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.

Anonymized example code can be found at `https://github.com/MilesCranmer/symbolic_deep_learning`

Preprint. Under review.

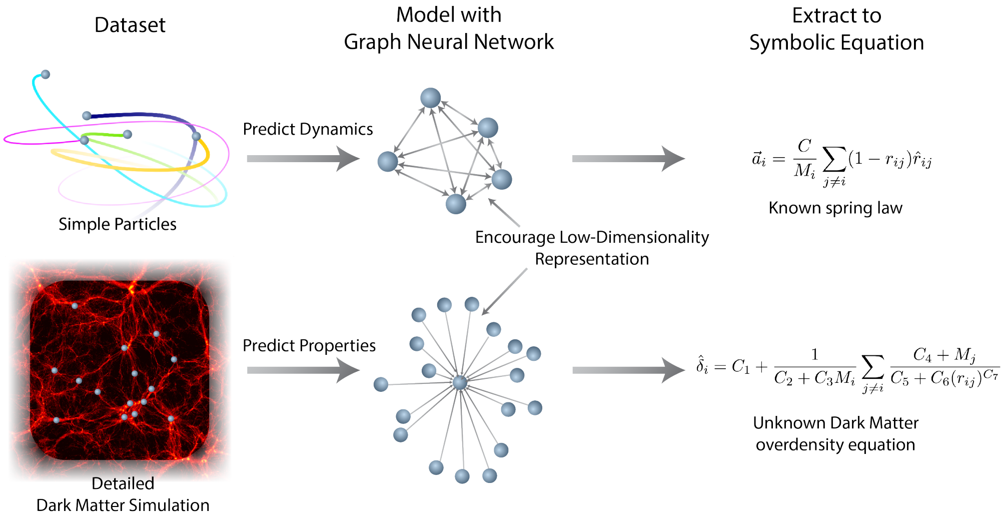

Figure 1: A cartoon depicting how we extract physical equations from a dataset.

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 [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) [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 [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, but we also derive a new interpretable closed-form analytical expression that can be useful in astrophysics.

2 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.

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) [6] in their generalized form, the *Graph Network*, a type of Graph Neural Network [7, 8, 9], to learn models of particle systems in many physical domains [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].

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 ϕ^e, maps from a pair of nodes ($\mathbf{v}_i, \mathbf{v}_j \in \mathbb{R}^{L_v}$) 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, ϕ^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 ϕ^u aggregates all messages and all updated nodes and computes a global property. ϕ^e, ϕ^v, ϕ^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.

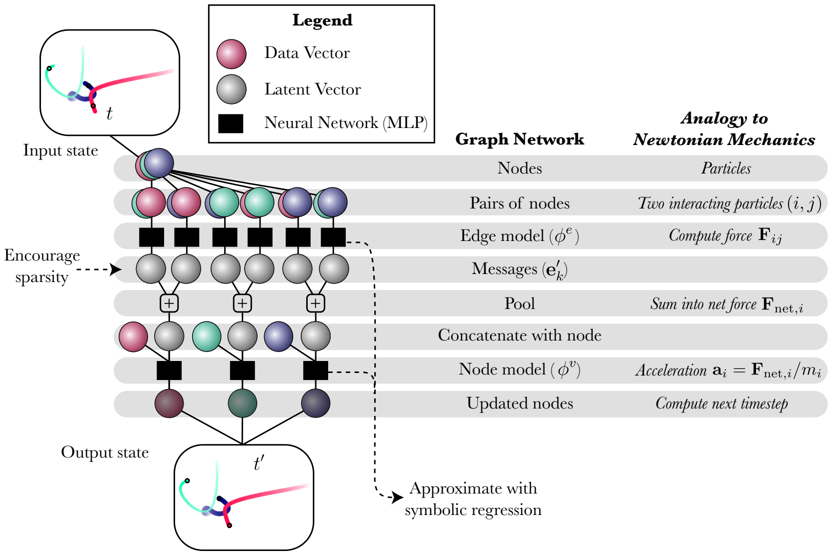
Legend
Data Vector
Latent Vector
Neural Network (MLP)
t
Input state
Graph Network
Analogy to Newtonian Mechanics
Nodes
Particles
Pairs of nodes
Two interacting particles (i, j)
Edge model (ϕ^e)
Compute force $\mathbf{F}_{ij}$
Encourage sparsity
Messages ($\mathbf{e}'_k$)
Pool
Sum into net force $\mathbf{F}_{\mathrm{net},i}$
Concatenate with node
Node model (ϕ^v)
Acceleration $\mathbf{a}_i = \mathbf{F}_{\mathrm{net},i}/m_i$
Updated nodes
Compute next timestep
Output state
t'
Approximate with symbolic regression

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, ϕ^e and ϕ^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 ϕ^e, ϕ^v, ϕ^u, which are our targets for the symbolic regression. GNs can also be embedded with additional symmetries as in [23, 24], 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 ϕ^e, ϕ^v, and ϕ^u independently. *eureqa* works by using a genetic algorithm to combine algebraic expressions stochastically. The technique is analogous 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, IF$(\cdot, \cdot, \cdot)$ 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. There are many other alternative approaches to *eureqa*, including [25, 26, 27, 28, 29, 30]. An alternative way of interpreting GNNs is given in [31].

A key advantage of fitting a symbolic model on internal GN functions is that the symbolic regression never needs to consider more than two particles at once. This makes the symbolic regression problem tractable.

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 L_1 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 [32] and PyTorch Geometric[33]. We train them with a decaying learning schedule using Adam [34]. The symbolic regression technique is described in section 4.1. 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 $-\frac{Gm_1m_2}{r^2}\hat{r}$) are defined on pairs of particles, analogous to the message function ϕ^e of our Graph Networks. The summation that aggregates messages is analogous to the calculation of the net force on a receiving particle. Finally, the node function, ϕ^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 defines a single scalar function for an entire system, corresponding to the total energy of the system, $\mathcal{H}$, that determines the derivatives of the canonical position ($\mathbf{q}$) and momentum ($\mathbf{p}$). In short, Hamilton's equations can be described as following: $\dot{\mathbf{q}} = \frac{\partial \mathcal{H}}{\partial \mathbf{p}}$ and $\dot{\mathbf{p}} = -\frac{\partial \mathcal{H}}{\partial \mathbf{q}}$.

For this model we will be using a Hamiltonian Neural Network inductive bias [35, 36], combining it with the GN inductive bias [37], which is called a Hamiltonian Graph Network (HGN). Specifically, the global model ϕ^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 $\mathcal{H}$ with respect to the position and momentum, $\mathbf{q}$ and $\mathbf{p}$, respectively, of the input nodes, we will be able to calculate the updates to the momentum and position via Hamilton equations.

