Contents

1 Introduction

The aim of this report is to reproduce the key results of the paper *Discovering Symbolic Models from Deep Learning with Inductive Biases* by *Cranmer et al.*. The main contributions of the paper are two fold - first the presentation of a systematic framework whereby inductive biases can be employed to distill low dimensional symbolic approximations to learned neural networks. The second is showing its successful application of the framework in 3 contexts: rediscovering force laws, rediscovering Hamiltonians and discovering a new equation for a nontrivial cosmology problem. Since the time of the paper's publication, there have been several successful applications of the framework to a variety of problems more complex than the ones presented in the paper (cite some examples). The scope of this report is limited to the reproducibility of the experiments relating to the rediscovery of force laws. However, in doing so, the wider contribution of the framework will be validated.

1.1 Motivation

First the motivation behind symbolic distillation is explored. Why would it be desirable to approximate a high dimensional neural network with a low dimensional symbolic model? The answer lies in the scientific method itself and the unreasonable effectiveness of mathematics in the natural sciences. When using neural networks as a tool for science, the lack of interpretability is a major drawback. Science is not just about making predictions, but also about understanding the underlying mechanisms that govern the phenomena being studied. Neural networks are famously known to be black boxes, with the decision-making process being opaque to the user. If they are to be used for science, there needs to be some distillation of what the network has learned in a form that is interpretable to humans. Symbolic models are a natural choice for this task, as they provide a compact and interpretable representation of the underlying mechanisms due to their low dimensionality. Further, they are likely to generalise better to out of distribution data than the neural network itself, indeed in the findings of the paper, the symbolic models generalised better than the neural networks to unseen data. This is due to the unreasonable effectiveness of mathematics in the natural sciences, for whatever reason, choosing to describe the world in terms of closed form low dimensional mathematical equations has been shown to be a very good inductive bias.

1.2 Inductive Biases

It is important to fully explore the concept of effective inductive biases. Inductive biases can be understood as the priors that a set of modelling assumptions impose on the space of possible functions. To illustrate the challenges neural networks face in scientific modelling, consider an experiment depicted in Figure 1 where a neural network, consisting of a single layer with 100 ReLU activations, is trained on data points sampled from a sinusoidal function. While this high-dimensional neural network fits the training data adequately, it struggles to generalise. Conversely, one can searching for an analytic equation using a technique known as symbollic regression. Symbolic regression is a supervised machine learning method that constructs analytic functions to model data. Applying symbolic regression to the same dataset yields the equation $y = sin(x)$, which perfectly describes the data. Thus, applying the inductive bias that the underlying function is a closed form low dimensional equation, results in a model that generalises, in this case, perfectly. This example highlights the advantages of analytic equations, which not only require far fewer parameters than neural networks to represent, but also lead to more reliable generalisation to out-of-distribution data.

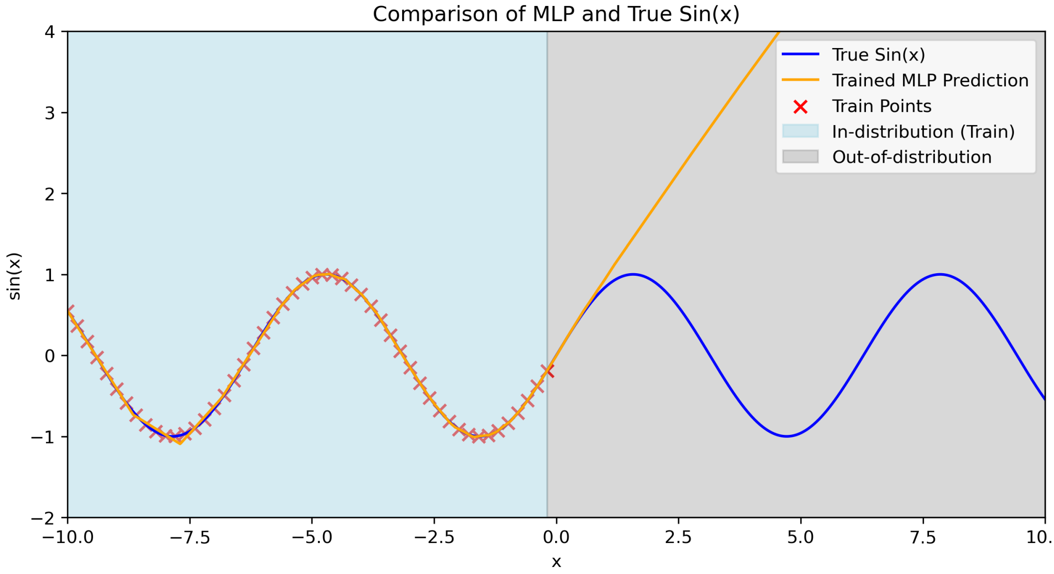


Figure 1: Neural network prediction (one layer network with width 100 and ReLU activations) versus symbolic model for sinusoidal dataset. The symbolic model is $y = sin(x)$.

