Discovering Symbolic Models from Deep Learning with Inductive Biases

Miles Cranmer$^1$ Alvaro Sanchez-Gonzalez$^2$ Peter Battaglia$^2$ Rui Xu$^1$

Kyle Cranmer$^3$ David Spergel$^{4,1}$ Shirley Ho$^{4,3,1,5}$

1 Princeton University, Princeton, USA  2 DeepMind, London, UK
3 New York University, New York City, USA  4 Flatiron Institute, New York City, USA
5 Carnegie Mellon University, Pittsburgh, USA

Abstract

We develop a general approach to distill symbolic representations of a learned deep model by introducing strong inductive biases. We focus on Graph Neural Networks (GNNs). The technique works as follows: we first encourage sparse latent representations when we train a GNN in a supervised setting, then we apply symbolic regression to components of the learned model to extract explicit physical relations. We find the correct known equations, including force laws and Hamiltonians, can be extracted from the neural network. We then apply our method to a non-trivial cosmology example—a detailed dark matter simulation—and discover a new analytic formula which can predict the concentration of dark matter from the mass distribution of nearby cosmic structures. The symbolic expressions extracted from the GNN using our technique also generalized to out-of-distribution-data better than the GNN itself. Our approach offers alternative directions for interpreting neural networks and discovering novel physical principles from the representations they learn.

1 Introduction

The miracle of the appropriateness of the language of mathematics for the formulation of the laws of physics is a wonderful gift which we neither understand nor deserve. We should be grateful for it and hope that it will remain valid in future research and that it will extend, for better or for worse, to our pleasure, even though perhaps also to our bafflement, to wide branches of learning.—Eugene Wigner “The Unreasonable Effectiveness of Mathematics in the Natural Sciences” [1].

For thousands of years, science has leveraged models made out of closed-form symbolic expressions, thanks to their many advantages: algebraic expressions are usually compact, present explicit interpretations, and generalize well. However, finding these algebraic expressions is difficult. Symbolic regression is one option: a supervised machine learning technique that assembles analytic functions to model a given dataset. However, typically one uses genetic algorithms—essentially a brute force procedure as in [2]—which scale exponentially with the number of input variables and operators. Many machine learning problems are thus intractable for traditional symbolic regression.

On the other hand, deep learning methods allow efficient training of complex models on high-dimensional datasets. However, these learned models are black boxes, and difficult to interpret. Code for our models and experiments can be found at [https://github.com/MilesCranmer/symbolic_deep_learning](https://github.com/MilesCranmer/symbolic_deep_learning).

Furthermore, generalization is difficult without prior knowledge about the data imposed directly on the model. Even if we impose strong inductive biases on the models to improve generalization, the learned parts of networks typically are linear piece-wise approximations which extrapolate linearly (if using ReLU as activation \cite{3}).

Here, we propose a general framework to leverage the advantages of both deep learning and symbolic regression. As an example, we study Graph Networks (GNs or GNNs) \cite{4} as they have strong and well-motivated inductive biases that are very well suited to problems we are interested in. Then we apply symbolic regression to fit different internal parts of the learned model that operate on reduced size representations. The symbolic expressions can then be joined together, giving rise to an overall algebraic equation equivalent to the trained GN. Our work is a generalized and extended version of that in \cite{5}.

We apply our framework to three problems—rediscovering force laws, rediscovering Hamiltonians, and a real world astrophysical challenge—and demonstrate that we can drastically improve generalization, and distill plausible analytical expressions. We not only recover the injected closed-form physical laws for Newtonian and Hamiltonian examples, we also derive a new interpretable closed-form analytical expression that can be useful in astrophysics.

# Framework

Our framework can be summarized as follows. (1) Engineer a deep learning model with a separable internal structure that provides an inductive bias well matched to the nature of the data. Specifically, in the case of interacting particles, we use Graph Networks as the core inductive bias in our models. (2) Train the model end-to-end using available data. (3) Fit symbolic expressions to the distinct functions learned by the model internally. (4) Replace these functions in the deep model by the symbolic expressions. This procedure with the potential to discover new symbolic expressions for non-trivial datasets is illustrated in fig. 1.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{framework.png}
\caption{A cartoon depicting how we extract physical equations from a dataset.}
\end{figure}

Particle systems and Graph Networks. In this paper we focus on problems that can be well described as interacting particle systems. Nearly all of the physics we experience in our day-to-day life can be described in terms of interactions rules between particles or entities, so this is broadly relevant. Recent work has leveraged the inductive biases of Interaction Networks (INs) \cite{6} in their generalized form, the Graph Network, a type of Graph Neural Network \cite{7,8,9}, to learn models of particle systems in many physical domains \cite{6,10,11,12,13,14,15,16}.

Therefore we use Graph Networks (GNs) at the core of our models, and incorporate into them physically motivated inductive biases appropriate for each of our case studies. Some other interesting
approaches for learning low-dimensional general dynamical models include \[17, \ 18, \ 19\]. Other related work which studies the physical reasoning abilities of deep models include \[20, \ 21, \ 22, \ 23\].

Internally, GNs are structured into three distinct components: an edge model, a node model, and a global model, which take on different but explicit roles in a regression problem. The edge model, or “message function,” which we denote by \( \phi^e \), maps from a pair of nodes \((v_i, v_j) \in \mathbb{R}^L\) connected by an edge in a graph together with some vector information for the edge, to a message vector. These message vectors are summed element-wise for each receiving node over all of their sending nodes, and the summed vector is passed to the node model. The node model, \( \phi^v \), takes the receiving node and the summed message vector, and computes an updated node: a vector representing some property or dynamical update. Finally, a global model \( \phi^u \) aggregates all messages and all updated nodes and computes a global property. \( \phi^e, \phi^v, \phi^u \) are usually approximated using multilayer-perceptrons, making them differentiable end-to-end. More details on GNs are given in the appendix. We illustrate the internal structure of a GN in fig. 2.

Figure 2: An illustration of the internal structure of the graph neural network we use in some of our experiments. Note that the comparison to Newtonian mechanics is purely for explanatory purposes, but is not explicit. Differences include: the “forces” (messages) are often high dimensional, the nodes need not be physical particles, \( \phi^e \) and \( \phi^v \) are arbitrary learned functions, and the output need not be an updated state. However, the rough equivalency between this architecture and physical frameworks allows us to interpret learned formulas in terms of existing physics.

GNs are the ideal candidate for our approach due to their inductive biases shared by many physics problems. (a) They are equivariant under particle permutations. (b) They are differentiable end-to-end and can be trained efficiently using gradient descent. (c) They make use of three separate and interpretable internal functions \( \phi^e, \phi^v, \phi^u \), which are our targets for the symbolic regression. GNs can also be embedded with additional symmetries as in \[24, \ 25\], but we do not implement these.

**Symbolic regression.** After training the Graph Networks, we use the symbolic regression package *eureqa* \[2\] to perform symbolic regression and fit compact closed-form analytical expressions to \( \phi^e, \phi^v, \phi^u \) independently. *eureqa* works by using a genetic algorithm to combine algebraic expressions stochastically. The technique is similar to natural selection, where the “fitness” of each expression is defined in terms of simplicity and accuracy. The operators considered in the fitting process are \(+, -, \times, /, >, <, ^, \exp, \log, \text{IF} \) as well as real constants. After fitting expressions to each part of the graph network, we substitute the expressions into the model to create an alternative analytic model. We then refit any parameters in the symbolic model to the data a second time, to avoid the accumulated approximation error. Further details are given in the appendix.

This approach gives us an explicit way of interpreting the trained weights of a structured neural network. An alternative way of interpreting GNNs is given in \[26\]. This new technique also allows
we to extend symbolic regression to high-dimensional datasets, where it is otherwise intractable. As an example, consider attempting to discover the relationship between a scalar and a time series, given data \( \{(z_i, \{x_{i,1}, x_{i,2}, \ldots x_{i,100}\}\) \}, where \( z_i \in \mathbb{R} \) and \( x_{i,j} \in \mathbb{R}^5 \). Assume the true relationship as \( z_i = y_i^2 \), for \( y_i = \sum_{j=1}^{100} y_{i,j}, y_{i,j} = \exp(x_{i,j,3}) + \cos(2x_{i,j,1}) \). Now, in a learnable model, assume an inductive bias \( z_i = f(\sum_{j=1}^{100} g(x_{i,j})) \) for scalar functions \( f \) and \( g \). If we need to consider \( 10^9 \) equations for both \( f \) and \( g \), then a standard symbolic regression search would need to consider their combination, leading to \( (10^9)^2 = 10^{18} \) equations in total. But if we first fit a neural network for \( f \) and \( g \), and after training, fit an equation to \( f \) and \( g \) separately, we only need to consider \( 2 \times 10^9 \) equations. In effect, we factorize high-dimensional datasets into smaller sub-problems that are tractable for symbolic regression.