Now, we impose a slightly different inductive bias on the global model ϕ^u than in [37], and name this the "Flattened HGN" or FlatHGN: instead of learning ϕ^u we fix it to be the sum of a pairwise interaction term, $\mathcal{H}_{\text{pair}}$, corresponding to the aggregated messages for the full graph, and a per-particle term, $\mathcal{H}_{\text{self}}$, corresponding the aggregated updated nodes. This is a Hamiltonian version of the Lagrangian Graph Network in [38], and is similar to [39]. 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 scalar 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 [40]. 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

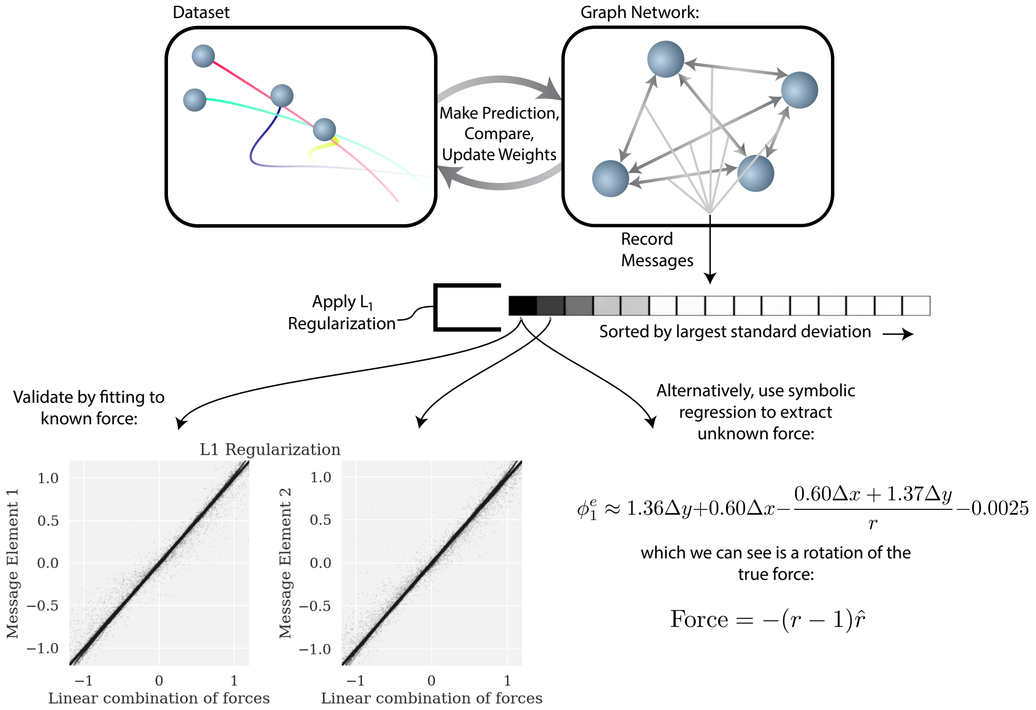
Dataset
Graph Network:
Make Prediction,
Compare,
Update Weights
Record
Messages
Apply L$_1$
Regularization
Sorted by largest standard deviation
Validate by fitting to
known force:
Alternatively, use symbolic
regression to extract
unknown force:
L1 Regularization
Message Element 1
Message Element 2
1.0
0.5
0.0
−0.5
−1.0
−1
0
1
Linear combination of forces
$\phi_1^e \approx 1.36\Delta y + 0.60\Delta x - \frac{0.60\Delta x + 1.37\Delta y}{r} - 0.0025$
which we can see is a rotation of the
true force:
$\text{Force} = -(r-1)\hat{r}$

Figure 3: A diagram showing how we implement and exploit our inductive bias on GNs. A video of this figure during training can be seen by going to the URL `https://github.com/MilesCranmer/symbolic_deep_learning/blob/master/video_link.txt`.

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 ϕ^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: $-m_1 m_2 \hat{r}/r$; (b) $1/r^2$ orbital force $-m_1 m_2 \hat{r}/r^2$; (c) charged particles force $q_1 q_2 \hat{r}/r^2$; (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. 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 dimensionalty, performs worse than the model using L_1 regularization. We speculate this is because the low-dimensionality of the messages at the start of training makes it harder to solve the optimization process via gradient descent in a fixed training time.

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.

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 ϕ^e, which we refer to as ϕ^e_1, over random samples of the training dataset. The inputs to the regression are $m_1, m_2, q_1, q_2, x_1, x_2, \ldots$ (mass, charge, x-position of receiving and sending node) as well as simplified variables to help the symbolic regression: e.g., Δx for x displacement, and r for distance.

We then use *eureqa* to fit the ϕ^e_1 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

Sim.	Standard	Bottleneck	L_1	KL
Charge-2	4.0×10^{-3}	$\mathbf{2.6 \times 10^{-5}}$	1.1×10^{-4}	3.9×10^{-4}
Charge-3	3.5×10^{-3}	1.1×10^{-4}	1.3×10^{-4}	$\mathbf{1.1 \times 10^{-5}}$
r^{-1}-2	4.0×10^{-4}	$\mathbf{5.4 \times 10^{-6}}$	7.3×10^{-6}	3.2×10^{-2}
r^{-1}-3	4.9×10^{-4}	1.0×10^{-5}	$\mathbf{8.9 \times 10^{-6}}$	2.0×10^{-2}
r^{-2}-2	7.1×10^{-4}	$\mathbf{3.4 \times 10^{-7}}$	7.3×10^{-5}	8.4×10^{-5}
r^{-2}-3	2.4×10^{-4}	$\mathbf{1.1 \times 10^{-6}}$	2.0×10^{-5}	3.0×10^{-4}
Spring-2	2.9×10^{-2}	3.4×10^{-5}	$\mathbf{3.2 \times 10^{-5}}$	4.3×10^{-2}
Spring-3	3.6×10^{-2}	2.1×10^{-3}	$\mathbf{6.5 \times 10^{-5}}$	9.1×10^{-2}

Table 1: The normalized mean square error of a fit of a linear combination of true force components to the message components for a given model (see text). An example of this comparison is shown in fig. 3. A small number indicates that the messages are strongly correlated with the true force vectors.

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.

From this, we recover many analytical expressions that are equivalent to the simulated force laws (a, b indicate learned constants):

- Spring, 2D, L_1 (expect $\phi_1^e \approx (\mathbf{a} \cdot (\Delta x, \Delta y))(r-1) + b$).

$$\phi_1^e \approx 1.36\Delta y + 0.60\Delta x - \frac{0.60\Delta x + 1.37\Delta y}{r} - 0.0025$$

- $1/r^2$, 3D, Bottleneck (expect $\phi_1^e \approx \frac{\mathbf{a}\cdot(\Delta x, \Delta y, \Delta z)}{r^3} + b$).

$$\phi_1^e \approx \frac{0.021\Delta x m_2 - 0.077\Delta y m_2}{r^3}$$

- Discontinuous, 2D, L_1 (expect $\phi_1^e \approx \mathrm{IF}(r > 2, (\mathbf{a} \cdot (\Delta x, \Delta y, \Delta z))r, 0) + b$).

$$\phi_1^e \approx \mathrm{IF}(r > 2, 0.15 r \Delta y + 0.19 r \Delta x, 0) - 0.038$$

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).

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 ($\mathcal{H}_{\text{pair}} \approx \frac{a q_1 q_2}{r}$), symbolic regression applied to the learned message function yields[1]:

$$\mathcal{H}_{\text{pair}} \approx \frac{0.0019 q_1 q_2}{r}$$

It is also possible to fit the per-particle term $\mathcal{H}_{\text{self}}$, 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.