1.3 Symbolic Distillation Framework

The general framework prescribed by the paper for symbolic distillation is as follows:

- Design a neural network with an architecture that has a seperable internal structure and an inductive bias that is well suited to modelling the underlying problem.

- Train the model with regularisation techniques that encourage the network to learn a low-dimensional representation of the data.
- Replace a component of the neural network with a symbolic model that approximates the neural network's output.
- Retrain the neural network with the symbolic model as a component, and repeat the process until all components of the neural network have been replaced with low dimensional symbolic models.

This framework may prompt the question: what is the necessity of employing neural networks initially? Why not directly utilize symbolic regression? The issue arises from the high dimensionality of modern datasets, which renders the application of symbolic regression, essentially a brute-force search over the space of all possible equations, computationally infeasible. To illustrate this point and the motivation behind this work, consider the problem of discovering a force law from a dataset of observed instantaneous accelerations of particles $\mathbf{a}_i \in \mathbb{R}^n$ and their positions $\mathbf{x}_i \in \mathbb{R}^n$. Assume a force law of the form $\mathbf{f}_{ij} = -k|\mathbf{x}_i - \mathbf{x}_j|$, with $\mathbf{a}_i = \frac{\mathbf{F}_i}{m_i}$, where $\mathbf{F}_i = \sum_j \mathbf{f}_{ij}$. To learn symbolic models for the acceleration $\mathbf{p}(\mathbf{x})$ and the pairwise force $\mathbf{q}(\mathbf{x})$, one can apply the inductive bias $\mathbf{a}_i = \mathbf{p}(\sum_j \mathbf{q}(\mathbf{x}_i, \mathbf{x}_j))$. If the symbolic regression model needs to search N equations to fit a given model, the search space scales as $O(N^2)$. However, if neural networks are first used to approximate the functions $\mathbf{p}(x)$ and $\mathbf{q}(x)$, and symbolic regression is subsequently applied to approximate these neural networks, the models can now be fit independently, reducing the search space to $O(N)$. If the dimensionality of the problem n increases or the force law becomes more complex, the number of equations N required to search also increases significantly. Therefore, this factorization of the search space is essential to make symbolic regression tractable for high-dimensional problems.

1.4 Graph Neural Networks

The work in the original paper and this report is primarily related to the class of problems which can be classed as interacting particle problems. This type of problem describes a very broad range of most problems encountered in physics. The graph neural network architecture applies a well suited inductive bias to these problems. This section will briefly describe the fundamental components of the graph neural network architecture and why it is well suited to interacting particle problems.

A graph neural network (GNN) is a specialised architecture designed to operate on graph data structures, where nodes represent particles and edges represent pairwise interactions between particles. In this context, each node has an associated feature vector encoding properties such as position, velocity, mass, and charge. The GNN operates through a scheme of message passing, involving two main components: the edge model (ϕ_e) and the node model (ϕ_v).

Edge Model (ϕ_e):

- The edge model is applied to each edge in the graph.
- It takes as input the feature vectors of the two nodes connected by the edge.
- Formally, $\phi_e : \mathbb{R}^{d_v} \times \mathbb{R}^{d_v} \to \mathbb{R}^{d_e}$, where d_v is the dimensionality of the node feature vectors, and d_e is the dimensionality of the edge messages.
- The output of the edge model is called an edge message.

Node Model (ϕ_v):

- Once every edge has an associated edge message, the node model is applied to each node in the graph.
- It takes as input the feature vector of the node and the aggregated edge messages from all inbound edges.
- Aggregation is performed using a summation operator.
- Formally, $\phi_v : \mathbb{R}^{d_v} \times \mathbb{R}^{d_e} \rightarrow \mathbb{R}^{d'_v}$, where d'_v is the dimensionality of the updated node feature vectors.
- The output of the node model is the updated node feature vector.

