Rediscovering orbital mechanics with machine learning

IOP Publishing Mach. Learn.: Sci. Technol. 4 (2023) 045002 https://doi.org/10.1088/2632-2153/acfa63

MACHINE LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
3 April 2023

REVISED
25 August 2023

ACCEPTED FOR PUBLICATION
15 September 2023

PUBLISHED
9 October 2023

PAPER

Rediscovering orbital mechanics with machine learning

Pablo Lemos[1,2,*], **Niall Jeffrey**[2,3,4], **Miles Cranmer**[5], **Shirley Ho**[5,6,7,8] **and Peter Battaglia**[8]

[1] Department of Physics and Astronomy, University of Sussex, Sussex House, Falmer, Brighton BN1 9RH, United Kingdom
[2] Department of Physics and Astronomy, University College London, Gower Street, London WC1E 6BT, United Kingdom
[3] Laboratoire de Physique de l'Ecole Normale Supérieure, ENS, Université PSL, CNR, Sorbonne Université Université de Paris, Paris 75005, France
[4] Department of Astrophysical Science, Princeton University, Peyton Hall, Princeton, NJ 08544, United States of America
[5] Flatiron Institute Center for Computational Astrophysics, 162 5th Ave, 3rd floor, New York, NY 10010, United States of America
[6] Center for Cosmology and Particle Physics, Department of Physics, New York University, New York, NY 10003, United States of America
[7] Department of Physics, Carnegie Mellon University, Pittsburgh, PA 15213, United States of America
[8] DeepMind, London, United Kingdom
[*] Author to whom any correspondence should be addressed.

E-mail: pablo.lemos@umontreal.ca

Keywords: scientific discovery, symbolic regression, AI scientist, graph neural network, inductive biases

Supplementary material for this article is available online

Abstract

We present an approach for using machine learning to automatically discover the governing equations and unknown properties (in this case, masses) of real physical systems from observations. We train a 'graph neural network' to simulate the dynamics of our Solar System's Sun, planets, and large moons from 30 years of trajectory data. We then use symbolic regression to correctly infer an analytical expression for the force law implicitly learned by the neural network, which our results showed is equivalent to Newton's law of gravitation. The key assumptions our method makes are translational and rotational equivariance, and Newton's second and third laws of motion. It did not, however, require any assumptions about the masses of planets and moons or physical constants, but nonetheless, they, too, were accurately inferred with our method. Naturally, the classical law of gravitation has been known since Isaac Newton, but our results demonstrate that our method can discover unknown laws and hidden properties from observed data.

1. Introduction

Machine learning (ML) has led to dramatic advances in many scientific disciplines, typically by helping to process large, complex sets of observations, and learn to predict key desired properties. From particle physics [1] to structural biology [2] to cosmology [3], ML methods help find patterns in large data sets [4, 5], classify different objects [6], and perform parameter inference [7–9], as well as groundbreaking applications such as predicting protein structure [10] and function [11], and producing language that is often indistinguishable from humans' [12]. However, there have been comparatively few applications of ML to one of the most fundamental parts of science: theory discovery. Here we demonstrate a new approach for using real data and established scientific frameworks to aid in the discovery of both physical laws and unobserved properties of a complex physical domain—our Solar System. We use real observations of the orbital trajectories of the Sun, planets, and moons to (re-)discover Newton's law of gravitation, as well as the masses of these bodies. Our approach is analogous to the process followed by scientists when they develop scientific theories—describing patterns in data, proposing symbolic formulae to explain them, evaluating these expressions against observation—while automating key aspects of this endeavour.

Our approach involves two stages: training a learned simulator on observed data, and then performing symbolic regression on components of the simulator which correspond to physical laws. In [13, 14] we described an initial version of our approach, applied to simulated data. Here we have extended and innovated on our general approach, incorporated new techniques for simultaneously inferring unobserved properties of

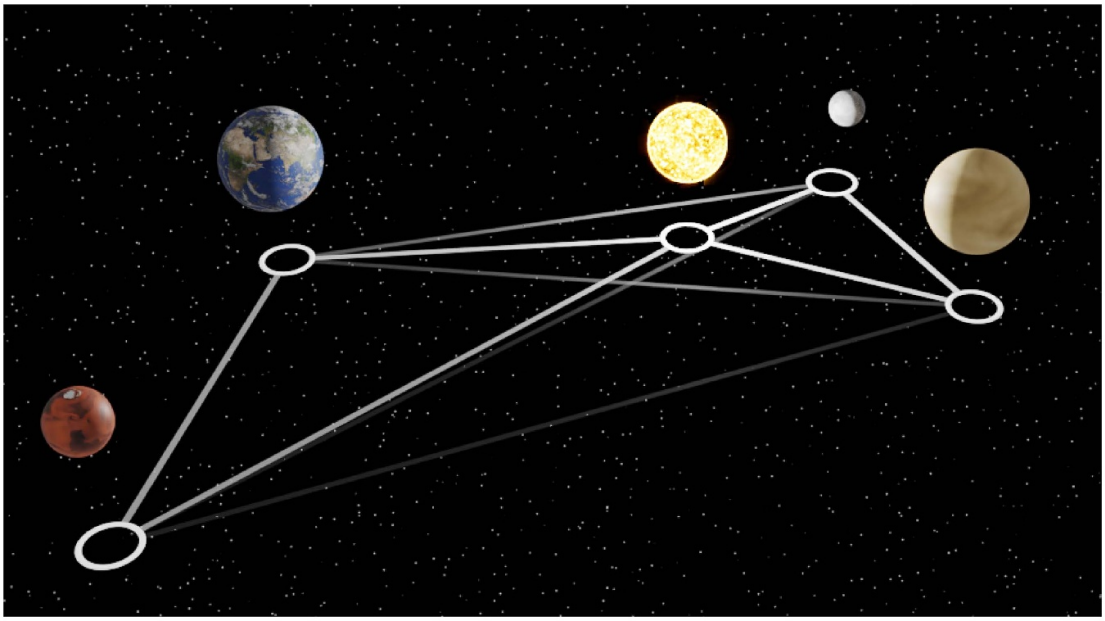

Figure 1. Schematic rendering of the Sun, Mercury, Venus, Earth, and Mars, with the corresponding graph structure our learned simulator uses. The graph's nodes represent the bodies, and the brightness of the edges is proportional to the strength of the gravitational interaction between them.

the system (e.g. the masses of the bodies), and, perhaps most importantly, applied it to real data and proved it can correctly recover one of the most widely known and important classical physical laws: gravitational force.

In the first stage of our approach, we train a learned simulator, based on graph networks (GNs) [15], which are deep neural networks that can be trained to approximate complex functions on graphs. Here the (relative) positions and velocities of the Solar System's Sun, planets, and moons are represented as nodes of the input graph, and possible physical interactions (e.g. forces) between the bodies are represented by the graph's edges. GN-based simulators have been trained to accurately model N-body and more complex particle- and mesh-based systems in recent years [16–18], though they have almost never been trained on real observations. Applying our approach to real data presented new challenges: the data are noisy, and their dynamic range spans several orders of magnitude; the dataset is partial (we only provide 31 objects; leaving out other massive bodies); and, crucially, the masses of the bodies are not observed, and therefore needed to be (re-)discovered at the same time. We fit the GN-based simulator to 30 years of observed Solar System trajectories, where the training procedure optimized the parameters of the GN's neural network 'edge function', which plays the role of computing forces [13, 14]. See, figure 1 for an illustration.

