DiffFind: Discovering Differential Equations from Time Series

Lalithsai Posam[1], Shubhranshu Shekhar[2], Meng-Chieh Lee[3], and Christos Faloutsos[3]

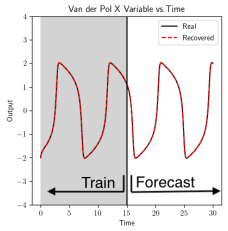
[1] University of California, Berkeley, USA lposam@berkeley.edu
[2] Brandeis University, Waltham, USA sshekhar@brandeis.edu
[3] Carnegie Mellon University, Pittsburgh, USA {mengchil, christos}@cs.cmu.edu

Abstract. Given one or more time sequences, how can we extract their governing equations? Single and co-evolving time sequences appear in numerous settings, including medicine (neuroscience - EEG signals, cardiology - EKG), epidemiology (covid/flu spreading over time), physics (astrophysics, material science), marketing (sales and competition modeling; market penetration), and numerous more. Linear differential equations will fail, since the underlying equations are often non-linear (SIR model for virus/product spread; Lotka-Volterra for product/species competition, Van der Pol for heartbeat modeling).
We propose DiffFind and we use genetic algorithms to find suitable, parsimonious, differential equations. Thanks to our careful design decisions, DiffFind has the following properties - it is: (a) *Effective*, discovering the correct model when applied on real and synthetic nonlinear dynamical systems, (b) *Explainable*, gives succinct differential equations, and (c) *Hands-off*, requiring no manual hyperparameter specification.
DiffFind outperforms traditional methods (like auto-regression), includes as special case and thus outperforms a recent baseline ('SINDy'), and wins first or second place for all *5* real and synthetic datasets we tried, often achieving excellent, zero or near-zero RMSE of *0.005*.

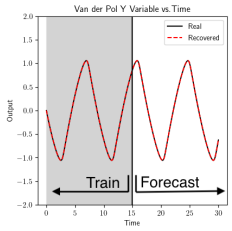
1 Introduction

Given long, co-evolving time sequences (say, flu patient counts, or EEG waves, or product-sales volumes), how to automatically extract the governing differential equations? Earlier methods [2,6,24] heavily rely on feature engineering to select careful features, thus sacrificing generality. Moreover, most of them are not able to handle time delay differential equations. Recent studies [15,23] focus on the forecasting problem and use deep learning models. Thanks to their large number of parameters, those models provide good forecasting, at the expense of explainability, and they are suffering from expensive hyperparameter and network architecture tuning.

Informal Problem 1 *The problem is as follows:*
- ***Given*** *a multivariate time series* $\boldsymbol{X}$ *(*$\boldsymbol{x_1}(t)$*,* $\boldsymbol{x_2}(t)$*, ...;* $t = 1, \ldots$*)*



(a) X Variable forecast



(b) Y Variable forecast

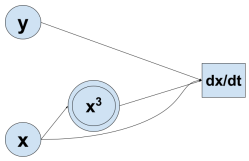
Notice the overlap of black (real) and red (our recovered version)

$$\dot{\boldsymbol{x}} = 3.00(1.00\,\boldsymbol{x} - \boldsymbol{y} - 0.33\,\boldsymbol{x}^3) \quad \#\text{Actual}$$
$$\uparrow\downarrow \quad \uparrow\downarrow \qquad \uparrow\downarrow \qquad \uparrow\downarrow$$
$$\dot{\boldsymbol{x}} = 3.03(1.08\,\boldsymbol{x} - \boldsymbol{y} - 0.34\,\boldsymbol{x}^3) \quad \#\text{Recovered}$$

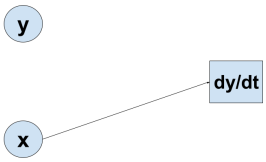
(c) near-perfect recovery of X variable equations

$$\dot{\boldsymbol{y}} = 0.33\,\boldsymbol{x} \quad \#\text{Actual}$$
$$\uparrow\downarrow$$
$$\dot{\boldsymbol{y}} = 0.33\,\boldsymbol{x} \quad \#\text{Recovered}$$

(d) near-perfect recovery of Y variable equations



(e) X variable dependency network



(f) Y variable dependency network

Fig. 1: DIFFFIND works: Recovers the underlying dynamics for Van der Pol: (a - b) The recursive forecasting matches reality; (c - d) The recovered equations match the actual ones; (e - f) Discovered network topology.

