Graph Neural Network-Based Symbolic Regression Using Deep Learning

Amber Li
amli@mit.edu

Samuel Kim
samkim@mit.edu

1. Introduction

We can train neural networks to be very good at certain tasks, but they generally fail at extrapolating far beyond training data and or providing us with interpretations of the data. Symbolic regression, on the other hand, is a technique for extracting meaning from data by producing models to explain and generalize beyond the training data.

The Equation Learner (EQL) architecture proposed in [1], [2] helps make neural network-based models more interpretable and better at extrapolation, particularly for symbolic regression tasks. Researchers have integrated the EQL network into other deep learning architectures and found that the system performs various learning and extrapolation tasks much better than a standard neural network-based architecture [3].

At the same time, researchers have also found symbolic models by applying symbolic regression to the internal components of graph neural network (GNN) models [4]. The intuition is that GNNs provide a strong inductive bias well-suited for data from interacting particle systems, which facilitates the discovery of analytical equations to describe such systems. This leads us to wonder if it is possible to build a neural network architecture that has the same capabilities, but which can be trained end-to-end to produce the same analytical equations without having to separately apply symbolic regression.

In the remaining sections, we describe the extension of the EQL network to learning the forces involved in interacting particle systems. We accomplish this by integrating an EQL network into the GNN architecture from [4], and we compare the resulting analytic expressions with those produced in the original work. Overall, the integrated architecture consistently produces the correct equations for the spring and $1/r$ forces in either 2 or 3 dimensions and with either 4 or 8 particles. Notably, it was able to produce a simpler, "unrotated" version of the force equations, which the original architecture was not able to do. Additionally, the new model was able to produce a mostly correct equation for the $1/r^2$ force in 2 dimensions with 4 particles.

2. Related Work

The EQL network learns functions in a supervised manner. The training data consists of inputs mapped to desired outputs, and the neural network contains specially-chosen activation functions, such as the identity, square, sin, and multiplication. An example with two hidden layers is shown in Figure 1, borrowed from [3], although the EQL may have more hidden layers and additional, different activation functions. In order for the EQL to produce interpretable results, sparsity must be enforced so that the model finds a simplest possible solution to fit the data. We enforce sparsity by adding regularization to the loss function as in [3].

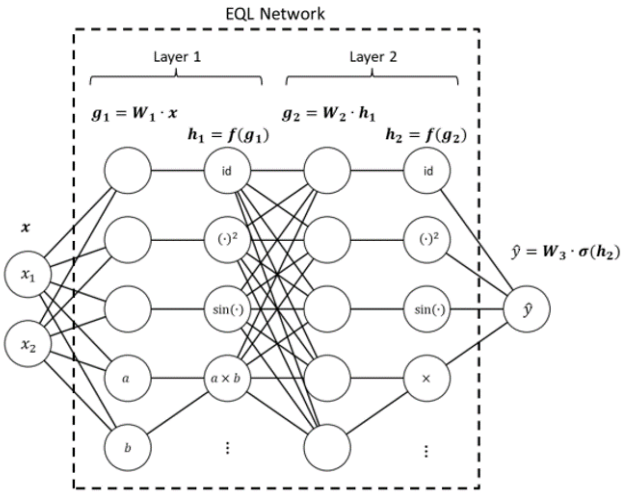

Figure 1: Equation Learner (EQL) Network Example

This project differs from the literature mentioned [4] in two key ways.

First, the architecture is modified. Figure 2 is borrowed from [4] and shows the structure of the original graph neural network used for extracting analytical relations from the training data. We're interested specifically in learning the force law between pairs of interacting particles, which is encoded in the graph network by the "edge model." The original edge model (or "message function") in [4] is a multilayer perceptron with two hidden layers and ReLU activa-

tions. In this work, we use the same overall GNN structure but replace the edge model with an EQL network with two hidden layers and our own activation functions.

This modified architecture creates our second key difference: since the EQL is easily integrated into other deep learning architectures such as the GNN, this means that the whole model trains end-to-end using backprop. After training, we do not need to perform extra symbolic regression steps using software as done in [4] but can simply extract the desired equations from the learned EQL weights.