In the second stage, we isolate the GN's edge function and use symbolic regression to fit an analytical formula to it. This allows us to decipher the otherwise uninterpretable neural network approximation to the force function, by expressing it symbolically. The best fitting force expression was the correct formula for Newton's universal law of gravitation. We then re-fit the unobserved (relative) masses of the bodies using our discovered equation and found a nearly perfect fit to the true masses. We could then simulate the Solar System dynamics using the discovered equation and re-learned masses, and found it corresponded very closely to the true observed trajectories.

The reason we adopt this two-step approach, instead of applying symbolic regression directly to the data, is that symbolic regression is not practical or efficient in many regimes like ours. Because the learned simulator is a neural network, and can be trained by gradient descent, fitting it to real data is very efficient and effective, mirroring the wide range of contemporary advances where neural networks are used to process real data. By contrast, the symbolic regression procedure involves a prohibitively expensive search using evolutionary algorithms, which would take orders of magnitude longer. So by fitting a neural network simulator first, and then applying symbolic regression to only that component of the learned simulator we were interested in, we reduced the cost of the equation discovery a great deal, and made the problem tractable for our symbolic regression code.

There are several reasons to prefer a symbolic expression, instead of settling for a learned simulator. Naturally, describing physical phenomena with compact symbolic formulations supports scientific interpretation, and can interface with existing symbolically defined physical theories. By contrast, the knowledge stored within trained neural networks cannot easily interface with existing theories. For example,

how would one interpret the thousands, millions, or even billions of weights within a neural network, or communicate that knowledge effectively to others? Beyond interpretability, the symbolic expression we extracted was more accurate than the predictions of the neural network, due to the strong bias toward simplicity, which was carried implicit within our symbolic regression method. Of course, because we recovered the true classical force law, we know it will generalize to any scale, while the neural network is only accurate when the statistics of the input match what it was trained on. In other words, we not only found a model of Solar System dynamics, but of galactic dynamics as well!

Symbolic regression, also known as automated equation discovery, has been explored for decades in the context of scientific discovery, for example, BACON [19, 20], COPER [21], FAHRENHEIT/EF [22, 23] and LAGRANGE [24]. More recent work [25, 26] introduced the symbolic regression package *eureqa*, which has been applied to finding symbolic formulae for Lagrangians, Hamiltonians, repeated sub-equations, etc without relying on known constants or strong priors on the physical nature of the system. Another notable recent development in symbolic regression is the *SINDy* technique [27] which searches for dynamical models which are sparse linear combinations of hand-designed expressions. Though, in addition to *SINDy*, there have been many additional advances in search techniques [28–43], as well as work in discovering symmetries using ML [44, 45].

Here we used the neural network-symbolic regression technique we first introduced in [14], which extends symbolic regression to high-dimensional input, such as graphs, by using a neural network as an intermediate stepping-stone model. However, our model has some key improvements on the model introduced in [14], perhaps most importantly its ability to learn hidden properties of the system, in our cases the masses of the bodies. By applying the right inductive biases, we will show how we can recover the masses of the planets and moons in the Solar System, up to an overall calibration. Previously [46] have used the same Solar System database used in this work to study the underlying mechanism of the Solar System dynamics, but there are key differences between our approaches: our model aims to recover a mathematical equation, while [46] use interaction kernels (learnable functions of the pair-wise distance between bodies) to model the dynamics. One issue with interaction kernels is the requirement to learn a kernel for each pair of bodies, therefore the complexity over the problem scales as $O(N^2)$, where N is the number of bodies. Meanwhile, our approach learns a single law that applies simultaneously to all edges, and therefore does not suffer from this scalability issue. In addition, interaction kernels lack the explainability of our formalism. Finally, our method infers the masses of the bodies as part of the learning, while the method introduced in [46] can only do so by relying on certain approximations. As our base symbolic regression technique, we use our open-source software *PySR*[9] [14, 47]. *PySR* is a genetic algorithm, taking as input only a basic set of operators, and a dataset, rather than predefined expressions as with *SINDy*, which means that we can avoid supplying additional prior knowledge over the space of possible expressions.

It is important to emphasize that there is no way to 'discover' new theories without imposing some constraints, inductive biases, or other assumptions on the process (this is a fundamental tenet of statistical learning theory, and generally consistent with the 'no free lunch' theorem). For example, mathematical axioms are required to define quantitative theories; the concepts of space and time are required to specify equations of motion; and a physical mechanics formalism, such as classical mechanics, is required to define specific dynamical laws, such as Hooke's law or the Hamiltonian of a many-body system. Here our approach leverages the fact that an N-body system can be represented as a graph; and that these systems are translationally equivariant [48]. Our learned simulator incorporates Newton's laws of motion in that the learned scalar for each node is multiplicative in scaling the model's output to acceleration; and finally, our equation search prioritizes simple expressions, which is analogous to Occam's razor. All of these are inductive biases that would be available to a scientist when formulating a new theory. Ultimately we believe our approach should be viewed as a tool which can help scientists make parts of their discovery process more efficient and systematic, rather than as a replacement for the rich domain knowledge, scientific methodology, and intuition which are essential to scientific discovery.

2. Model

Our two-step approach first fits a GN-based learned simulator to model the observed trajectories, and then uses symbolic regression to fit analytical formulae to internal components of the learned simulator, which we designed to have direct correspondences to classical mechanics' force law. Within our learned simulator, we used one trainable scalar value per body, which scaled the predicted acceleration for the body. This implements the assumption of Newton's second law, i.e. $F = ma$, where the predicted acceleration, a, is computed by a neural network prediction that plays the role of force, F, divided by a scalar that plays the role of mass, m.

[9] https://github.com/MilesCranmer/PySR.

2.1. GN-based learned simulator

The input to our GN-based learned simulator, g_θ, is a graph, (V, E), which represents the physical system, where the set of N^v bodies are represented as nodes, $V = \{v_i\}_{i=1:N^v}$, and relationships between pairs of bodies are represented as directed edges, $E = \{(s_k, r_k, \mathbf{e}_k)\}_{k=1:N^e}$. Each v_i node attribute contains a trainable scalar variable that is fixed across all input graphs and is analogous to mass, as described below. The s_k and r_k edge attributes are integers that index the sender and receiver nodes, respectively. The $\mathbf{e}_k$ edge attribute is the spatial displacement vector between the two corresponding bodies. Because we assume we do not know which bodies interact, we instantiate edges from each body to every other body, which allows us to model all possible pairwise interactions.