– ***Find*** *the nonlinear differential equations that fit the sequences,* ***accurately*** *and* ***automatically****, that is, with no hyperparameters required by the user.*

To overcome the aforementioned shortcomings, we propose DIFFFIND, which automatically evolves to find correct representation via activation functions. Fig. 1 illustrates the effectiveness and explainability of DIFFFIND using the Van der Pol oscillator. In Fig. 1(a) and 1(b), the forecast by DIFFFIND (in red) is almost identical to the actual one (in black): the gray shaded region shows the data used for training, and the white shaded region shows the forecast using the recovered equations. DIFFFIND provides the explanations by visualizing the evolved network topology in Fig. 1(e) and 1(f), and demonstrating the equations in Fig. 1(c) and 1(d). Notice that DIFFFIND not only discovered the correct terms (like x^3), but it also recovered the coefficients, which are very close to the actual ones (e.g., 0.34 instead of 1/3 for the coefficient of the x^3 term in Fig. 1(c)).

In summary, the proposed DIFFFIND has the following properties:

(a) **Effective**, discovering the correct model when tested on real and synthetic nonlinear dynamical systems,
(b) **Explainable**, providing a succinct differential equation that governs the given time sequences - we explicitly aim for succinctness by penalizing model complexity (Eq 1),
(c) **Hands-off**, requiring no manual hyperparameter specification.

Finally, our results are reproducible:

Reproducibility: The code and data used in the experiments along with the supplementary material are available at `https://github.com/Lalithsai853/DIFFIND-PAKDD-2024`.

2 Background and Related Work

Here we first review the existing works for sequential modeling and learning differential equations; and then we present genetic algorithms for architecture learning. In short, DiffFind is the *only* method that matches all the specifications, as shown in Table 1.

2.1 Related work

Time Series Forecasting: Time series forecasting has been studied for many decades [3,8,18]. Classic methods for time series forecasting include the family of auto-regression (AR)-based methods such as exponential smoothing [5], ARMA [18], ARIMA [3], ARMAX [8] models. However, all these models assume linearity, and thus on non-linear dynamical systems may not work well. In contrast, recent, deep-learning models [15,23,26,27] do not need the linearity assumption, but they are often black-box models, with limited or no explainability.

Dynamical System Modeling: Early work on nonlinear system identification [2,24] construct families of candidate nonlinear functions for the rate of change of state variables in time. More recently, SINDy [6] approximates derivatives using a spline over the data points and applies sparse regression to recover equations from a set of pre-specified basis functions, e.g. polynomials. Neural networks have been proposed to model complex dynamics that incorporate prior scientific knowledge via differential equations [9,21]. Similarly, [28,19,29] learn to estimate the parameters for domain-expert specified dynamical systems, which is different from the problem of discovering equations. Recently, system identification from underlying data through neural networks have been studied in [10,24]. Sahoo et al. [22] address a slightly different problem of recovering equations relating input and output variables using a neural network based regression approach, which can be extended to our setting of system identification. However, these methods require careful selection of network architecture (complex architecture tuning), and have limited interpretability.

In conclusion, *only* DiffFind fulfills all the specs, as shown in Table 1.

2.2 Background - Genetic Algorithms for Architecture Search

For deep neural networks, several articles propose architecture search algorithms [11], [12], [30]. The NEAT [25] algorithm stands out, proposing to use a genetic algorithm [14] to optimize the weights and structure of networks simultaneously.

Table 1: DiffFind matches all specs, while competitors miss one or more of the features. '?' indicates 'unclear' or 'depends on implementation'.

Property \ Method	Dynamical Sys. Modeling[17,24,2,6,22]	Neural [23,15]	Fore. [8,18,3]	DiffFind
Hands-off				✔
Explainable	✓		✓	✔
Effective	?	?	?	✔
Scalable(O(n))	✓	✓	✓	✔
Architecture Learning				✔

3 Proposed Method: DiffFind

Preliminaries Table 2 shows the symbols used in this paper. We are given a multivariate time series $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d\}$ containing observations for d sequences, and we want to discover the (non-)linear differential equations Each individual

Table 2: Symbols and definitions

Symbol	Interpretation
$\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d\}$	input d-dimensional multivariate time series data
$\boldsymbol{x}_i(t)$	value at time step t in ith sequence of $\boldsymbol{X}$
G_{node}	number of node genes in the genome
G_{conn}	number of connection genes in the genome
$A_i(f)$	amplitude at frequency f in the Fourier transform of $\boldsymbol{x_i}$

sequence of observation is given as $\boldsymbol{x} = \{x_1, \ldots, x_T\}$ for T equidistant time-ticks. In this work, our goal is to learn the underlying dynamics of $\boldsymbol{X}$ that (1) succinctly compresses the given data, and (2) represents the data dynamics in terms of a system of differential equations.