We emphasize that this method is not a new symbolic regression technique by itself; rather, it is a way of extending any existing symbolic regression method to high-dimensional datasets by the use of a neural network with a well-motivated inductive bias. While we chose eureqa for our experiments based on its efficiency and ease-of-use, we could have chosen another low-dimensional symbolic regression package, such as our new high-performance package PySR\(^1\)\(^2\). Other community packages such as \( 28, 29, 30, 31, 32, 33, 34, 35, 36 \), could likely also be used and achieve similar results (although \( 32 \) is unable to fit the constants required for the tasks here). Ref. \( 29 \) is an interesting approach that uses gradient descent on a pre-defined equation up to some depth, parametrized with a neural network, instead of genetic algorithms; \( 35 \) uses gradient descent on a latent embedding of an equation; and \( 36 \) demonstrates Monte Carlo Tree Search as a symbolic regression technique, using an asymptotic constraint as input to a neural network which guides the search. These could all be used as drop-in replacements for eureqa here to extend their algorithms to high-dimensional datasets.

We also note several exciting packages for symbolic regression of partial differential equations on gridded data: \( 37, 38, 39, 40, 41, 42 \). These either use sparse regression of coefficients over a library of PDE terms, or a genetic algorithm. While not applicable to our use-cases, these would be interesting to consider for future extensions to gridded PDE data.

**Compact internal representations.** While training, we encourage the model to use compact internal representations for latent hidden features (e.g., messages) by adding regularization terms to the loss (we investigate using L1 and KL penalty terms with a fixed prior, see more details in the Appendix). One motivation for doing this is based on Occam’s Razor: science always prefers the simpler model or representation of two which give similar accuracy. Another stronger motivation is that if there is a law that perfectly describes a system in terms of summed message vectors in a compact space (what we call a linear latent space), then we expect that a trained GN, with message vectors of the same dimension as that latent space, will be mathematical rotations of the true vectors. We give a mathematical explanation of this reasoning in the appendix, and emphasize that while it may seem obvious now, our work is the first to demonstrate it. More practically, by reducing the size of the latent representations, we can filter out all low-variance latent features without compromising the accuracy of the model, and vastly reducing the dimensionality of the hidden vectors. This makes the symbolic regression of the internal models more tractable.

**Implementation details.** We write our models with PyTorch \( 43 \) and PyTorch Geometric\(^4\)\(^4\). We train them with a decaying learning schedule using Adam \( 45 \). The symbolic regression technique is described in section 4.7. More details are provided in the Appendix.

### 3 Case studies

In this section we present three specific case studies where we apply our proposed framework using additional inductive biases.

**Newtonian dynamics.** Newtonian dynamics describes the dynamics of particles according to Newton’s law of motion: the motion of each particle is modeled using incident forces from nearby particles, which change its position, velocity and acceleration. Many important forces in physics (e.g., gravitational force \( -\frac{Gm_1 m_2}{r^2} \)) are defined on pairs of particles, analogous to the message function \( \phi \) of our Graph Networks. The summation that aggregates messages is analogous to the

\(^{1}\)https://github.com/MilesCranmer/PySR
calculation of the net force on a receiving particle. Finally, the node function, $\phi^v$, acts like Newton’s law: acceleration equals the net force (the summed message) divided by the mass of the receiving particle.

To train a model on Newtonian dynamics data, we train the GN to predict the instantaneous acceleration of the particle against that calculated in the simulation. While Newtonian mechanics inspired the original development of INs, never before has an attempt to distill the relationship between the forces and the learned messages been successful. When applying the framework to this Newtonian dynamics problem (as illustrated in fig. 1), we expect the model trained with our framework to discover that the optimal dimensionality of messages should match the number of spatial dimensions. We also expect to recover algebraic formulas for pairwise interactions, and generalize better than purely learned models. We refer our readers to section 4.1 and the Appendix for more details.

**Hamiltonian dynamics.** Hamiltonian dynamics describes a system’s total energy $H(q, p)$ as a function of its canonical coordinates $q$ and momenta $p$—e.g., each particle’s position and momentum. The dynamics of the system change perpendicularly to the gradient of $H$: $\frac{dq}{dt} = \frac{\partial H}{\partial p}$, $\frac{dp}{dt} = -\frac{\partial H}{\partial q}$.

Here, we will use a variant of a Hamiltonian Graph Network (HGN) [46] to learn $H$ for the Newtonian dynamics data. This model is a combination of a Hamiltonian Neural Network [47, 48] and GN. In this case, the global model $\phi^u$ of the GN will output a single scalar value for the entire system representing the energy, and hence the GN will have the same functional form as a Hamiltonian. By then taking the partial derivatives of the GN-predicted $H$ with respect to the position and momentum, $q$ and $p$, respectively, of the input nodes, we will be able to calculate the updates to the momentum and position. We impose a modification to the HGN to facilitate its interpretability, and name this the “Flattened HGN” or FlatHGN; instead of summing high-dimensional encodings of nodes to calculate $\phi^u$, we instead set it to be a sum of scalar pairwise interaction terms, $H_{pair}$ and a per-particle term, $H_{self}$. This is because many physical systems can be exactly described this way. This is a Hamiltonian version of the Lagrangian Graph Network in [49], and is similar to [50]. This is still general enough to express many physical systems, as nearly all of physics can be written as summed interaction energies, but could also be relaxed in the context of the framework.

Even though the model is trained end-to-end, we expect our framework to allow us to extract analytical expressions for both the per-particle kinetic energy, and the pairwise potential energy. We refer our readers to our section 4.2 and the Appendix for more details.

**Dark matter halos for cosmology.** We also apply our framework to a dataset generated from state-of-the-art dark matter simulations [51]. We predict a property (“overdensity”) of a dark matter blob (called a “halo”) from the properties (positions, velocities, masses) of halos nearby. We would like to extract this relationship as an analytic expression so we may interpret it theoretically. This problem differs from the previous two use cases in many ways, including (1) it is a real-world problem where an exact analytical expression is unknown; (2) the problem does not involve dynamics, rather, it is a regression problem on a static dataset; and (3) the dataset is not made of particles, but rather a grid of density that has been grouped and reduced to handmade features. Similarly, we do not know the dimensionality of interactions, should a linear latent space exist. We rely on our inductive bias to find the optimal dimensional of the problem, and then yield an interpretable model that performs better than existing analytical approximations. We refer our readers to our section 4.3 and the Appendix for further details.

### 4 Experiments & results

#### 4.1 Newtonian dynamics

We train our Newtonian dynamics GNs on data for simple N-body systems with known force laws. We then apply our technique to recover the known force laws via the representations learned by the message function $\phi^e$.

**Data.** The dataset consists of N-body particle simulations in two and three dimensions, under different interaction laws. We used the following forces: (a) $1/r$ orbital force: $-m_1m_2 \hat{r}/r$; (b) $1/r^2$ orbital force $-m_1m_2 \hat{r}/r^2$; (c) charged particles force $q_1q_2 \hat{r}/r^2$; (d) damped springs with $|r - 1|^2$
potential and damping proportional and opposite to speed; (e) discontinuous forces, \(-\{0, r^2\}\hat{r}\), switching to 0 force for \(r < 2\); and (f) springs between all particles, a \((r - 1)^2\) potential. The simulations themselves contain masses and charges of 4 or 8 particles, with positions, velocities, and accelerations as a function of time. Further details of these systems are given in the appendix, with example trajectories shown in fig. 4.