To simulate the bodies' dynamics, the model predicts the per-body accelerations, a_i, by explicitly imposing Newton's second and third laws of motion. The GN contains an 'edge function', $\mathbf{e}'_k = f_{\text{GN}}(v_{r_k}, v_{s_k}, \mathbf{e}_k; \theta)$, with trainable parameters θ, which computes an interaction vector, $\mathbf{e}'_k$, along each edge, which is analogous to a force. For the two directed edges between a pair of bodies, $(i, j, \mathbf{e}_k)$ and $(j, i, \mathbf{e}_l)$, rather than computing distinct $\mathbf{e}'_k$ and $\mathbf{e}'_l$ we instead compute just one and set the other equal to its negative, $\mathbf{e}'_l = -\mathbf{e}'_k$, in accordance with Newton's third law's 'equal and opposite' principle. Next, for each body, i, all of its incoming interaction vectors are summed, $\bar{\mathbf{e}}'_i = \sum_{\{k|r_k=i\}} \mathbf{e}'_k$, analogous to superposition of forces to compute net force. Finally, the per-node output accelerations, $\hat{a}_i = \bar{\mathbf{e}}'_i / v_i$ are computed by dividing each node's pooled interactions by the scalar node attribute, v_i, which, following Newton's second law's $F = ma$, gives v_i the semantics of 'mass' and $\bar{\mathbf{e}}'_i$ the semantics of 'net force'[10]. The Sun's scalar attribute is fixed to 1 to fix the degeneracy of scale between the learnable GN and learnable scale. The details of the neural networks are described in the Experimental Methods below.

Our learned simulator g_θ is trained by supervised learning, where the discrepancies between the model's predicted accelerations and the true observed accelerations are minimized with respect to the trainable model parameters using gradient descent,

$$\theta^*, V^* = \text{argmin}_{\theta, V} \mathbb{E}_{(E,A)\sim\mathcal{D}_{\text{train}}} \ell_{\text{GN}}\left(g\left(V, E; \theta\right), A\right), \tag{1}$$

where A are the true observed accelerations associated with some input (V, E), ℓ_{GN} is an error metric, and $\mathcal{D}_{\text{train}}$ is the empirical distribution which represents the observed system states (represented by the relative displacements between bodies, E) and accelerations used for training. While the edge attributes, E, vary as the positions of the bodies in the system change, the scalar per-node attributes, V, are trainable variables which are constant across inputs. By minimizing the error with respect to V, we are fitting the masses for each body in the system, which we will compare to the known masses of the Solar System bodies in the results.

2.2. Symbolic regression of force function

Once the learned simulator was trained, we performed symbolic regression to fit an explicit symbolic formula to the GN-based force function. We created a dataset of force function inputs, $(v_{r_k}, v_{s_k}, \mathbf{e}_k) \in \mathcal{D}_{\text{SR}}$, and used the symbolic regression procedure to search for an expression, f_{SR}, which minimizes,

$$f^*_{\text{SR}} = \text{argmin}_{f_{\text{SR}}} \mathbb{E}_{x\sim\mathcal{D}_{\text{SR}}} \ell_{\text{SR}}\left(f_{\text{SR}}\left(x\right), f_{\text{GN}}\left(x; \theta^*\right)\right), \tag{2}$$

where $x = (v_{r_k}, v_{s_k}, \mathbf{e}_k)$ sampled from the empirical symbolic regression training distribution, $\mathcal{D}_{\text{train}}$, and ℓ_{SR} is an error metric.

The symbolic regression procedure explores a space of analytic expressions and selects one or more which predict the target, $f_{\text{GN}}(v_{r_k}, v_{s_k}, \mathbf{e}_k; \theta^*)$, accurately, while also minimizing the complexity of the discovered expression. The space of symbolic expressions is large due to the combinatorial number of ways the operators, variables, and constants can be composed (e.g. if there are M possible discrete symbols, then there are M^L possible symbol strings of length L, but actually the constants are effectively real-valued rather than discrete). Because it is fundamentally a discrete problem, we cannot compute gradients or perform gradient descent, as with the GN-based simulator's training.

3. Experimental methods

3.1. Data

We use Solar System data from NASA's HORIZONS On-Line Ephemerys System[11] [49, 50]. We extract orbits for 31 bodies: the Sun, all planets, and those moons which have a mass above 10^{18} kg. Whilst more bodies

[10] Note, in practice we use $\log(v_i)$, in order to reduce the dynamic range of v_i.

[11] http://ssd.jpl.nasa.gov/?horizons.

could have been considered, we expect their gravitational influence to be small, therefore we do not expect that their omission will affect our results. We use data from January 1980 to January 2013 with a time step of 30 min, and use the first 30 years of data (approximately one full orbit of Saturn) for training, and the last three for validation. From the HORIZONS interface, we extract positions and velocities in Cartesian coordinates, with the Solar System barycenter as the reference frame. The same data was used by [46], but using only data from the Sun and the eight planets, without any of the moons. Of course, using data in Cartesian coordinates is an easier problem than using observations from Earth's point of view. Repeating this task using terrestrial observations would require learning the correct coordinates, as well as the equations, and is left for future work.

From this data, we extract the pair-wise displacement vectors between bodies and each body's acceleration vector (calculated from changes in the velocities) at every step. Relative displacements serve as the input to our model, meaning that our model is equivariant to a translated reference frame. The accelerations serve as the truth for our model training.

Therefore, Our input graph has $N^v = 31$ nodes; each node with one trainable scalar, and a single edge connecting every pair of bodies $N^e = N^v(N^v - 1)/2 = 465$; each containing three coordinates giving the distances between bodies along each spatial axis.

3.2. GN implementation details

The GN uses a TensorFlow [51] model with three-layer multilayer perceptrons (MLPs) and 128 hidden nodes per layer. The model also contains the trainable scalar properties of the nodes v_i, which are backpropagated simultaneously with the weights of the neural network. Furthermore, our model has the following properties:

- **Activation function**: We use a hyperbolic tangent ('tanh') as the activation function in our networks. While this is slower than the very commonly used Rectified Linear Unit (ReLU) activation function [52], our problem is very susceptible to the dying ReLU problem [53] due to the very different values of both inputs and outputs.
- **Loss function**: For the loss function, we use the relative mean weighted error:

$$\text{Loss} = \sum \frac{(A - g(V,E;\theta))^2}{A^2}. \tag{3}$$

  The reason we use the relative mean weighted error is again due to the large dynamic ranges experienced in our dataset so that every body is emphasised equally during training, not only the ones with large accelerations.
- **Spherical coordinates**: Our GN takes as inputs a three-vector for every pair of bodies representing the displacements and outputs a second three-vector which corresponds to the force However, due to the large dynamic range of input displacements, we transform the input displacement from Cartesian into spherical coordinates, using $\log_{10}$ to transform the magnitude, as inputs. Similarly, the output force is assumed to be in spherical coordinates, whose magnitude component is transformed through an exponential function back into Cartesian. This allows the GN to learn forces of very different magnitudes, without requiring the parameter distribution inside the GN to have a large dynamic range itself.
- **Data augmentation**: a random three-dimensional rotation is applied to the input graph at every training iteration. This serves as data augmentation, useful for our limited-size training data. It also prevents biases from being created inside the model, and encourages a learned rotational equivariance: for example, the Solar System is largely confined to a plane (which could bias along the rotational axis), and some planets are not observed to complete an entire orbit in the training set (which could bias in their particular direction).
- **Training noise**: During training, we corrupted the input states with Gaussian noise to improve the model's robustness to error over long rollouts at test time. This technique has been used widely in GN-based learned simulators [17, 18]: it is believed to help the model close the gap between the distribution of training input states, which are always from the true observations, and rollout input states, which are predicted by the model and incur some error.
- **Early stopping**: We stop training once a threshold was reached where 20 epochs experienced no improvement in the validation loss, to prevent overfitting.
- **Multiple runs**: To estimate the uncertainty in our estimation of the masses, we repeat the minimization procedure for ten different random seeds, and calculate a mean and standard deviation in the mass estimates from the different best fits.
- **Local minima**: Our loss function has multiple local minima, in which gradient descent is at risk of becoming 'stuck'. Therefore, we restart the minimization when training stops with a validation loss larger than 0.5 (for reference, the best fit loss is typically close to 0.05).