Difficulty of the problem For ease of presentation, let's assume we have only two coevolving sequences, $\boldsymbol{x}(t)$ and $\boldsymbol{y}(t)$, and we want to find the two functions $f(\boldsymbol{x}, \boldsymbol{y})$ and $g(\boldsymbol{x}, \boldsymbol{y})$ so that $\dot{\boldsymbol{x}} = f()$ and $\dot{\boldsymbol{y}} = g()$ generate the two input sequences.

How should we search for the possible $f()$ and $g()$? We could look for polynomials of lag w, eg., for lag w=1, we would consider:

$$f(\boldsymbol{x}, \boldsymbol{y}) = a_1\boldsymbol{x}(t) + a_2\boldsymbol{y}(t) + a_3\boldsymbol{x}(t) * \boldsymbol{y}(t) + a_4\boldsymbol{x}(t)^2 + \ldots$$

for lag w=2, we would consider

$$f(\boldsymbol{x}, \boldsymbol{y}) = b_1\boldsymbol{x}(t) + b_2\boldsymbol{x}(t-1) + b_3\boldsymbol{x}(t)^2 + b_4\boldsymbol{y}(t) + \ldots$$

How could we search the infinite possible powers and combinations of $\boldsymbol{x}(t)$, $\boldsymbol{x}(t-1)$, $\boldsymbol{y}(t)$, etc? Should we put an upper limit to the exponent of the powers? Linear methods like AR, do exactly that, setting to 1 the maximum exponent.

We propose to do better, by using a genetic algorithm, to search through the possibilities. This creates the need for the following design decisions (DD):
DD1) what is the 'genome'
DD2) which operations (mutation, crossover) do we allow
DD3) what is the fitness function

Next, we describe our design decisions. Once these decisions are finalized, the rest of the algorithm is based on genetic optimization: create a population of genomes (DD1), do mutations and crossovers on the fittest (DD2), and repeat until the fitness function is stabilized (DD3).

DD1 - Proposed Genome Our genome is carefully constructed to ensure that we cover a large family of differential equations. For example, we include polynomial functions, and interaction nodes to represent nonlinear equations. Each genome has a genetic encoding scheme that includes a list of connections. A connection comprises two node genes, and their edge weight. Node gene contains the following:

- response coefficient
- activation functions (this is how we get our power terms from polynomial functions, etc.)
- aggregation function (allows us to combine variables together through addition, product, etc.)

Fig. 2 gives examples of our proposed genome representation.

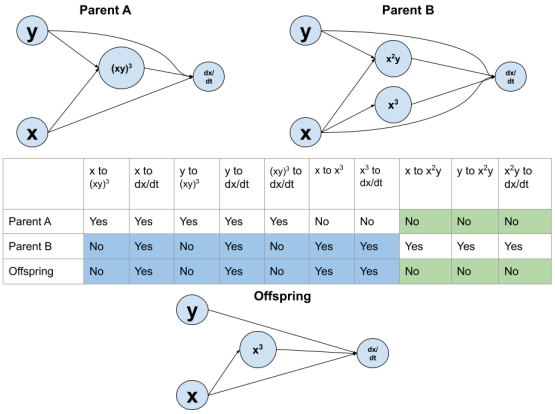
DD2 - Proposed Mutation and Crossover The initial population is composed of sparsely connected networks with no hidden nodes and few of the possible connections between input sequence and output. We allow the following mutations to create new networks by modifying the existing structure:

- addition, deletion of node genes and connections
- update of activation functions, including no activation, quadratic, cubic, and quartic powers

These mutations modify connections between nodes, respecting the feed-forward property of the network. Each new gene that appears through mutation is assigned a global identifier[4] that represents the chronology of the appearance of every gene in the system.

Fig. 2 gives an example of the proposed crossover mechanism: During crossover, matching genes (genes with the same identifier) from both parents are chosen randomly to include in the child, and any excess or disjoint genes from the more fit parent are automatically added to the child. If the two parents have the same performance, then the excess and disjoint genes are added randomly. This allows for equations with different numbers of terms to automatically crossover and form new types of equations that combine certain aspects of both parents.

[4] Identical mutations are assigned the same identifier.



	x to $(xy)^3$	x to dx/dt	y to $(xy)^3$	y to dx/dt	$(xy)^3$ to dx/dt	x to x^3	x^3 to dx/dt	x to x^2y	y to x^2y	x^2y to dx/dt
Parent A	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No
Parent B	No	Yes	No	Yes	No	Yes	Yes	Yes	Yes	Yes
Offspring	No	Yes	No	Yes	No	Yes	Yes	No	No	No

Fig. 2: Proposed genome and crossover scheme. The proposed genome is the list of edges; the crossover picks some edges from parent A (blue) and some others from parent B (green).

