Symbolic Regression Using a Generative Transformer Model

Anonymous Author(s)
Affiliation
Address
email

Abstract

Symbolic regression is the task of identifying a mathematical expression that best fits a provided dataset of input and output values. Due to the richness of the space of mathematical expressions, symbolic regression is generally a challenging problem. While conventional approaches based on genetic evolution algorithms have been used for decades, deep learning-based methods are relatively new and an active research area. In this work, we present a novel transformer-based language model for symbolic regression. This model exploits the strength and other possible flexibilities that have been provided by probabilistic language models like GPT. Through comprehensive experiments, we show that our model is not only scalable but also comparable in terms of performance with other models.

1 Introduction

Deep learning and neural networks have earned an esteemed reputation for being capable tools for solving a wide variety of problems over countless application domains. Notably, deep language models have made an enormous impact in the field of linguistics and natural language processing. With the advances in technology like GPT, the scope of problems now accessible to automated methods continues to grow. It is particularly interesting when language models are used for tasks that, at first glance, do not seem to have any relationship with language at all.

Symbolic regression, the problem of finding a mathematical equation to fit a set of data, is one such task. This objective of symbolic regression is to obtain a closed-form symbolic mathematical expression to describe the relationship between specified predictor and response variables, where the mathematical expression is allowed to be flexible without being restricted to a particular structure or family. More precisely, the goal in symbolic regression is to recover a mathematical function f in terms of the input variables $\boldsymbol{x} = [x_1 \dots x_d]^T$, given a set of datapoint vectors of the form $D = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, such that $\mathbb{E}(y_i|\boldsymbol{x}_i) = f(\boldsymbol{x}_i)$ for all i. Here, $x_1, \dots, x_d, y_i$ are scalars and $\boldsymbol{x}_i \in \mathbb{R}^d$.

By not imposing any structural constraints on the shape of the desired equation, symbolic regression is a much more difficult task compared to other kinds of regression, such as linear regression or multinomial regression, as the search space of candidate expressions is so much larger.

The most common approach for symbolic regression is based on genetic programming, where numerous candidate parse trees are generated, evaluated, combined, and mutated in an evolutionary way until a tree is produced that models an expression that fits the dataset up to a required accuracy level. In essence, it is a search strategy over the vast space of mathematical expressions, seeking the formula that would optimize an objective function.

In this typical framework, which applies not only to genetic methods but also many deep-learning based approaches to symbolic regression, the goal is to identify a mathematical expression that fits

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

most optimally given a single input dataset. This dataset is the basis over which all the training occurs. Consequently, when presented with any new dataset (as a fresh instance of the task of symbolic regression), the entire training procedure must begin again from scratch.

In this work, we explore an alternative approach to symbolic regression by considering it as a task in language modelling. Symbolic mathematics behaves as a language in its own right, with well-formed mathematical expressions treated as valid "sentences" in this language. It is natural, therefore, to consider using deep language models to address tasks involving symbolic mathematics.

We can frame the regression problem as an exercise in captioning, where we are taking input in the form of a cloud of points in $\mathbb{R}^{d+1}$ and returning a statement in the language of symbolic mathematics to describe the point set. By training a model to correctly "caption" datasets with the equations underlying them, we will have a system for performing symbolic regression quickly and accurately.

In this work, we present a method that operates based on this idea. Our method makes use of deep language models in a framework that represents a major shift in the way symbolic regression is performed. We move the task of symbolic regression from being a strictly quantitative problem into a language one. Effectively, we propose a system that not only learns the language of symbolic mathematics, but also the underlying relationship between point clouds and mathematical expressions that define them.

As part of our method, we make use of a T-net model [15] to represent the input point cloud in an order-invariant way. This allows us to obtain vector embeddings of the entire input dataset for symbolic regression instances without depending on the number of points in the dataset or the order in which they are given.