Note that these changes were adopted to optimize the dynamics learned by the GNN before starting the symbolic regression. Therefore, they should not bias our results, only improve the GNN's ability to predict particle positions.

3.3. Symbolic regression implementation details

We used the *PySR* [14, 47] library[12] for symbolic regression which was developed by some of the authors. *PySR* is an open-source analogue to *eureqa* [26], which has a Python API and also supports distributed computation and custom operators and losses.

PySR uses a tree search algorithm to produce a set of candidate equations, that go from some input features (in our cases displacements and learned masses) to some outputs (in our case forces). The objective of *PySR* in this case, is to find a 'simple' and interpretable equation that resembles the interaction predicted by the learned simulator. We do this because we are looking for physical laws that can explain nature with simple equations, as opposed to the high complexity of a neural network. To accomplish this target simplicity, *PySR* assigns to each proposed equation a score, calculated as the ratio between the increase in accuracy (in our case, the decrease in our error metric $\ell_{\rm SR}$) and the increase in complexity with respect to the previous proposed equation. The complexity is calculated from the number of terms and operators that are used in the equation. More details about this can be found in a coming paper. It is clear that different options could be used for both the complexity, accuracy, and score calculations. Therefore, we do not claim that *PySR* uniquely obtains the perfect equation. Instead, its role is to produce a set of candidate equations from the infinite set of possible ones, with a complexity that is orders of magnitude lower than the complexity of a deep neural network.

For this application, we select a dataset $\mathcal{D}_{\rm SR}$ consisting of 500 data points that were not used during training of the learned simulator. Each of these points contains as inputs the learned scalar variables and displacements between a pair of randomly selected bodies at a random time step $x = (v_{r_k}, v_{s_k}, \mathbf{e}_k)$, and as outputs the corresponding interaction learned by the GN $f_{\rm GN}(x)$. We add the norm of the displacement vector $|\mathbf{e}_k|$ as an extra input. The allowed operators between these input quantities are addition, subtraction, multiplication and division. The maximum complexity allowed for the equations is 40. We use as our constant optimizer the Broyden–Fletcher–Goldfarb–Shanno algorithm [54–57] with 10 iterations, and our error metric $\ell_{\rm SR}$ is a MSE loss function between the GN interaction $f_{\rm GN}(x)$ and the proposed equation $f_{\rm SR}(x, \theta^*)$.

4. Results

4.1. Learned simulator performance

Our model learns to predict interactions between bodies which generally agree with the observed accelerations. The predicted next-step relative accelerations, $(x_{\rm truth} - x_{\rm pred})^2/x_{\rm truth}^2$, have error of around 0.2% on the validation data, averaged over all bodies and time steps. These accelerations can be time-integrated to roll out predicted trajectories, as shown for the trajectories of the Sun, Venus, and Earth in the top panel of figure 2 (2A and 2B). The predicted dynamics agree with the ground truth observations over short time intervals, and begin to deviate after several months. This is not surprising because the model is trained to predict only the next time step, 30 min in the future, and the strong non-linearity of N-body dynamics lead to small errors rapidly growing over the rollout. Figure 3 shows how the rollout error accumulates over the three years of validation data for the same bodies used in figure 2. Similar figures for all other bodies are available in the supplementary information.

This shows that the GN-based simulator can learn dynamics from real data, rather than simulated data as in previous work. Because our focus here was on symbolic discovery, the learned simulator we used was relatively simple compared to recent GN-based models [17, 18], but with more powerful methods we expect the accuracy would be even greater.

4.2. Hidden property inference

The learned scalar properties v_i, which scale the predicted accelerations and thus play the role of mass, are shown in the top left part of figure 4 (4A) along with the masses per body. The multiple plotted values for each body represent the fit masses from different training runs with different random initializations (discarding runs that get stuck in local minima as described in the previous section), and help give a sense of the uncertainty in the estimates. The results indicate that the scalar quantities learned by our model roughly match the true masses for the bodies represented in our dataset with a mean percent error ~9.1% calculated over all bodies and all different initializations.

[12] https://github.com/MilesCranmer/pysr.