DD3 - Proposed Fitness Function We design our fitness function to learn a model of the sequence data that simultaneously minimizes reconstruction error and promotes a simple, compact model. Motivated by Occam's razor, in our approach, we regularize the neural architecture by including a term for model complexity. In particular, DiffFind utilizes the following fitness function to evaluate the performance of a given genome:

$$\mathcal{F} = C - \underbrace{\sqrt{\frac{\sum_{j=1}^{T}(x_j - \hat{x}_j)^2}{T}}}_{\text{Reconstruction error}} - \underbrace{\lambda(G_{conn} + G_{node})}_{\text{Model complexity penalty}} \tag{1}$$

where $C = 100$ is a large constant, $\hat{x}_j$ is the j_{th} predicted value for an individual sequence of multivariate time series based on the given genome, G_{conn} and G_{node} represents the complexity of genome in terms of number of connection and node genes, and λ controls the penalty for model (genome) complexity (our recommended, default value is λ=0.1).

Implementation Details – Algorithm We present the pseudocode of the DiffFind in Algorithm 1. We learn a genetic neural network for each variable within the input multivariate time series $\boldsymbol{X}$. In our architecture learning, nodes represent variables or operations. For example, input nodes represent values at the current time step, while the output node represents the differential value at that time step. Hidden nodes are created to incorporate operations and functions such as addition, subtraction, square, cube, and so on. Connections dictate flow between nodes, and end in the output node. Aggregation and activation functions provide the network an array of options for creating terms such as polynomial expressions.

```
   Data: An initial set of genomes 𝒮
   Result: The best genome winner in 𝒮
 1 Initialization;
 2 while stopping criterion is not met do
 3 |   for currGenome ∈ 𝒮 do
 4 |   |   Calculate fitness for currentGenome;
 5 |   |   Assign fitness value to currentGenome.fitness;
 6 |   |   if currentGenome.fitness > winner.fitness then
 7 |   |   |   winner ← currentGenome;
 8 |   |   else
 9 |   |   |   Continue;
10 |   |   end
11 |   end
12 |   Apply mutation and crossover (reproduction) schemes on proposed genome tree
   |     structure;
13 |   Remove any stagnant speciesx;
14 end
15 Return winner;
```

Algorithm 1: Pseudocode of DiffFind

4 Experiments

We design our experiment to answer the following research questions:

Q1. Effective: How effective is DiffFind in learning the data dynamics? How accurate is the DiffFind in recursive forecast

Q2. Explainable: How to explain the results of DiffFind? How compact is the system of equations learned by DiffFind?

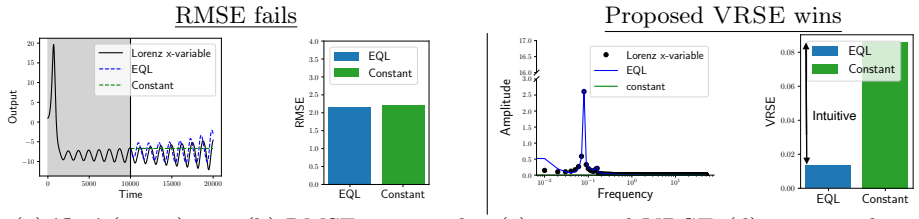
Q3. Scalable: How fast is DiffFind? How does the running-time of DiffFind scale w.r.t. length of time sequences?

Datasets We evaluate DiffFind through extensive experiments on four carefully chosen dynamical systems, which relate to real phenomena from atmospheric convection to spikes in human brain signals, and two real data, including a case study on solar cycles. Dataset description is provided in Supplementary A.

Baselines We compare DiffFind to the following baselines.

1. AutoReg expresses each variable as a linear function of its own past values as well as the past values of all other variables in the model.
2. SINDy [6] approximates derivatives of each variable and applies sparse regression to recover equations from a set of prespecified basis functions.
3. EQL [22] recovers equations relating input and output variables using a neural network based regression approach. We extend the method to recover differential equations by setting the output as derivative of the variables.