[1] We 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.

	Test	Formula	Summed Component	$\left\langle \left\lvert \delta_i - \hat{\delta}_i \right\rvert \right\rangle$
Old	Constant	$\hat{\delta}_i = C_1$	N/A	0.421
	Simple	$\hat{\delta}_i = C_1 + (C_2 + M_i C_3) e_i$	$e_i = \sum_{j \neq i}^{\lvert \mathbf{r}_i - \mathbf{r}_j \rvert < 20} M_j$	0.121
New	Best, without mass	$\hat{\delta}_i = C_1 + \frac{e_i}{C_2 + C_3 e_i \lvert \mathbf{v}_i \rvert}$	$e_i = \sum_{j \neq i} \frac{C_4 + \lvert \mathbf{v}_i - \mathbf{v}_j \rvert}{C_5 + (C_6 \lvert \mathbf{r_i} - \mathbf{r_j} \rvert)^{C_7}}$	0.120
	Best, with mass	$\hat{\delta}_i = C_1 + \frac{e_i}{C_2 + C_3 M_i}$	$e_i = \sum_{j \neq i} \frac{C_4 + M_j}{C_5 + (C_6 \lvert \mathbf{r}_i - \mathbf{r}_i \rvert)^{C_7}}$	0.0882

Table 2: A comparison of both known and discovered formulas for dark matter overdensity. C_i indicates fitted parameters, which are given in the appendix.

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 [41]. 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 [42, 43], 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 δ_i for halo i might correlate δ 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): $\sum_{j \neq i}^{\lvert \mathbf{r}_i - \mathbf{r}_i \rvert < 20} M_j$. The intuition behind this scaling is described in [44]. 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 [40]. 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 $\mathbf{r}_i$, and velocity $\mathbf{v}_i$. We also compute the smoothed overdensity δ_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 δ_i for each halo with a GN. Training details are the same as for the Newtonian simulations, but we switch 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.

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 [45].

Here we study this on the cosmology example by masking 20% of the data: halos which have $\delta_i > 1$. 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: $\hat{\delta}_i = C_1 + e_i/(C_2 + M_i); e_i = \sum_{i \neq j} \frac{C_3 + M_j}{C_4 + C_5 |\mathbf{r}_i - \mathbf{r}_i|^{C_6}}$. We fit these C_i constants to the same masked portion of data on which the graph network was trained. The graph network itself obtains an average error $\left\langle \left| \delta_i - \hat{\delta}_i \right| \right\rangle$ of 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 or energy from these models: symbolic regression is capable of fitting explicit equations to our trained model's message function. We introduced the Flattened Hamiltonian Graph Network, which allowed us to learn symbolic forms of pairwise interaction energies. 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.

Acknowledgment: Miles 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.

Our code made use of the following Python packages: `numpy`, `scipy`, `sklearn`, `jupyter`, `matplotlib`, `pandas`, `torch`, `tensorflow`, `jax`, and `torch_geometric` [46, 47, 48, 49, 50, 51, 32, 52, 53, 33].

References

[1] Eugene P. Wigner. The unreasonable effectiveness of mathematics in the natural sciences. Richard courant lecture in mathematical sciences delivered at New York University, May 11, 1959. *Communications on Pure and Applied Mathematics*, 13(1):1–14, 1960. doi: 10.1002/cpa.3160130102. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/cpa.3160130102`.

[2] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *science*, 324(5923):81–85, 2009.

[3] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, pages 2924–2932, 2014.

[4] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[5] Miles D Cranmer, Rui Xu, Peter Battaglia, and Shirley Ho. Learning symbolic physics with graph networks. *arXiv preprint arXiv:1909.05862*, 2019.

[6] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pages 4502–4510, 2016.

[7] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[8] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34 (4):18–42, 2017.

[9] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

[10] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. *arXiv preprint arXiv:1612.00341*, 2016.

[11] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. *arXiv preprint arXiv:1806.01242*, 2018.

[12] Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li F Fei-Fei, Josh Tenenbaum, and Daniel L Yamins. Flexible neural representation for physics prediction. In *Advances in Neural Information Processing Systems*, pages 8799–8810, 2018.

[13] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. *arXiv preprint arXiv:1810.01566*, 2018.

[14] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. *arXiv preprint arXiv:1802.04687*, 2018.

[15] Victor Bapst, Thomas Keck, A Grabska-Barwińska, Craig Donner, Ekin Dogus Cubuk, SS Schoenholz, Annette Obika, AWR Nelson, Trevor Back, Demis Hassabis, et al. Unveiling the predictive power of static structure in glassy systems. *Nature Physics*, 16(4):448–454, 2020.

[16] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W Battaglia. Learning to simulate complex physics with graph networks. *arXiv preprint arXiv:2002.09405*, 2020.

[17] Norman H Packard, James P Crutchfield, J Doyne Farmer, and Robert S Shaw. Geometry from a time series. *Physical review letters*, 45(9):712, 1980.

[18] Bryan C. Daniels and Ilya Nemenman. Automated adaptive inference of phenomenological dynamical models. *Nature Communications*, 6(1):1–8, August 2015. ISSN 2041-1723.

[19] Miguel Jaques, Michael Burke, and Timothy Hospedales. Physics-as-Inverse-Graphics: Joint Unsupervised Learning of Objects and Physics from Video. *arXiv:1905.11169 [cs]*, May 2019.

[20] Michael Janner, Sergey Levine, William T. Freeman, Joshua B. Tenenbaum, Chelsea Finn, and Jiajun Wu. Reasoning about physical interactions with object-centric models. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=HJx9EhC9tQ`.

[21] Anton Bakhtin, Laurens van der Maaten, Justin Johnson, Laura Gustafson, and Ross Girshick. Phyre: A new benchmark for physical reasoning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 5082–5093. Curran Associates, Inc., 2019. URL `http://papers.nips.cc/paper/8752-phyre-a-new-benchmark-for-physical-reasoning.pdf`.

[22] Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S. Du, Ken ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rJxbJeHFPS`.

[23] Erik J Bekkers. B-spline cnns on lie groups. 2019.

[24] Marc Finzi, Samuel Stanton, Pavel Izmailov, and Andrew Gordon Wilson. Generalizing Convolutional Neural Networks for Equivariance to Lie Groups on Arbitrary Continuous Data. 2020.

[25] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. Deap: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13(70):2171–2175, 2012.

[26] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.

[27] Samuel H Rudy, Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Data-driven discovery of partial differential equations. *Science Advances*, 3(4):e1602614, 2017.

[28] Steven Atkinson, Waad Subber, Liping Wang, Genghis Khan, Philippe Hawi, and Roger Ghanem. Data-driven discovery of free-form governing differential equations. *arXiv preprint arXiv:1910.05117*, 2019.

[29] Silviu-Marian Udrescu and Max Tegmark. Ai feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, 2020.

[30] Gert-Jan Both, Subham Choudhury, Pierre Sens, and Remy Kusters. Deepmod: Deep learning for model discovery in noisy data, 2019.

