2D). Figure 3 shows that the required terms (i.e. u_{xx} and uu_x) considerably stand out relative to the others. We show that up to moderate noise levels $< 20\%$ (Fig. a and b) the exact value of the threshold does not significantly impacts which terms are eventually pruned. The difference between the largest non-featuring and smallest featuring terms is typically up to an order of magnitude. For very high noise levels $> 20\%$ levels (See Fig. 3 c) the accuracy of the pruning is much more sensitive to the exact value and more advanced sparsity algorithms could be required to obtain more robust model selection (See Discussion).

We then train the network one final time without L_1 penalty and with the regression term only containing the terms selected in the first cycle, to find an unbiased estimate of the coefficients of the underlying PDE.

Results

We test the performance of DeepMoD on a set of case studies: the Burgers' equation with and without shock, the Korteweg - de Vries equation, the 2D advection-diffusion equation and the Keller-Segel model for chemotaxis. These examples show the ability of DeepMoD to handle (1) non-linear equations, (2) solutions containing a shock wave, (3) coupled PDEs and finally (4) higher dimensional and experimental data.

Non-linear PDEs

We apply DeepMoD to recover various non-linear and higher order differential equations. As examples we consider Burgers' equation (in the SI we the Korteweg-de Vries equation, which contains a third-order derivative). The Burgers' equation occurs in various areas of gas dynamics, fluid mechanics and applied mathematics and is evoked as a prime example to benchmark model discovery (14, 16) and coefficient inference algorithms (9, 10, 12), as it contains a non-linear term as well as second order spatial derivative,

$$\partial_t u = -uu_x + \nu u_{xx}. \quad (11)$$

Here ν is the viscosity of the fluid and u its velocity field. We use the dataset produced by Rudy et al. (16), where $\nu = 0.1$. The numerical simulations for the synthetic data were performed on a dense grid for numerical accuracy. DeepMoD requires significantly less datapoints than this grid and we hence construct a smaller dataset for DeepMoD by randomly sampling the results through space and time. From now on, we will refer to randomly sampling from this dense grid simply as sampling. Also note that this shows that our method does not require the data to be regularly spaced or stationary in time. For the data in Fig. 2 we add 10% white noise and sampled 2000 points for DeepMoD to be trained on.

We train the neural network using an Adam optimizer (see SI for details) and plot the different contributions of the cost function as a function of the training epoch in Fig. 2a and we show the value of each component of ξ as a function of the training epoch in Fig. 2b. Note that for this example, after approximately 2000 epochs, the MSE is converged, while at the same time we observe the components of ξ only start to converge after this point. We can thus identify three 'regimes': in the initial regime (0 - 2000 epochs), the MSE is trained. Since the output of the neural network is far

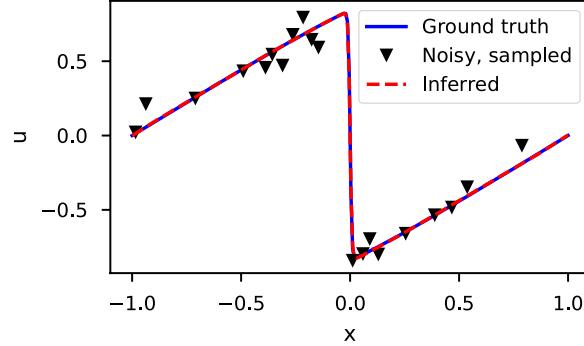


Figure 4: Ground truth, Noisy and Inferred data at $t = 0.8$ for the Burgers' equation with a shock wave (10% noise and 2000 sample points).

from the real solution, so is Θ , and the regression task cannot converge (See first 2000 epochs in Fig. 2b). After the MSE has converged, \hat{u} is sufficiently accurate to perform the regression task and ξ starts to converge. After this second regime (2000 - 6000 epochs), all components of the cost converged (>6000 epochs) and we can determine the solution. From this, we obtain a reconstructed solution (see Fig. 2b) and at the same time recover the underlying PDE, with coefficients as little as 1% error in the obtained coefficients. We show the impact of including the regression term in the cost function in Fig. 2c, where the obtained solution of DeepMoD is compared with a neural network, solely trained on the MSE, to reconstruct the data. Including the regression term regularizes the network and prevents overfitting, despite the many terms in the library. We conclude that it is the inclusion of the regression in the neural network which makes DeepMoD robust to noisy data and prevents overfitting.

Next, we characterize the robustness of DeepMoD in Fig. 2d, where we run DeepMod for five times (differently sampled data set) for a range of sample sizes and noise levels. The color in Fig. 2d shows how many of the five runs return the correct equation and the value in the grid displays the mean error over all correct runs. Observe that at vanishing noise levels, we recover Eq. 11 with as little as 100 data-points, while for 5000 data points we recover the PDE with noise levels of up to 75%. Between the domain where we recover the correct equation for all five runs and the domain where we do not recover a single correct equation, we observe an intermediate domain where only a fraction of the runs return the correct equation, indicating the importance of sampling (See SI 2 for further discussion).

To benchmark DeepMoD, we can directly compare the performance of our algorithm with respect to two state of the art methods, (i): PDE-Find by Rudy et al. (16) and (ii) PDE-Stride by Maddu et al. (15). We considered an identical Burgers' data set and for 10^5 data points, approach (i) recovers the correct equation for up to 1% Gaussian noise (16) while method (ii) discovers the correct equation up to 5% noise (15). Compared to the results in Fig. 2d we note that even for two orders of magnitude fewer samples points, 10^3 w.r.t. 10^5 , DeepMoD recovers the correct equation up to noise levels $> 50\%$ Gaussian noise. DeepMoD allows up to two orders of magnitude higher noise-levels and smaller sample sizes with respect to state-of-the-art model discovery algorithms. The reason DeepMoD is considerably more robust w.r.t. noise and sample size is two-fold: (i) numerical differentiation or denoising requires a relatively fine sampling grid to accurately approximate the derivatives present in the library function. (ii) Since the functions in the library of the neural network are calculated with respect to the inferred solution, our approach is considerably less sensitive to elevated noise levels (See Fig. 2c). We show in the SI that DeepMod has similar performance for the KdV equation, which contains a third order spatial derivative.

Shock wave solutions

If the viscosity is too low, the Burgers' equation develops a discontinuity called a shock (See Fig. 4). Shocks are numerically hard to handle due to divergences in the numerical derivatives. Since DeepMoD uses automatic differentiation we circumvent this issue. We adapt the data from Raissi et al (10), which has $\nu = 0.01/\pi$, sampling 2000 points and adding 10 % white noise (See Fig. 4). We recover ground truth solution of the Burgers' equation as well as the corresponding PDE,

$$\partial_t u = -0.99uu_x + 0.0035u_{xx}, \quad (12)$$

with a relative error of 5% on the coefficients. In Fig. 4 we show the inferred solution for $t = 0.8$.