Evaluation Measures – Pitfalls of RMSE We report the performance using root mean square error (RMSE) which has been used as a standard statistical metric in various earlier works [7]. However, we note that RMSE often fails to quantify the performance gains for different models. For example, Fig. 3 (a – b)



(a) 'flat' (green) vs non-linear (EQL blue) (b) RMSE incorrectly declares tie (c) proposed VRSE (d) ... correctly picks the winner

Fig. 3: VRSE wins over RMSE. VRSE captures the visual similarity, and correctly picks the winner (see (d)). RMSE declares a tie (see (b)) between the green, flat line, and the blue, wavy reconstruction (see (a)). Our insight is that VRSE should look for similarities in the *Fourier* domain (see (c)), not in the time domain.

shows that the constant forecast is equivalent to forecast made by EQL, while we clearly observe that EQL predictions approximate the ground truth better as compared to constant model. To address the issue, and to capture visual similarity between sequences, we propose VRSE (visual relative squared error) to evaluate and compare model performance. Fig. 3 (c – d) clearly shows that VRSE assigns lower error comparatively.

Definition 1 (VRSE). VRSE*(visual relative squared error) is defined as follows:*

$$\text{VRSE} = \frac{\sum_f (\, A_x(f) - A_y(f)\,)^2}{\sum_f (A_x(f))^2} \tag{2}$$

where $A_x(f)$ ($A_y(f)$) is the amplitude at frequency f of time sequence $\boldsymbol{x}$ ($\boldsymbol{y}$).

Experimental Settings are included in Supplementary B.

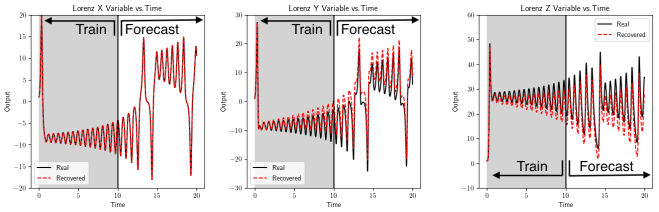
4.1 Q1 - DiffFind is Effective

In this subsection, we show the most difficult version of forecast, the so-called 'recursive' or 'k-step-ahead' forecast for each of the test-bed dynamical systems. Recursive forecast is defined as follows:

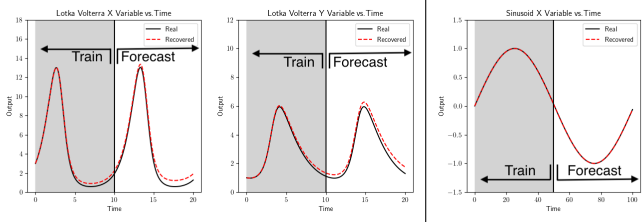
Definition 2 (Recursive Forecast). *The forecast of the next value is based on the prediction of the previous time step, instead of using actual observation.*

The easier alternative is the one-step-ahead forecast - despite the difficulty of the task, DiffFind does very well, even on nonlinear (chaotic) sequences, as we discuss next.

Observation 1 DiffFind *includes* SINDy *as a special case.*



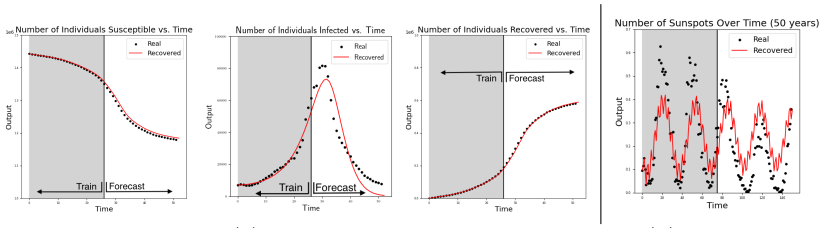
(a) Lorenz System



(b) Lotka-Volterra System

(c) Sinusoid System

Fig. 4: DiffFind is effective. The recursive forecasting from the recovered equations is shown for different dynamical systems.



(a) Influenza

(b) Sunspots

Fig. 5: DiffFind is effective on real data. (a) Our recovered equation (red) match the ground truth (black), for all three (susceptible, infected, recovered) in Influenza data. (b) DiffFind recovers a simple equation underlying sunspot phenomenon.

Reason: SINDy considers polynomials of a user-specified maximum degree, while DiffFind automatically searches for the best exponents and products.

Notice that, in Fig. 4 and Fig. 5 (real data), for all the systems, namely, Lorenz, Lotka-Volterra, Sinusoid, real influenza data and real sunpots data, DiffFind forecast (shown in red) follows actual observations (shown in black) closely. Furthermore, notice that DiffFind recovers the time-delay equations governing the sinusoid system. Table 4 shows that DiffFind outperforms the baselines with respect to both evaluation measures RMSE and VRSE. DiffFind consistently ranks in top two against all baselines across all datasets.