A major advantage of our approach is that we are no longer training a model to learn an equation for an individual dataset in each instance of symbolic regression. Instead, we train a single language model one time, and use that trained model to rapidly solve instances of symbolic regression as individual captioning tasks. We will show that our method, SymbolicGPT, not only presents a running time speed boost of an order of magnitude or more, but also provides competent performance in accurately reconstructing mathematical equations to fit numerical datasets, presenting a new frontier for language models and a novel direction for approaching symbolic regression.

2 Related Work

Symbolic regression has conventionally been approached using methods based on genetic algorithms [11, 1, 19, 12, 21]. In this framework, the task is seen as a search space optimization problem where symbolic expressions are candidates and the expression with the greatest fitness, or fitting accuracy on the training data, is obtained through a process of mutation and evolution. Although this approach has shown success in practice, it is computationally expensive, highly randomized, requires instance-based training, and struggles when learning equations that contain many variables and values of constants.

More recently, newer approaches to symbolic regression have arisen that make use of neural networks. The EQL (Equation Learner) model [10, 18] is an example of performing symbolic regression by training a neural network that represents a symbolic expression. This method, and others based on it [5, 7], take advantage of advances in deep learning as an alternative to genetic approaches. However, they still approach symbolic regression as an instance-based problem, training a model from scratch given every new input dataset for a regression task.

A recent study [2] presents a novel, language-based method for handling symbolic regression as a machine translation task, similar to the approach used by [9] for performing symbolic integration and solving differential equations. Given an input dataset, the algorithm treats the input as a text string and passes it through a trained sequence-to-sequence LSTM to produce an output text string that is parsed as the desired symbolic expression. Although this method overcomes the cost of per-instance training, its interpretation of the input dataset as a textual string limits its usability, as the input data must follow specific constraints, such as fitting a 1-dimensional mesh of fixed size. Consequently, this method can only be used only in one-dimensional space. However, in most problems, more than one variable is involved and we need to find a multivariate function. In this work, we propose a method that removes such limitations on the structure of input data. This can be applied easily to symbolic regression problems in high-dimensional spaces and when many variables are involved.

Another active area of research is to use deep reinforcement learning methods to tackle this problem [7, 14]. The method proposed in [14] uses a hybrid approach between traditional genetic algorithms and deep learning methods. Here, the authors use deep RNNs to generate samples of candidate equations. Numerical optimization is then used to optimize for the constants of each candidate equations. A reinforcement learning algorithm is applied to train the RNN to generate better candidate equations at every iteration. However, this method still relies on the iterative nature of traditional genetic algorithms as well as the numerical optimization. This results in a computationally intensive process in order to generate predictions.

3 Method

Our model for symbolic regression, SymbolicGPT, consists of three main stages: obtaining an order-invariant embedding of the input dataset using a T-net [15], obtaining a skeleton equation using a GPT language model [17], and optimizing constant values to fill in the equation skeleton. In addition to discussing each of these steps, we also present the method for generating our training set, which we will introduce first.

3.1 Equation generation

To train our language model, we need a large dataset of solved instances of symbolic regression. This dataset is a collection of input-label pairs where each input is in the form of a numerical dataset, itself a set of input and output pairs $\{(\boldsymbol{x}, y)\}$, and the corresponding label is a string encoding the symbolic expression governing the relationship between variables in the numerical dataset.

In order to ensure that the language model is able generalize to unseen equations, having good training data is key. It is necessary to train the model over a wide, diverse set of training equations in order to prevent overfitting of the language model.

There are a number of different ways to randomly sample symbolic mathematical expressions. One approach, as used in [3], is to consider symbolic expressions as constructed by rules in a context-free grammar, and randomly sampling from rules until reaching a string containing only terminal values. Another approach, taken in [9], uses parse-tree representations of symbolic formulas, presenting a method that samples uniformly from all trees of n nodes and then filling in nodes with valid operator or variable values.