**Model training.** The models are trained to predict instantaneous acceleration for every particle given the current state of the system. To investigate the importance of the size of the message representations for interpreting the messages as forces, we train our GN using 4 different strategies: 1. Standard, a GN with 100 message components; 2. Bottleneck, a GN with the number of message components matching the dimensionality of the problem (2 or 3); 3. L\(_1\), same as “Standard” but using a L\(_1\) regularization loss term on the messages with a weight of \(10^{-2}\); and 4. KL same as “Standard” but regularizing the messages using the Kullback-Leibler (KL) divergence with respect to Gaussian prior. Both the L\(_1\) and KL strategies encourage the network to find compact representations for the message vectors, using different regularizations. We optimize the mean absolute loss between the predicted acceleration and the true acceleration of each node. Additional training details are given in the appendix and found in the codebase.

**Performance comparison.** To evaluate the learned models, we generate a new dataset from a different random seed. We find that the model with L\(_1\) regularization has the greatest prediction performance in most cases (see table 3). It is worth noting that the bottleneck model, even though it has the correct dimensionality, performs worse than the model using L\(_1\) regularization under limited training time. We speculate that this may connect to the lottery ticket hypothesis [52].

**Interpreting the message components.** As a first attempt to interpret the information in the message components, we pick the \(D\) message features (where \(D\) is the dimensionality of the simulation) with the highest variance (or KL divergence), and fit each to a linear combination of the true force components. We find that while the GN trained in the Standard setting does not show strong
correlations with force components (also seen in fig. 5), all other models for which the effective message size is constrained explicitly (bottleneck) or implicitly (KL or $L_1$) to be low dimensional yield messages that are highly correlated with the true forces (see table 1 which indicates the fit errors with respect to the true forces), with the model trained with $L_1$ regularization showing the highest correlations. An explicit demonstration that the messages in a graph network learn forces has not been observed before our work.

The messages in these models are thus explicitly interpretable as forces. The video at https://github.com/MilesCranmer/symbolic_deep_learning/blob/master/video_link.txt (fig. 3) shows a fit of the message components over time during training, showing how the model discovers a message representation that is highly correlated with a rotation of the true force vector in an unsupervised way.

<table>
<thead>
<tr>
<th>Sim.</th>
<th>Standard</th>
<th>Bottleneck</th>
<th>$L_1$</th>
<th>KL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Charge-2</td>
<td>0.016</td>
<td>0.947</td>
<td>0.004</td>
<td>0.185</td>
</tr>
<tr>
<td>Charge-3</td>
<td>0.013</td>
<td>0.980</td>
<td>0.002</td>
<td>0.425</td>
</tr>
<tr>
<td>$r^{-1}$.2</td>
<td>0.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.796</td>
</tr>
<tr>
<td>$r^{-1}$.3</td>
<td>0.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.332</td>
</tr>
<tr>
<td>$r^{-2}$.2</td>
<td>0.004</td>
<td>0.993</td>
<td>0.990</td>
<td>0.770</td>
</tr>
<tr>
<td>$r^{-2}$.3</td>
<td>0.002</td>
<td>0.994</td>
<td>0.977</td>
<td>0.244</td>
</tr>
<tr>
<td>Spring-2</td>
<td>0.032</td>
<td>1.000</td>
<td>1.000</td>
<td>0.883</td>
</tr>
<tr>
<td>Spring-3</td>
<td>0.036</td>
<td>0.995</td>
<td>1.000</td>
<td>0.214</td>
</tr>
</tbody>
</table>

Table 1: The $R^2$ value of a fit of a linear combination of true force components to the message components for a given model (see text). Numbers close to 1 indicate the messages and true force are strongly correlated. Successes/failures of force law symbolic regression are tabled in the appendix.

**Symbolic regression on the internal functions.** We now demonstrate symbolic regression to extract force laws from the messages, without using prior knowledge for each force’s form. To do this, we record the most significant message component of $\phi_e^i$, which we refer to as $\phi_e^i$, over random samples of the training dataset. The inputs to the regression are $m_1, m_2, q_1, q_2, \Delta x_1, \Delta x_2, \ldots$ (mass, charge, x-position of receiving and sending node) as well as simplified variables to help the symbolic regression: e.g., $\Delta x$ for $x$ displacement, and $r$ for distance.

We then use *eureqa* to fit the $\phi_e^i$ to the inputs by minimizing the mean absolute error (MAE) over various analytic functions. Analogous to Occam’s razor, we find the “best” algebraic model by asking *eureqa* to provide multiple candidate fits at different complexity levels (where complexity is scored as a function of the number and the type of operators, constants and input variables used), and select the fit that maximizes the fractional drop in mean absolute error (MAE) over the increase in complexity from the next best model: $\left(-\Delta \log(\text{MAE}_c)/\Delta c\right)$. From this, we recover many analytical expressions (this is tabled in the appendix) that are equivalent to the simulated force laws ($a, b$ indicate learned constants):

- Spring, 2D, $L_1$ (expect $\phi_e^i \approx (a \cdot (\Delta x, \Delta y))(r - 1) + b$).
  \[
  \phi_e^i \approx 1.36\Delta y + 0.60\Delta x - \frac{0.60\Delta x + 1.37\Delta y}{r} - 0.0025
  \]

- $1/r^2$, 3D, Bottleneck (expect $\phi_e^i \approx a \cdot (\Delta x, \Delta y, \Delta z)/r^3 + b$).
  \[
  \phi_e^i \approx \frac{0.021\Delta x m_2 + 0.077\Delta y m_2}{r^3}
  \]

- Discontinuous, 2D, $L_1$ (expect $\phi_e^i \approx \text{IF}(r > 2, (a \cdot (\Delta x, \Delta y, \Delta z)) r, 0) + b$).
  \[
  \phi_e^i \approx \text{IF}(r > 2, 0.15r\Delta y + 0.19r\Delta x, 0) - 0.038
  \]

Note that reconstruction does not always succeed, especially for training strategies other than $L_1$ or bottleneck models that cannot successfully find compact representations of the right dimensionality (see some examples in Appendix).

7
### 4.2 Hamiltonian dynamics

Using the same datasets from the Newtonian dynamics case study, we also train our “FlatHGN,” with the Hamiltonian inductive bias, and demonstrate that we can extract scalar potential energies, rather than forces, for all of our problems. For example, in the case of charged particles, with expected potential \( H_{\text{pair}} \approx \frac{q_1 q_2}{r} \), symbolic regression applied to the learned message function yields \( H_{\text{pair}} \approx 0.0019 q_1 q_2 r \).

It is also possible to fit the per-particle term \( H_{\text{self}} \), however, in this case the same kinetic energy expression is recovered for all systems. In terms of performance results, the Hamiltonian models are comparable to that of the L\(_1\) regularized model across all datasets (See Supplementary results table).

Note that in this case, by design, the “FlatHGN” has a message function with a dimensionality of 1 to match the output of the Hamiltonian function which is a scalar, so no regularization is needed, as the message size is directly constrained to the right dimension.

### 4.3 Dark matter halos for cosmology

Now, one may ask: “will this strategy also work for general regression problems, non-trivial datasets, complex interactions, and unknown laws?” Here we give an example that satisfies all four of these concerns, using data from a gravitational simulation of the Universe.

Cosmology studies the evolution of the Universe from the Big Bang to the complex galaxies and stars we see today [53]. The interactions of various types of matter and energy drive this evolution. Dark Matter alone consists of \( \approx \) 85% of the total matter in the Universe [54, 55], and therefore is extremely important for the development of galaxies. Dark matter particles clump together and act as gravitational basins called “halos” which pull regular baryonic matter together to produce stars, and form larger structures such as filaments and galaxies. It is an important question in cosmology to predict properties of dark matter halos based on their “environment,” which consist of other nearby dark matter halos. Here we study the following problem: how can we predict the excess amount of matter (in comparison to its surroundings, \( \delta = \frac{\rho - \langle \rho \rangle}{\langle \rho \rangle} \)) for a dark matter halo based on its properties and those of its neighboring dark matter halos?

A hand-designed estimator for the functional form of \( \delta_i \) for halo \( i \) might correlate \( \delta \) with the mass of the same halo, \( M_i \), as well as the mass within 20 distance units (we decide to use 20 as the smoothing radius): \( \sum_{j \neq i, |r_i - r_j| < 20} M_j \). The intuition behind this scaling is described in [56]. Can we find a better equation that we can fit better to the data, using our methodology?