Table 3: DiffFind is effective and explainable. It precisely recovers the actual equations of 4 test-bed dynamical systems.

Systems	**Actual Equations**	**DiffFind Recovered Equations**
Lorenz System	$\dot{\boldsymbol{x}} = 10(\boldsymbol{x} - \boldsymbol{y})$ $\dot{\boldsymbol{y}} = \boldsymbol{x}(28 - \boldsymbol{z}) - \boldsymbol{y}$ $\dot{\boldsymbol{z}} = \boldsymbol{xy} - \frac{8}{3}\boldsymbol{z}$	$\dot{\boldsymbol{x}} = 9.996(\boldsymbol{x} - \boldsymbol{y})$ $\dot{\boldsymbol{y}} = \boldsymbol{x}(24.298 - 1.121\boldsymbol{z}) - 0.922\boldsymbol{y}$ $\dot{\boldsymbol{z}} = 0.933\boldsymbol{xy} - 2.506\boldsymbol{z}$
Van der Pol Oscillator	$\dot{\boldsymbol{x}} = 3(\boldsymbol{x} - \boldsymbol{y} - \frac{1}{3}\boldsymbol{x}^3)$ $\dot{\boldsymbol{y}} = \frac{1}{3}\boldsymbol{x}$	$\dot{\boldsymbol{x}} = 3.03(1.08\boldsymbol{x} - \boldsymbol{y} - 0.34\boldsymbol{x}^3)$ $\dot{\boldsymbol{y}} = 0.33\boldsymbol{x}$
Lotka-Volterra System	$\dot{\boldsymbol{x}} = 1.1\boldsymbol{x} - 0.4\boldsymbol{xy}$ $\dot{\boldsymbol{y}} = 0.1\boldsymbol{xy} - 0.4\boldsymbol{y}$	$\dot{\boldsymbol{x}} = 1.070\boldsymbol{x} - 0.373\boldsymbol{xy}$ $\dot{\boldsymbol{y}} = 0.099\boldsymbol{xy} - 0.387\boldsymbol{y}$
Sinusoid System	$\dot{\boldsymbol{x}}_{\boldsymbol{t}} = (2 - (\frac{2\pi}{100})^2)\boldsymbol{x}_{\boldsymbol{t-1}} - \boldsymbol{x}_{\boldsymbol{t-2}}$	$\dot{\boldsymbol{x}}_{\boldsymbol{t}} = 1.996\boldsymbol{x}_{\boldsymbol{t-1}} - 1.001\boldsymbol{x}_{\boldsymbol{t-2}}$

Table 4: DiffFind wins: Consistently first or second, against all baselines and on all datasets. Winners in and runner-ups in .

	Systems↓		**AutoReg**	**SINDy**	**EQL**	**DiffFind**
Synthetic	Lorenz System	RMSE	5.7279	0.4028	6.1186	1.1864
		VRSE	0.2670	0.0002	0.2878	0.1420
	Van der Pol Oscillator	RMSE	0.2712	0.0019	2.4400	0.0561
		VRSE	0.2857	0.0001	2.9812	0.0
	Lotka-Volterra System	RMSE	2.5272	0.5902	2.0000	0.1408
		VRSE	10.2739	0.2488	4.3889	0.1415
	Sinusoid System	RMSE	0.0	0.7688	0.6921	0.0078
		VRSE	0.0	0.9939	0.8445	0.0
Real	Influenza (imputed real)	RMSE	4.60e7	6.52e6	1.57e5	1.27e4
		VRSE	4358.29	4157.26	0.8107	0.0059

4.2 Q2 - DiffFind is Explainable

In Fig. 1, DiffFind successfully recovers the dynamics of the Van Der Pol system. The evolved neural networks are visualized in Fig. 1(e) and 1(f), corresponding to the recovered equations in Fig. 1(c) and 1(d), respectively. For all other test-bed dynamical systems, we report the actual equations used to generate the observations from each system, and the corresponding DiffFind recovered system of equations in Table 3. Notice that DiffFind successfully recovers the correct model family along with the coefficients.

The recovered equations are a compact representation in terms of a differential equation for the given observational data. The differential equations are easily understandable by the domain experts, which can be used to explain the interrelations among variables of the system and underlying dynamics.

4.3 Q3 - DiffFind is Scalable

Fig. 6 shows that DiffFind scales linearly with the training set size. To quantify the scalability of our method, we empirically vary the number of observations in the training data for the Van der Pol Oscillator, and plot against the wall-clock training time for DiffFind.