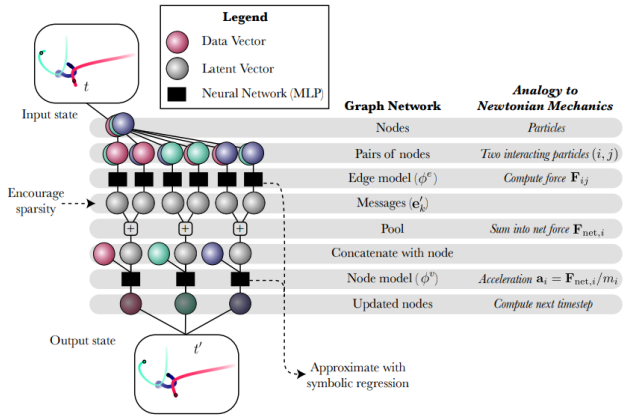

Figure 2: Graph Neural Network Architecture and Interpretation

3. Methods

The main contributions of this work were (1) creating the integrated GNN+EQL architecture and (2) producing successful experimental results for the spring force, $1/r$, and $1/r^2$ forces.

3.1. Symbolic Regression Pipeline

The steps needed to perform symbolic regression using the integrated GNN+EQL architecture are:

1. **Generate training data**
   Produce the accelerations of n bodies in d dimensions over 1000 time steps.

2. **Train the model**
   Specify the input variables to the message function. Save models and messages over time during training.

3. **Get the force equations**
   Choose the top d message features by largest standard deviation. Extract EQL weights, filter, and pretty print.

3.2. Architecture and Training Details

The activation functions used inside the EQL for all experiments were:

- 8 constant functions
- 16 identity functions
- 8 product functions

For the spring and $1/r$ experiments, the input variables to the EQL message function were $[dx, dy, r, 1/r, m_1, m_2]$. We give both r and $1/r$ as inputs due to the limitations of the EQL network in performing division. For the the $1/r^2$ experiments, the input variables were $[dx, dy, 1/r^3, m_1, m_2]$.

In all experiments, the EQL network within the message function of the GNN used 2 hidden layers and 100 output message features. Normally, for an experiment with particles interacting in d dimensions, we would expect the message function to output d meaningful features, but [4] found that using 100 message features and applying L1 regularization on the messages to constrain message dimensionality worked the best, so we took the same approach. The regularization weight used in all experiments, unless otherwise explicitly stated, was 1.0.

To get the final result of a symbolic regression task, the force equation is extracted from the trained model by sorting the 100 message features in order of decreasing variance and taking the top d features. The intuition is that these output features contain the most information and therefore are the most significant features.

For the spring and $1/r$ experiments, models were trained for 200 epochs. For the $1/r^2$ experiments, models were trained for 1000 epochs. All other hyperparameters and training settings were kept the same as in [4]. Specifically, we use batch gradient descent with an Adam optimizer. The learning rate is set according to the 1cycle learning rate policy with initial learning rate 0.001. See code implementation for more details. Other training schedule hyperparameter settings were tried, but they did not work as well.

3.3. Force Equations

Experiments used the spring, $1/r$, and $1/r^2$ forces. Below, we give the potential functions, U, used to generate the training data, while ϕ^e refers to the edge model of the GNN (see Figure 2), also known as the message function, representing the expected force equations. The variable r is the distance between two particles plus 0.01 to avoid singularities. The message functions given below are for particles in 2 dimensions, so the expected ϕ^e each have two message features.

3.3.1 Spring force

Potential:

$$U = (r-1)^2$$

Message function, unrotated (simplest form):

$$\phi^e = \begin{bmatrix} c_1(\Delta x)(1-1/r)+b_1 \\ c_2(\Delta y)(1-1/r)+b_2 \end{bmatrix}$$

Message function, rotated:

$$\phi^e = \begin{bmatrix} c_1(\Delta x)\cos\theta(1-1/r) - c_2(\Delta y)\sin\theta(1-1/r) + b_1 \\ c_1(\Delta y)\sin\theta(1-1/r) + c_2(\Delta y)\cos\theta(1-1/r) + b_2 \end{bmatrix}$$

3.3.2 $1/r$ Force

Potential:

$$U = m_1 m_2 \log r$$

Message function, unrotated:

$$\phi^e = \begin{bmatrix} c_1(\Delta x)\frac{m_1 m_2}{r^2} + b_1 \\ c_2(\Delta y)\frac{m_1 m_2}{r^2} + b_2 \end{bmatrix}$$

3.3.3 $1/r^2$ Force

Potential:

$$U = -\frac{m_1 m_2}{r}$$

Message function, unrotated:

$$\phi^e = \begin{bmatrix} c_1(\Delta x)\frac{m_1 m_2}{r^3} + b_1 \\ c_2(\Delta y)\frac{m_1 m_2}{r^3} + b_2 \end{bmatrix}$$

4. Results

The experiments and their results are organized by by number of particles n and dimensions d.