**Data, training and symbolic regression.** We study this problem with the open sourced N-body dark matter simulations from [51]. We choose the zeroth simulation in this dataset, at the final time step (current day Universe), which contains 215,854 dark matter halos. Each halo has mass \( M_i \), position \( r_i \), and velocity \( v_i \). We also compute the smoothed overdensity \( \delta_i \) at the location of the center of each halo. We convert this set of halos into a graph by connecting halos within fifty distance units (each distance unit is approximately 3 million light years long) of each other. This results in 30,437,218 directional edges between halos, or 71 neighbors per halo on average. We then attempt to predict \( \delta_i \) for each halo with a GN. Training details are the same as for the Newtonian simulations, but we switch to 500 hidden units after hyperparameter tuning based on GN accuracy.

The GN trained with L\(_1\) appears to have messages containing only 1 informative feature, so we extract message samples for this component of the messages over the training set for random pairs of halos, and node function samples for random receiving halos and their summed messages. The formula extracted by the algorithm is given in table 2 as “Best, with mass”. The form of the formula is new and it captures a well-known relationship between halo mass and environment: bias-mass relationship. We refit the parameters in the formula on the original training data to avoid accumulated approximation error from the multiple levels of function fitting. We achieve a loss of 0.0882 where the hand-designed formula achieves a loss of 0.121. It is quite surprising that our formula extracted by our approach is able to achieve a better fit than the formula hand-designed by scientists.

---

2We have removed constant terms that don’t depend on the position or momentum as those are just arbitrary offsets in the Hamiltonian which don’t have an impact on the dynamics. See Appendix for more details.
The flattened Hamiltonian graph network. Because GNs have more explicit substructure than their
δ
We then proceed through the same training procedure as before, learning a GN to predict
was trained. The graph network itself obtains an average error
⟨
Table 2: A comparison of both known and discovered formulas for dark matter overdensity. C
indicates fitted parameters, which are given in the appendix.

The formula makes physical sense. Halos closer to the dark matter halo of interest should influence
its properties more, and thus the summed function scales inversely with distance. Similar to the
hand-designed formula, the overdensity should scale with the total matter density nearby, and we see
this in that we are summing over mass of neighbors. The other differences are very interesting, and
less clear; we plan to do detailed interpretation of these results in a future astrophysics study.

As a followup, we also calculated if we could predict the halo overdensity from only velocity and
position information. This is useful because the most direct observational information available is
in terms of halo velocities. We perform an identical analysis without mass information, and find
a curiously similar formula. The relative speed between two neighbors can be seen as a proxy for
mass, which is seen in table 2. This makes sense as a more massive object will have more gravity,
accelerating falling particles near it to faster speeds. This formula is also new to cosmologists, and
can in principle help push forward cosmological analysis.

Symbolic generalization. As we know that our physical world is well described by mathematics,
we can use it as a powerful prior for creating new models of our world. Therefore, if we distill a
neural network into a simple algebra, will the algebra generalize better to unseen data? Neural nets
excel at learning in high-dimensional spaces, so perhaps, by combining both of these types of models,
one can leverage the unique advantages of each. Such an idea is discussed in detail in [57].

Here we study this on the cosmology example by masking 20% of the data: halos which have
δ
> 1. We then proceed through the same training procedure as before, learning a GN to predict δ
with L
1
regularization, and then extracting messages for examples in the training set. Remarkably, we obtain
a functionally identical expression when extracting the formula from the graph network on this subset
of the data. We fit these constants to the same masked portion of data on which the graph network
was trained. The graph network itself obtains an average error
⟨
0.0634 on the training set, and 0.142 on the out-of-distribution data. Meanwhile, the symbolic expression achieves 0.0811
on the training set, but 0.0892 on the out-of-distribution data. Therefore, for this problem, it seems
a symbolic expression generalizes much better than the very graph neural network it was extracted
from. This alludes back to Eugene Wigner’s article: the language of simple, symbolic models is
remarkably effective in describing the universe.

5 Conclusion

We have demonstrated a general approach for imposing physically motivated inductive biases on
GNs and Hamiltonian GNs to learn interpretable representations, and potentially improved zero-shot
generalization. Through experiment, we have shown that GN models which implement a bottleneck
or L
1
regularization in the message passing layer, or a Hamiltonian GN flattened to pairwise and
self-terms, can learn message representations equivalent to linear transformations of the true force
vector or energy. We have also demonstrated a generic technique for finding an unknown force
law from these models: symbolic regression is capable of fitting explicit equations to our trained
model’s message function. We repeated this for energies instead of forces via introduction of the
“Flattened Hamiltonian Graph Network.” Because GNs have more explicit substructure than their
more homogeneous deep learning relatives (e.g., plain MLPS, convolutional networks), we can draw
more fine-grained interpretations of their learned representations and computations. Finally, we have
demonstrated our algorithm on a non-trivial dataset, and discovered a new law for cosmological dark
matter.
Acknowledgments and Disclosure of Funding

Miles Cranmer would like to thank Christina Kreisch and Francisco Villaescusa-Navarro for assistance with the cosmology dataset; and Christina Kreisch, Oliver Philcox, and Edgar Minasyan for comments on a draft of the paper. The authors would like to thank the reviewers for insightful feedback that improved this paper. Shirley Ho and David Spergel’s work is supported by the Simons Foundation.

Our code made use of the following Python packages: numpy, scipy, sklearn, jupyter, matplotlib, pandas, torch, tensorflow, jax, and torch_geometric [58, 59, 60, 61, 62, 63, 64, 65, 66, 67].

References


Supplementary

A Model Implementation Details

Code for our implementation can be found at https://github.com/MilesCranmer/symbolic_deep_learning. Here we describe how one can implement our model from scratch in a deep learning framework. The main argument in this paper is that one can apply strong inductive biases to a deep learning model to simplify the extraction of a symbolic representation of the learned model. While we emphasize that this idea is general, in this section we focus on the specific Graph Neural Networks we have used as an example throughout the paper.

A.1 Basic Graph Representation

We would like to use the graph $G = (V, E)$ to predict an updated graph $G' = (V', E')$. Our input dataset is a graph $G = (V, E)$ consisting of $N^v$ nodes with $L^v$ features each: $V = \{v_i\}_{i=1}^{N^v}$, with each $v_i \in \mathbb{R}^{L^v}$. The nodes are connected by $N^e$ edges: $E = \{(r_k, s_k)\}_{k=1}^{N^e}$, where $r_k, s_k \in \{1 : N^v\}$ are the indices for the receiving and sending nodes, respectively. We would like to use this graph to predict another graph $V' = \{v'_i\}_{i=1}^{N^v}$, where each $v'_i \in \mathbb{R}^{L^{v'}}$ is the node corresponding to $v_i$. The number of features in these predicted nodes, $L^{v'}$, need not necessarily be the same as for the input nodes ($L^v$), though this could be the case for dynamical models where one is predicting updated states of particles. For more general regression problems, the number of output features is arbitrary.

Edge model. The prediction is done in two parts. We create the first neural network, the edge model (or “message function”), to compute messages from one node to another: $\phi^e : \mathbb{R}^{L^v} \times \mathbb{R}^{L^v} \rightarrow \mathbb{R}^{L^{e'}}$. Here, $L^{e'}$ is the number of message features. In the bottleneck model, one sets $L^{e'}$ equal to the known dimension of the force, which is 2 or 3 for us. In our models, we set $L^{e'} = 100$ for the standard and $L_1$ models, and 200 for the KL model (which is described separately later on). We create $\phi^e$ as a multi-layer perceptron with ReLU activations and two hidden layers, each with 300 hidden nodes. The mapping is $e'_k = \phi^e(v_{r_k}, v_{s_k})$ for all edges indexed by $k$ (i.e., we concatenate the receiving and sending node features).

Aggregation. These messages are then pooled via element-wise summation for each receiving node $i$ into the summed message, $\bar{e}'_i \in \mathbb{R}^{L^{e'}}$. This can be written as $\bar{e}'_i = \sum_{k \in \{1 : N^e | r_k = i\}} e'_k$.