This message-passing process is repeated for a number of layers, with the output of the final layer being the output of the graph neural network. The computational graph of the GNN is shown in Figure 2, illustrating the interactions between nodes and edges through the edge and node models.

The graph neural network is an appropriate inductive bias for a variety of reason. Firstly, many important forces in physics are defined on pairs of particles, analogous to the message function of the Graph Networks. Further, the summation that aggregates messages is analogous to the calculation of the net force on a receiving particle. Finally, the node function is analogous to the application of Newton's second law: acceleration equals the net force (the summed message) divided by the mass of the receiving particle. Further, the graph neural network architecture is invariant under particle permutations, which is a desirable property for a model which represents interacting particle problems.

It will be the edge and node models of the graph neural network that will be replaced by symbolic models in the symbolic distillation framework. The goal will be to recover the underlying force law used to generate the data.

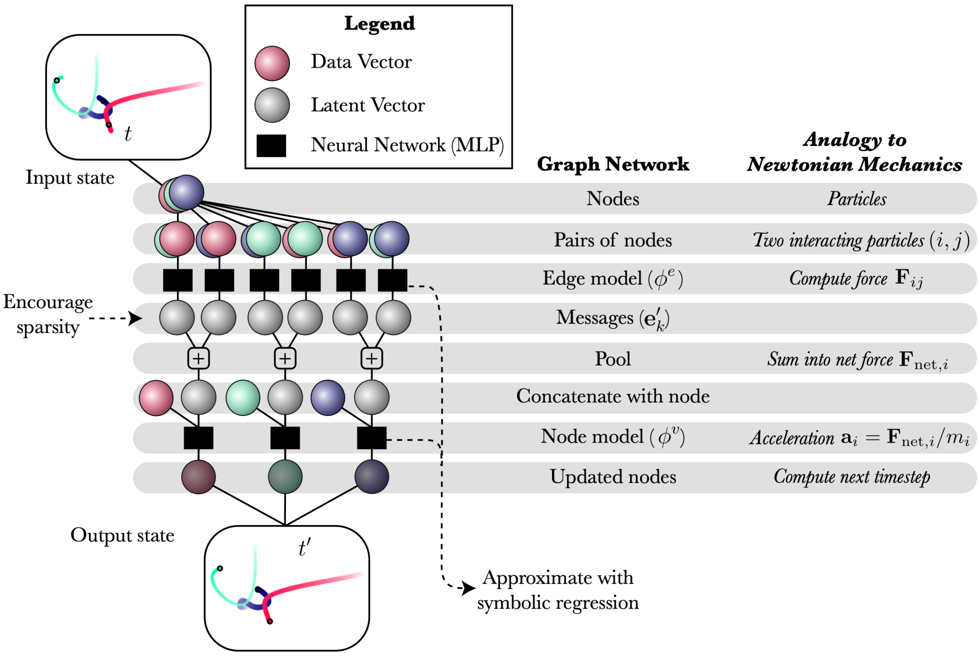

Legend
Data Vector
Latent Vector
Neural Network (MLP)
t
Input state
Graph Network
Analogy to Newtonian Mechanics
Nodes
Particles
Pairs of nodes
Two interacting particles (i, j)
Edge model (ϕ^e)
Compute force $\mathbf{F}_{ij}$
Encourage sparsity
Messages ($\mathbf{e}'_k$)
Pool
Sum into net force $\mathbf{F}_{\text{net},i}$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

Figure 2: Computational graph of the graph neural network architecture. Diagram taken from Cranmer et al. [?]