4.1. Spring Force

4.1.1 n=4, d=2

For reference, this is the rotated version of the expected message function again:

$$\phi^e = \begin{bmatrix} c_1(\Delta x)\cos\theta(1-1/r) - c_2(\Delta y)\sin\theta(1-1/r) + b_1 \\ c_1(\Delta y)\sin\theta(1-1/r) + c_2(\Delta y)\cos\theta(1-1/r) + b_2 \end{bmatrix}$$

Example result from the original GNN [4]:

$$\begin{bmatrix} 0.60(\Delta x)(1-1/r) + 1.36(\Delta y)(1-1/r) \\ ? \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Example result from the GNN+EQL:
Trial 1:

$$\begin{bmatrix} 0.02(\Delta x)(1-1/r) - 0.05(\Delta y)(1-1/r) \\ 0.05(\Delta x)(1-1/r) + 0.02(\Delta y)(1-1/r) \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Trial 2:

$$\begin{bmatrix} 0.05(\Delta x)(1-1/r) \\ -0.05(\Delta y)(1-1/r) \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

The results of this experimental setting were correct across all trials, with 1 out of 5 trials producing a simplest, unrotated version of the message function (as in the trial 2 example). This is an improvement over the model in [4], which was not able to produce an unrotated version of the spring force.

Note that all but $d = 2$ of the message features produce an expression of 0, showing that we have successfully enforced sparsity through regularization on the edge model.

4.1.2 n=8, d=2

For reference, this is the rotated version of the expected message function:

$$\phi^e = \begin{bmatrix} c_1(\Delta x)\cos\theta(1-1/r) - c_2(\Delta y)\sin\theta(1-1/r) + b_1 \\ c_1(\Delta y)\sin\theta(1-1/r) + c_2(\Delta y)\cos\theta(1-1/r) + b_2 \end{bmatrix}$$

No example result from the original GNN was given in [4].

Example result from the integrated GNN+EQL:

$$\begin{bmatrix} -0.06(\Delta x)(1-1/r) + 0.04(\Delta y)(1-1/r) \\ 0.04(\Delta x)(1-1/r) + 0.06(\Delta y)(1-1/r) \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

The results for this experiment were also consistently correct, although the GNN+EQL model was not able to produce an unrotated equation.

4.1.3 n=4, d=3

This is the unrotated form of the expected message function:

$$\begin{bmatrix} c_1(\Delta x)(1-1/r) + b_1 \\ c_2(\Delta y)(1-1/r) + b_2 \\ c_3(\Delta z)(1-1/r) + b_3 \end{bmatrix}$$

Example results from the GNN+EQL:
Trial 1:

$$\begin{bmatrix} -0.01(\Delta x)(1-1/r) + 0.03(\Delta y)(1-1/r) \\ 0.05(\Delta x)(1-1/r) - 0.02(\Delta z)(1-1/r) \\ 0.05(\Delta y)(1-1/r) + 0.03(\Delta z)(1-1/r) \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Trial 2:

$$\begin{bmatrix} 0.05(\Delta x)(1-1/r)+0.03(\Delta y)(1-1/r) \\ -0.03(\Delta x)(1-1/r)+0.05(\Delta y)(1-1/r) \\ 0.05(\Delta z)(1-1/r) \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

The GNN+EQL model produced correct results consistently. The extracted equations took on different rotated forms, some simpler and other more complicated: for example, the "trial 2" example above is a rotation about the z-axis, while the "trial 1" example has some rotation about all three axes.

4.2. $1/r$ Force

4.2.1 n=4, d=2

Unrotated form of expected message function:

$$\begin{bmatrix} c_1(\Delta x)\frac{m_1 m_2}{r^2}+b_1 \\ c_2(\Delta y)\frac{m_1 m_2}{r^2}+b_2 \end{bmatrix}$$

Example result from the GNN+EQL:

$$\begin{bmatrix} -0.02(\Delta x)\frac{m_2}{r^2} \\ -0.02(\Delta y)\frac{m_2}{r^2} \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

The results from this experimental setting were consistently unrotated and in the same form as the example shown. However, note that the m_1 is missing from the expression.