[31] Rex Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnn explainer: A tool for post-hoc explanation of graph neural networks. *arXiv preprint arXiv:1903.03894*, 2019.

[32] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[33] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[34] Diederik P Kingma and J Adam Ba. A method for stochastic optimization. arxiv 2014. *arXiv preprint arXiv:1412.6980*, 2014.

[35] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. In *Advances in Neural Information Processing Systems*, pages 15353–15363, 2019.

[36] Peter Toth, Danilo Jimenez Rezende, Andrew Jaegle, Sébastien Racanière, Aleksandar Botev, and Irina Higgins. Hamiltonian generative networks. *arXiv preprint arXiv:1909.13789*, 2019.

[37] Alvaro Sanchez-Gonzalez, Victor Bapst, Kyle Cranmer, and Peter Battaglia. Hamiltonian graph networks with ode integrators. *arXiv preprint arXiv:1909.12790*, 2019.

[38] Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

[39] Jiang Wang, Simon Olsson, Christoph Wehmeyer, Adrià Pérez, Nicholas E Charron, Gianni De Fabritiis, Frank Noé, and Cecilia Clementi. Machine learning of coarse-grained molecular dynamics force fields. *ACS central science*, 5(5):755–767, 2019.

[40] Francisco Villaescusa-Navarro, ChangHoon Hahn, Elena Massara, Arka Banerjee, Ana Maria Delgado, Doogesh Kodi Ramanah, Tom Charnock, Elena Giusarma, Yin Li, Erwan Allys, et al. The quijote simulations. *arXiv preprint arXiv:1909.05273*, 2019.

[41] Scott Dodelson. *Modern cosmology*. 2003.

[42] David N Spergel, Licia Verde, Hiranya V Peiris, E Komatsu, MR Nolta, CL Bennett, M Halpern, G Hinshaw, N Jarosik, A Kogut, et al. First-year wilkinson microwave anisotropy probe (wmap)* observations: determination of cosmological parameters. *The Astrophysical Journal Supplement Series*, 148(1):175, 2003.

[43] Planck Collaboration, N. Aghanim, Y. Akrami, M. Ashdown, J. Aumont, C. Baccigalupi, M. Ballardini, A. J. Banday, R. B. Barreiro, N. Bartolo, S. Basak, R. Battye, K. Benabed, J. P. Bernard, M. Bersanelli, P. Bielewicz, J. J. Bock, J. R. Bond, J. Borrill, F. R. Bouchet, F. Boulanger, M. Bucher, C. Burigana, R. C. Butler, E. Calabrese, J. F. Cardoso, J. Carron, A. Challinor, H. C. Chiang, J. Chluba, L. P. L. Colombo, C. Combet, D. Contreras, B. P. Crill, F. Cuttaia, P. de Bernardis, G. de Zotti, J. Delabrouille, J. M. Delouis, E. Di Valentino, J. M. Diego, O. Doré, M. Douspis, A. Ducout, X. Dupac, S. Dusini, G. Efstathiou, F. Elsner, T. A. Enßlin, H. K. Eriksen, Y. Fantaye, M. Farhang, J. Fergusson, R. Fernandez-Cobos, F. Finelli, F. Forastieri, M. Frailis, A. A. Fraisse, E. Franceschi, A. Frolov, S. Galeotta, S. Galli, K. Ganga, R. T. Génova-Santos, M. Gerbino, T. Ghosh, J. González-Nuevo, K. M. Górski, S. Gratton, A. Gruppuso, J. E. Gudmundsson, J. Hamann, W. Handley, F. K. Hansen, D. Herranz, S. R. Hildebrandt, E. Hivon, Z. Huang, A. H. Jaffe, W. C. Jones, A. Karakci, E. Keihänen, R. Keskitalo, K. Kiiveri, J. Kim, T. S. Kisner, L. Knox, N. Krachmalnicoff, M. Kunz, H. Kurki-Suonio, G. Lagache, J. M. Lamarre, A. Lasenby, M. Lattanzi, C. R. Lawrence, M. Le Jeune, P. Lemos, J. Lesgourgues, F. Levrier, A. Lewis, M. Liguori, P. B. Lilje, M. Lilley, V. Lindholm, M. López-Caniego, P. M. Lubin, Y. Z. Ma, J. F. Macías-Pérez, G. Maggio, D. Maino, N. Mandolesi, A. Mangilli, A. Marcos-Caballero, M. Maris, P. G. Martin, M. Martinelli, E. Martínez-González, S. Matarrese, N. Mauri, J. D. McEwen, P. R. Meinhold, A. Melchiorri, A. Mennella, M. Migliaccio, M. Millea, S. Mitra, M. A. Miville-Deschênes, D. Molinari, L. Montier, G. Morgante, A. Moss, P. Natoli, H. U. Nørgaard-Nielsen, L. Pagano, D. Paoletti, B. Partridge, G. Patanchon, H. V. Peiris, F. Perrotta, V. Pettorino, F. Piacentini, L. Polastri, G. Polenta, J. L. Puget, J. P. Rachen, M. Reinecke, M. Remazeilles, A. Renzi, G. Rocha, C. Rosset, G. Roudier, J. A. Rubiño-Martín, B. Ruiz-Granados, L. Salvati, M. Sandri, M. Savelainen, D. Scott, E. P. S. Shellard, C. Sirignano, G. Sirri, L. D. Spencer, R. Sunyaev, A. S. Suur-Uski, J. A. Tauber, D. Tavagnacco, M. Tenti, L. Toffolatti, M. Tomasi, T. Trombetti, L. Valenziano, J. Valiviita, B. Van Tent, L. Vibert, P. Vielva, F. Villa, N. Vittorio, B. D. Wand elt, I. K. Wehus, M. White, S. D. M. White, A. Zacchei, and A. Zonca. Planck 2018 results. VI. Cosmological parameters. *arXiv e-prints*, art. arXiv:1807.06209, July 2018.

[44] Carlos S. Frenk, Simon D. M. White, Marc Davis, and George Efstathiou. The Formation of Dark Halos in a Universe Dominated by Cold Dark Matter. *Astrophysical Journal*, 327:507, April 1988. doi: 10.1086/166213.

[45] Gary Marcus. The next decade in ai: four steps towards robust artificial intelligence. *arXiv preprint arXiv:2002.06177*, 2020.

[46] S. van der Walt, S. C. Colbert, and G. Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science Engineering*, 13(2):22–30, 2011.

[47] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: https://doi.org/10.1038/s41592-019-0686-2.

[48] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011. URL `http://jmlr.org/papers/v12/pedregosa11a.html`.

[49] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. Jupyter Notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press, 2016.

[50] J. D. Hunter. Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, 9 (3):90–95, 2007.

[51] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010. doi: 10.25080/Majora-92bf1922-00a.