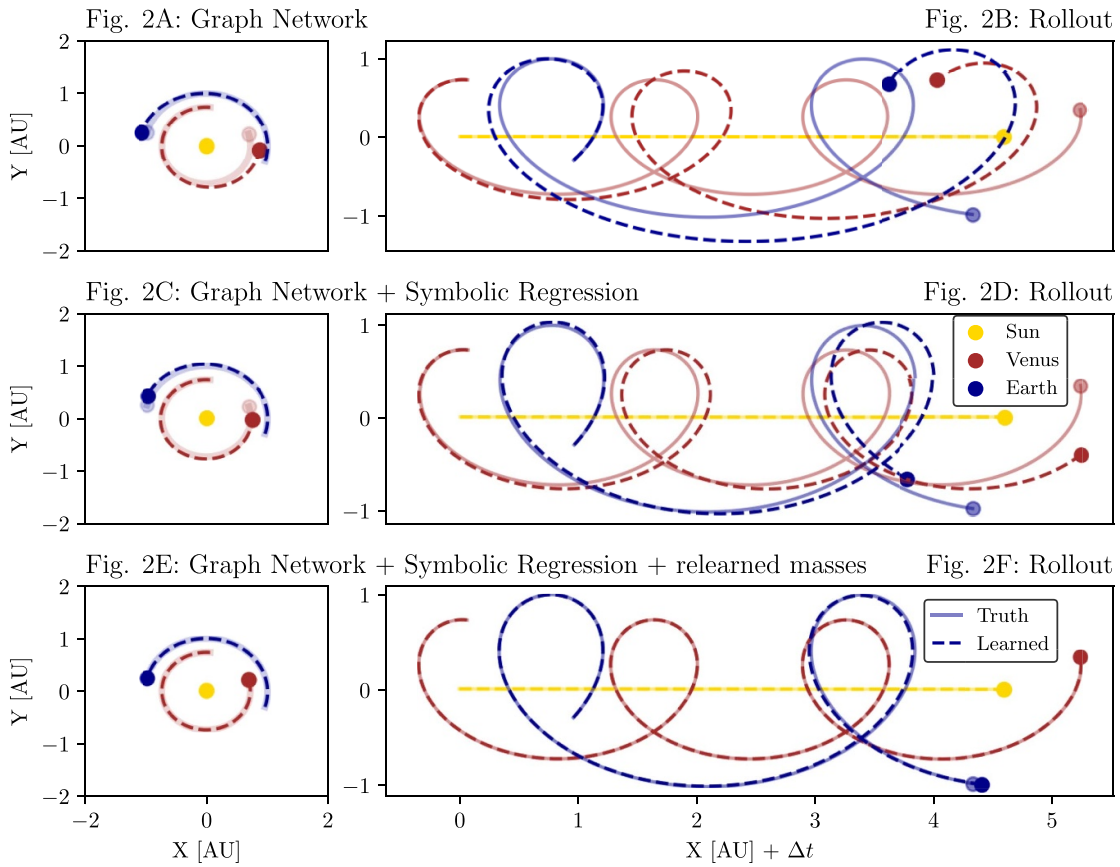

Figure 2. Comparison between data (continuous line) and integrated prediction (dashed line) of same bodies evolved from the same initial conditions, for the interactions predicted by the learned simulator (top, 2A and 2B), the symbolic regression expression (center, 2C and 2D) and the same expression, after re-estimating the masses (bottom, 2E and 2F). The left panels show the orbits for six months. The right panels show the same orbits but for 21 months, with a time displacement along the x-axis, which allows us to visualize the trajectory for a longer time period.

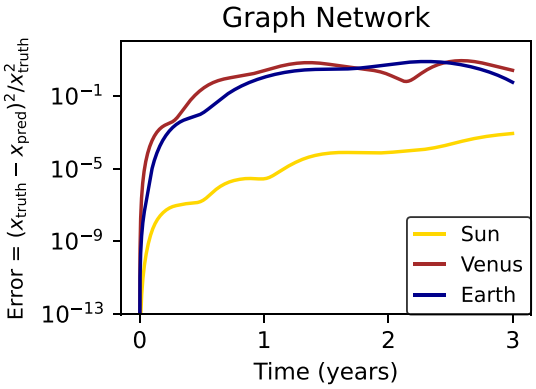

Figure 3. The loss per step, calculated using equation (3), from integrating the trajectories for the bodies shown in figure 2 using the learned simulator. A version of this figure for all other videos can be found in the supplementary information.

Note that the assumption of Newton's second law is key for the purpose of learning the masses. If we do not assume $\vec{F} = M\vec{a}$, the algorithm can learn the correct force, but replace the masses M by $f(M)$; where f is a black box function, and then learn $\vec{F} = f^{-1}(M)\vec{F}$. Therefore, while we can learn the correct dynamics without assuming Newton's second law, we loose the interpretability of our results.

The errors between our model's inferred masses and the true ones demonstrate an interesting pattern: bodies which have little effect on other bodies' accelerations tend to have higher mass error. We computed

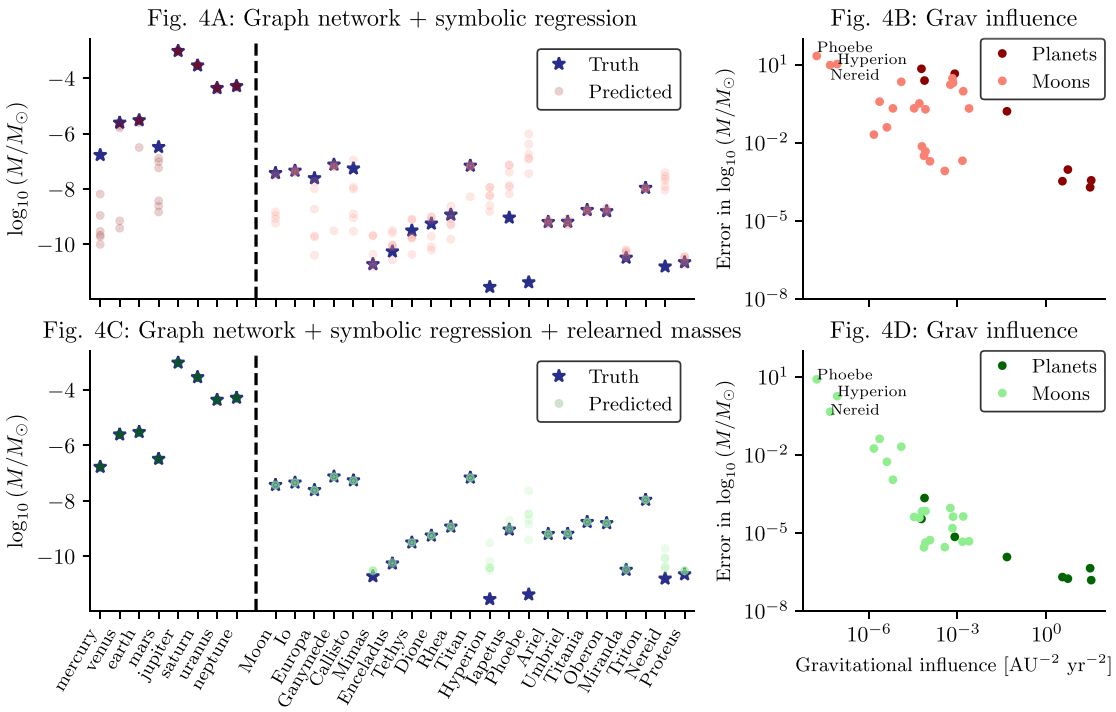

Figure 4. Left: comparison of the learned scalars v_k relative to the Sun v_0 and the known logarithmic masses of the Solar System bodies (in units of solar mass). The multiple red points represent the different mass values for each random seed, and are an attempt to capture the model's uncertainty. The dashed line separates planets and satellites). Right: negative correlation between the gravitational influence exerted by each body, and the error in its mass estimate. The plot shows clearly how the masses of the bodies that have a negligible gravitational influence on others cannot be estimated by our two-step algorithm. The top two panels use the masses trained with the learned simulator, while the bottom two use the masses re-estimated using the symbolic expression.

the gravitational influence of a body n as the sum of gravitational potentials experienced by all other bodies that result from body n,

$$\text{grav. influence}_n = \sum_{i \neq n} V_{\text{grav}}(i,n) = \sum_{i \neq n} -\frac{GM_n}{|\vec{x}_n - \vec{x}_i|}, \tag{4}$$

which sums over all bodies except for n, and where V_{grav} is the gravitational potential and G is the gravitational constant. We calculate this gravitational influence for each body, as the mean of the gravitational influence summed over time, to account for the fact that bodies with eccentric orbits might have a larger gravitational influence at certain points as they get closer to other bodies.