2 Data

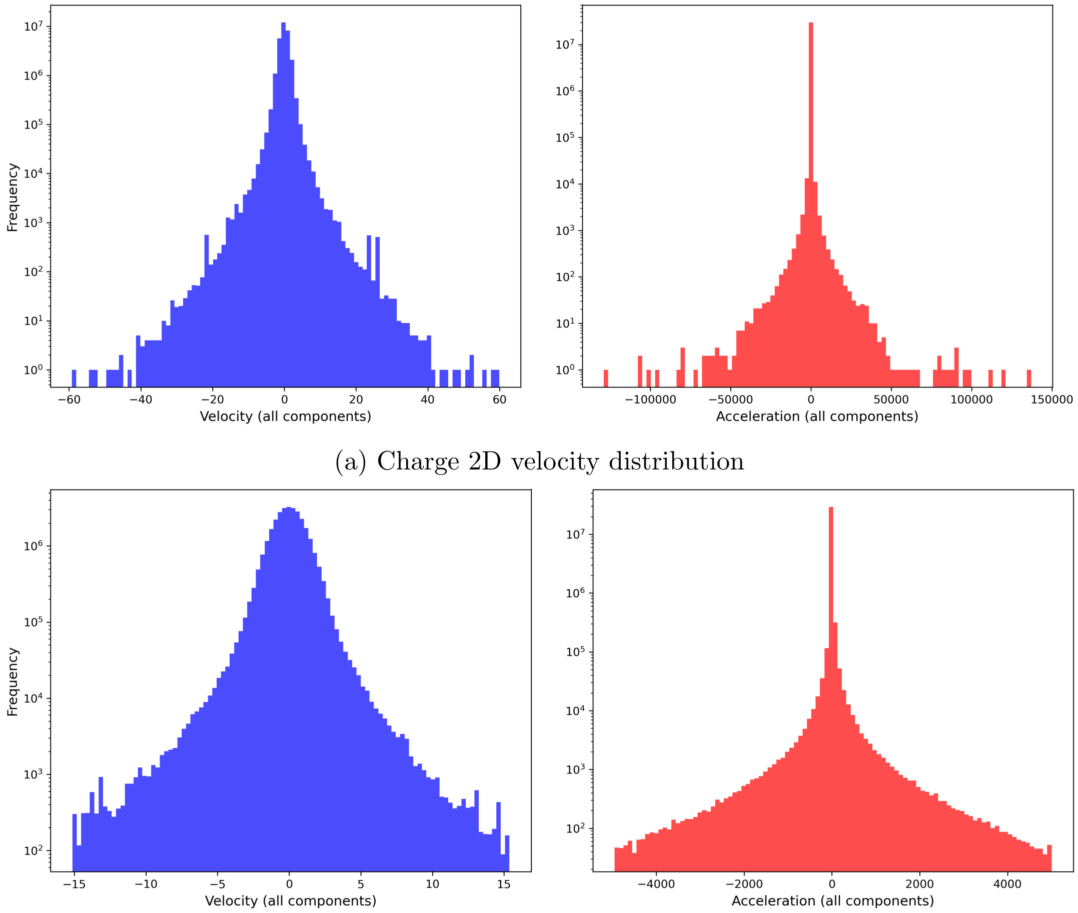
This section will outline the datasets used in the experiments and the pipeline for the problem. The dataset consists of N-body particle simulations for a variety of force laws. From the original paper, the following force laws are investigated:

- Spring: $\mathbf{f}_{ij} = -(|\mathbf{r}_{ij} - 1|)\hat{\mathbf{r}}_{ij}$
- $\frac{1}{r}$ Orbital force: $\mathbf{f}_{ij} = -\frac{m_i m_j \hat{\mathbf{r}}_{ij}}{|\mathbf{r}_{ij}|}$
- $\frac{1}{r^2}$ Orbital force: $\mathbf{f}_{ij} = -\frac{m_i m_j \hat{\mathbf{r}}_{ij}}{|\mathbf{r}_{ij}|^2}$
- Charge: $\mathbf{f}_{ij} = -\frac{q_i q_j \hat{\mathbf{r}}_{ij}}{|\mathbf{r}_{ij}|^2}$

Each force law was simulated in 2D and 3D, with 4 and 8 particles respectively. For each experiment, 10000 simulations were run in parallel for 500 time steps. This was split 3:1 to obtain the training and validation set. The test set was generated separately. The initial positions and velocities were randomly sampled from a normal distribution, the mass was sampled from a log normal distribution and the charge was drawn randomly from the set $\{-1, 1\}$. It is worth noting that the original paper ran each simulation for 1000 time steps, however, to avoid model training times becoming prohibitively long, the simulations were run for half the number of time steps. The step size varies for each simulation due to the differences in scale. It is: 0.005 for $\frac{1}{r}$, 0.001 for $\frac{1}{r^2}$, 0.01 for Spring, 0.001 for Charge. The code used to generate the simulations is available in the repository under the `simulations` directory.