[52] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[53] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

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: $V = \{\mathbf{v}_i\}_{i=1:N^v}$, with each $\mathbf{v}_i \in \mathbb{R}^{L^v}$. 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 $V' = \{\mathbf{v}'_i\}_{i=1:N^v}$, where each $\mathbf{v}'_i \in \mathbb{R}^{L^{v\prime}}$ is the node corresponding to $\mathbf{v}_i$. The number of features in these predicted nodes, $L^{v\prime}$, 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: $\phi^e : \mathbb{R}^{L^v} \times \mathbb{R}^{L^v} \to \mathbb{R}^{L^{e'}}$. 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 ϕ^e as a multi-layer perceptron with ReLU activations and two hidden layers, each with 300 hidden nodes. The mapping is $\mathbf{e}'_k = \phi^e(\mathbf{v}_{r_k}, \mathbf{v}_{s_k})$ 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, $\bar{\mathbf{e}}'_i \in \mathbb{R}^{L^{e'}}$. This can be written as $\bar{\mathbf{e}}'_i = \sum_{k\in\{1:N^e|r_k=i\}} \mathbf{e}'_k$.

Node model. We create a second neural network to predict the output nodes, $\mathbf{v}'_i$, for each i from the corresponding summed message and input node. This net can be written as $\phi^v : \mathbb{R}^{L^v} \times \mathbb{R}^{L^{e'}} \to \mathbb{R}^{L^{v\prime}}$, and has the mapping: $\hat{\mathbf{v}}'_i = \phi^v(\mathbf{v}_i, \bar{\mathbf{e}}'_i)$, where $\hat{\mathbf{v}}'_i$ is the prediction for $\mathbf{v}'_i$. We also create ϕ^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 L$_1$ models as:

$$\begin{aligned}
&\text{Input graph } G = (V, E) \text{ with}\\
&\text{nodes (e.g., positions of particles) } V = \{\mathbf{v}_i\}_{i=1:N^v};\ \mathbf{v}_i \in \mathbb{R}^{L^v},\ \text{and}\\
&\text{edges (indices of connected nodes) } E = \{(r_k, s_k)\}_{k=1:N^e};\ r_k, s_k \in \{1 : N^v\}.\\
&\text{Compute messages for each edge: } \mathbf{e}'_k = \phi^e(\mathbf{v}_{r_k}, \mathbf{v}_{s_k}),\\
&\mathbf{e}'_k \in \mathbb{R}^{L^{e'}},\ \text{then}\\
&\text{sum for each receiving node } i:\ \bar{\mathbf{e}}'_i = \sum_{k \in \{1:N^e | r_k = i\}} \mathbf{e}'_k,\\
&\bar{\mathbf{e}}'_i \in \mathbb{R}^{L^{e'}}.\\
&\text{Compute output node prediction: } \hat{\mathbf{v}}'_i = \phi^v(\mathbf{v}_i, \bar{\mathbf{e}}'_i)\\
&\hat{\mathbf{v}}'_i \in \mathbb{R}^{L^{v\prime}}.
\end{aligned}$$

Loss. We jointly optimize the parameters in ϕ^v and ϕ^e via mini-batch gradient descent with Adam as the optimizer. Our total loss function for optimizing is:

$$\begin{aligned}
&\mathcal{L} = \mathcal{L}_v + \alpha_1 \mathcal{L}_e + \alpha_2 \mathcal{L}_n,\ \text{where}\\
&\text{the prediction loss is } \mathcal{L}_v = \frac{1}{N^v} \sum_{i \in \{1:N^v\}} \left| \mathbf{v}'_i - \hat{\mathbf{v}}'_i \right|,\\
&\text{the message regularization is } \mathcal{L}_e = \frac{1}{N^e} \left\{ \begin{array}{cc} \sum_{k \in \{1:N^e\}} \left| \mathbf{e}'_k \right|, & \text{L}_1 \\ 0, & \text{Standard} \\ 0, & \text{Bottleneck} \end{array} \right.,\\
&\text{with the regularization constant } \alpha_1 = 10^{-2},\ \text{and the}\\
&\text{regularization for the network weights is } \mathcal{L}_n = \sum_{l = \{1:N^l\}} |w_l|^2,\\
&\text{with } \alpha_2 = 10^{-8},
\end{aligned}$$

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`. 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 L$_1$ 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 ϕ^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 ϕ^e now represent the means, and the second half of the outputs represent the log variance of a particular message component. In other words,

$$\begin{aligned}
\boldsymbol{\mu}'_k &= \phi^e_{1:100}(\mathbf{v}_{r_k}, \mathbf{v}_{s_k}),\\
\boldsymbol{\sigma}'^2_k &= \exp\big(\phi^e_{101:200}(\mathbf{v}_{r_k}, \mathbf{v}_{s_k})\big),\\
\mathbf{e}'_k &\sim \mathcal{N}(\boldsymbol{\mu}'_k, \mathrm{diag}(\boldsymbol{\sigma}'^2_k)),\\
\bar{\mathbf{e}}'_i &= \sum_{k\in\{1:N^e|r_k=i\}} \mathbf{e}'_k,\\
\hat{\mathbf{v}}'_i &= \phi^v(\mathbf{v}_i, \bar{\mathbf{e}}'_i),
\end{aligned}$$

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 $\mathbf{e}'_k$, and pass those samples through a sum to compute a sample of $\bar{\mathbf{e}}'_i$ and then pass that value through the edge function to compute a sample of $\hat{\mathbf{v}}'_i$. The loss is calculated normally, except for $\mathcal{L}_e$, which becomes the KL divergence with respect to our Gaussian prior of $\mu = 0,\ \sigma = 1$:

$$\mathcal{L}_e = \frac{1}{N^e} \sum_{k=\{1:N^e\}} \sum_{j=\{1:L^{e'}/2\}} \frac{1}{2}\left(\mu'^2_{k,j} + \sigma'^2_{k,j} - \log\left(\sigma'^2_{k,j}\right)\right),$$

with $\alpha_1 = 1$ (equivalent to $\beta = 1$ for the loss of a β-Variational Autoencoder; simply the standard VAE). The KL-divergence loss also encourages sparsity in the messages $\mathbf{e}'_k$ similar to the L_1 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 $\mathbf{e}'_k$ 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.

Here, we sketch a rough mathematical explanation of our hypothesis that we will reconstruct the true force in the graph network given our inductive biases. Newtonian mechanics prescribes that force vectors, $\mathbf{f}_k \in \mathcal{F}$, can be summed to produce a net force, $\sum_k \mathbf{f}_k = \bar{\mathbf{f}} \in \mathcal{F}$, which can then be used to update the dynamics of a body. Our model uses the i-th body's pooled messages, $\bar{\mathbf{e}}'_i$ to update the body's state via $\mathbf{v}'_i = \phi^v(\mathbf{v}_i, \bar{\mathbf{e}}'_i)$. If we assume our GN is trained to predict accelerations perfectly for any number of bodies, this means (ignoring mass) that $\bar{\mathbf{f}}_i = \sum_{r_k=i} \mathbf{f}_k = \phi^v(\mathbf{v}_i, \sum_{r_k=i} \mathbf{e}'_k) = \phi^v(\mathbf{v}_i, \bar{\mathbf{e}}'_i)$. Since this is true for any number of bodies, we also have the result for a single interaction: $\bar{\mathbf{f}}_i = \mathbf{f}_{k,r_k=i} = \phi^v(\mathbf{v}_i, \mathbf{e}'_{k,r_k=i}) = \phi^v(\mathbf{v}_i, \bar{\mathbf{e}}'_i)$. Thus, we can substitute this expression into the multi-interaction case: $\sum_{r_k=i} \phi^v(\mathbf{v}_i, \mathbf{e}'_k) = \phi^v(\mathbf{v}_i, \bar{\mathbf{e}}'_i) = \phi^v(\mathbf{v}_i, \sum_{r_k=i} \mathbf{e}'_k)$. From this relation, we see that ϕ^v has to be a linear transformation conditioned on $\mathbf{v}_i$. Therefore, for cases where $\phi^v(\mathbf{v}_i, \bar{\mathbf{e}}'_i)$ is invertible in $\bar{\mathbf{e}}'_i$ (which becomes true when $\bar{\mathbf{e}}'_i$ is the same dimension as the output of ϕ^v), we can write $\mathbf{e}'_k = (\phi^v(\mathbf{v}_i, \cdot))^{-1}(\mathbf{f}_k)$, which is also a linear transform, meaning that the message vectors are linear transformations of the true forces when $L^{e'}$ is equal to the dimension of the forces.