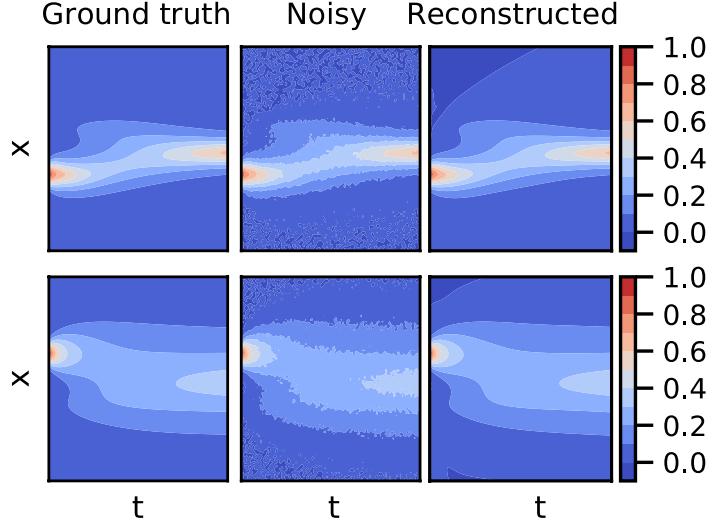


Figure 5: Ground truth, noisy and reconstructed solutions for the density of cells, u (top row) and the density of secreted chemicals w (bottom row) in the Keller Segel model for 5 % white noise and 10000 samples.

Coupled differential equations

Next, we apply DeepMod to a set of coupled PDE's in the form of the Keller-Segel (KS) equations, a classical model for chemotaxis (19, 20). Chemotactic attraction is one of the leading mechanisms that accounts for the morphogenesis and self-organization of biological systems. The KS model describes the evolution of the density of cells u and the secreted chemical w ,

$$\begin{aligned} \partial_t u &= \nabla \cdot (D_u \nabla u - \chi u \nabla w) \\ \partial_t w &= D_w \Delta w - kw + hu. \end{aligned} \quad (13)$$

Here the first equation represents the drift-diffusion equation with a diffusion coefficient of the cells, D_u and a chemotactic sensitivity χ , which is a measure for the strength of their sensitivity to the gradient of the secreted chemical w . The second equation represents the reaction diffusion equation of the secreted chemical w , produced by the cells at a rate h and degraded with a rate k . For a 1D system, we sample 10000 points of u and w for parameter values of $D_u = 0.5$, $D_v = 0.5$, $\chi = 10.0$, $k = 0.05$ and $h = 0.1$ and add 5 % white noise. We choose a library consisting of all spatial derivatives (including cross terms) as well as first order polynomial terms, totalling 36 terms. For these conditions we recover the correct set of PDEs,

$$\begin{aligned} \partial_t u &= 0.50u_{xx} - 9.99uw_{xx} - 10.02u_xw_x \\ \partial_t w &= 0.48w_{xx} - 0.049w + 0.098u, \end{aligned} \quad (14)$$

as well as the reconstructed fields for u and w (See Fig. 5). Note that even the coupled term, u_xw_x , which becomes vanishingly small over most of the domain, is correctly identified by the algorithm, even in the presence of considerable noise levels.

Experimental data

To showcase the robustness of DeepMod on high-dimensional and experimental input data, we consider a 2D advection diffusion process described by,

$$\partial_t u = -\nabla \cdot (-D\nabla u + \vec{v} \cdot u), \quad (15)$$

where \vec{v} is the velocity vector describing the advection and D is the diffusion coefficient. In the SI we apply *DeepMod* on a simulated data-set of Eq. 15, with as initial condition, a 2D Gaussian with $D = 0.5$ and $\vec{v} = (0.25, 0.5)$. For as little as 5000 randomly sampled points we recover the correct form of the PDE as well as the vector \vec{v} for noise levels up to $\approx 25\%$. In the absence of noise the correct equation is recovered with as little as 200 sample points through space and time (See SI 2). This number is surprisingly small considering this is an 2D equation.

Finally, we apply DeepMoD on a time-series of images from an electrophoresis experiment, tracking the advection-diffusion of a charged purple loading dye under the influence of a spatially uniform electric field (See SI for further details). In Fig. 6a we show time-lapse images of the experimental setup where we measure the time-evolution of two initial localised purple dyes. Fig. 6b shows the resultant 2D density field for three separate time-frames (in arbitrary units), corresponding to the red square in Fig. 6a by subtracting the reference image (no dye present). The dye displays a diffusive and advective motion with constant velocity v , which is related to the strength of the applied electric field.

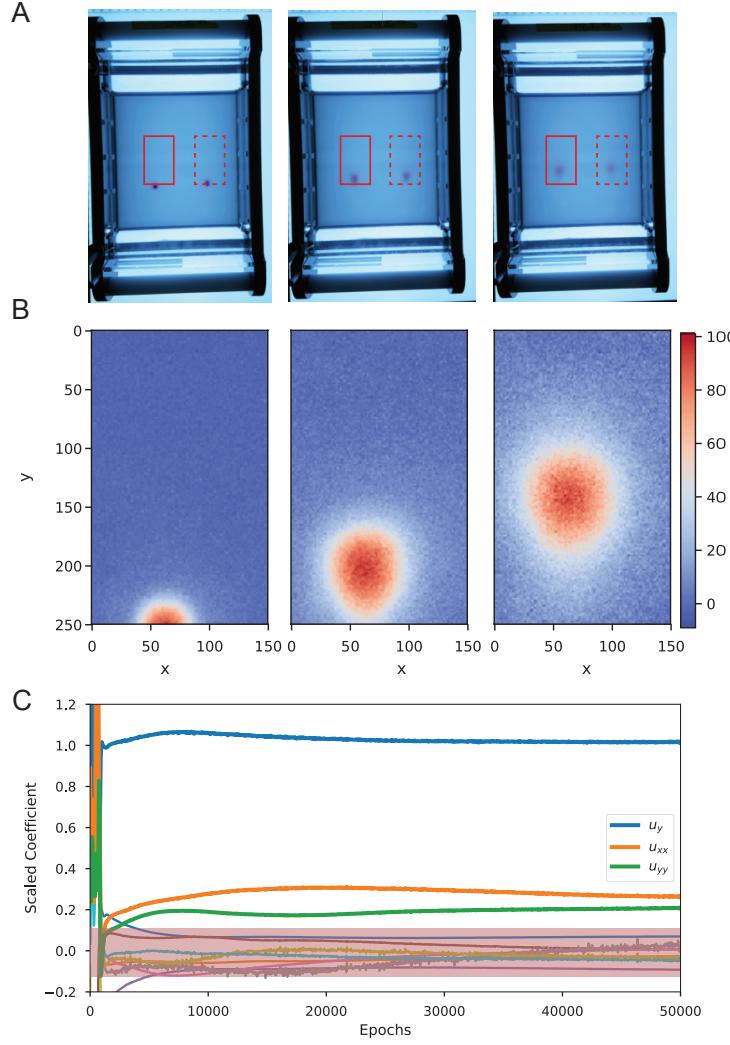


Figure 6: **(a)** Time-series images of the electrophoresis essay. The two red boxes indicate the analysed region. **(b)** Region indicated in the solid red box of (a) showing the density of the dye at three different time-frames (in pixels). **(c)** Scaled coefficients values of all the candidate for a single run. The pink region indicates the terms with scaled coefficient $|\xi| < 0.1$.

We apply DeepMoD on 5000 sampled data-points sampled through space and time and consistently recover the advection term u_y as well as the two diffusive components (u_{xx} and u_{yy}). In Fig. 6c we show the scaled coefficients as function of the number of epochs. After thresholding the scaled coefficients ($|\xi| < 0.1$), we obtain for three unscaled coefficients, the resultant advection diffusion equation,

$$0.31u_y + 0.011u_{xx} + 0.009u_{yy} = 0. \quad (16)$$

Analysing the second diffusing dye (dashed box in Fig. 6) result in nearly identical values for the drift velocity, $v \approx 0.3$, and the diffusion coefficients, $D \approx 0.01$ (See SI) indicating the robustness of the obtained value of D and v . In contrast to the artificial data presented in previous paragraphs, some higher-order non-linear terms, in particular uu_{yy} and uu_{xx}

remain small, yet non-zero. This suggests that an automatic threshold strategy may not guarantee the desired sparse solution. Fixing a threshold of the scaled coefficients ($|\xi| < 0.1$ in this particular case) or other thresholding strategies such as coefficient cluster detection would be better suited for this task.