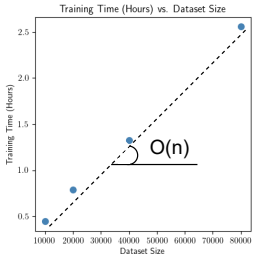


Fig. 6: Linear in (wall-clock) training time, vs seq. length.

Case Study Next, we apply DiffFind to solar spotting data to understand the dynamics of solar activity. Note that unlike other datasets, there is no ground truth available. Fig. 5(b) shows that DiffFind accurately finds the underlying pattern in the solar activity. The recovered equation $\dot{\boldsymbol{x}}_{\boldsymbol{t}} = -0.26\boldsymbol{x}_{\boldsymbol{t}} + 0.763\boldsymbol{x}_{\boldsymbol{t-2}} - 0.68\boldsymbol{x}_{\boldsymbol{t-4}}$ provides an interpretable model of this periodic natural phenomenon.

5 Conclusions

We present DiffFind to model and extract governing equations from a given set of observational data. Our proposed DiffFind has the following properties:
(a) **Effective**, discovering correct model for several real and synthetic nonlinear dynamical systems,
(b) **Explainable**, providing a succinct differential equation, thanks to our proposed fitness function (Eq. 1), that deliberately penalizes model complexity,
(c) **Hands-off**, requiring no manual hyperparameter specification.
An additional contribution is also the VRSE error function, which outperforms the traditional RMSE, as it agrees better with our intuition (see Eq (2), Fig. 3)

References

1. Anisiu, M.C.: Lotka, volterra and their model. Didáctica mathematica **32**, 9–17 (2014)
2. Bongard, J., Lipson, H.: Automated reverse engineering of nonlinear dynamical systems. PNAS **104**(24), 9943–9948 (2007)
3. Box, G.E., Jenkins, G.M., MacGregor, J.F.: Some recent advances in forecasting and control. Journal of the Royal Statistical Society: Series C (Applied Statistics) **23**(2), 158–179 (1974)
4. Brauer, F., Castillo-Chavez, C., Castillo-Chavez, C.: Mathematical models in population biology and epidemiology, vol. 2. Springer (2012)
5. Brown, R.G.: Statistical forecasting for inventory control (1959)
6. Brunton, S.L., Proctor, J.L., Kutz, J.N.: Discovering governing equations from data by sparse identification of nonlinear dynamical systems. PNAS **113**(15), 3932–3937 (2016)
7. Chai, T., Draxler, R.R.: Root mean square error (rmse) or mean absolute error (mae)?–arguments against avoiding rmse in the literature. Geoscientific model development **7**(3), 1247–1250 (2014)

8. Chatfield, C.: Time-series forecasting. Chapman and Hall/CRC (2000)
9. Chen, R.T., Rubanova, Y., Bettencourt, J., Duvenaud, D.K.: Neural ordinary differential equations. NeurIPS **31** (2018)
10. Cranmer, M., Sanchez Gonzalez, A., Battaglia, P., Xu, R., Cranmer, K., Spergel, D., Ho, S.: Discovering symbolic models from deep learning with inductive biases. NeurIPS **33**, 17429–17442 (2020)
11. Ding, S., Li, H., Su, C., Yu, J., Jin, F.: Evolutionary artificial neural networks: a review. Artificial Intelligence Review **39**(3), 251–260 (2013)
12. Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: A survey. JMLR **20**(1), 1997–2017 (2019)
13. FitzHugh, R.: Impulses and physiological states in theoretical models of nerve membrane. Biophysical journal **1**(6), 445–466 (1961)
14. Kramer, O., Kramer, O.: Genetic algorithms. Springer (2017)
15. Lim, B., Zohren, S.: Time-series forecasting with deep learning: a survey. Philosophical Transactions of the Royal Society A **379**(2194), 20200209 (2021)
16. Lorenz, E.N.: Deterministic nonperiodic flow. Journal of atmospheric sciences **20**(2), 130–141 (1963)
17. Luo, Y., Xu, C., Liu, Y., Liu, W., Zheng, S., Bian, J.: Learning differential operators for interpretable time series modeling. In: ACM SIGKDD. pp. 1192–1201 (2022)
18. Makridakis, S., Hibon, M.: Arma models and the box–jenkins methodology. Journal of forecasting **16**(3), 147–163 (1997)
19. Park, N., Kim, M., Hoai, N.X., McKay, R.B., Kim, D.K.: Knowledge-based dynamic systems modeling: A case study on modeling river water quality. In: ICDE. pp. 2231–2236. IEEE (2021)
20. Van der Pol, B.: Lxxxviii. on "relaxation-oscillations". The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science **2**(11), 978–992 (1926)
21. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational physics (2019)
22. Sahoo, S., Lampert, C., Martius, G.: Learning equations for extrapolation and control. In: ICML. pp. 4442–4450. PMLR (2018)
23. Salinas, D., Flunkert, V., Gasthaus, J., Januschowski, T.: Deepar: Probabilistic forecasting with autoregressive recurrent networks. International Journal of Forecasting **36**(3), 1181–1191 (2020)
24. Schmidt, M., Lipson, H.: Distilling free-form natural laws from experimental data. science **324**(5923), 81–85 (2009)
25. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary computation **10**(2), 99–127 (2002)
26. Tealab, A.: Time series forecasting using artificial neural networks methodologies: A systematic review. Future Computing and Informatics J. **3**(2), 334–340 (2018)
27. Torres, J.F., Hadjout, D., Sebaa, A., Martínez-Álvarez, F., Troncoso, A.: Deep learning for time series forecasting: a survey. Big Data **9**(1), 3–21 (2021)
28. Wang, R., Maddix, D., Faloutsos, C., Wang, Y., Yu, R.: Bridging physics-based and data-driven modeling for learning dynamical systems. In: Learning for Dynamics and Control. pp. 385–398. PMLR (2021)
29. Wang, R., Robinson, D., Faloutsos, C., Wang, Y.B., Yu, R.: Autoode: Bridging physics-based and data-driven modeling for covid-19 forecasting. In: NeurIPS 2020 Workshop on Machine Learning in Public Health (2020)
30. Zhou, X., Qin, A.K., Gong, M., Tan, K.C.: A survey on evolutionary construction of deep neural networks. IEEE Tran. on Evolutionary Computation **25**(5), 894–912 (2021)