2.1 Preprocessing

Due to the nature of orbital and charge force laws, their dependence on r with a negative power can lead to exploding force values when particles are too close. This causes the data to vary over a scale too large for a neural network to learn effectively. To mitigate this, a small epsilon (10^{-2}) was added to the denominator, squashing r values into a more manageable range. Despite this, some experiments were found to have velocity distributions that still contaned extremely large outliers. To address this, a preprocessing step was introduced. The velocities and accelerations were pruned based on upper and lower percentile bounds. If any particle in a given timestep had a velocity or acceleration outside these bounds, the entire graph for that timestep was removed. This ensured that only timesteps where all particles had acceptable values were retained, improving data quality and consistency. Note, while this breaks the time continuity of the data, it is important to note that the data need not be continuous in time, as each time step defines a complete graph input to the neural network. The impact of this preprocessing step for the 2D Charge data in 3. To see more detailed summary statistics showing the effect of this step, refer to the appendix.



(a) Charge 2D velocity distribution

(b) Charge 2D velocity distribution (pruned)

Figure 3: Velocity and acceleration distributions for Charge 2D data before and after preprocessing

3 Method

This section outlines the method used to train a graph neural network to predict instantaneous accelerations of particles.

3.1 Training strategies

This report investigates 4 of the training strategies outlined in the original paper. The training strategies are as follows: standard, KL, bottleneck and L1.

Standard Strategy The standard training strategy serves as the baseline case where the edge message dimensionality is 100 and the loss does not include any regularisation terms. Here the loss function is given by the mean absolute error between the predicted and true accelerations.

KL Strategy The KL strategy involves sampling edge messages from a multivariate normal distribution as defined by the output of the edge model, which is now a 200-dimensional vector, with the first 100 dimensions representing the mean and the second 100 dimensions representing the log variance. Additionally, the average Kullback-Leibler divergence penalty over all edge messages, using a standard normal as the prior, is added to the loss. The weight of this term is set to 1.0.

L1 Strategy The L1 strategy is the same as the standard, except it now incorporates the L1 norm of the edge message as a regularisation term, averaged over all edge messages, in the loss. The weight of this term is set to 1.0×10^{-2}.

Bottleneck Strategy The bottleneck strategy is the same as the standard, however now aligns the dimensionality of the edge message with the dimensionality of the problem (2 or 3).

3.2 Model Architecture

The architecture of the graph neural network (GNN) employed across various training strategies is consistent, except for variations in the output dimension of the edge model. The GNN comprises two multi-layer perceptrons (MLPs): the edge model and the node model.

Edge Model This model consists of two hidden layers, each with 300 units and ReLU activation functions. It takes as input the concatenated feature vectors from a pair of nodes connected by an edge in the graph. Each input vector is a $(2d + 2)$-dimensional representation of position, velocity, mass, and charge, where d represents the problem's dimensionality (2 or 3). The output dimension of the edge model varies depending on the training strategy.

Node Model Mirroring the edge model in structure, the node model takes as input the summed edge messages from all inbound edges to a given node, concatenated with the node's feature vector. Its output is a vector of dimension 2 or 3, corresponding to the predicted instantaneous acceleration of the node.

3.3 Model Training

The network is trained over 100 epochs using the Adam optimizer with an initial learning rate of 0.001 and weight decay set to 1.0×10^{-8}. It is worth noting that this is half the number of epochs used in the original paper. The learning rate is modulated by the 1cycle learning rate policy [2], which increases the learning rate to a peak of 0.001 before gradually decreasing it by a factor of $100,000$, adhering to a pre-defined schedule. Random translation is used as a form of data augmentation to improve the model's generalisation capabilities. The model is trained on a single NVIDIA A100 GPU.

3.4 Symbolic Regression of the Network

After training the model end to end, the edge and node model are symbolically distilled using the `PySR` package [1]. PySR's internal search algorithm is a multi-population evolutionary algorithm, which consists of a unique evolve-simplify-optimise loop, designed for optimisation of unknown scalar constants in newly-discovered empirical expressions. The operators considered in the fitting process are $+, -, \times, /$ as well as real constants. Note, this is a subset of the operators used in the original paper when fitting for symbolic models. This was done to reduce the search space and quickly check that the symbolic distillation framework was working as expected.

3.4.1 Model Selection Criteria

The best symbolic model is selected by evaluating the mean absolute error (MAE) and the complexity of the model. The complexity is scored by counting the number of occurrences of each operator, constant, and input variable. PySR outputs several equations at each complexity level. The chosen formula is one that maximises the fractional drop in MAE over an increase in complexity from the next best model.