If the dimension of the force vectors (or what the minimum dimension of the message vectors "should" be) is unknown, one can encourage the messages to be sparse by applying L_1 or Kullback-Leibler regularizations to the messages in the GN. The aim is for the messages to learn the minimal vector space required for the computation automatically. This is a more mathematical explanation of why the message features are linear combinations of the force vectors, when our inductive bias of a bottleneck or sparse regularization is applied. We emphasize that this is a new contribution: never before has previous work explicitly identified the forces in a graph network.

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 ϕ^e. 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 ϕ^e with a symbolic model with included input edge parameters, and also fit the global model, denoted ϕ^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, $\mathcal{H}$, which is a scalar. We label pairwise interaction energy $\mathcal{H}_{\text{pair}}$ and the energy of individual particles as $\mathcal{H}_{\text{self}}$. Thus, using our previous graph notation, we can write the total energy of a system as:

$$\mathcal{H} = \sum_{i=1:N^v} \mathcal{H}_{\text{self}}(\mathbf{v}_i) + \sum_{k\in\{1:N^e\}} \mathcal{H}_{\text{pair}}(\mathbf{v}_{r_k}, \mathbf{v}_{s_k}). \tag{1}$$

For particles interacting via gravity, this would be

$$\mathcal{H} = \sum_i \frac{p_i^2}{2m_i} - \frac{1}{2}\sum_{i\neq j} \frac{m_i m_j}{|\mathbf{r}_i - \mathbf{r}_j|}, \tag{2}$$

where $\mathbf{p}_i, m_i, \mathbf{r}_i$ indicates the momentum, mass, and position of particle i, respectively, and we have set the gravitational constant to 1. Following [35, 37], we could model $\mathcal{H}$ as a neural network, and apply Hamilton's equations to create a dynamical model. More specifically, as in [37], we can predict $\mathcal{H}$ 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 $\mathcal{H}$ 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:

$$\mathcal{H}_{\text{self}} : \mathbb{R}^{L^v} \to \mathbb{R}, \tag{3}$$

and one for each edge:

$$\mathcal{H}_{\text{pair}} : \mathbb{R}^{L^v} \times \mathbb{R}^{L^v} \to \mathbb{R}. \tag{4}$$

Note that the derivatives of $\mathcal{H}$ now propagate through the pool, e.g.,

$$\begin{aligned}\frac{\partial \mathcal{H}(V)}{\partial \mathbf{v}_i} = \frac{\partial \mathcal{H}_{\text{self}}(\mathbf{v}_i)}{\partial \mathbf{v}_i} &+ \sum_{r_k=i} \frac{\partial \mathcal{H}_{\text{pair}}(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k})}{\partial \mathbf{v}_i} \\ &+ \sum_{s_k=i} \frac{\partial \mathcal{H}_{\text{pair}}(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k})}{\partial \mathbf{v}_i}.\end{aligned} \tag{5}$$

This model is similar to the Lagrangian Graph Network proposed in [38]. 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 $\mathcal{H}_{\text{self}}$ and $\mathcal{H}_{\text{pair}}$ 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 $\mathcal{H}$. One difference is that we also regularize $\mathcal{H}_{\text{pair}}$, since it is degenerate with $\mathcal{H}_{\text{self}}$ 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'_{12} 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

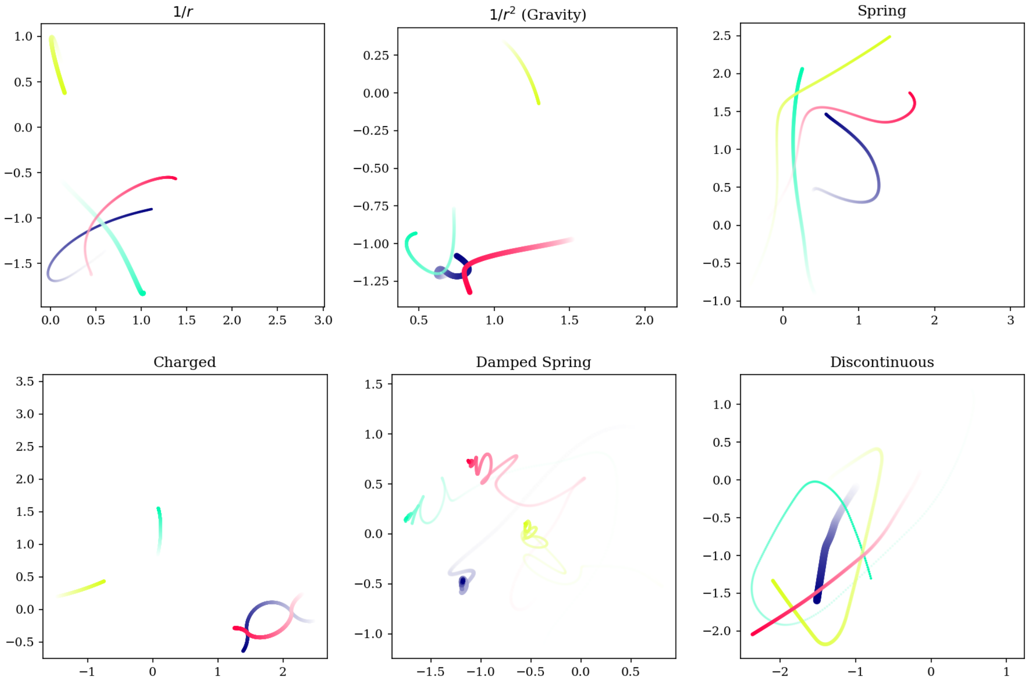

Figure 4: Examples of a selection of simulations, for 4 nodes and two dimensions. Decreasing transparency shows increasing time, and size of points shows mass.

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

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

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

$$\ddot{\mathbf{r}}_i = -\frac{1}{m_i} \sum_j \nabla_{\mathbf{r}_i} U_{ij}. \tag{6}$$

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 1, and the fit itself is visualized in fig. 5 for various models on the 2D spring simulation.