The top right panel of figure 4 (4B) plots the error in the estimate of the mass

$$\frac{1}{N_{\text{inits}}} \sum_{\text{inits}} \left(\log_{10} M_{\text{true}} - \log_{10} M_{\text{pred}}\right)^2, \tag{5}$$

where the sum is over all ten random initializations, as a function of gravitational influence in equation (4). The figure shows a clear negative correlation (Pearson correlation coefficient of -0.64 in log space) between the error in the mass estimate and the gravitational influence. In other words, bodies that have a strong influence on the rest have very accurate masses, while those that are not very influential have poor mass estimates. For example, Mercury and Venus do not have moons, and Mars' moons were too small to be included in our dataset, and thus they do not have nearby bodies to affect. Similarly, the moons Phoebe, Hyperion, and Nereid have small masses and thus have little influence on their planet and nearby moons. Thus the mass errors are to be expected: the 'equivalence principle'[13] holds that for bodies which impart negligible gravitational influence on other bodies in the system, and thus do not influence other bodies' accelerations, their masses are, in ML parlance, 'unidentifiable', meaning such masses are difficult to estimate accurately.

[13] Because $F = m_{\text{body}}a$, and the F function includes m_{body} in the numerator, m_{body} cancels and is not required to compute its own acceleration.

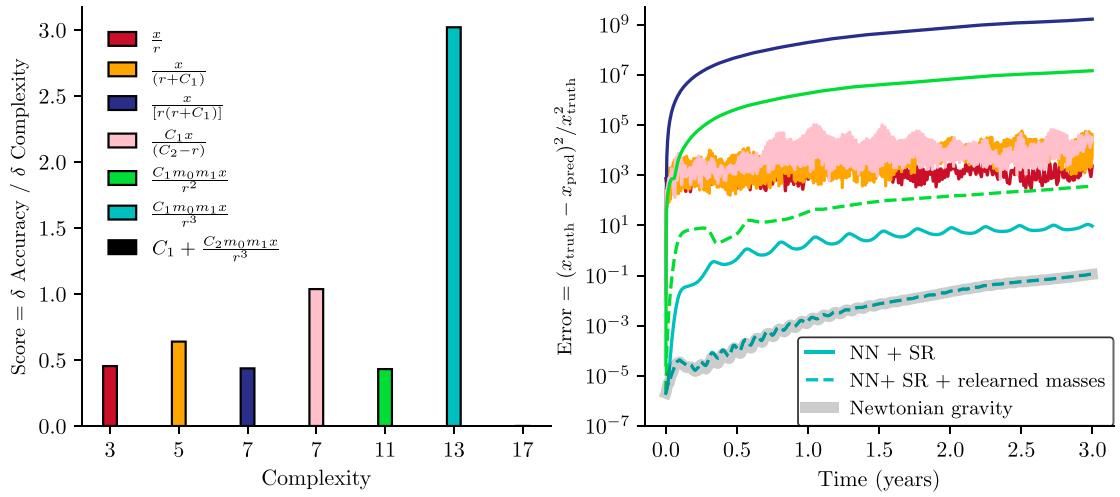

Figure 5. Left: discovered equations from our learned simulator across different random initializations. The different equations are sorted in order of increasing complexity along the x-axis. The output variable in all cases is F_x, but similarly results are obtained for F_y and F_z. The y-axis shows the equation score, which balances loss and complexity, as described in [47]. Right: loss per step on predicted orbits using each equation, summed over all planets. Note that we do not show the complexity 17 equation, as the results perfectly overlap with those of the complexity 13 equation. For the two equations that have masses, we plot the error using the masses from the learned simulator (continuous line) and from refitting the masses after obtaining the equation (dashed). The grey, thicker line is the error obtained using Newtonian gravity with the correct masses.

4.3. Symbolic discovery

Our symbolic regression procedure correctly discovered Newton's law of gravity. The left panel in figure 5 shows candidate equations obtained by *PySR*. The top six equations were those with the highest score, of the over one hundred million tested, sorted in order of increasing complexity. The highest bar corresponds to one with the same form as Newton's gravity. The seventh, rightmost bar, was the best equation which had higher complexity than the best equation, which we plotted in order to demonstrate that increasing complexity does not necessarily provide a better score.

To show how each equation fared in predicting orbital trajectories, the right panel of figure 5 plots their respective rollout errors over the three years of validation data, and compares with the true data. The lowest error is the cyan line's equation, which corresponds to Newton's law of gravity,

$$\vec{F} = -\frac{G_{\text{learned}} M_1 M_2}{r^3}\vec{r}, \tag{6}$$

where M_i are the masses learned by our learned simulator, shown in the top panel of figure 4.

Our symbolic regression method also learns a value of the gravitational constant which is very similar to the true one. Note, similar to how we fit masses relative to the Sun's mass, constants which include a mass unit are also relative to some reference mass.

We plotted the rollout trajectories over the three years of validation data using the best-fit equation in the second row of figure 2 (2C and 2D). The orbits predicted by the discovered symbolic expression are more accurate over time than those from the learned simulator. This means that despite *PySR*'s fitted formula being simpler than the learned simulator's neural network-based one, it yields more accurate predictions.

4.4. Relearning the masses

Having determined the correct form of the interactions between bodies, we can then *re*-estimate the hidden properties. We replaced the MLP edge function within the learned simulator's GN with the best-fit symbolic expression (equation (6)), and re-trained the mass and gravitational variables in the same manner as we originally trained the learned simulator. The bottom-left panel of figure 4 (4C) shows how the mass estimates are far more accurate than they were from the original learned simulator training, with a mean percent error calculated over all bodies and iterations of $\sim$1.6%, more than a factor of five lower than before relearning the masses. Similarly, the bottom-right panel of figure 4 (4D) clearly shows how the negative correlation between error in mass estimate and gravitational influence in other bodies is greatly strengthened by re-learning the masses, with the Pearson correlation coefficient in log-space going from -0.64 (before re-learning the masses) to -0.87 (after re-learning).

We also predicted new trajectories with the symbolic equation and re-learned masses (bottom panel of figure 2, 2E and 2F), which shows the trajectories are far more accurate than those obtained directly from both the learned simulator, and the symbolic equation with original masses. This shows that our learned simulator tends to overfit, compared to the simpler model given by the equation. The blue dashed curve in the right part of figure 5 shows the difference between data and prediction for these re-estimated masses. The figure clearly shows how this outperforms the learned simulator and symbolic regression, and performs just as well as Newtonian gravity using the correct parameters (thick grey curve). Therefore, our algorithm obtains the correct equation for Newtonian gravity (figure 5) and very accurate values for the masses of the bodies (bottom part of figure 4), using only the orbit data, graph structure, and some inductive biases as input information.

5. Discussion and conclusions

Our results show that our two-step approach—training a neural network simulator with physical inductive biases, then interpreting what it has learned using symbolic regression—is a powerful tool for discovering physical laws from real observations. We (re-)discovered Newton's formula for gravitational force from observed trajectories of the Sun, planets, and moons of our Solar System. Furthermore, our method shows a novel aspect of automated scientific discovery, the ability to learn complex properties of the system. We recover the masses of the bodies in the Solar System with very high precision.

While our method allows us to re-discover Newton's formula and the masses, it is important to note that this was only possible through the use of inductive biases, particularly Newton's second and third law, and spherical symmetry. Furthermore, we made use of choices such as spherical coordinates and logarithmic units, which facilitated the learning. This illustrates that while automated theory formation with ML is possible, it does require some prior knowledge. Our understanding of the system can therefore greatly facilitate the task at hand.