Discussion

In this paper we presented *DeepMoD*, a novel model-discovery algorithm, utilising neural networks to discover the underlying PDE of a spatio-temporal dataset. We demonstrate the algorithm on 5 artificially obtained case studies: Burgers' (with/without shock), Korteweg-de Vries, advection diffusion and Keller-Segel equations as well as on an experimental data-set of the advection diffusion equation. In contrast to many of the state of the art model discovery algorithms DeepMoD is very robust with respect to elevated noise levels and resilient to small data set sizes, demonstrating an automated model selection task directly from an experimental obtained time-series measurement. DeepMoD allows higher dimensional input/output as well as coupled PDEs as demonstrated with the 2D advection diffusion and Keller-Segel equation.

Through the use of automatic differentiation, combined with a regression-based regularization, the approximation of the spatio-temporal derivatives in noisy data is strongly enhanced. DeepMoD combines two previously established ideas, (i) a regression-based approach to model discovery (pioneered by e.g. Rudy et al. (16, 17)) and (ii) the ability of neural networks to infer system parameters in the context of Physics Informed Neural Networks (Raissi et al. (9, 10, 12)). We show that combining both approaches strongly improves the model discovery task at hand and results in an increased robustness with respect to noise-levels and sample size for model discovery tasks. This approach, for the first time, allows model selections on highly noisy and hence low spatial/temporal resolution experimental data, which to date is one of the prime challenges of this field. DeepMoD also allows to infer the various type of diffusive, chemo-tactic equations, from single particle tracking (SPT) data by following a similar approach as (16), which will advance existing approaches to infer potentially anomalous diffusive processes from SPT data (21, 22).

The success of this approach however strongly relies on (i) the completeness of the library functions in Θ and (ii) a threshold of small yet non-zero terms. (i) If the underlying functions are not present in the library, DeepMoD will not return the correct underlying equation. This problem however can be identified via a cross-validation procedure on a set on a spatial/temporal domain that is not present in the training data. If correct, the resultant equation should perform well outside the spatio-temporal domain of training. Conversely, since we use neural networks as function approximator, the function library, Θ , can be tailored to the problem at hand and thus contain non-linear functions of the network's output, e.g. $\sin u$, $1/(1+u)$. This can be used to model the spatio-temporal evolution of e.g. genetic activation networks (23). We have empirically found that including these extensive libraries does not result in over-fitting the sparse coefficient vector of the data, even though the optimisation contains more degrees of freedom. (ii) While the threshold criterion based on the standard deviation of the coefficient vector provides consistent results throughout the artificial data-sets, this approach understandably fails when either the data is very noisy or when experimental artefacts introduce non-zero contributions of higher order terms. While we have shown in Fig. 3 that up to moderate noise levels the exact value of the threshold does not impact the results significantly, for very high noise levels more advanced sparsity selections algorithms like PDE-find (16) or coefficient clustering schemes would be more appropriate.

Besides the model selection capabilities, DeepMoD demonstrates its usefulness to denoise data and allows accurately approximating derivatives from noisy data, a notoriously difficult task to solve with classical interpolation and finite difference schemes. Employing this "function library based" regulation of neural network architecture may boost the enhancement of e.g. super resolution images through physics informed regularisation (24, 25).

Acknowledgements

Thanks to the Bettencourt Schueller Foundation long term partnership, this work was partly supported by CRI Research Fellowship to Remy Kusters. We acknowledge the support of NVidia through their academic GPU grant program. We thank Jonathan Grizou for his valuable input on the manuscript, Ariel Lindner and Pascal Hersen for suggesting the gel electrophoresis experiments and Thea Chrysostomou for helping with the experiments.

G.B. and R.K. developed the code and tested it on all examples. S.C. helped testing the algorithm on the last example. P.S. and R.K. supervised the project. G.B. and R.K. wrote the paper.

Code and examples available at <https://github.com/PhIMaL/DeePyMoD>.

References

- [1] Anuj Karpatne, Gowtham Atluri, James H Faghmous, Michael Steinbach, Arindam Banerjee, Auroop Ganguly, Shashi Shekhar, Nagiza Samatova, and Vipin Kumar. Theory-guided data science: A new paradigm for scientific discovery from data. *IEEE Transactions on Knowledge and Data Engineering*, 29(10):2318–2331, 2017.
- [2] Nicholas Wagner and James M Rondinelli. Theory-guided machine learning in materials science. *Frontiers in Materials*, 3:28, 2016.
- [3] Anuj Karpatne, William Watkins, Jordan Read, and Vipin Kumar. Physics-guided neural networks (pgnn): An application in lake temperature modeling. *arXiv preprint arXiv:1710.11431*, 2017.
- [4] Markus Reichstein, Gustau Camps-Valls, Bjorn Stevens, Martin Jung, Joachim Denzler, Nuno Carvalhais, et al. Deep learning and process understanding for data-driven earth system science. *Nature*, 566(7743):195, 2019.
- [5] Qingkai Kong, Daniel T Trugman, Zachary E Ross, Michael J Bianco, Brendan J Meade, and Peter Gerstoft. Machine learning in seismology: Turning data into insights. *Seismological Research Letters*, 90(1):3–14, 2018.
- [6] Emmanuel de Bezenac, Arthur Pajot, and Patrick Gallinari. Deep learning for physical processes: Incorporating prior scientific knowledge. *arXiv preprint arXiv:1711.07970*, 2017.
- [7] Romit Maulik and Omer San. A neural network approach for the blind deconvolution of turbulent flows. *Journal of Fluid Mechanics*, 831:151–181, 2017.
- [8] Tharindu P Miyanawala and Rajeev K Jaiman. An efficient deep learning technique for the navier-stokes equations: Application to unsteady wake flow dynamics. *arXiv preprint arXiv:1710.09099*, 2017.
- [9] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part ii): data-driven discovery of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10566*, 2017.
- [10] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017.
- [11] Maziar Raissi. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *arXiv preprint arXiv:1801.06637*, 2018.
- [12] M Raissi, P Perdikaris, and GE Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.
- [13] Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. Pde-net: Learning pdes from data. *arXiv preprint arXiv:1710.09668*, 2017.
- [14] Zichao Long, Yiping Lu, and Bin Dong. Pde-net 2.0: Learning pdes from data with a numeric-symbolic hybrid deep network. *arXiv preprint arXiv:1812.04426*, 2018.
- [15] Suryanarayana Maddu, Bevan L Cheeseman, Ivo F Sbalzarini, and Christian L Müller. Stability selection enables robust learning of partial differential equations from limited noisy data. *arXiv preprint arXiv:1907.07810*, 2019.
- [16] Samuel H Rudy, Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Data-driven discovery of partial differential equations. *Science Advances*, 3(4):e1602614, 2017.
- [17] Samuel Rudy, Alessandro Alla, Steven L Brunton, and J Nathan Kutz. Data-driven identification of parametric partial differential equations. *arXiv preprint arXiv:1806.00732*, 2018.
- [18] Niall M Mangan, Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Inferring biological networks by sparse identification of nonlinear dynamics. *IEEE Transactions on Molecular, Biological and Multi-Scale Communications*, 2(1):52–63, 2016.
- [19] Evelyn F Keller and Lee A Segel. Initiation of slime mold aggregation viewed as an instability. *Journal of theoretical biology*, 26(3):399–415, 1970.
- [20] Pierre-Henri Chavanis. A stochastic keller-segel model of chemotaxis. *Communications in Nonlinear Science and Numerical Simulation*, 15(1):60–70, 2010.
- [21] Mohamed El Beheiry, Maxime Dahan, and Jean-Baptiste Masson. Inferencemap: mapping of single-molecule dynamics with bayesian inference. *Nature methods*, 12(7):594, 2015.
- [22] Naor Granik, Lucien E. Weiss, Maayan Shalom, Michael Chein, Eran Perlson, Yael Roichman, and Yoav Shechtman. Single particle diffusion characterization by deep learning. *bioRxiv*, 2019. doi: 10.1101/588533.
- [23] Anton Crombach, Mónica A García-Solache, and Johannes Jaeger. Evolution of early development in dipterans: reverse-engineering the gap gene network in the moth midge clogmia albipunctata (psychodidae). *Biosystems*, 123:74–85, 2014.