4.2.2 n=8, d=2

Unrotated form of expected message function again, for reference:

$$\begin{bmatrix} c_1(\Delta x)\frac{m_1 m_2}{r^2}+b_1 \\ c_2(\Delta y)\frac{m_1 m_2}{r^2}+b_2 \end{bmatrix}$$

Example result from the GNN+EQL:

$$\begin{bmatrix} -0.02(\Delta x)\frac{m_2}{r^2} \\ -0.02(\Delta y)\frac{m_2}{r^2} \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Similar to the $n = 4, d = 2$ experiments, these results were consistently unrotated but also missing the m_1.

4.2.3 n=4, d=3

Unrotated form of expected message function:

$$\begin{bmatrix} c_1(\Delta x)\frac{m_1 m_2}{r^2}+b_1 \\ c_2(\Delta y)\frac{m_1 m_2}{r^2}+b_2 \\ c_3(\Delta z)\frac{m_1 m_2}{r^2}+b_3 \end{bmatrix}$$

Example result from the GNN+EQL:

$$\begin{bmatrix} 0.026(\Delta x)\frac{m_2}{r^2} \\ 0.026(\Delta y)\frac{m_2}{r^2} \\ -0.026(\Delta z)\frac{m_2}{r^2} \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Again, the results for these experiments were always unrotated but missing the m_1.

4.3. $1/r^2$ Force

4.3.1 n=4, d=2

For reference, here is the unrotated form of the expected message function again:

$$\phi^e = \begin{bmatrix} c_1(\Delta x)\frac{m_1 m_2}{r^3}+b_1 \\ c_2(\Delta y)\frac{m_1 m_2}{r^3}+b_2 \end{bmatrix}$$

Example results from the GNN+EQL:
Trial 1:

$$\begin{bmatrix} -0.06(\Delta x)\frac{m_2}{r^3} \\ -0.06(\Delta y)\frac{m_2}{r^3} \\ -0.02\frac{(\Delta x)^2}{r^9}-0.02\frac{(\Delta y)^2}{r^9}+0.01\frac{m_2}{r^3} \\ \vdots \\ 0 \end{bmatrix}$$

Trial 2:

$$\begin{bmatrix} 0.04(\Delta x)\frac{m_2}{r^3} \\ 0.04(\Delta y)\frac{m_2}{r^3}+0.01 \\ 0.02\frac{(\Delta y)^2}{r^9}-0.01\frac{m_2}{r^3} \\ \vdots \\ 0 \end{bmatrix}$$

The results from these experiments do not demonstrate perfect sparsity of messages as in the spring and $1/r$ force experiments. However, the most significant message features do provide the correct force equation, although m_1 is still missing.

4.4. Examples of Failed Results

To contrast with the successful results shown above, we provide a few examples of failed results.

When the regularization weight is too small, the message features produced are not sparse in the way we would expect, and the expressions have too many terms. One setting where this happened is an experiment using the $1/r$ force with regularization weight 0.01:

$$\begin{bmatrix} 0.04(\Delta x)r - 0.41\frac{\Delta x}{r} - 0.14\frac{\Delta x}{r^2} + 0.02(\Delta y) + \dots \\ 0.02(\Delta x)r + 0.2(\Delta x) - 0.22\frac{\Delta x}{r} + 0.02\frac{\Delta x}{r^2} + \dots \\ \vdots \end{bmatrix}$$

On the other hand, when the regularization weight is set too large, the resulting message features all get squeezed to 0 and carry no information. A setting where this happened is an experiment using the $1/r^2$ force with regularization weight 10. The message function produced was simply:

$$\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

To demonstrate the difficulty of recovering the $1/r^2$ force, here is an incorrect result from one of the trials run with the same parameters that produced the correct result in a different trial (see Section 4.3.1):

$$\begin{bmatrix} -0.05(\Delta x)\frac{m_2}{r^3} \\ -0.01(\Delta x)\frac{m_2}{r^3} - 0.06(\Delta y)\frac{m_2}{r^3} + 0.01 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

