Deep Learning for Symbolic Regression

Abstract

Symbolic regression is a task that aims to discover the underlying equations from a given sample of data or observation. Due to the richness of the space of mathematical expressions, symbolic regression is generally a challenging problem. Current approaches, including genetic programming and deep neural method, usually follow a two-step procedure: first predicting the symbols of function, then approximating the constants. We modify these conventional methods by proposing an end-to-end transformer-based language model to directly predict the complete mathematical expressions. The proposed model exploits the advantages of a large-scale model and global training strategy. Through comprehensive experiments, we show that our model performs robustly, eventually reaching around 78% accuracy.

1. Introduction

In recent years, neural network has achieved state-of-art performance in various fields in computer vision, speech recognition, natural language processing, etc. However, only a few studies investigated the capacity and application of neural networks to deal with symbolic regression problems.

Symbolic regression (i.e., function identification) is a central problem in natural science, which aims to find a model, in symbolic form, that fits a given sample of data or observations. To be more specific, for a given finite sampling of value pairs of the independent variables and the associated dependent variables, the goal of symbolic regression is to find a mathematics expression, involving both the functional form and the numerical coefficients that can provide a good fit.

Genetic programming (GP) is currently the main conventional algorithm for symbolic regression, which was introduced based on the Darwinian evolutionary process [1]. It should be noted that, genetic algorithm, while achieving reasonable prediction accuracy, cannot improve with experience because the solutions to each problem needs to be learned from scratch.

In recent years, neural network has shown to have wide applications fields in computer vision, speech recognition, natural language processing, etc. However, in symbolic regression, these deep learning-based methods are relatively new and are developing into an active research area. For example, transformer-based large-scale methods have been proposed in [2,3]. These methods Inherited from GP follow a two-step procedure, first predicting the symbolic expressions using a neural network, and then fitting constants through by non-linear optimizer such as BFGS. We argue that there are some shortcomings in the two-step procedure. Firstly, it may not be suited to neural network approaches in some ways, as the neural network itself is a good non-linear optimizer. In addition, merely optimizing on functional form cannot provide sufficient supervision, for example, different instances of the same symbolic mathematical form may results in different shapes of mathematical curves. Therefore, we propose an end-to-end numerical method and symbolic prediction to implement the complete mathematical equation.

In this report, we train a transformer model over a synthetic dataset to achieve end-to-end symbolic regression. The base 10 positional encoding (p10) method is used to represent numbers as a sequence of five tokens so that they can be processed by the model. After 2 days training on a single GeForce RTX 2080 GPU, the models is shown to achieve an accuracy of 78%.

2.Related work

2.1 Genetic Programming for Symbolic Regression

The traditional approach to symbolic regression is Genetic Programming (GP) based on Genetic Algorithms [1]. In each generation, a set of candidate mathematical expressions are generated and evaluated from the data, and the fittest ones are selected for mutation and recombination to construct the next generation. This procedure is iterated until a satisfactory level of accuracy is achieved. While GP is the first algorithm to demonstrate the potential of symbolic regression for data-driven solutions, it is notoriously difficult to scale well to high-dimensional problems and exhibits high sensitivity to hyperparameters [4].

2.2 Deep learning for symbolic regression.

Several recent approaches leverage deep learning for symbolic regression. Udrescu and Tegmark developed AI Feynman at 2020 [5], which first used the physics-inspired strategies enabled by neural networks to discover hidden simplicity such as symmetry or separability in the dataset, and then recursively break harder problems into simpler ones with fewer variables. In grammar VAE, Kusner et al. [6] proposed a generative model to learn latent representations for discrete data such as arithmetic expressions. They showed that this latent representation can be used for symbolic regression. Biggio [2] introduced a large-scale pre-training method that applies a transformer through a two-step process to solve the symbolic regression problem. Once the model is pretrained, the skeleton is predicted with a simple forward pass and BFGS is called to approximate the constant.

2.3 Transformers for symbolic mathematics.

Transformers was originally designed for machine translation [15], thus it quickly shows great capacity in symbolic manipulation, which can be manipulated as words in natural language. As for now, the transformer has achieved great performance on symbolic integration [7], theorem proving [8], formal logic [9], SAT solving [10], and dynamical systems [11]. All these problems rely on the advantages of pretraining models on large datasets [12, 13, 14]. Our work is inspired by the work of Lample & Charton (2019) [7], Transformers are trained to solve complex mathematical problems, namely functional integration and ordinary differential equations (ODEs). However, in this report, we present the additional challenge of numerical computation rather than limited symbolic manipulation.

3. Generating Datasets

A vast synthetic dataset is a key part of training a pre-training language model. In our approach, the training example is a pair of datapoints and function such as . We first generate a random function , then sample a set of N input values and compute .