- [24] Jianchao Yang, John Wright, Thomas S Huang, and Yi Ma. Image super-resolution via sparse representation. *IEEE transactions on image processing*, 19(11):2861–2873, 2010.
- [25] Lucas von Chamier, Romain F Laine, and Ricardo Henriques. Artificial intelligence for microscopy: What you should know. 2019.

SUPPLEMENTARY INFORMATION FOR DEEPMOD: DEEP LEARNING FOR MODEL DISCOVERY IN NOISY DATA

Gert-Jan Both
CRI Research
Universite Paris Descartes
Paris, France

Subham Choudhury
CRI Research
Universite Paris Descartes
Paris, France

Pierre Sens
Institut Curie, CNRS UMR168
PSL Research University
Paris, France

Remy Kusters*
CRI Research
Universite Paris Descartes
Paris, France
`remy.kusters@cri-paris.org`

February 25, 2021

In this Supplementary Information we discuss DeepMoD and the synthetic datasets on which we have benchmarked the algorithm. To recapitulate, the core idea behind DeepMoD is to find a sparse representation of the underlying PDE by generating a *library*, Θ , composed of polynomial basis functions with their corresponding spatial derivatives. This essentially reduces the model selection task to finding a sparse vector, ξ , corresponding to,

$$\partial_t u = \Theta \xi, \quad (1)$$

which best fits the data. DeepMoD aims to recover a sparse vector, ξ , using a neural network and by implementing the regression scheme within the network. The algorithm can be divided into three parts: First, a feed-forward neural network approximates the mapping from the input data $\{x, t\}$ to the output $u(\{x, t\})$. The derivatives of $u(\{x, t\})$ are determined from this mapping using automatic differentiation and are used to construct a library containing all possible terms in the PDE. The coefficient vector ξ is obtained by performing Lasso directly within the neural network. After the total loss of the neural network has converged we threshold the resulting weight vector and obtain the sparse vector ξ . The final sparse weight vector only contains non-zero terms corresponding to active terms, i.e. the terms that feature in the PDE, with their corresponding value. In the following sections we discuss the various parts of the algorithm in detail. The full code as well as the data-sets are available on Github: <https://github.com/PhIMaL/DeePyMoD>.

Cost function

A fully connected feed-forward network is used to map the input coordinates of the problem $\{x, \dots, t\}$ to a scalar or vector output $u(\{x, \dots, t\})$. We use the tanh activation function as it is infinitely differentiable; the chosen activation function has to be differentiable at least as many times as the highest order derivative. For example, a ReLu activation would henceforth not work for libraries containing second-order (and higher) derivatives. The neural network learns the mapping by minimizing the total cost function, which is comprised of the mean squared error, \mathcal{L}_{MSE} , a regression based cost function, \mathcal{L}_{Reg} and a sparsity promoting cost function, \mathcal{L}_{L_1} ,

$$\mathcal{L} = \mathcal{L}_{MSE} + \mathcal{L}_{reg} + \mathcal{L}_{L_1} = \frac{1}{N} \sum_{i=1}^N |u(\{x, t\}_i) - \hat{u}_i|^2 + \frac{1}{N} \sum_{i=1}^N |f(\{x, t\}_i)|^2 + \lambda \sum_{i=2}^M |\xi_i| \cdot \frac{\|\Theta_i\|}{\|\partial_t \hat{u}\|} \quad (2)$$

*Corresponding Author.

The cost function related to the mapping $\{x, \dots, t\}$ to a scalar or vector output $u(\{x, \dots, t\})$ is written as,

$$\mathcal{L}_{mse} = \frac{1}{N} \sum_{i=1}^N |u(\{x, t\}_i) - \hat{u}_i|^2, \quad (3)$$

where N is the number of samples. Automatic differentiation is then used to find the derivatives of the outputs with respect to the inputs, e.g. $\partial_t u, u_x, u_{xx}, \dots$. These derivatives are then combined with polynomial basis functions to yield the function library Θ . For a 1D+1 input $\{x, t\}$ and single output \hat{u} , we select a library constructing all possible combinations of spatial derivatives $\partial_x^n u$ up to order n and polynomials u^m up to order m , which, including a constant term, gives a library of $(n+1) \times (m+1) = M$ terms. We implement the regression problem by first rewriting it as $f = \partial_t u - \mathcal{F}(\dots)$ and include it in the cost function by minimizing,

$$\mathcal{L}_{reg} = \frac{1}{N} \sum_{i=1}^N |f(\{t, x\}_i)|^2. \quad (4)$$

To showcase that the algorithm is not sensitive to the number of terms in the library, we plot the evolution of the different terms of ξ for a Burgers' and a Korteweg-de Vries (KdV) data set of 2000 data points and 10 % of white noise in Fig. 1.

Here we increased the amount of library terms up to fourth order derivatives and third order polynomials, and thus a total of 20 terms, compared to 12 in the example in the main text. We still obtain the correct equation with identical accuracy of the obtained coefficients but including higher order derivatives increases the computational time significantly.

Finally, we add a sparsity-promoting component in the form of an L_1 regularization on ξ^* :

$$\mathcal{L}_{L_1} = \lambda \sum_{i=2}^M |\xi_i^*|. \quad (5)$$

Here, ξ_i^* represents the i -th element of the scaled coefficient vector (we discuss the scaling in the next paragraph). Note that the regularization is not applied to the constant term, hence the summation starts from $i = 2$. The parameter λ in front of this cost (See Eq. 2) is set to $\lambda = 10^{-5}$ for all the presented examples. To showcase the importance of including the L_1 regularization however, we show in Fig. 2 the evolution of ξ as function of the epoch for the KdV equation using a dataset of 2000 points and 10% white noise. (a) Shows this without L_1 regularization and (b) including an L_1 term with $\lambda = 10^{-5}$. Essentially it increases the convergence rate of the network: whereas the network with L_1 regularization has converged after roughly 60.000 epochs, the one without is not converged after 100.000 epochs.

Training the Neural network and the role of noise on the MSE

To train the neural network, we minimize the total cost function using the Adam optimizer with slightly adjusted parameters for faster convergence: we use a increased learning rate of 0.002 (except of the Keller Segel where we used the default value: 0.001) and set β_1 to 0.99 (default value: 0.9). We also explored the L-BFGS-B optimizer and found that while it was faster than Adam, it sometimes suffered from convergence issues.