While automated theory formation is a very promising and exciting field of work, it is important to consider the limitations of this procedure. First, while we can provide a rough estimate of the uncertainty in the mass estimates by running with multiple random seeds, this does not produce a true estimate of the errors, and instead shows multiple local minima where the algorithm terminates. To perform Bayesian inference on the mass estimates, we would need to model the posterior distribution on each mass, which cannot be done with our current GN algorithm, which uses gradient descent to produce point estimates. Bayesian neural networks could provide a future avenue for this. Second, while our algorithm can provide a scientist with candidate equations that produce a good fit to the data, as shown in figure 5, the specific scoring function used to measure the quality of equations (e.g. complexity vs. accuracy) warrants further exploration. The scientist's preferences for what makes a 'good' equation should be expressed, and more generally, the candidate equations should be viewed as a narrower palette of choices, which should be subject to further experimentation.

This work offers a new way of marrying modern ML methods with automatic theory formation and demonstrates its efficacy in the context of complex real-world data. Even though the law we (re-)discovered is already known, of course, the purpose of this work is to confirm that known laws and hidden properties are *discoverable* with our method. This demonstrates the potential for using ML techniques to aid in scientific discovery and theory evaluation.

Data availability statement

The data that support the findings of this study will be openly available following an embargo at the following URL/DOI: https://github.com/Pablo-Lemos/orbits.

Acknowledgments

We organize the referees at the ML4Astro Machine Learning for Astrophysics Workshop at the Thirty-ninth International Conference on Machine Learning (ICML 2022), for comments and feedback in previous versions of this work. P L acknowledges support by the UK STFC Grant ST/T000473/1.

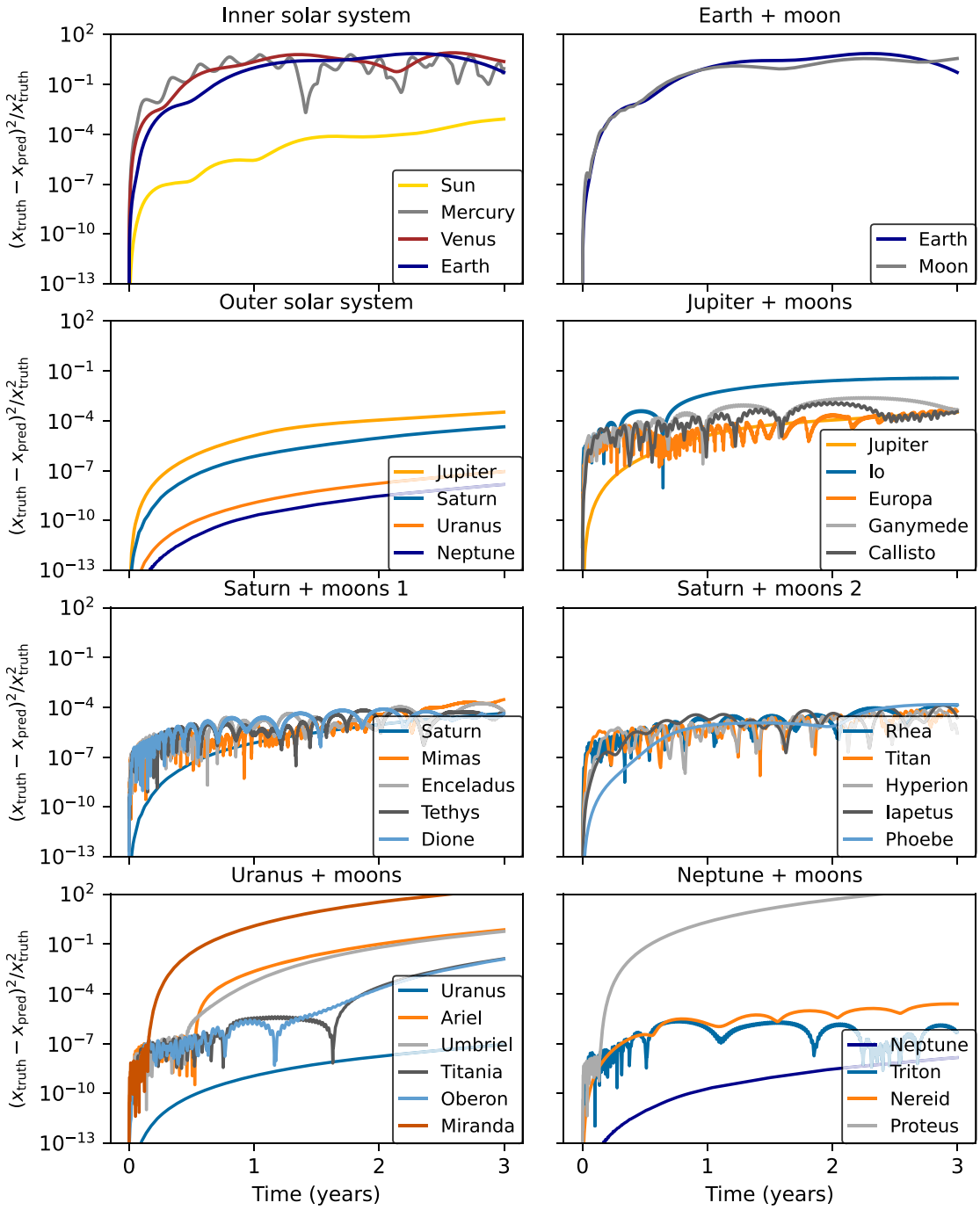

Figure A1. The loss per step, calculated using equation (3), from integrating the trajectories for the bodies shown in figure 2 using the learned simulator.

Appendix A. Rollout errors

Figure A1 shows the rollout errors for all the bodies used in this paper, defined as $(x_{\text{truth}} - x_{\text{pred}})^2/x^2_{\text{truth}}$ where x_{pred} is the trajectory obtained when integrating using the interaction learned by the GN, and x_{truth} is the real data. This curves are calculated from data that was not used during training. This plot is similar to figure 3 in the text, but showing all bodies in our system.

Appendix B. Inductive biases

Table B1 summarizes the inductive biases assumed in this work, and described throughout the body of the paper.

Table B1. A list of inductive biases assumed in this work.