Node model. We create a second neural network to predict the output nodes, $v'_i$, for each $i$ from the corresponding summed message and input node. This net can be written as $\phi^v : \mathbb{R}^{L^v} \times \mathbb{R}^{L^{e'}} \rightarrow \mathbb{R}^{L^{v'}}$, and has the mapping: $v'_i = \phi^v(v_i, \bar{e}'_i)$, where $v'_i$ is the prediction for $v_i$. We also create $\phi^v$ as a
multi-layer perceptron with ReLU activations and two hidden layers, each with 300 hidden nodes. This model is then trained with the loss function as described later in this section.

Summary. We can write out our forward model for the bottleneck, standard, and L1 models as:

\[ G = (V,E) \]

nodes (e.g., positions of particles) \( V = \{ \mathbf{v}_i \}_{i=1:N_v}; \mathbf{v}_i \in \mathbb{R}^{L_v} \), and edges (indices of connected nodes) \( E = \{ (r_k,s_k) \}_{k=1:N_e}; r_k, s_k \in \{ 1 : N_v \} \).

Compute messages for each edge: \( e'_k = \phi_e(\mathbf{v}_{r_k}, \mathbf{v}_{s_k}) \), \( e'_k \in \mathbb{R}^{L'_e} \), then sum for each receiving node \( i \) : \( \bar{e}'_i = \sum_{k \in \{ 1 : N_e \} | r_k = i \}} e'_k \), \( \bar{e}'_i \in \mathbb{R}^{L'_e} \).

Compute output node prediction: \( \hat{\mathbf{v}}'_i = \phi_v(\mathbf{v}_i, \bar{e}'_i) \), \( \hat{\mathbf{v}}'_i \in \mathbb{R}^{L'_v} \).

Loss. We jointly optimize the parameters in \( \phi_v \) and \( \phi_e \) via mini-batch gradient descent with Adam as the optimizer. Our total loss function for optimizing is:

\[ \mathcal{L} = \mathcal{L}_v + \alpha_1 \mathcal{L}_e + \alpha_2 \mathcal{L}_n, \]

where the prediction loss is \( \mathcal{L}_v = \frac{1}{N_v} \sum_{i \in \{ 1 : N_v \}} | \mathbf{v}'_i - \hat{\mathbf{v}}'_i | \),

the message regularization is \( \mathcal{L}_e = \frac{1}{N_e} \left\{ \sum_{k \in \{ 1 : N_e \}} | e'_k |, \begin{array}{ll} \text{L-1} & \text{Standard} \\ \text{Bottleneck} & \end{array} \right\} \),

with the regularization constant \( \alpha_1 = 10^{-2} \), and the regularization for the network weights is \( \mathcal{L}_n = \sum_{l \in \{ 1 : N_l \}} | w_l |^2 \),

with \( \alpha_2 = 10^{-8} \),

where \( \mathbf{v}'_i \) is the true value for the predicted node \( i \). \( w_l \) is the \( l \)-th network parameter out of \( N_l \) total parameters. This implementation can be visualized during training in the video [https://github.com/MilesCranmer/symbolic_deep_learning](https://github.com/MilesCranmer/symbolic_deep_learning) During training, we also apply a random translation augmentation to all the particle positions to artificially generate more training data.

Next, we describe the KL variant of this model. Note that for the cosmology example in section 4.3, we use the L1 model described above with 500 hidden nodes (found with coarse hyperparameter tuning to optimize accuracy) instead of 300, but other parameters are set the same.

A.2 KL Model

The KL model is a variational version of the GN implementation above, which models the messages as distributions. We choose a normal distribution for each message component with a prior of \( \mu = 0, \sigma = 1 \). More specifically, the output of \( \phi_e \) should now map to twice as many features as it is predicting a mean and variance, hence we set \( L'_e = 200 \). The first half of the outputs of \( \phi_e \) now represent the means, and the second half of the outputs represent the log variance of a particular
message component. In other words,
\[
\begin{align*}
\mu_k' &= \phi_{1:100}(v_{r_k}, v_{s_k}), \\
\sigma^2_k' &= \exp(\phi_{101:200}(v_{r_k}, v_{s_k})), \\
e_k' &\sim \mathcal{N}(\mu_k', \text{diag}(\sigma^2_k')), \\
\hat{e}_j' &= \sum_{k \in \{1:N' \mid r_k = i\}} e_k', \\
\hat{v}_j' &= \phi^v(v_i, \hat{e}_j'),
\end{align*}
\]
where \(\mathcal{N}\) is a multinomial Gaussian distribution. Every time the graph network is run, we calculate the mean and log variance of messages, sample each message once to calculate \(e_k'\), and pass those samples through a sum to compute a sample of \(\hat{e}_j'\) and then pass that value through the edge function to compute a sample of \(\hat{v}_j'\). The loss is calculated normally, except for \(\mathcal{L}_e\), which becomes the KL divergence with respect to our Gaussian prior of \(\mu = 0, \sigma = 1\):
\[
\mathcal{L}_e = \frac{1}{N_e} \sum_{k = \{1:N'\}} \sum_{j = \{1:L'/2\}} \frac{1}{2} \left( \mu^2_{k,j} + \sigma^2_{k,j} - \log(\sigma^2_{k,j}) \right),
\]
with \(\alpha_1 = 1\) (equivalent to \(\beta = 1\) for the loss of a \(\beta\)-Variational Autoencoder; simply the standard VAE). The KL-divergence loss also encourages sparsity in the messages \(e_k'\) similar to the L1 loss.

The difference is that here, an uninformative message component will have \(\mu = 0, \sigma = 1\) (a KL of 0) rather than a small absolute value. We train the networks with a decaying learning schedule as given in the example code.

A.3 Constraining Information in the Messages

The hypothesis which motivated our graph network inductive bias is that if one minimizes the dimension of the vector space used by messages in a GN, the components of message vectors will learn to be linear combinations of the true forces (or equivalent underlying summed function) for the system being learned. The key observation is that \(e_k'\) could learn to correspond to the true force vector imposed on the \(r_k\)-th body due to its interaction with the \(s_k\)-th body.


General Graph Neural Networks. In all of our models here, we assume the dataset does not have edge-specific features, such as a different coupling constants between different particles, but these could be added by concatenating edge features to the receiving and sending node input to \(\phi^v\). We also assume there are no global properties. The graph neural network is described in general form in \([4]\). All of our techniques are applicable to the general form: one would approximate \(\phi^v\) with a symbolic model with included input edge parameters, and also fit the global model, denoted \(\phi^u\).
A.4 Flattened Hamiltonian Graph Network.

As part of this study, we also consider an alternate dynamical model that is described by a linear latent space other than force vectors. In the Hamiltonian formalism of classical mechanics, energies of pairwise interactions and kinetic and potential energies of particles are pooled into a global energy value, $H$, which is a scalar. We label pairwise interaction energy $H_{\text{pair}}$ and the energy of individual particles as $H_{\text{self}}$. Thus, using our previous graph notation, we can write the total energy of a system as:

$$H = \sum_{i=1}^{N} H_{\text{self}}(v_i) + \sum_{k \in \{1:N^e\}} H_{\text{pair}}(v_{r_k}, v_{s_k}).$$ (1)

For particles interacting via gravity, this would be

$$H = \sum_{i} \frac{p_i^2}{2m_i} - \frac{1}{2} \sum_{i \neq j} m_i m_j \left| \mathbf{r}_i - \mathbf{r}_j \right|,$$ (2)

where $p_i$, $m_i$, $\mathbf{r}_i$ indicates the momentum, mass, and position of particle $i$, respectively, and we have set the gravitational constant to 1. Following [47, 46], we could model $H$ as a neural network, and apply Hamilton’s equations to create a dynamical model. More specifically, as in [46], we can predict $H$ as the global property of a GN (this is called a Hamiltonian Graph Network or HGN). However, energy, like forces in Cartesian coordinates, is a summed quantity. In other words, energy is another “linear latent space” that describes the dynamics.