To gain more insight into the minimization of the neural network, we trace the different components of the cost function (i.e. the MSE, regression and L_1 part) as a function of the epoch for the Burgers dataset in Fig. 3 for a data set without noise (a) and a dataset with 5% noise (2000 randomly sampled data points). Panels (a) and (c) contain the absolute cost whereas panels (b) and (d) plot the corresponding relative cost (the neural network consisted of five hidden layers of twenty neurons each and $\lambda = 10^{-5}$).

We find that for both datasets, the network converges after approximately 8000 epochs. For the noiseless case, we observe in panel (b) that the L_1 component makes up the largest fraction of the cost. In contrast, for the noisy case, the total cost is dominated by the MSE. For the noiseless case both MSE and the regression term should in principle approach zero, leaving the L_1 as the main component of the total cost. Given that the L_1 regularization acts on the scaled coefficients, which are on the order $\mathcal{O}(1)$ and that only few components should be significant, the total cost should be of the order of λ : $\mathcal{O}(10^{-5})$, which we indeed observe. In the case of the noisy data set however, the regression prevents overfitting and denoises the data. This results in a residual cost of the MSE cost (see panel (d)). In other words, the MSE cost is a measure of the difference between the true underlying and measured dynamics due to the noise. With this measure we can directly estimate the noise level from the MSE by writing the data $u(x, t)$ as the sum of the ground truth u_0 and additive Gaussian noise $\sigma \mathcal{N}(0, 1)$,

$$MSE = \langle (\hat{u} - u_0 - \sigma \mathcal{N}(0, 1))^2 \rangle. \quad (6)$$

If the network learns the *exact* solution, $\hat{u} \rightarrow u_0$, Eq. 6 becomes

$$MSE = \sigma^2 \langle \mathcal{N}^2(0, 1) \rangle = \sigma^2. \quad (7)$$

For panel (c), we obtain a value of $\approx 10^{-4}$, which indeed corresponds to Fig. 3(c). If a network converged at a MSE lower than this value, it would be a sign of overfitting; Eq. 7 is the minimal MSE. We empirically tested the robustness against overfitting by making our network significantly larger (10 layers of 100 neurons) and did not observe overfitting.

To determine if the network converged, we implement a convergence criterion. As stated in the main text, the MSE converges before the coefficients and we thus determine the convergence based on ξ :

$$\max \left(\frac{\partial \mathcal{L}}{\partial w_i} \frac{\|\partial_t u\|}{\|\Theta_i\|} \right) < \text{tol}. \quad (8)$$

We calculate the derivative of the loss function with respect to ξ and normalize it to fairly compare the gradients. We then require the maximum value of this vector to be smaller than some tolerance, as all components have to be converged. Typically, we choose $\text{tol} = 10^{-6}$. Note however that while convergence is typically reached within ± 25000 epochs it can sometimes take considerably longer for all the coefficients to properly converge, especially when the function library is very large (See e.g. the example in Fig. 1 for the KdV equations with an extended library).

Scaling and thresholding

After the network has converged we obtain the vector ξ . Despite the L_1 regularization, many terms are close to but non-zero. We thus need to threshold ξ in order to obtain the true sparse vector. Since each component has different units and possibly different absolute scales, we normalize the components before thresholding. We calculate the norm of each column in Θ and the norm of the time derivative and scale Θ and ξ accordingly:

$$\begin{aligned} \Theta &\rightarrow \Theta^* \cdot \|\Theta\|^{-1}, \\ \xi &\rightarrow \xi^* \frac{\|\Theta\|}{\|\partial_t u\|}, \\ \partial_t u &\rightarrow \partial_t u^* \|\partial_t u\|^{-1} \end{aligned} \quad (9)$$

where $\|\Theta\|$ is a row vector containing the norms of each column in Θ and ξ^* and Θ^* are the normalized weights vector and library. By normalizing accordingly, the regression problem will only consist of unit-length vectors, which allows for an easier interpretation of ξ^* ; typically, it will be $\mathcal{O}(10^{-1} - 10^0)$. The normalization also allows DeepMoD to select terms with different orders of magnitude.

Next, we threshold this scaled vector. Since most components are close to zero, we interpret the thresholding operation as finding outliers from the median. We hence set any component smaller than the standard deviation σ to zero:

$$\xi_i = \begin{cases} \xi_i & \text{if } \xi_i^* > \text{median}(\xi^*) + \sigma(\xi^*) \\ 0 & \text{if } (\text{median}(\xi^*) - \sigma(\xi^*)) < \xi_i^* < (\text{median}(\xi^*) + \sigma(\xi^*)) \\ \xi_i & \text{if } \xi_i^* < \text{median}(\xi^*) - \sigma(\xi^*). \end{cases} \quad (10)$$

Although this criterion is somewhat arbitrary, in all the benchmarks we presented here it performs well. To showcase the importance of the normalization in the thresholding, we plot the normalized and unnormalized values of ξ , obtained from the Burgers equation with shock after 10^5 epochs. For this dataset, the non-linear term and the diffusion term differ two orders of magnitude (1 versus $0.01/\pi$). Observe that due to the small value of the diffusion term it cannot be distinguished from the non-active terms. By normalizing the coefficients however, both components are significantly larger in magnitude relative to all other coefficients.

After the thresholding we have determined the sparsity pattern, i.e. which terms are present in the PDE. Due to our use of L_1 regularization our estimates of the coefficients are not unbiased however. We thus run the network again with only the components of the sparsity pattern and without L_1 , similar to a Physics Informed Neural Network.

Computational cost

The computational cost of DeepMoD strongly depends on e.g. the size of the data set, the order of the derivatives, the hardware (CPU or GPU) and optimisation options. A typical run needs 5 - 10 minutes; on a machine with a CPU (1950X AMD Threadripper) with 16GB RAM took 3 minutes on a data set of 1000 points and 10 minutes on a data set of 5000 points for a third-order library. The increase in computational cost due to the increase in data set size can be circumvented by using a GPU, as long as the whole data set and model can be loaded into its memory. The bulk of the computational costs come from calculating the library, which is re-evaluated every single iteration. Higher order derivatives require subsequent backwards passes through the neural network, which is computationally expensive. Nonetheless, improved optimiser settings, smarter initialisation and potentially batching could further reduce the computational time.

Synthetic data

DeepMoD was tested on five synthetic case studies: the Burgers' equation with and without shock formation, the Korteweg-de Vries equation (KdV), the 2D advection diffusion equation and the Keller-Segel equations. To allow a direct comparison with existing algorithms, we used the data-set provided in Rudy et al. for the Burgers' equation without shock and the KdV equation. For the Burgers' dataset with shock formation, we consider the dataset of Raissi et al. and for the 2D advection diffusion and the Keller-Segel equations we generate the data using standard numerical solvers in Mathematica.

In this SI we focus our attention on the performance of the algorithm to discover the correct PDE with respect to the level of (white) noise and the dataset size. We add white noise (Gaussian) on the scale of the standard deviation, χ , (typically in a range between 5% and 75%),

$$y_{noisy} = y + \chi\sigma(y)\mathcal{N}(0, 1) \quad (11)$$

where \mathcal{N} is a Gaussian distribution.