For our training dataset, we use an approach similar to the latter, where we start with a blank parse tree and then "decorate" the nodes with choices of operators and variables. In contrast with [9], we do not constrain our parse trees by number of nodes, but by number of levels. This enables more control over the maximum level of complexity in the equations used in our training set, as the number of levels in the parse tree correspond to the number of potential function nestings, a measure of how complex an equation can be.

We begin by fixing k, the maximum number of levels in the parse tree for the equations we wish to encounter in our training set. We also begin with a pre-specified number of variables, d, and a pre-selected set of operators, $P = \{u_1, \ldots, u_m\}$, that are allowed to appear in any training equation. Then, for each data-equation pair in our training set, we generate a perfectly balanced binary tree of depth k, having $2^{k-1} - 1$ internal nodes and 2^{k-1} leaf nodes. These nodes originally start off empty to form the template of a symbolic expression.

The template is filled in by randomly selecting valid choices to occupy each node in the parse tree. For leaf nodes, each node is randomly assigned with a variable from the set $\{x_1, \ldots, x_d\}$. For interior nodes, operators from the set P are randomly chosen. Once filled in, the parse tree can naturally be interpreted as a symbolic expression. For nodes filled in by binary operators, both of their child nodes are used as input; in the case of unary operators, only the left child is used as input, and the right child is ignored. Importantly, the unary operator "id($\cdot$)", which returns its input argument unchanged, is included in P, which effectively allows for equations with shallower or unbalanced parse trees to still be represented using this template.

In order to prefer simpler equations with fewer transcendental functions, operators such as id($\cdot$) and mult($\cdot, \cdot$) are given higher weighting in the probability distribution for selecting operators than choices like sin($\cdot$), exp($\cdot$), and log($\cdot$).

As a final step for the equation generation procedure, constants are incorporated into the equation by inserting them at nodes in the parse tree. Given a specified value $r \in [0, 1]$ and constant bounds c_{min} and c_{max}, for each node in the tree, a random real-valued constant is selected between c_{min} and c_{max} and, with probability r, is inserted as a multiplicative factor the subtree rooted at that node. Similarly, a second random constant is selected between c_{min} and c_{max} and, with probability r, is inserted as an additive bias to the subtree rooted at that node. By varying the constant ratio r, the equations can be customized to include many constants, few constants, or none at all.

Once an equation is generated, an input dataset for symbolic regression can be produced by evaluating the symbolic expression at n different vectors $\boldsymbol{x}$ randomly sampled from some region of interest in $\mathbb{R}^d$. The label value for the symbolic regression instance would be the symbolic expression. This process can be repeated many times to construct the training set by which our SymbolicGPT model will learn how to perform symbolic regression.

3.2 Order-invariant embedding

Once the training set of input data and output equations is generated, it is used to train our model for converting numerical datasets into equation strings.

The first step is to convert the input dataset $D = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n \subset \mathbb{R}^{d+1}$ into a single vector embedding $\boldsymbol{w}_D \in \mathbb{R}^e$. For the conversion to be useful, it must have two properties. First, it should not strictly depend on the number of points in the input dataset, n. In practice, the datasets provided as input to a symbolic regression solver may have varying sizes, and we do not want our method to be restricted to cases with a fixed number of input points.

Second, the conversion method should not be sensitive to order in which the points of the dataset are given. The input to a symbolic regression instance is a collection of datapoints, rather than a sequence, and the optimal symbolic expression to fit the dataset should not depend on the order in which the points are listed. Thus, the vector embedding of the dataset should be similarly order-invariant.

Our approach for converting the dataset D into a vector embedding is to use a T-net, a kind of deep network that makes use of a global max pooling layer to provide order-invariance over its arbitrarily-sized input [15]. The T-net takes as input the dataset D, consisting of n datapoints over d variables, represented in matrix format as $X \in \mathbb{R}^{n\times(d+1)}$, where n can be any number and d is a fixed hyperparameter of the model. Any symbolic regression instance with fewer than d variables can be padded with 0 values, bringing the total number of variables up to d.