Therefore, we argue that an HGN will be more interpretable if we explicitly sum up energies over the system, rather than compute $H$ as a global property of a GN. Here, we introduce the “Flattened Hamiltonian Graph Network,” or “FlatHGN”, which uses eq. (1) to construct a model that works on a graph. We set up two Multi-Layer Perceptrons (MLPs), one for each node:

$$H_{\text{self}} : \mathbb{R}^{L^v} \rightarrow \mathbb{R},$$ (3)

and one for each edge:

$$H_{\text{pair}} : \mathbb{R}^{L^e} \times \mathbb{R}^{L^v} \rightarrow \mathbb{R}.$$ (4)

Note that the derivatives of $H$ now propagate through the pool, e.g.,

$$\frac{\partial H(V)}{\partial v_i} = \frac{\partial H_{\text{self}}(v_i)}{\partial v_i} + \sum_{r_k = i} \frac{\partial H_{\text{pair}}(e_k, v_{r_k}, v_{s_k})}{\partial v_i}$$

$$+ \sum_{s_k = i} \frac{\partial H_{\text{pair}}(e_k, v_{r_k}, v_{s_k})}{\partial v_i}.$$ (5)

This model is similar to the Lagrangian Graph Network proposed in [49]. Now, should this FlatHGN learn energy functions such that we can successfully model the dynamics of the system with Hamilton’s equations, we would expect that $H_{\text{self}}$ and $H_{\text{pair}}$ should be analytically similar to parts of the true Hamiltonian. Since we have broken the traditional HGN into a FlatHGN, we now have pairwise and self energies, rather than a single global energy, and these are simpler to extract and interpret. This is a similar inductive bias to the GN we introduced previously. To train a FlatHGN, one can follow our strategy above, with the output predictions made using Hamilton’s equations applied to our $H$. One difference is that we also regularize $H_{\text{pair}}$, since it is degenerate with $H_{\text{self}}$ in that it can pick up self energy terms.

B Simulations

Our simulations for sections 4.1 and 4.2 were written using the JAX library (https://github.com/google/jax) so that we could easily vectorize computations over the entire dataset of 10,000 simulations. Example “long exposures” for each simulation in 2D are shown in fig. 4. To create each simulation, we set up the following potentials between two particles, 1 (receiving) and 2 (sending). Here, $r'_{ij}$ is the distance between two particles plus 0.01 to prevent singularities. For particle $i$, $m_i$ is the mass, $q_i$ is the charge, $n$ is the number of particles in the simulation, $\mathbf{r}_i$ is the position of a
Figure 4: Examples of a selection of simulations, for 4 nodes and two dimensions. Decreasing transparency shows increasing time, and size of points shows mass.

particle, and \( \dot{\mathbf{r}}_i \) is the velocity of a particle.

\[
\frac{1}{r^2} : U_{12} = -m_1 m_2/r'_{12} \\
\frac{1}{r} : U_{12} = m_1 m_2 \log(r'_{12}) \\
\text{Spring : } U_{12} = (r'_{12} - 1)^2 \\
\text{Damped : } U_{12} = (r'_{12} - 1)^2 + \mathbf{r}_1 \cdot \dot{\mathbf{r}}_1/n \\
\text{Charge : } U_{12} = q_1 q_2/r'_{12} \\
\text{Discontinuous : } U_{12} = \begin{cases} 
0, & r'_{12} < 2 \\
(r'_{12} - 1)^2, & r'_{12} \geq 2
\end{cases}
\]

All variables lack units. Here, \( m_i \) is sampled from a log-normal distribution with \( \mu = 0, \sigma = 1 \). Each component of \( \mathbf{r}_i \) and \( \dot{\mathbf{r}}_i \) is randomly sampled from a normal distribution with \( \mu = 0, \sigma = 1 \). \( q_i \) is randomly drawn from a set of two elements: \( \{-1, 1\} \), representing charge. The acceleration of a given particle is then

\[
\ddot{\mathbf{r}}_i = -\frac{1}{m_i} \sum_j \nabla_{\mathbf{r}_i} U_{ij}.
\]

This is integrated over 1000 time steps of a fixed step size for a given random initial configuration using an adaptive RK4 integrator. The step size varies for each simulation due to the differences in scale. It is: 0.005 for \( 1/r \), 0.001 for \( 1/r^2 \), 0.01 for Spring, 0.02 for Damped, 0.001 for Charge, and 0.01 for Discontinuous. Each simulation is performed in two and three dimensions, for 4 and 8 bodies. We store these simulations on disk. For training, the simulations for the particular problem being studied are loaded, and each instantaneous snapshot of each simulation is converted to a fully connected graph, with the predicted property (nodes of \( V' \), see appendix A) being the acceleration of the particles at that snapshot.

The test loss of each model trained on each simulation set is given in table 3.

As described in the text (and visualized in the drive video), we can fit linear combinations of the true force components to each of the significant features of a message vector. This fit is summarized by table 4, and the fit itself is visualized in fig. 5 for various models on the 2D spring simulation.
<table>
<thead>
<tr>
<th>Sim.</th>
<th>Standard</th>
<th>Bottleneck</th>
<th>L₁</th>
<th>KL</th>
<th>FlatHGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Charge-2</td>
<td>49</td>
<td>50</td>
<td>52</td>
<td>60</td>
<td>55</td>
</tr>
<tr>
<td>Charge-3</td>
<td>1.2</td>
<td>0.99</td>
<td>0.94</td>
<td>4.2</td>
<td>3.5</td>
</tr>
<tr>
<td>Damped-2</td>
<td>0.30</td>
<td>0.33</td>
<td>0.30</td>
<td>1.5</td>
<td>0.35</td>
</tr>
<tr>
<td>Damped-3</td>
<td>0.41</td>
<td>0.45</td>
<td>0.40</td>
<td>3.3</td>
<td>0.47</td>
</tr>
<tr>
<td>Disc.-2</td>
<td>0.064</td>
<td>0.074</td>
<td>0.044</td>
<td>1.8</td>
<td>0.075</td>
</tr>
<tr>
<td>Disc.-3</td>
<td>0.20</td>
<td>0.18</td>
<td>0.13</td>
<td>4.2</td>
<td>0.14</td>
</tr>
<tr>
<td>r⁻¹-2</td>
<td>0.077</td>
<td>0.069</td>
<td>0.079</td>
<td>3.5</td>
<td>0.05</td>
</tr>
<tr>
<td>r⁻¹-3</td>
<td>0.051</td>
<td>0.050</td>
<td>0.055</td>
<td>3.5</td>
<td>0.017</td>
</tr>
<tr>
<td>r⁻²-2</td>
<td>1.6</td>
<td>1.6</td>
<td>1.2</td>
<td>9.3</td>
<td>1.3</td>
</tr>
<tr>
<td>r⁻²-3</td>
<td>4.0</td>
<td>3.6</td>
<td>3.4</td>
<td>9.8</td>
<td>2.5</td>
</tr>
<tr>
<td>Spring-2</td>
<td>0.047</td>
<td>0.046</td>
<td>0.045</td>
<td>1.7</td>
<td>0.016</td>
</tr>
<tr>
<td>Spring-3</td>
<td>0.11</td>
<td>0.11</td>
<td>0.090</td>
<td>3.8</td>
<td>0.010</td>
</tr>
</tbody>
</table>

Table 3: Test prediction losses for each model on each dataset in two and three dimensions. The training was done with the same batch size, schedule, and number of epochs.

Figure 5: The most significant message components of each model compared with a linear combination of the force components: this example, the spring simulation in 2D with eight nodes for training. These plots demonstrate that the GN’s messages have learned to be linear transformations of the vector components of the true force, in this case a springlike force, after applying an inductive bias to the messages.

C Symbolic Regression Details

After training a model on each simulation, we convert a deep learning model to a symbolic expression by approximating subcomponents of the model with symbolic regression, over observed inputs and outputs. For our aforementioned GNN implementation, we can record the outputs of φₑ and φᵥ for various data points in the training set.

For models other than the bottleneck and Hamiltonian model (where we explicitly limit the features) we calculate the most significant output features of φₑ (we also refer to the output features as “message components”). For the L₁ and standard model, this is done by sorting the message components with the largest standard deviation; the most significant feature is the one with the largest standard deviation, which are the features we study. For the KL model, we consider the feature with the largest KL-divergence: µ² + σ² − log(σ²). These features are the ones we consider to be containing information used by the GN, so are the ones we fit symbolic expressions to.