Burgers' equation

The Burgers' equation in one dimension is given by

$$\partial_t u + uu_x - \nu u_{xx} = 0, \quad (12)$$

where u is the velocity of the fluid and ν is the viscosity of the fluid. Here we consider $\nu = 0.1$ and we obtain a solution without a shock wave. We use the data-set presented in Rudy et al where the equation is solved for 101 time points between 0 and 10 and 256 spatial points between -8 and 8, given a total amount of 25856 points. N points are randomly sampled and $\chi\%$ white noise is added (we set the maximal number of epochs to 50000). In Fig. 5(a) we show, for one specific dataset of 2000 samples and 20% of noise the ground truth, noisy and reconstructed equation. In Fig. 5(b,c) we show for 5 randomly sampled datasets the accuracy of the algorithm as function of (left) the noise level (at $N = 500$) and (right) number of sample points (at vanishing noise levels). The number inside each square and its colour represent the mean relative error in the coefficients, if the right PDE was found. Red squares indicate the correct equation was not recovered.

Korteweg-de Vries

We consider the Korteweg-de Vries (KdV) equation as a second benchmark problem. It describes waves in shallow water and is of particularl interest for model selection due to its third order term,

$$\partial_t u = 6uu_x - u_{xxx}. \quad (13)$$

We use the data-set of Rudy et al. for 201 time steps between 0 and 20 and spatial resolution of 512 cells between -30 and 30, giving a total dataset size of 120912. We randomly sample 2000 points and add 20% white noise, resulting in Fig. 6(a). This example shows that even for higher order equations the algorithm is capable of recovering the correct PDE. While for 2000 sample points we recover the correct PDE we observe in Fig. 6(b,c) that this can strongly depend on the sampling (we set the maximal number of epochs to 100.000). The performance with respect to the number of sample points is slightly lower compared to the Burgers' equation due to the localized solution: the area outside the two waves is close to zero and hence does not contain information on the solution. These effects could be overcome by more advanced sampling such as latin hypercube sampling.

Higher dimensional input

To showcase the robustness of DeepMoD on high-dimensional input data, we consider the 2D advection diffusion process described by,

$$\partial_t u = -\nabla \cdot (-D \nabla u + \vec{v} \cdot u), \quad (14)$$

where \vec{v} is the velocity vector describing the advection and D is the diffusion coefficient. Our initial condition is a 2D Gaussian and set $D = 0.5$ and $\vec{v} = (0.25, 0.5)$. For as little as 5000 randomly sampled points we recover the correct form of the PDE as well as the vector \vec{v} for noise levels up to $\approx 25\%$ (See SI 2). Our library consists of all spatial derivatives (including cross terms) as well as first order polynomial terms, totalling 12 terms. At vanishing noise levels we obtain the correct equations with as little as 200 sample points through space and time. This number is surprisingly small considering this is an 2D equation. To illustrate this, Fig. 8 shows the ground truth, sampled points with 10% noise and the reconstructed profile at two different times. Note that while it is impossible to identify the underlying process by sight, the algorithm correctly discovers the underlying PDE. After the network learned the mapping $x, t \rightarrow u$ on 5000 sample points, we can reconstruct the solution on the entire domain (Fig. 8 right), highlighting the ability of DeepMoD to act as a highly-accurate physics-informed interpolater and denoiser.

Advection-Diffusion equation

To model the temporal evolution of a concentration field in two-dimensions, subject to advection-diffusion, we numerically solve

$$\partial_t u = -\nabla \cdot (-D \nabla u + \vec{v} \cdot u), \quad (15)$$

where \vec{v} is the velocity vector describing the advection and D is the diffusion coefficient. We consider a Gaussian initial condition $u(x, y)|_{t=0} = 20 \exp - (x^2 + y^2)$ over time and space where we set $D = 0.5$ and $\vec{v} = (0.25, 0.5)$. For 5000 data-points randomly sampled through space (41 time-frames) we recover the correct form of the PDE as well as the vector \vec{v} for noise levels up to $\approx 25\%$ (See Fig. 7). At vanishing noise levels we obtain the correct equations with as little as 200 sample points through space and time. This number is surprisingly little considering this is an 2D equation, with respect to the 1D Burgers' and KdV equations (See Fig. 7).

Gel electrophoresis

0.1 Advection-Diffusion

To test the performance of DeepMoD on experimental data we perform a gel electrophoresis experiment with a (purple) charged loading dye ($10\mu l$), inserted inside an agarose gel (1g agarose /100ml TAE buffer) and tracked in an electrophoresis assay. A low voltage of 12 V is applied (lowest voltage the power-supply allows). For this voltage, the electric field drives the charged loading dye with a constant velocity through the gel. For this low voltage, the advective motion due to the electric field is accompanied by a diffusive motion of the dye in the gel which can both be observed in Fig. 5 of the main text.

We analyse the density profile of two initially localised dyes (See Fig. 5 in the main text) to infer the underlying advection-diffusion equation (Eq. 14, using a library of 12 candidate terms, similarly as for the artificial advection diffusion data. The neural network we used contained 3 hidden layers of 20 neurons ($\lambda = 10^{-5}$). 5000 data-points are randomly sampled through space ($x = 300$ pixels in the direction of advection and $y = 150$ pixels perpendicular) and time (50 time-frames with 10 min interval). As discussed in the main text, an automatic threshold of the coefficient vector does not render a sparse solution due to small yet non-zero higher order terms. Therefore we fix the threshold to $|\xi| < 0.1$. In Table. S1 we present the resultant equation for both experiments with five different sampling seeds. In 8/10 cases the correct equation is obtained, while in 2 cases two small, yet non-zero additional uu_x, uu_y terms are found. To illustrate the robustness of the algorithm with respect to the precise architecture we show in Table SI2 that increasing or decreasing the number of layers in the neural network did not yield different results (except for one single layer, which is not sufficient to serve as function approximator).

Electrophoresis experiment		
Run	Experiment 1 (solid line)	Experiment 2 (dashed line)
1	$u_t = 0.31u_y + 0.011u_{xx} + 0.009u_{yy}$	$u_t = 0.28u_y + 0.007u_{xx} + 0.016u_{yy} + 0.001uu_{xx} - 0.002uu_{yy}$
2	$u_t = 0.30u_y + 0.009u_{xx} + 0.009u_{yy}$	$u_t = 0.28u_y + 0.009u_{xx} + 0.009u_{yy}$
3	$u_t = 0.31u_y + 0.010u_{xx} + 0.010u_{yy}$	$u_t = 0.29u_y + 0.008u_{xx} + 0.011u_{yy}$
4	$u_t = 0.31u_y + 0.010u_{xx} + 0.011u_{yy}$	$u_t = 0.29u_y + 0.009u_{xx} + 0.011u_{yy}$
5	$u_t = 0.31u_y + 0.007u_{xx} + 0.007u_{yy}$	$u_t = 0.29u_y + 0.008u_{xx} + 0.015u_{yy} + 0.001uu_{xx} - 0.002uu_{yy}$

Table 1: Equations obtained for the two electrophoresis experiment (See Fig. 5 in the main text) for five different seeds. The red terms that are indicated are the terms that in 2/10 cases are incorrectly retained after thresholding the coefficient vector.

Impact of neural network architecture	
Layers	Result
1	$u_t = 0.31u_y + 0.19 - 0.16u$
2	$u_t = 0.30u_y + 0.009u_{xx} + 0.008u_{yy}$
3	$u_t = 0.30u_y + 0.009u_{xx} + 0.009u_{yy}$
4	$u_t = 0.30u_y + 0.009u_{xx} + 0.009u_{yy}$
5	$u_t = 0.30u_y + 0.009u_{xx} + 0.009u_{yy}$

Table 2: Equations obtained for the two electrophoresis experiment (See Fig. 5 in the main text) for networks with different amount of layers consisting of 20 neurons with a tanh activation function. The red terms are incorrect. Note that for at least two layers, the size of the network has negligible effect on the results.