Sim.	Standard	Bottleneck	L_1	KL	FlatHGN
Charge-2	**49**	50	52	60	55
Charge-3	1.2	0.99	**0.94**	4.2	3.5
Damped-2	**0.30**	0.33	**0.30**	1.5	0.35
Damped-3	0.41	0.45	**0.40**	3.3	0.47
Disc.-2	0.064	0.074	**0.044**	1.8	0.075
Disc.-3	0.20	0.18	**0.13**	4.2	0.14
r^{-1}-2	0.077	0.069	0.079	3.5	**0.05**
r^{-1}-3	0.051	0.050	0.055	3.5	**0.017**
r^{-2}-2	1.6	1.6	**1.2**	9.3	1.3
r^{-2}-3	4.0	3.6	3.4	9.8	**2.5**
Spring-2	0.047	0.046	0.045	1.7	**0.016**
Spring-3	0.11	0.11	0.090	3.8	**0.010**

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.

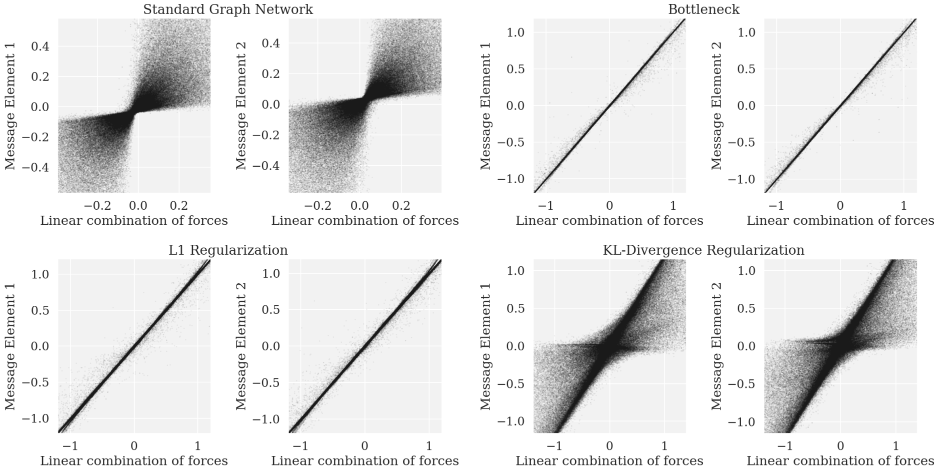

Figure 5: The most significant message components of each model compared with a linear combination of the force components: this example, the spring simulation in 2D with eight nodes for training. These plots demonstrate that the GN's messages have learned to be linear transformations of the vector components 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 sub-components of the model with symbolic regression, over observed inputs and outputs. For our aforementioned GNN implementation, we can record the outputs of ϕ^e and ϕ^v 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 ϕ^e (we also refer to the output features as "message components"). For the L_1 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: $\mu^2 + \sigma^2 - \log(\sigma^2)$. 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 ϕ_1^e, over random examples of the training dataset. We do this for the particle simulations in section 4.1. The inputs to the actual ϕ_1^e neural network are: $m_1, m_2, q_1, q_2, x_1, x_2, \ldots$ (mass, charge, and Cartesian positions of receiving and sending node), leaving us with many examples of $(m_1, m_2, q_1, q_2, x_1, x_2, \ldots, \phi_1^e)$. We would like to fit a symbolic expression to map $(m_1, m_2, q_1, q_2, x_1, x_2, \ldots) \to \phi_1^e$. 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 $(m_1, m_2, q_1, q_2, \Delta x, \Delta y, (\Delta z,) r, \phi_1^e)$ examples (we take 5000 examples for each of our tests) to *eureqa*, and ask it to fit ϕ_1^e as a function of the others by minimizing the mean absolute error (MAE). We allow it to use the operators $+, -, \times, /, >, <,$ ^$, \exp, \log, \text{IF}(\cdot,\cdot,\cdot)$ 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 ^$, \exp, \log, \text{IF}(\cdot,\cdot,\cdot)$ 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 4 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. 6. We find experimentally that this score produces the best-recovered solutions in a variety of tests on different generating equations.

Following this process, we fit a single analytic expression to model ϕ_1^e 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 $\phi_1^e \approx (\mathbf{a} \cdot (\Delta x, \Delta y))(r-1) + b$).

$$\phi_1^e \approx 1.36\Delta y + 0.60\Delta x - \frac{0.60\Delta x + 1.37\Delta y}{r} - 0.0025$$

- $1/r^2$, 3D, Bottleneck (expect $\phi_1^e \approx \frac{\mathbf{a}\cdot(\Delta x, \Delta y, \Delta z)}{r^3} + b$).

$$\phi_1^e \approx \frac{0.021\Delta x m_2 - 0.077\Delta y m_2}{r^3}$$

- Discontinuous, 2D, L$_1$ (expect $\phi_1^e \approx \text{IF}(r > 2, (\mathbf{a} \cdot (\Delta x, \Delta y, \Delta z))r, 0) + b$).

$$\phi_1^e \approx \text{IF}(r > 2, 0.15r\Delta y + 0.19r\Delta x, 0) - 0.038$$

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 $\phi_1^e \approx (\mathbf{a} \cdot (\Delta x, \Delta y, \Delta z))(r-1) + b$).

$$\phi_1^e \approx 0.57\Delta y + 0.32\Delta z$$

- $1/r$, 3D, Standard (expect $\phi_1^e \approx \frac{\mathbf{a}\cdot(\Delta x, \Delta y, \Delta z)}{r^2} + b$).

$$\phi_1^e \approx \frac{0.041 + m_2\text{IF}(\Delta z > 0, 0.021, 0.067)}{r}$$

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.

Discovering potentials using FlatHGN. Lastly, we also show an example of a successful reconstruction of a pairwise Hamiltonian from data. We treat the $\mathcal{H}_{\text{pair}}$ just as we would ϕ_1^e, and fit it to data. The one difference here is that there are potential $\mathcal{H}_{\text{pair}}$ 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 $\mathcal{H}_{\text{pair}}$ are used. Thus, we cannot simply fit a linear transformation of the true $\mathcal{H}_{\text{pair}}$ to data to verify it has learned our generating equation: we must rely on symbolic regression

Solutions extracted for the 2D $1/r^2$ Simulation	MAE	Complexity
$\phi_1^e = 0.162 + (5.62 + 20.3 m_2 \Delta x - 153 m_2 \Delta y)/r^3$	17.954713	22
$\phi_1^e = (6.07 + 19.9 m_2 \Delta x - 154 m_2 \Delta y)/r^3$	18.400224	20
$\phi_1^e = (3.61 + 20.9 \Delta x - 154 m_2 \Delta y)/r^3$	42.323236	18
$\phi_1^e = (31.6 \Delta x - 152 m_2 \Delta y)/r^3$	69.447467	16
$\phi_1^e = (2.78 - 152 m_2 \Delta y)/r^3$	131.42547	14
$\phi_1^e = -142 m_2 \Delta y/r^3$	160.31243	12
$\phi_1^e = -184 \Delta y/r^2$	913.83751	8
$\phi_1^e = -7.32 \Delta y/r$	1520.9493	6
$\phi_1^e = -0.282 m_2 \Delta y$	1551.3437	5
$\phi_1^e = -0.474 \Delta y$	1558.9756	3
$\phi_1^e = 0.0148$	1570.0905	1