3.4.2 Symbolic Distillation of the Edge Model

To symbolically distil the edge model, its inputs and outputs are recorded over the test set. The input comprises the concatenated feature vectors of two nodes connected by an edge, including the position, velocity, mass, and charge of each node $((m_1, m_2, q_1, q_2, x_1, x_2, \ldots))$. Given that force laws depend on the relative positions of the nodes, To simplify, the positions are converted to $\Delta x = x_2 - x_1$ for displacement, similarly for y (and z in 3D), and $r = \sqrt{\Delta x^2 + \Delta y^2 + (\Delta z^2)}$ for distance. This array of $(m_1, m_2, q_1, q_2, \Delta x, \Delta y, (\Delta z), r)$ comprises as the input to the symbolic model. Next, the d most significant components of the corresponding edge message are selected as the output labels for the symbolic regression model. Significance here is defined by the components of the edge message with the highest standard deviation across the samples, where d is the dimensionality (2 or 3). In the bottleneck strategy, the edge message is already of the correct dimensionality, so no further processing is needed. 5000 such samples are collected from the data in the test set, and the symbolic model is fit to these data points.

3.4.3 Symbolic Distillation of the Node Model

The symbolic distillation of the node model is similar to that of the edge model. The input to the symbolic model is the node's feature vector, which includes the position, velocity, mass, and charge of the node $((m_1, q_1, x_1, \ldots))$, and the aggregated inbound edge messages. The aggregation is not performed over the full 100 dimensions of the edge message, but rather the d most significant components, where d is the dimensionality of the problem (2 or 3). The output of the symbolic model is the predicted instantaneous acceleration of the node. The symbolic model is fit to 5000 samples from the test set.

3.5 Evaluation metrics

To evaluate model performance, the mean absolute error (MAE) between the predicted and true accelerations is calculated over the test set. This metric assesses the predictive capabilities of the trained model. Additionally, the standard deviation of the top 15 edge messages is computed to evaluate the compactness of the learnt representations. This metric indicates whether the training resulted in a model that correctly captured the problem's dimensionality. The R^2 metric between the significant edge message components and a linear transformation of the true pairwise force is also calculated. The appendix of the original paper provides a mathematical argument for why the model would learn a linear transformation of the pairwise force. Furthermore, a qualitative binary classification of the models is conducted, considering a model successful if it can recover the correct force law. This is determined by comparing the symbolic model to the true force law and verifying if the symbolic model represents a linear transformation of the true force law.

4 Results

Sim	Standard	Bottleneck	L_1	KL
Charge-3	**0.110**	0.140	0.157	3.325
Charge-2	0.466	**0.462**	0.483	3.267
r^{-2}-3	0.1753	**0.1746**	0.2128	3.8264
r^{-2}-2	0.475	0.461	**0.450**	4.0380
r^{-1}-3	0.0283	**0.0261**	0.0408	3.251
r^{-1}-2	0.0254	**0.0238**	0.0326	2.3365
Spring-3	**0.0566**	0.0595	0.0728	2.6839
Spring-2	0.0200	**0.0182**	0.0249	1.7455

Table 1: MAE loss on test set. Best value across training strategy shown in bold.

Sim	Standard	Bottleneck	L_1	KL
Charge-3	0.429	**0.817**	0.683	0.251
Charge-2	0.006	**0.700**	0.006	0.381
r^{-2}-3	0.502	**0.518**	0.505	0.323
r^{-2}-2	0.001	**0.411**	0.181	0.291
r^{-1}-3	0.402	**0.403**	0.364	0.333
r^{-1}-2	0.314	**0.324**	0.324	0.250
Spring-3	0.430	0.430	**0.990**	0.918
Spring-2	0.422	0.999	**1.000**	0.768

Table 2: R^2 metric between the significant edge message components and a linear transformation of the true pairwise force. Best value across training strategy shown in bold.

Sim	Standard	Bottleneck	L_1	KL
Charge-3	0.404	1.000	0.474	0.550
Charge-2	0.180	1.000	0.263	0.667
r^{-2}-3	0.583	1.000	0.795	0.925
r^{-2}-2	0.374	1.000	0.525	0.712
r^{-1}-3	0.877	1.000	0.743	0.719
r^{-1}-2	0.591	1.000	0.925	0.990
Spring-3	0.889	1.000	0.999	0.993
Spring-2	0.737	1.000	1.000	0.989

Table 3: Percent of total edge message variance across all components explained by the top d components, where d is the dimensionality of the problem.

5 Implementation

6 Results

7 Reproducibility

References

[1] Miles Cranmer. Interpretable machine learning for science with pysr and symbolicregression.jl, 2023.

[2] Leslie N. Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates, 2018.