As an example, here we fit the most significant feature, which we refer to as φₑ¹, over random examples of the training dataset. We do this for the particle simulations in section 4.1. The inputs to the actual φₑ¹ neural network are: m₁, m₂, q₁, q₂, x₁, x₂, ... (mass, charge, and Cartesian positions of receiving...
and sending node), leaving us with many examples of \((m_1, m_2, q_1, q_2, x_1, x_2, \ldots, \phi_1^c)\). We would like to fit a symbolic expression to map \((m_1, m_2, q_1, q_2, x_1, x_2, \ldots) \rightarrow \phi_1^c\). To simplify things for this symbolic model, we convert the input position variables to a more interpretable format: \(\Delta x = x_2 - x_1\) for \(x\) displacement, likewise for \(y\) (and \(z\), if it is a 3D simulation), and \(r = \sqrt{\Delta x^2 + \Delta y^2 + (\Delta z^2)}\) for distance.

We then pass these \((m_1, m_2, q_1, q_2, \Delta x, \Delta y, (\Delta z, r), \phi_1^c)\) examples (we take 5000 examples for each of our tests) to eureqa, and ask it to fit \(\phi_1^c\) as a function of the others by minimizing the mean absolute error (MAE). We allow it to use the operators +, -, \(\times\), /, \(\triangleright\), \(\triangleleft\), \(^\wedge\), \(\exp\), \(\log\), IF(\(\cdot\), \(\cdot\)) as well as real constants in its solutions. We score complexity by counting the number of occurrences of each operator, constant, and input variable. We weight \(^\wedge\), \(\exp\), \(\log\), IF(\(\cdot\), \(\cdot\)) as three times the other operators, since these are more complex operations. eureqa outputs the best equation at each complexity level, denoted by \(c\). Example outputs are shown in table 5 for the \(1/r\) and \(1/r^2\) simulations. We select a formula from this list by taking the one that maximizes the fractional drop in mean absolute error (MAE) over an increase in complexity from the next best model. This is analogous to Occam’s Razor: we jointly optimize for simplicity and accuracy of the model. The objective itself can be written as maximizing \((-\Delta \log(\text{MAE}_c)/\Delta c)\) over the best model at each maximum complexity level, and is schematically illustrated in fig. 3. We find experimentally that this score produces the best-recovered solutions in a variety of tests on different generating equations.

Following the process of fitting analytic equations to the messages, we fit a single analytic expression to model \(\phi_1^c\) as a function of the simplified input variables. We recover many analytical expressions that were used to generate the data, examples of which are listed below (\(a, b\) indicate learned constants):

- Spring, 2D, \(L_1\) (expect \(\phi_1^c \approx (a \cdot (\Delta x, \Delta y))(r - 1) + b\)).
  \[
  \phi_1^c \approx 1.36 \Delta y + 0.60 \Delta x - \frac{0.60 \Delta x + 1.37 \Delta y}{r} - 0.0025
  \]

- \(1/r^2\), 3D, Bottleneck (expect \(\phi_1^c \approx a((\Delta x, \Delta y, \Delta z))/r^2 + b\)).
  \[
  \phi_1^c \approx \frac{0.021 \Delta x m_2 - 0.077 \Delta y m_2}{r^3}
  \]

- Discontinuous, 2D, \(L_1\) (expect \(\phi_1^c \approx \text{IF}(r > 2, (a \cdot (\Delta x, \Delta y, \Delta z))r, 0) + b\)).
  \[
  \phi_1^c \approx \text{IF}(r > 2, 0.15r \Delta y + 0.19r \Delta x, 0) - 0.038
  \]

**Examples of failed reconstructions.** Note that reconstruction does not always succeed, especially for training strategies other than \(L_1\) or bottleneck models that cannot successfully find compact representations of the right dimensionality. We demonstrate some failed examples below:

- Spring, 3D, KL (expect \(\phi_1^c \approx (a \cdot (\Delta x, \Delta y, \Delta z))(r - 1) + b\)).
  \[
  \phi_1^c \approx 0.57 \Delta y + 0.32 \Delta z
  \]

- \(1/r\), 3D, Standard (expect \(\phi_1^c \approx a((\Delta x, \Delta y, \Delta z))/r + b\)).
  \[
  \phi_1^c \approx \frac{0.041 + m_2 \text{IF}(\Delta z > 0, 0.021, 0.067)}{r}
  \]

We do not attempt to make any general statements about when symbolic regression applied to the message components will fail or succeed in extracting the true law. Simply, we show that it is possible, for a variety of physical systems, and argue that reconstruction is more likely by the inclusion of a strong inductive bias in the network.

A full table of successes and failures in reconstructing the force law over the different n-body experiments is given in table [5]. While the equations given throughout the paper were generated with eureqa, to create this table in particular, we switched from eureqa to PySR. This is because PySR allows us to configure a controlled experiment with fixed hyperparameters and total mutation steps for each force law, whereas Eureqa makes these controls inaccessible. However, given enough training time, we found eureqa and PySR produced equivalent results for equations at this simplicity level.
where the indices refer to particle number. Despite Eureqa with a gray background has the largest fractional drop in mean absolute error in their tables, which

\begin{table}
\centering
\begin{tabular}{|l|c|c|}
\hline
Solutions extracted for the 2D $1/r^2$ Simulation & MAE & Complexity \\
\hline
$\phi_1^e = 0.162 + (5.62 + 20.3m_2 \Delta x - 153m_2 \Delta y)/r^3$ & 17.954713 & 22 \\
$\phi_2^e = (6.07 + 19.9m_2 \Delta x - 154m_2 \Delta y)/r^3$ & 18.400224 & 20 \\
$\phi_3^e = (3.61 + 20.9 \Delta x - 154m_2 \Delta y)/r^3$ & 42.323236 & 18 \\
$\phi_4^e = (31.6 \Delta x - 152m_2 \Delta y)/r^3$ & 69.447467 & 16 \\
$\phi_5^e = (2.78 - 152m_2 \Delta y)/r^3$ & 131.42547 & 14 \\
\hline
Solutions extracted for the 2D $1/r$ Simulation & MAE & Complexity \\
\hline
$\phi_1^e = (4.53m_2 \Delta y - 1.53 \Delta x - 15.0m_2 \Delta x)/r^2 - 0.209$ & 0.37839388 & 22 \\
$\phi_2^e = (4.58m_2 \Delta y - \Delta x - 15.2m_2 \Delta x)/r^2 - 0.227$ & 0.38 & 20 \\
$\phi_3^e = (4.55m_2 \Delta y - 15.5m_2 \Delta x)/r^2 - 0.238$ & 0.42 & 18 \\
$\phi_4^e = (4.59m_2 \Delta y - 15.5m_2 \Delta x)/r^2$ & 0.46575519 & 16 \\
$\phi_5^e = (10.7 \Delta y - 15.5m_2 \Delta x)/r^2$ & 2.48 & 14 \\
$\phi_6^e = (\Delta y - 15.6m_2 \Delta x)/r^2$ & 6.96 & 12 \\
$\phi_7^e = -15.6m_2 \Delta x/r^2$ & 7.93 & 10 \\
$\phi_8^e = -34.82x/r^2$ & 31.17 & 8 \\
$\phi_9^e = -8.71 \Delta x/r$ & 68.345174 & 6 \\
$\phi_{10}^e = -0.36m_2 \Delta x$ & 85.743106 & 5 \\
$\phi_{11}^e = -0.632 \Delta x$ & 93.052677 & 3 \\
$\phi_{12}^e = -\Delta x$ & 96.708906 & 2 \\
$\phi_{13}^e = -0.303$ & 103.29053 & 1 \\
\hline
\end{tabular}
\caption{Results of using symbolic regression to fit equations to the most significant (see text) feature of $\phi_r$, denoted $\phi_r^e$, for the $1/r^2$ (top) and $1/r$ (bottom) force laws, extracted from the bottleneck model. We expect to see $\phi_r^e \approx \frac{a (\Delta x, \Delta y, \Delta z)}{r^n} + b$, for arbitrary $a$ and $b$, and $\alpha = 2$ for the $1/r$ simulation and $\alpha = 3$ for the $1/r^2$ simulation, which is approximately what we recover. The row with a gray background has the largest fractional drop in mean absolute error in their tables, which according to our parametrization of Occam’s razor, represents the best model. This demonstrates a technique for learning an unknown “force law” with a constrained graph neural network.}
\end{table}

**Pure eureqa experiment** To demonstrate that eureqa by itself is not capable of finding many of the equations considered from the raw high-dimensional dataset, we ran it on the simulation data without our GN’s factorization of the problem, giving it the features of every particle. As expected, even after convergence, it cannot find meaningful equations; all of the solutions it provides for the $n$-body system are very poor fits. One such example of an equation, for the acceleration of particle 2 along the $x$ direction in a 6-body system under a $1/r^2$ force law, is:

$$\ddot{x}_2 = \frac{0.426}{367y_4 - 1470} + \frac{2.88 \times 10^5 x_1}{2.08 \times 10^3 y_4 + 446g_4^2} - 5.98 \times 10^{-5} x_6 - 109 x_1,$$

where the indices refer to particle number. Despite eureqa converging, this equation is evidently meaningless and achieves a poor fit to the data. Thus, we argue that raw symbolic regression is intractable for the problems we consider, and only after factorization with a neural network do these problems become feasible for symbolic regression.