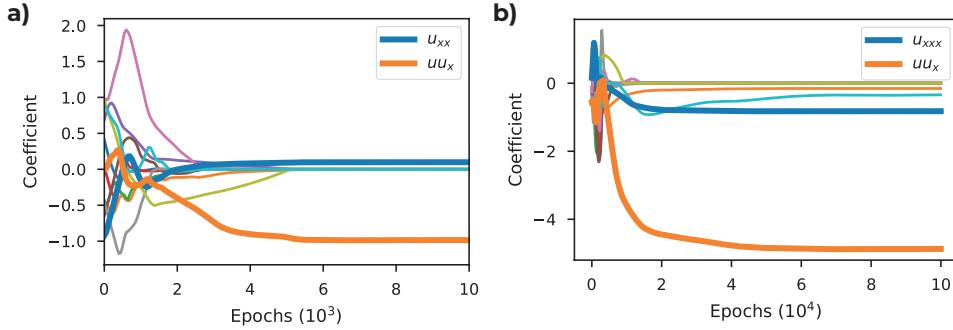


Figure 1: Evolution of unscaled components of ξ for a library containing fourth order derivatives and third order polynomial. (a) shows the Burgers' equation, (b) the KdV, both for a 2000 point dataset with 10% white noise.

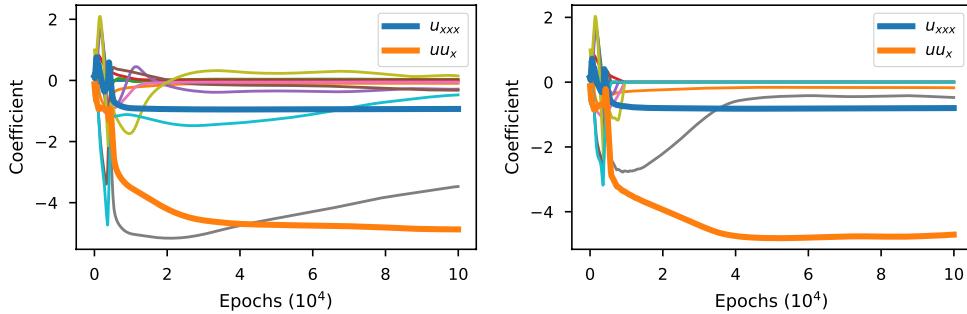


Figure 2: Evolution of unscaled components of ξ for a KdV dataset containing 2000 points with 10% white noise. (a) shows without L_1 regularization (i.e. $\lambda = 0$), (b) shows the evolution with $\lambda = 10^{-5}$.

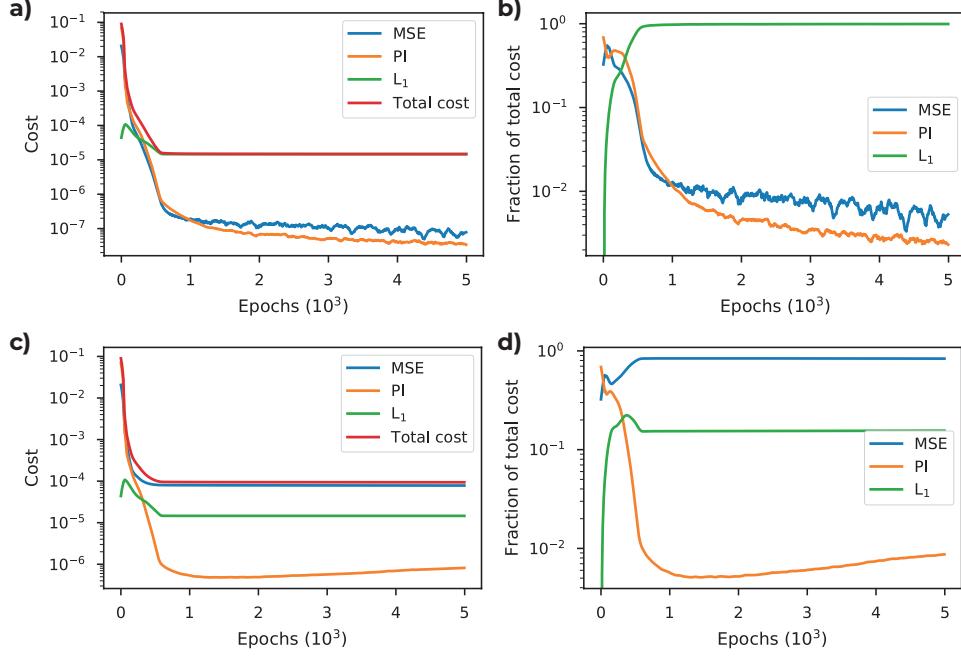


Figure 3: **a)** Absolute cost of each component of the cost function for a noiseless Burgers' dataset of 2000 datapoints. **b)** Relative cost of the Burgers' dataset of panel (a). **c)** Absolute cost of each component of the cost function for a noisy (5% white noise) Burgers' dataset of 2000 datapoints. **d)** Relative cost of the Burgers' dataset of panel (c). Note that these figures were smoothed using a first order Savitzky-Golay filter with a window length of 15.

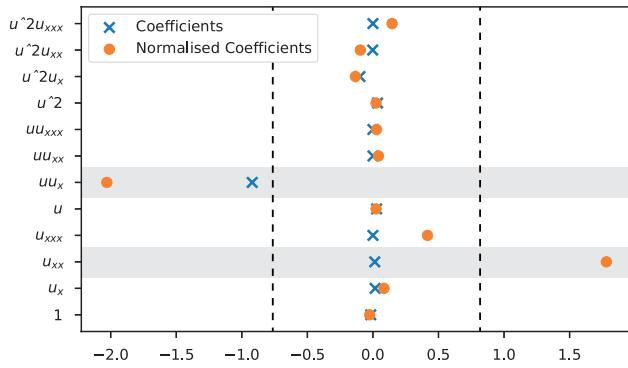


Figure 4: Scaled (orange dots) and unscaled (blue crosses) components of ξ after 10^5 epochs of training on the Burgers with shock dataset of 2000 samples and 10% white noise. The grey bars denote the terms to be discovered and the vertical dashed lines the area which will be thresholded.

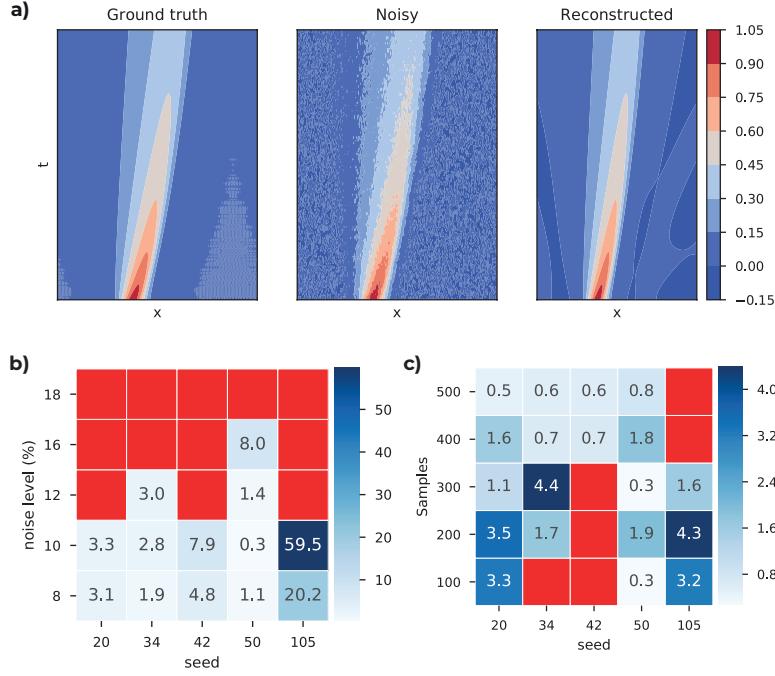


Figure 5: a) Ground truth (left), Noisy (middle) and reconstructed (right) solution for the Burgers' equation. b) Accuracy of the algorithm for the Burgers' equation as function of the noise levels for 500 sample points and c) as function of sample size (vanishing noise levels). Annotation and colors indicate the mean error on the coefficients, if the correct PDE was found. Red squares indicate the right PDE was not found.