The matrix X is first normalized using a learnable normalization layer in order to regulate extreme values from the input. The normalized input points are then passed through three stages of MLP networks. Within each stage, each of the n rows of X are passed individually, albeit in parallel, through a single fully connected layer, where weights are shared between the networks for all points for that stage. The first stage results in n points encoded in e-dimensional space; the second stage takes them into $2e$ dimensions, and the output at after the third stage are n points having $4e$ dimensions each.

The next layer in the T-net is a global max pool, which reduces the $n \times 4e$ output of the previous step down to a $1 \times 4e$-dimensional vector. The max pooling eliminates the dependence on both n and the order of the input points, achieving both goals needed for our vector embedding. Finally, the output of the global max pool is passed through two more fully connected layers, resulting in a single output vector $\boldsymbol{w}_D$, an e-dimensional embedding of the input dataset. The overall structure of the T-net is shown in the left part of Figure 1.

3.3 Generative model architecture

The main component of SymbolicGPT is the deep network for producing symbolic equations, as implemented using a GPT language model [16, 17, 4]. This framework takes in two pieces of input: the order-invariant embedding of the point cloud $\boldsymbol{w}_D$ as produced by the T-net, representing the input dataset, and a sequence of tokens, $X_{(eq)}$, used to initialize the output formula string. In the typical regression case where no information is provided about the output symbolic expression in advance, this token sequence would be the singleton Start-of-Sequence token $\langle SOS \rangle$, although in general it can be any desired prefix of the output equation. The input token sequence is tokenized at a character level and encoded as the matrix W_t using a trainable embedding as part of the GPT model.

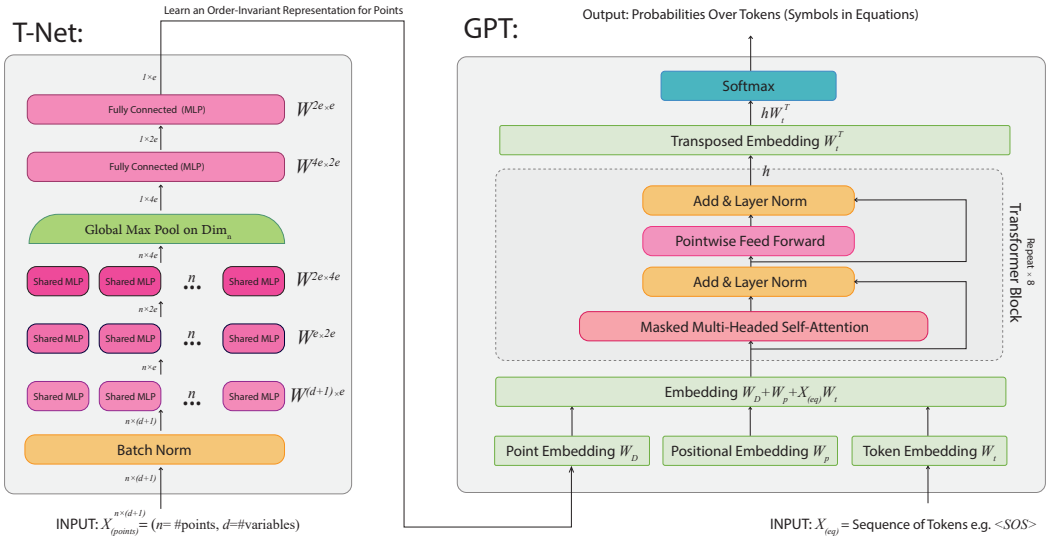

Figure 1: The architecture of SymbolicGPT. The left box illustrates the structure of our order-invariant T-net for obtaining a vector representation of the input dataset, and the right box shows the structure of the GPT language model for producing symbolic equation skeletons.

The first step in the GPT model is to combine the two inputs $\boldsymbol{w}_D$ and W_t together, along with the positional embedding vector W_p. Based on empirical support, we chose to obtain the combined embedding by taking the sum $W_p + W_D + X_{eq}W_t$, where W_D is the dataset representation vector $\boldsymbol{w}_D$ expanded to fit a matrix matching the dimensions of the other embeddings.