To improve the results for the $1/r^2$ force, we also tried using 3 hidden layers and adding skip connections to the EQL, which we discuss in the next section (5). However, this did not succeed. Here are examples of incorrect message features produced using regularization weight 0.1 and no skip connections in the last hidden layer:

Trial 1:

$$\begin{bmatrix} 0.014 - \frac{0.05}{r^3} \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Trial 2:

$$\begin{bmatrix} -0.02(\Delta x)m_2 + 0.031 - \frac{0.09}{r^3} \\ -0.03(\Delta x)m_2 + 0.01 \\ 0.02(\Delta x)m_2 \\ 0.012 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

5. Discussion

To summarize, the integrated GNN+EQL model produced the correct force equations for the spring, $1/r$, and $1/r^2$ forces. However, the equations for the $1/r$ and $1/r^2$ forces were missing the m_1 term and the $1/r^2$ message features were not perfectly sparse in those aside from the d most significant output features.

It makes sense that the model was most successful on experiments with the spring force because its expected message function has the simplest composition of variables, i.e. there is only a product of two input variables (like $\Delta x \cdot \frac{1}{r}$), while the expected message function for the $1/r$ force contains a product of five input variables (as in $\Delta x \cdot m_1 \cdot m_2 \cdot \frac{1}{r} \cdot \frac{1}{r}$). The underlying reason for this is that the complexity of functions that the EQL network is able to represent is limited by the number of hidden layers and the choice of activation functions. As detailed in Section 3.2, the EQL within the integrated model had 2 hidden layers for all experiments. This combined with the choice of activation functions means that the largest composition of variables that the EQL could possibly produce is a product of four input variables. This may explain why the m_1 term is missing from the extracted equations for the $1/r$ force.

As for the $1/r^2$ experiments, the correct message function requires the EQL to produce a product of exactly four input variables: $\Delta x \cdot m_1 \cdot m_2 \cdot \frac{1}{r^3}$, which proved to be difficult. This may also explain the absence of the m_1 term from the extracted equations.

To expand the expressiveness of the EQL network, we can increase the number of hidden layers from 2 to 3. However, this makes the EQL much harder to train, and it becomes trickier to enforce sparsity. Experiments conducted using 3 hidden layers were not successful. We also explored adding skip connections between the hidden layers to encourage the model to find simpler functions, but it did not produce the correct equations.

6. Conclusions

This work demonstrated an extension of the Equation Learner (EQL) network to the problem of predicting particle interactions by integrating the EQL with a Graph Neural Network (GNN) deep learning architecture. The resulting model successfully and consistently produced the correct force equation in experiments with the spring force, and nearly correct equations in experiments with the $1/r$ and $1/r^2$ forces. Notably, the integrated GNN+EQL model is an improvement over existing work because it can be trained end-to-end and does not require separate symbolic regression, and it is able to produce a simplest, unrotated form of the force equation for particles interacting under a spring force. A limitation of the current model is that the EQL network within only contains 2 hidden layers, which constrains

the complexity of the functions it can produce; although an EQL network with 3 hidden layers can produce more complex functions, that model is much more difficult to train.

Future work may further investigate skip connections and other ways to increase the expressiveness of the EQL network so that it can more easily represent functions of greater complexity. We have also received a suggestion to try variations of this GNN+EQL model architecture that take greater advantage of the graph structure, i.e. by introducing asymmetry into the graph rather than having a fully connected graph. Finally, there is more potential for the EQL network to be integrated into other deep learning architectures, such as the Siamese Neural Network.

References

[1] G. Martius and C. H. Lampert, "Extrapolation and learning equations," oct 2016.
http://arxiv.org/abs/1610.02995 1

[2] S. S. Sahoo, C. H. Lampert, and G. Martius, "Learning Equations for Extrapolation and Control," jun 2018.
https://arxiv.org/abs/1806.07259 1

[3] S. Kim, P. Y. Lu, S. Mukherjee, M. Gilbert, L. Jing, V. Čeperić, and M. Soljačić, "Integration of Neural Network-Based Symbolic Regression in Deep Learning for Scientific Discovery," dec 2019.
https://arxiv.org/abs/1912.04825 1

[4] M. Cranmer, A. Sanchez-Gonzalez, P. Battaglia, R. Xu, K. Cranmer, D. Spergel and S. Ho, "Discovering Symbolic Models from Deep Learning with Inductive Biases," jun 2020.
https://arxiv.org/abs/2006.11287 1, 2, 3