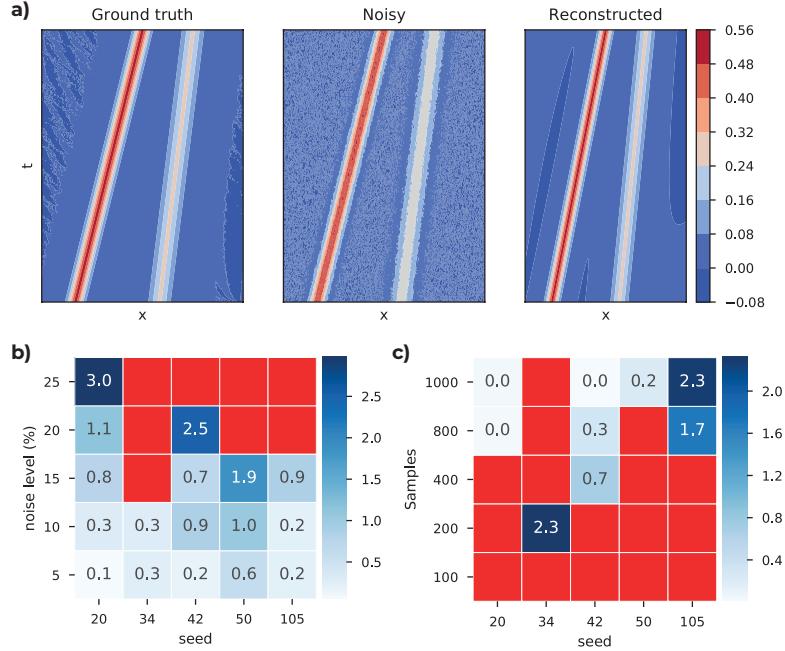


Figure 6: a) Ground truth (left), Noisy (middle) and reconstructed (right) solution for the Korteweg-de Vries equation. Accuracy of the algorithm for the Korteweg - de Vries equation (b) as function of the noise levels for 2000 sample points and (c) as function of sample size (vanishing noise levels). Annotation and colors indicate the mean error on the coefficients, if the correct PDE was found. Red squares indicate the right PDE was not found.

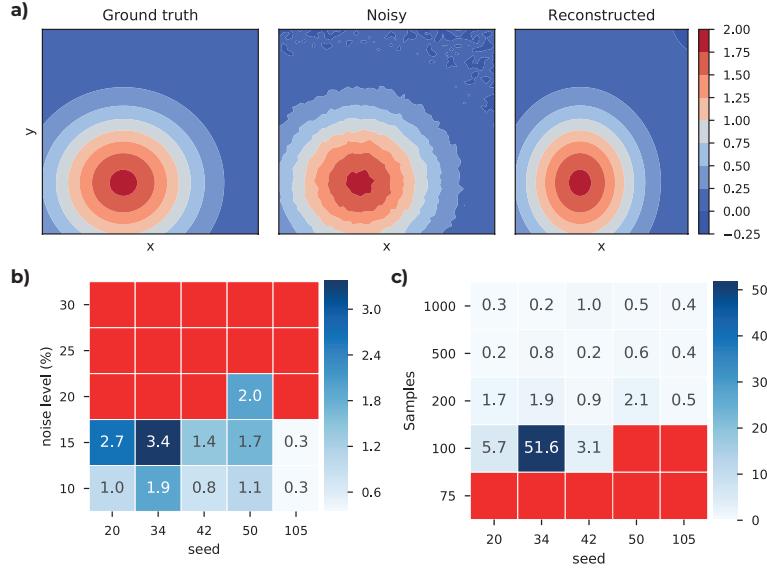


Figure 7: a) Ground truth (left), Noisy (middle) and reconstructed (right) solution for the 2D advection diffusion equation. Accuracy of the algorithm for the 2D advection diffusion equation, (b) as function of noise levels for 5000 sample points and (c) as function of sample size (vanishing noise levels). Annotation and colors indicate the mean error on the coefficients, if the correct PDE was found. Red squares indicate the correct PDE was not found.

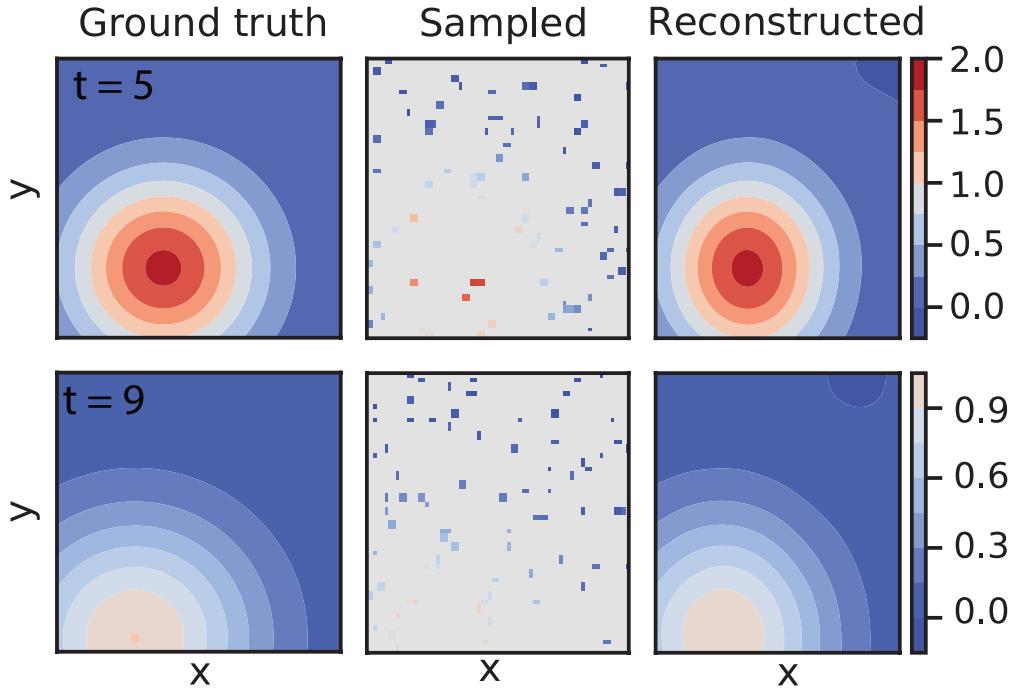


Figure 8: Ground truth (left), Sampled (middle) and reconstructed (right) solution for the 2D advection diffusion equation at $t = 5, 9$ (10 % white noise and 5000 samples). Note that, even for an extremely low sampling rate, the full concentration profile is recovered, as well as the corresponding PDE of the process.