The combined vector is then passed through $l = 8$ successive transformer blocks, using the standard format of GPT models [17]. Each transformer is a sequential combination of a 1-hidden-layer MLP block and a self-attention block, with all blocks feeding into a central residual pathway, similar to ResNets [6].

After l layers of the transformer block, the resulting output vector h is passed through a final decoder in the form of a linear projection into a vanilla softmax classifier. The projection uses the transposed token embedding matrix W_t^T to map the hidden state vector back into the space of tokens for symbolic expressions. The result of the softmax is a probability vector over tokens in the symbolic equation, which can be used for a beam search to produce the equation to describe the input dataset.

Although the symbolic equation used to generate the data can contain constant values, we do not train the GPT model to recover these values exactly. Instead, constant values in the equation are masked by $\langle C \rangle$ tokens during the training phase, and the output of the GPT model is a "skeleton equation" which leaves these constant placeholders in the output string. This is because it is unnecessary to burden the language model with the additional task of learning precise constant values, as this can be easily handled as a separate step.

3.4 Learning constants

Once the GPT model generates a skeleton equation, we learn values of constants to decorate the skeleton as a post-processing step. This division of tasks is a common approach for string-based regression methods [8, 2].

To learning the values of constants in the symbolic equation, we employ BFGS optimization, similar to [2], using an implementation from SciPy [20]. The learned constant values then replace the $\langle C \rangle$ placeholder tokens in the skeleton equation, resulting in the final symbolic expression to represent the given dataset.

3.5 Training Metric

Normally, regression tasks use the mean square error as a metric for measuring predictive accuracy of an equation. For data following equations with large values, however, this can be problematic, as the residuals can grow very large even when the predicted equation is very close to the true underlying one. To resolve this issue, we chose to use a hybrid metric that uses relative mean square error whenever residuals are large. Let $\hat{y}_i = \hat{f}(\boldsymbol{x}_i)$ and $y = f(\boldsymbol{x}_i)$ where $\hat{f}$ is the predicted equation. First define

$$h(y, \hat{y}) = \begin{cases} (y - \hat{y})^2 & y < 1 \\ \left(\frac{y-\hat{y}}{y}\right)^2 & \text{otherwise} \end{cases}$$

Then the relative mean square error, MSE_R, is given by

$$MSE_R(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{n} \sum_{i=1}^{n} h(y_i, \hat{y}_i).$$

3.6 Strengths and Advantages

Our method exhibits the following strengths and advantages.

3.6.1 One-time training

In contrast with most approaches for symbolic regression, our method does not start training from scratch given every new problem instance. All of the model training is performed as a one-time procedure that takes place before the GPT transformer is ever used. After the model is trained, every instance of symbolic regression can be solved rapidly as a problem in inference, with running time dependent only on the step of reading in the input dataset and obtaining the PointNet embedding.

3.6.2 GPT technology

Our approach to symbolic regression is based on a probabilistic language model as implemented by GPT. As state-of-the-art language models continue to evolve, our method is expected to organically improve accordingly, with no extra effort in design or implementation, by simply replacing the GPT architecture with any newer and more powerful alternative.

3.6.3 Scalability

Our approach addresses two of the main problems with traditional methods. First, our model is able to scale to multiple variables. Iterative methods that choose the best candidate equation at each iteration struggle as the dimension d of the inputs increase since the search space of functions grow exponentially with respect to d. By passing in the data points directly as inputs, the model is able to infer the dimension and produce equations accordingly. Second, our model is able to scale in terms of the speed in which we generate predictions. Existing methods that train from scratch for each regression instance are slow compared to our model. Existing methods incrementally update their model based on computing many candidate equations, all of which need to be optimized in terms of the constants. This results in a bottleneck in terms of the number of constant optimizations that needs to be performed in order to generate a final prediction. SymbolicGPT only performs this constant optimization once which results in significantly faster inference times. We show empirically that SymbolicGPT produces comparable results using significantly less computation time in Section 4.