**Discovering potentials using FlatHGN.** Lastly, we also show an example of a successful reconstruction of a pairwise Hamiltonian from data. We treat the $H_{\text{pair}}$ just as we would $\phi_r^e$, and fit it to data. The one difference here is that there are potential $H_{\text{pair}}$ values offset by a constant function of the non-dynamical parameters (fixed properties like mass) which still produce the correct dynamics, since only the derivatives of $H_{\text{pair}}$ are used. Thus, we cannot simply fit a linear transformation of the true $H_{\text{pair}}$ to data to verify it has learned our generating equation: we must rely on symbolic regression.
Figure 6: A plot of the data for the $1/r$ simulation in Table 4 indicating mean absolute error versus complexity in the top plot and fractional drop in mean absolute error over the next-best model in the bottom plot. As indicated, we take the largest drop in log-loss over a single increase in complexity as the chosen model—it is our parametrization of Occam’s Razor.

Table 5: Success/failure of a reconstruction of the force law by symbolic regression, corresponding to the values in Table 1.

<table>
<thead>
<tr>
<th>Sim.</th>
<th>Standard</th>
<th>Bottleneck</th>
<th>L₁</th>
<th>KL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Charge-2</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Charge-3</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>$r^{-1.2}$</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$r^{-1.3}$</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$r^{-2.2}$</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>$r^{-2.3}$</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Spring-2</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Spring-3</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 5: Success/failure of a reconstruction of the force law by symbolic regression, corresponding to the values in Table 1.

The neural network has learned the correct form of the Hamiltonian.

$$H_{\text{pair}} \approx \frac{0.0019q_1q_2}{r} - 0.0112 - 0.00143q_1 - 0.00112q_1q_2,$$

where we expect $H_{\text{pair}} \approx a \frac{q_1q_2}{r} + f(q_1, q_2, m_1, m_2)$, for constant $a$ and arbitrary function $f$, which shows that the neural network has learned the correct form of the Hamiltonian.
**Hyperparameters.** Since the hyperparameters used internally by *eureqa* are opaque and not tunable, here we discuss the parameters used in *PySR* [27], which are common among many symbolic regression tools. At a given step of the training, there is a set of active equations in the “population”. The number of active equations is a tunable hyperparameter, and is related to the diversity of the discovered equations, as well as the number of compute cores being used. The max size of equations controls the maximum complexity considered, and can be controlled to prevent the algorithm from wasting cycles on over-complicated equations. The operators used in the equations depends on the specific problem considered, and is another hyperparameter specified by the user. Next, there is a set of tunable probabilities associated with each mutation: how frequently to mutate an operator into a different operator, add an operator with arguments, replace an operator and its arguments with a constant, and so on. In some approaches such as with *PySR*, the best equations found over the course of training are randomly reintroduced back into the population. The frequency at which this occurs is controlled by another hyperparameter.

**D Video Demonstration and Code**

We include a video demonstration of the central ideas of our paper at https://github.com/MilesCranmer/symbolic_deep_learning. It shows the message components of a graph network converging to be equal to a linear combination of the force components when L1 regularization is applied. Time in each clip of the video is correlated with training epoch. In this video, the top left corner of the fully revealed plot corresponds to a single test simulation that is 300 time steps long. Four particles of different masses are initiated with random positions and velocities, and evolved according to the potential of a spring with an equilibrium position of 1: \((r - 1)^2\), where \(r\) is the distance between two particles. The evaluation trajectories are shown on the right, with the gray particles indicating the true locations. The 15 largest message components in terms of standard deviation over a test set are represented in a sorted list below the graph network in gray, where darker color corresponds to a larger standard deviation. Since we apply L1 regularization to the messages, we expect this list to grow sparser over time, which it does. Of these messages, the two largest components are extracted, and each is fit to a separate linear combination of the true force components (bottom left). A better fit to the true force components — indicating that the messages represent the force — are indicated by dots (each dot is a single message) that lie closer along the \(y = x\) line in the bottom middle two scatter plots.

As can be seen in the video, as the messages grow increasingly sparse, the messages eventually converge to be almost exactly linear combinations of the true forces. Finally, once the loss is converged, we also fit symbolic regression to the largest message component. The video was created using the same training procedure as used in the rest of the paper. The dataset that the L1 model was trained on is the 4-node Spring-2. Finally, we include the full code required to generate the animated clips in the above figure. This code contains all of the models and simulators used in the paper, along with the default training parameters. This code can also be accessed in the drive.

**E Cosmological Experiments**

For the cosmological data graph network, we do a coarse hyperparameter tuning based on predictions of \(\delta_i\) and select a GN with 500 hidden units, two hidden layers per node function and message function. We choose 100 message dimensions as before. We keep other hyperparameters the same as before: L1 regularization with a regularization scale of \(10^{-2}\).

Remarkably, the vector space discovered by this graph network is 1 dimensional. This is indicated by the fact that only one message component has standard deviation of about \(10^{-2}\) and all other 99 components have a standard deviation of under \(10^{-8}\). This suggests that the \(\delta_i\) prediction is a sum over some function of the center halo and each neighboring halo. Thus, we can rewrite our model as a sum over a function \(\phi_e\) which takes the central halo and each neighboring halo, and passes it to \(\phi^v\) which predicts \(\delta_i\) given the central halo properties.

**Best-fit parameters.** We list best-fit parameters for the discovered models in the paper in table 6. The functional forms were extracted from the GN by approximating both \(\phi_e\) and \(\phi^v\) over training data with a symbolic regression and then analytically composing the expressions. Although the symbolic regression fits constants itself, this accumulates error from the two levels of approximation (graph net
|
|---|---|---|---|
|Old| Test| Formula| Summed Component| \( \langle |\delta_i - \hat{\delta}_i| \rangle \) |
|Constant| \( \delta_i = C_1 \)| N/A| 0.421 |
|Simple| \( \delta_i = C_1 + (C_2 + M_i C_3) e_i \)| \( e_i = \sum_{j\neq i} |r_i - r_j| < 20 M_j \)| 0.121 |
|New| Best, without mass| \( \delta_i = C_1 + \frac{e_i}{C_2 + C_3 e_i |v_i|} \)| \( e_i = \sum_{j\neq i} \frac{C_4 + |v_i - v_j|}{C_5 + (C_6 |r_i - r_j|) e_i} \)| 0.120 |
|Best, with mass| \( \delta_i = C_1 + \frac{e_i}{C_2 + C_3 M_i} \)| \( e_i = \sum_{j\neq i} \frac{C_4 + M_j}{C_5 + (C_6 |r_i - r_j|) e_i} \)| 0.0882 |

Table 6: Best-fit parameters for the functional forms used to estimate the overdensity of dark matter halos. The functional forms are given in the upper table for reference. Here we use the same formula as “Best, with mass,” since we found an equivalent formula by only looking at the 80% chunk of the data. The constants in that functional form are also fit by only training on that fraction of the data.

Thus, we take out the functional forms as given in Table 6 and refit the parameters directly to the training data. This results in the parameters given, which are used to calculate accuracy of the symbolic models.