1.	All bodies are observed, and there are no unobserved bodies.
2.	Permutation equivariance for the bodies (the reason for using graph network)
3.	Newton's second law
4.	Newton's third law
5.	A single scalar parameter for each body
6.	Sum aggregation for the forces
7.	Rotational equivariance
8.	Three-vector for edge message output (with the semantics of 'force')
9.	Use displacement as an input feature for message passing

ORCID iD

Pablo Lemos https://orcid.org/0000-0002-4728-8473

References

[1] Bourilkov D 2020 *Int. J. Mod. Phys.* A **34** 1930019
[2] Jumper J *et al* 2021 *Nature* **596** 583–9
[3] He S, Li Y, Feng Y, Ho S, Ravanbakhsh S, Chen W and Póczos B 2019 *Proc. Natl Acad. Sci.* **116** 13825–32
[4] Rashed E A and Samir A E S M 2020 arXiv:2003.04480
[5] Rashed E A and Samir A E S M 2021 *Nucl. Instrum. Methods Phys. Res.* A **985** 164652
[6] Sooknunan K, Lochner M, Bassett B A, Peiris H V, Fender R, Stewart A J, Pietka M, Woudt P A, McEwen J D and Lahav O 2021 *Mon. Not. R. Astron. Soc.* **502** 206–24
[7] Nolan S P, Smerzi A and Pezzè L 2020 arXiv:2006.02369
[8] Green S R and Gair J 2021 *Mach. Learn.: Sci. Technol.* **2** 03LT01
[9] Jeffrey N, Alsing J and Lanusse F 2021 *Mon. Not. R. Astron. Soc.* **501** 954–69
[10] Jumper J M *et al* 2021 *Nature* **596** 583–9
[11] Gligorijevic V, Renfrew P D, Kosciolek T, Leman J K, Cho K, Vatanen T, Berenberg D, Taylor B, Fisk I M, Xavier R J, Knight R and Bonneau R 2019 *bioRxiv Preprint* www.biorxiv.org/content/early/2019/10/04/786236 (Accessed 4 October 2019)
[12] Brown T B *et al* 2020 arXiv:2005.14165
[13] Cranmer M D, Xu R, Battaglia P and Ho S 2019 arXiv:1909.05862
[14] Cranmer M, Sanchez-Gonzalez A, Battaglia P, Xu R, Cranmer K, Spergel D and Ho S 2020 arXiv:2006.11287
[15] Battaglia P W *et al* 2018 arXiv:1806.01261
[16] Battaglia P W, Pascanu R, Lai M, Rezende D and Kavukcuoglu K 2016 arXiv:1612.00222
[17] Sanchez-Gonzalez A, Godwin J, Pfaff T, Ying R, Leskovec J and Battaglia P 2020 Learning to simulate complex physics with graph networks *Int. Conf. on Machine Learning (PMLR)* pp 8459–68
[18] Pfaff T, Fortunato M, Sanchez-Gonzalez A and Battaglia P W 2020 arXiv:2010.03409
[19] Langley P, Simon H A and Bradshaw G L 1987 *Heuristics for Empirical Discovery* (Springer) pp 21–54
[20] Langley P 1977 Bacon: a production system that discovers empirical laws *IJCAI'77: Proc. 5th Int. Joint Conf. on Artificial Intelligence* vol 1 p 344
[21] Kokar M 1986 *Mach. Learn.* **1** 403–22
[22] Langley P and Zytkow J M 1989 *Artif. Intell.* **40** 283–312
[23] Zembowicz R and Żytkow J M 1992 Discovery of equations: experimental evaluation of convergence *Proc. 10th National Conf. on Artificial Intelligence AAAI'92* (AAAI Press) pp 70–75
[24] Todorovski L C 1997 Declarative bias in equation discovery *ICML'97: Proc. 14th Int. Conf. on Machine Learning (July 1997)* pp 376–84
[25] Bongard J and Lipson H 2007 *Proc. Natl Acad. Sci.* **104** 9943–8
[26] Schmidt M and Lipson H 2009 *Science* **324** 81–85
[27] Brunton S L, Proctor J L and Kutz J N 2016 *Proc. Natl Acad. Sci.* **113** 3932–7
[28] Sahoo S, Lampert C and Martius G 2018 Learning equations for extrapolation and control *Proc. Machine Learning Research (PMLR)* vol 80, ed J Dy and A Krause pp 4442–50
[29] Kusner M J, Paige B and Hernández-Lobato J M 2017 Grammar variational autoencoder arXiv:1703.01925
[30] Udrescu S M and Tegmark M 2020 *Sci. Adv.* **6** eaay2631
[31] Lusch B, Kutz J N and Brunton S L 2018 *Nat. Commun.* **9** 4950
[32] Lange H, Brunton S L and Kutz N 2020 arXiv:2004.00574
[33] Both G J, Choudhury S, Sens P and Kusters R 2019 DeepMoD: deep learning for model discovery in noisy data (arXiv:1904.09406)
[34] Atkinson S, Subber W, Wang L, Khan G, Hawi P and Ghanem R 2019 arXiv:1910.05117
[35] Rackauckas C, Ma Y, Martensen J, Warner C, Zubov K, Supekar R, Skinner D and Ramadhan A 2020 arXiv:2001.04385
[36] Chen Z, Liu Y and Sun H 2020 Deep learning of physical laws from scarce data (arXiv:2005.03448)
[37] Vaddireddy H, Rasheed A, Staples A E and San O 2020 *Phys. Fluids* **32** 015113
[38] Guimerà R, Reichardt I, Aguilar-Mogas A, Massucci F A, Miranda M, Pallarès J and Sales-Pardo M 2020 *Sci. Adv.* **6** eaav6971
[39] Virgolin M, Alderliesten T, Witteveen C and Bosman P A N 2021 *Evol. Comput.* **29** 211–37
[40] Champion K, Lusch B, Kutz J N and Brunton S L 2019 arXiv:1904.02107
[41] Wu T and Tegmark M 2019 *Phys. Rev.* E **100** 033311
[42] Udrescu S M, Tan A, Feng J, Neto O, Wu T and Tegmark M 2020 *Advances in Neural Information Processing Systems* vol 33 pp 4860–71
[43] Mundhenk T *et al* 2021 *Advances in Neural Information Processing Systems* vol 34 pp 24912–23
[44] Wetzel S J, Melko R G, Scott J, Panju M and Ganesh V 2020 *Phys. Rev. Res.* **2** 033499

[45] Wetzel S J and Scherzer M 2017 *Phys. Rev.* B **96** 184410
[46] Zhong M, Miller J and Maggioni M 2021 arXiv:2108.11894
[47] Cranmer M 2020b Pysr: fast & parallelized symbolic regression in python/julia (available at: https://zenodo.org/record/4052869)
[48] Cohen T S and Welling M 2016 arXiv:1602.07576
[49] Giorgini J D, Yeomans D K, Chamberlin A B, Chodas P W, Jacobson R A, Keesey M S, Lieske J H, Ostro S J, Standish E M and Wimberly R N 1996 JPL's on-line solar system data service *AAS/Division for Planetary Sciences Meeting Abstracts #28* vol 28 p 25.04
[50] Giorgini J D, Chodas P W and Yeomans D K 2001 Orbit uncertainty and close-approach analysis capabilities of the horizons on-line ephemeris system *AAS/Division for Planetary Sciences Meeting Abstracts #33* vol 33 p 58.13
[51] Abadi M *et al* 2015 TensorFlow: large-scale machine learning on heterogeneous systems software available from tensorflow.org (available at: http://tensorflow.org/)
[52] Agarap A F 2018 arXiv:1803.08375
[53] Lu L, Shin Y, Su Y and Karniadakis G E 2019 arXiv:1903.06733
[54] Broyden C G 1970 *IMA J. Appl. Math.* **6** 76–90
[55] Fletcher R 1970 *Comput. J.* **13** 317–22
[56] Goldfarb D 1970 *Math. Comput.* **24** 23–26
[57] Shanno D F 1970 *Math. Comput.* **24** 647–56