4 Experiments and Results

To test our model, we implemented SymbolicGPT and trained it in a number of different settings, which we detail below. In all cases, we trained SymbolicGPT over 4 epochs using a batch size of 128. The embedding size for the T-net vector representation is $e = 512$, and the maximum equation output length was capped at 100 tokens.

The SymbolicGPT model was trained and run using an Intel(R) Core(TM) i9-9900K CPU @ 3.60GHz with a single NVIDIA GeForce RTX 2080 11 GB GPU and using 32.0 GB Ram. It is noteworthy

Experiment	GP	DSR	SymbolicGPT
General	48.01 ± 26.72	78.79 ± 42.79	$\mathbf{5.06} \pm 12.00$
One variable	44.59 ± 32.95	15.07 ± 2.05	$\mathbf{1.06} \pm 0.93$
Two variables	47.29 ± 29.42	76.71 ± 39.75	$\mathbf{3.46} \pm 8.97$
Three variables	60.04 ± 32.91	73.34 ± 56.36	$\mathbf{10.30} \pm 26.21$

Table 1: Average running times (in seconds) for an instance of symbolic regression during each of the four experiments.

that our performance scores were achieved using only a single GPU, and scaling up is expected to improve training and running times even further.

Our experimental framework consists of a large-scale comparison test where we test our model on 1000 different, randomly generated instances of symbolic regression and evaluate performance based on MSE_R. We repeat this test on four different setting, based on choice of the dimension d: datasets with one input variable, two variables, three variables, and a random selection between one and five variables. This last setting will be referred to as the "general" experiment.

In each experimental setting, SymbolicGPT was trained using 10,000 randomly generated symbolic regression instances belonging to the associated dimensional configuration, each consisting of an input dataset and an equation label. A further 1000 dataset-equation pairs were generated and used for validation, and 1000 new dataset-equation pairs were generated for a test set. As in [2], training and validation datasets used values of $\boldsymbol{x}$ from $[0.0, 3.0]^d$, and test datasets took $\boldsymbol{x}$ from $[3.1, 6.0]^d$. In the one-variable setting, each dataset contained 30 points; over two variables, 200 points; over three variables, 500 points; and in the general setting, the number of points was randomly picked between 100 and 500.

The parse tree templates, as described in Section 3.1, contained a depth of $k = 2$ levels and allowable operators coming from the set [id$(\cdot)$, add$(\cdot, \cdot)$, mul$(\cdot, \cdot)$, sqrt$(\cdot)$, sin$(\cdot)$]. Constant values selected from the interval [-1, 1] were randomly inserted using a constant ratio $r = 0.5$.

We compared our methods with three existing models for nonlinear regression:

1. Deep Symbolic Regression (**DSR**): We use the method in [14] to represent the most recent developments in deep learning methods for symbolic regression.

2. Genetic Programming (**GP**): We chose to use Python's GPLearn package to represent genetic evolution algorithms for symbolic regression.

3. Neural Network (**MLP**): We use a standard Multilayer Perceptron to act as a non-symbolic, nonlinear regressor to use as a baseline for comparison, as implemented in the Python package Scikit-Learn [13].

For each method, we evaluated its performance on 1000 test instances of symbolic regression in each of the four experiment settings, using MSE_R as the fitness metric. We summarized the results in the cumulative distribution plots of Figure 2, showing the proportion of the test cases that attained error less than any given threshold value. Methods corresponding to curves positioned higher in the plot achieved higher accuracy on more test equations, and hence are better regressors. However, the most important region of the plot is the far left side, as the number of test cases that achieved the lowest possible error is an indication of how often the method would find a highly accurate fitting equation.