A Appendix: Datasets

Test-bed Dynamical Systems We evaluate DiffFind through extensive experiments on four carefully-chosen dynamical systems, which relate to real phenomena from atmospheric convection to spikes in human brain signals. We generate observations for our experiment from the following, famous, nonlinear dynamical systems. Their exact equations are in Table 3.

- **Lorenz System** models atmospheric convection and contains three equations that model dynamics that are evolving on an attractor. This is a famous system, because it is the first that exhibited *deterministic chaos*, colloquially known as 'the butterfly effect' [16]. We set $[x_0, y_0, z_0] = [1, 1, 1]$
- **Van Der Pol Oscillator** [20] is famous for modeling the pulse in the heart. It is a special case of the also-famous FitzHugh-Nagumo model [13], which models brain spikes. We set $[x_0, y_0] = [1, 1]$.
- **Lotka-Volterra System** [1] is a pair of nonlinear differential equations that describe the predator-prey dynamics in an ecosystem. It is a special case of the Kolmogorov model [4] which provides a framework to describe competition and disease. We set $[x_0, y_0] = [3, 1]$.

For sanity check, we also used a linear dynamical system, obeying Hooke's law and resulting into a sinusoid.

- **Sinusoid System** (or 'pendulum'): is a second-order differential equation, or equivalently, a first order time-delay equation, where the derivative dependents on the previous two values. We set the initial conditions to $[\boldsymbol{x}[t], \boldsymbol{x}[t-1]] = [\frac{2\pi}{100}, 0]$.

In addition to the above dynamical systems, we evaluate DiffFind on a real data on influenza infections, and a case study on understanding the dynamics of solar cycles described as follows.

Imputed Real includes number of influenza infections in the USA for the past year. We further impute the counts of individuals who are susceptible to the infection, and the counts of individuals who recover based on a transmission and recovery parameters of 0.2 and 0.4 respectively. Data are available at `https://gis.cdc.gov/grasp/fluview/fluportaldashboard.html`

Sunspots data records sunspot numbers as a way to measure solar cycle mechanisms and solar forcing on the Earth's climate. The data contain sunspot numbers for the last 250 years recorded for each month. In our analysis, we consider quarterly sunspots, and use last 50 years of the data as test set. In our experiments, we transform the data using min-max normalization. Data are available at `https://www.sidc.be/SILSO/datafiles`

B Appendix: Experimental Settings

The parameters of all experiments are set to reasonable defaults. Activation options are input (no activation), square, and cube; aggregation options are

addition and product; connection addition and removal probabilities are at 0.10; node addition and removal probabilities are at 0.10. All the experiments are run on a stock desktop with 3.4 GHz 4-core CPU and 16 GB RAM, running Ubuntu 20.04 LTS.