Solutions extracted for the 2D $1/r$ Simulation	MAE	Complexity
$\phi_1^e = (4.53 m_2 \Delta y - 1.53 \Delta x - 15.0 m_2 \Delta x)/r^2 - 0.209$	0.37839388	22
$\phi_1^e = (4.58 m_2 \Delta y - \Delta x - 15.2 m_2 \Delta x)/r^2 - 0.227$	0.38	20
$\phi_1^e = (4.55 m_2 \Delta y - 15.5 m_2 \Delta x)/r^2 - 0.238$	0.42	18
$\phi_1^e = (4.59 m_2 \Delta y - 15.5 m_2 \Delta x)/r^2$	0.46575519	16
$\phi_1^e = (10.7 \Delta y - 15.5 m_2 \Delta x)/r^2$	2.48	14
$\phi_1^e = (\Delta y - 15.6 m_2 \Delta x)/r^2$	6.96	12
$\phi_1^e = -15.6 m_2 \Delta x/r^2$	7.93	10
$\phi_1^e = -34.8 \Delta x/r^2$	31.17	8
$\phi_1^e = -8.71 \Delta x/r$	68.345174	6
$\phi_1^e = -0.360 m_2 \Delta x$	85.743106	5
$\phi_1^e = -0.632 \Delta x$	93.052677	3
$\phi_1^e = -\Delta x$	96.708906	2
$\phi_1^e = -0.303$	103.29053	1

Table 4: Results of using symbolic regression to fit equations to the most significant (see text) feature of ϕ^e, denoted ϕ_1^e, for the $1/r^2$ (top) and $1/r$ (bottom) force laws, extracted from the bottleneck model. We expect to see $\phi_1^e \approx \frac{\mathbf{a}\cdot(\Delta x, \Delta y, \Delta z)}{r^\alpha} + b$, for arbitrary $\mathbf{a}$ 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.

to extract the full functional form. We follow the same procedure as before, and successfully extract the potential for a charge simulation:

$$\mathcal{H}_{\text{pair}} \approx \frac{0.0019 q_1 q_2}{r} - 0.0112 - 0.00143 q_1 - 0.00112 q_1 q_2,$$

where we expect $\mathcal{H}_{\text{pair}} \approx a \frac{q_1 q_2}{r} + f(q_1, q_2, m_1, m_2)$, for constant a and arbitrary function f, which shows that the neural network has learned the correct form of the Hamiltonian.

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 L_1 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

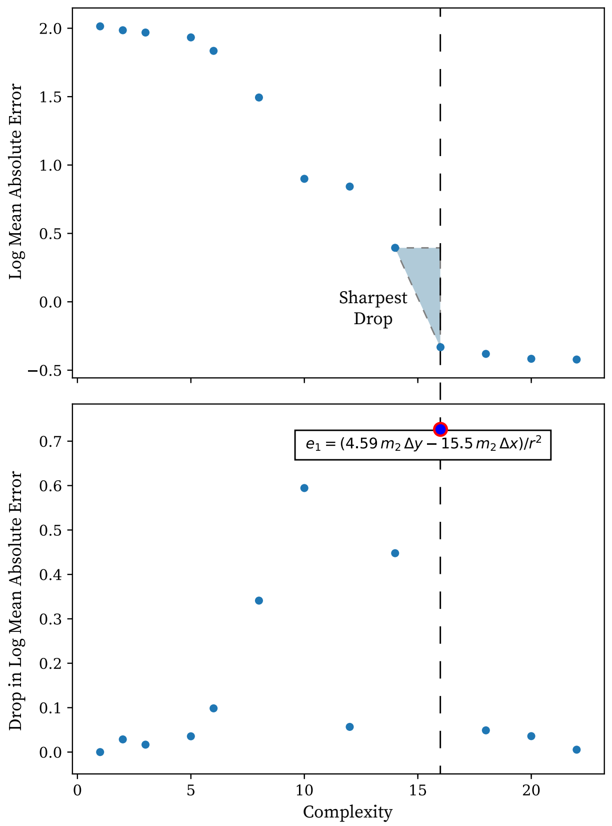

Figure 6: A plot of the data for the $1/r$ simulation in table 4, indicating mean absolute error versus complexity in the top plot and fractional drop in mean absolute error over the next-best model in the bottom plot. As indicated, we take the largest drop in log-loss over a single increase in complexity as the chosen model—it is our parametrization of Occam's Razor.

darker color corresponds to a larger standard deviation. Since we apply L_1 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 L_1 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 δ_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: L_1 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

	Test	Formula	Summed Component	$\left\langle\left\|\delta_i - \hat{\delta}_i\right\|\right\rangle$
Old	Constant	$\hat{\delta}_i = C_1$	N/A	0.421
	Simple	$\hat{\delta}_i = C_1 + (C_2 + M_i C_3) e_i$	$e_i = \sum_{j \neq i}^{\left\|\mathbf{r}_i - \mathbf{r}_j\right\| < 20} M_j$	0.121
New	Best, without mass	$\hat{\delta}_i = C_1 + \frac{e_i}{C_2 + C_3 e_i \|\mathbf{v}_i\|}$	$e_i = \sum_{j \neq i} \frac{C_4 + \left\|\mathbf{v}_i - \mathbf{v}_j\right\|}{C_5 + (C_6 \left\|\mathbf{r_i} - \mathbf{r_j}\right\|)^{C_7}}$	0.120
	Best, with mass	$\hat{\delta}_i = C_1 + \frac{e_i}{C_2 + C_3 M_i}$	$e_i = \sum_{j \neq i} \frac{C_4 + M_j}{C_5 + (C_6 \|\mathbf{r}_i - \mathbf{r}_i\|)^{C_7}}$	0.0882

Test	Best-fit Parameters
Simple	$C_1 = 0.415$
Traditional	$C_1 = -0.0376, C_2 = 0.0529, C_3 = 0.000927$
Best, without mass	$C_1 = -0.199, C_2 = 1.31, C_3 = 0.027,$ $C_4 = 1.54, C_5 = 50.165, C_6 = 18.94, C_7 = 13.21$
Best, with mass	$C_1 = -0.156, C_2 = 3.80, C_3 = 0.0809,$ $C_4 = 0.438, C_5 = 7.06, C_6 = 15.5, C_7 = 20.3$
Best, with mass and cutoff*	$C_1 = -0.149, C_2 = 3.77, C_3 = 0.0789,$ $C_4 = 0.442, C_5 = 7.09, C_6 = 15.5, C_7 = 21.3$

Table 5: 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.

components have a standard deviation of under 10^{-8}. This suggests that the δ_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 ϕ_1^e which takes the central halo and each neighboring halo, and passes it to ϕ^v which predicts δ_i given the central halo properties.

Best-fit parameters. We list best-fit parameters for the discovered models in the paper in table 5. The functional forms were extracted from the GN by approximating both ϕ_1^e and ϕ^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 to data, symbolic regression to graph net). Thus, we take out the functional forms as given in table 5, 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.