To give an indication of the speedy performance of SymbolicGPT, we measured and compared the average running time to solve an instance of symbolic regression in the general experiment setting. The mean running times, along with standard deviations, are shown in Table 1. We did not include the MLP regressor in order to make a fair comparison between strictly symbolic regression methods. The results show that SymbolicGPT requires significantly less time to solve an instance of symbolic regression compared with other methods, often by an order of magnitude or more, due to most of the computation being shifted to the offline step of setting up the pre-trained model.

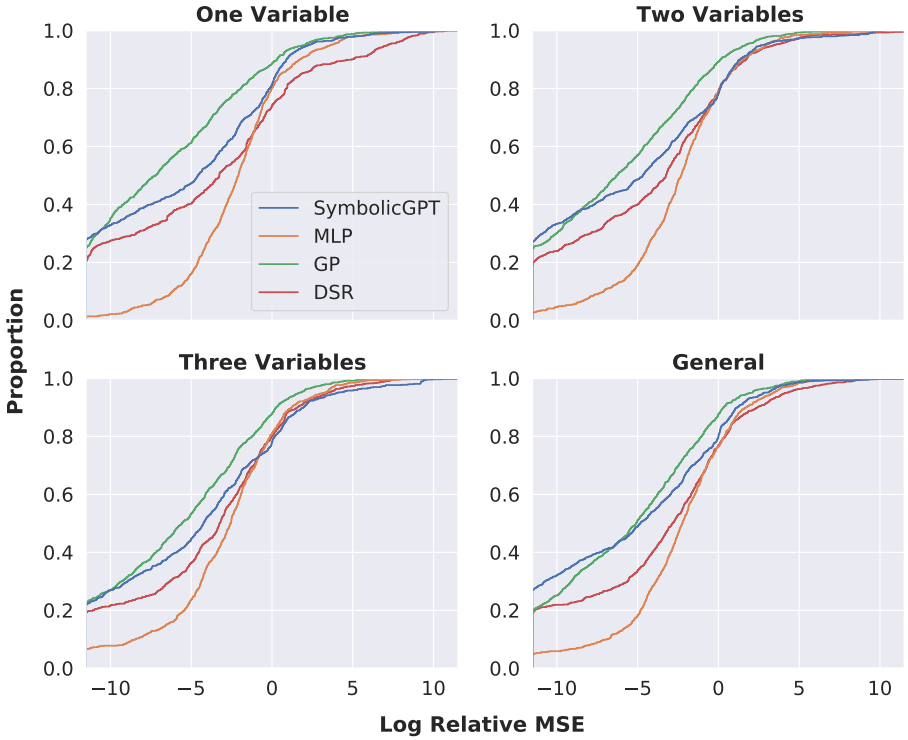

Figure 2: Cumulative $\log MSE_R$ over all methods and experiments. Each curve shows the proportion of test cases that attained an error score less than every given threshold. SymbolicGPT finds better fitting equations for more test cases than DSR and finds more highly accurate equations (with $\log MSE_R < -10$) than any other method tested.

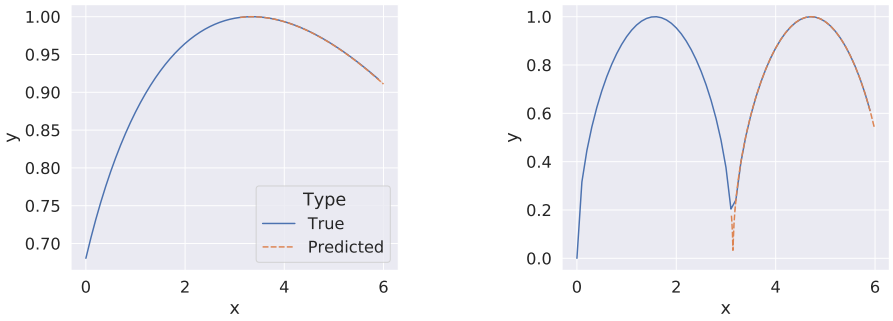

Figure 3: Selected predicted equations generated by SymbolicGPT, along with the true underlying curves.
Left: $y = \sin(\sqrt{0.79x})$. **Right:** $y = \sqrt{|\sin(x)|}$

.

5 Conclusions and Next Steps

In this work, we have presented a method that pushes the boundaries of language models and approaches the problem of symbolic regression from a new and powerful direction. We have employed language models in a novel way and with a novel approach, combining them with symbolic mathematics and order-invariant representations of point clouds. Our approach eliminates the per-instance computation expense of most regression methods, and resolves the input restrictions imposed by other language-based regression models. Moreover, our method is fast, scalable, and performs competently on several kinds of symbolic regression problems when compared with existing approaches.

In future work, we would like to explore the limits of SymbolicGPT to measure the extent of its scalability. Our experiments showed the competency of our model, but did not establish the capacity of the SymbolicGPT framework. There is also room to improve in situations where exact equation recovery is required, a problem whose current bottleneck is the constant optimization step in our algorithm.

References

[1] Douglas Adriano Augusto and Helio JC Barbosa. Symbolic regression via genetic programming. In *Neural Networks, 2000. Proceedings. Sixth Brazilian Symposium on*, pages 173–178. IEEE, 2000.

[2] Luca Biggio, Tommaso Bendinelli, Aurelien Lucchi, and Giambattista Parascandolo. A seq2seq approach to symbolic regression.

[3] Jure Brence, Ljupčo Todorovski, and Sašo Džeroski. Probabilistic grammars for equation discovery. *Knowledge-Based Systems*, page 107077, 2021.

[4] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[5] Gang Chen. Learning symbolic expressions via gumbel-max equation learner network. *arXiv preprint arXiv:2012.06921*, 2020.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] Samuel Kim, Peter Y Lu, Srijon Mukherjee, Michael Gilbert, Li Jing, Vladimir Čeperić, and Marin Soljačić. Integration of neural network-based symbolic regression in deep learning for scientific discovery. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[8] Michael Kommenda, Bogdan Burlacu, Gabriel Kronberger, and Michael Affenzeller. Parameter identification for symbolic regression using nonlinear least squares. *Genetic Programming and Evolvable Machines*, pages 1–31, 2019.

[9] Guillaume Lample and François Charton. Deep learning for symbolic mathematics. *International Conference on Learning Representations*, 2020.

[10] Georg S Martius and Christoph Lampert. Extrapolation and learning equations. In *5th International Conference on Learning Representations, ICLR 2017-Workshop Track Proceedings*, 2017.

[11] Ben McKay, Mark J Willis, and Geoffrey W Barton. Using a tree structured genetic algorithm to perform symbolic regression. In *Genetic Algorithms in Engineering Systems: Innovations and Applications, 1995. GALESIA. First International Conference on (Conf. Publ. No. 414)*, pages 487–492. IET, 1995.

[12] A Murari, E Peluso, M Gelfusa, I Lupelli, M Lungaroni, and P Gaudio. Symbolic regression via genetic programming for data driven derivation of confinement scaling laws without any assumption on their mathematical form. *Plasma Physics and Controlled Fusion*, 57(1):014008, 2014.

[13] 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.

[14] Brenden K Petersen, Mikel Landajuela Larma, Terrell N. Mundhenk, Claudio Prata Santiago, Soo Kyung Kim, and Joanne Taery Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *International Conference on Learning Representations*, 2021.

[15] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017.

[16] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[17] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[18] Subham Sahoo, Christoph Lampert, and Georg Martius. Learning equations for extrapolation and control. In *International Conference on Machine Learning*, pages 4442–4450. PMLR, 2018.

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

[20] 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, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, 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.

[21] Yiqun Wang, Nicholas Wagner, and James M Rondinelli. Symbolic regression in materials science. *MRS Communications*, 9(3):793–805, 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Please check section
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? Yes, supplemental materials and after publication, the github repo. [Yes] In the supplemental materials, you'll find all the codes, data, models, and results.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Yes, all configuration files are available for reproducing the complete results. In the case of other methods, we used default values.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes]

(b) Did you mention the license of the assets? [Yes]

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]