3.1 generating functions

We sample expressions following the framework introduced by Lample and Charton [7]. A mathematical expression is regarded as a tree where internal nodes are unary and binary operators and leaves are variables and constants. We list the operators, variables, and constants used in our experiment in table 1 and show the examples of generated functions in Figure 1. Note that to simplify the form of expressions, we limit the number operators to 2

|  |  |
| --- | --- |
| Operators |  |
| constants |  |
| variables |  |

Table 1. operators, constants, and variables in our experiments

Text, letter

Description automatically generated

Figure 1. examples of generated functions

3.2 generating datapoints

For each equation , we randomly sample input values, and then compute the corresponding output values using SymPy [17], which is a python library for symbolic mathematical computation. In our experiment, the value of is set to 512. In order to avoid invalid operations, we drop out input-output pairs containing NaNs.

3.3 Datapoint Preprocessing

To process the numerical values by transformers in the equations, it is necessary to convert them into tokens. In the experiment, we use the base 10 positional encoding (P10) method to represent a number as a sequence of five tokens: one sign token (+ or -), 3 digits (from 0 to 9) for the mantissa, and a symbolic exponent (from E-100 to E+100). For example, the expression is represented as .

4. model

In the experiments, we pretrain a Transformer that is as close as possible to the original model. The model consists of an encoder and a decoder, and an overview of the model is shown in Figure 2.

The encoder of the standard Transformer receives as input a 1D sequence of token embedding. To handle numerical values, we reshape the datapoints into a sequence of small parts, and each part serves as the token. After that we applied a trainable linear projection to map each part to D dimensions vectors, which function like the embedding vectors. The output of the encoder is a list of latent vectors, representing the compressed information of data points.

The decoder of the transformer receives the latent vector as the conditional information for multi-head attention. Besides, during the training phase, the sequence of equations served as the ground truth is sent to the decoder, in a masked way to prevent information leakage. While in the inference, the decoder is only provided with the latent vectors and generates a prediction autoregressively. The whole diagram is shown on Figure 2.

Diagram

Description automatically generated

Figure 2.1 diagram of training

Diagram

Description automatically generated

Figure 2.2 diagram of predicting

5. experiment and results

In our experiment, we use a transformer model with 4 attention heads, 6 layers, and dimensionality of 512, to predict the mathematical equations based on the corresponding data points. Parameters are optimized by minimizing the cross-entropy loss between the ground truth expressions and the predicted one. We use the Adam optimizer with a learning rate of , and a batch size of 256. More details about hyperparameters are shown on Table 2.

|  |  |  |
| --- | --- | --- |
| **parameters** | **number** | **description** |
| token size | 64 | The number of datapoints in a token |
| Embedding size | 128 | The size of embedding layer, and liner projecion layer |
| Encoder layer | 6 | The number of encoders |
| Decoder layer | 6 |  |
| Attention head | 4 |  |
| Optimizer | Adam |  |
| Learning Rate | 1e-4 |  |
| Batch size | 128 |  |
| Number of parameters | 173280 (encoder)  186205 (decoder) |  |
| Training set size | 720295 | The number of equations in training dataset |
| Test/val set size | 2000 | The number of equations in testing and validation dataset |

Table 2 hyperparameters for model training

After training on a single GeForce RTX 2080 GPU for 2 days (around 250 epoches of 720K examples), we test our model on a validation set of examples from the same data generator, the accuracy can reach to around 78%, as shown on Figure 3.1. Considering the limited dataset in our experiment, it is believed the accuracy can be higher if we increase the size of the dataset.

Chart, scatter chart

Description automatically generated

Figure 3.1 accuracy curve on test dataset

Chart

Description automatically generated

Figure 3.1 loss curve on training dataset

Chart, histogram

Description automatically generated

Figure 4.1 loss curve on validation data set

6. conclusion and future work

In this work, we independently propose a deep learning model for symbolic regression. We show that end-to-end methods can directly predict complete mathematical expressions without using skeleton estimation as an intermediate step.

We achieved good results in our experiments. In addition, our coding ability has been greatly improved during the implementation. In the future, we will further refine our model and compare with baselines in this field.

7. reference

1. John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992. ISBN 0-262-11170-5.
2. Mojtaba Valipour et al. ‘SymbolicGPT: A Generative Transformer Model for Symbolic Regression’. In: arXiv preprint arXiv:2106.14131 (2021).
3. Luca Biggio et al. Neural Symbolic Regression that Scales. 2021. arXiv: 2106.06427 [cs.LG].
4. Petersen, B. K. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. International Conference on Learning Representations, 2021
5. Silviu-Marian Udrescu and Max Tegmark. Ai feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, 2020.
6. Matt J Kusner, Brooks Paige, and Jose ́ Miguel Herna ́ndez-Lobato. Grammar variational autoencoder. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1945– 1954. JMLR. org, 2017
7. Guillaume Lample and Franc ̧ois Charton. Deep learning for symbolic mathematics. arXiv preprint arXiv:1912.01412, 2019.
8. Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. arXiv preprint arXiv:2009.03393, 2020.
9. Christopher Hahn, Frederik Schmitt, Jens U. Kreber, Markus N. Rabe, and Bernd Finkbeiner. Teaching temporal logics to neural networks. arXiv preprint arXiv:2003.04218, 2021
10. Feng Shi, Chonghan Lee, Mohammad Khairul Bashar, Nikhil Shukla, Song-Chun Zhu, and Vijaykrishnan Narayanan. Transformer-based machine learning for fast sat solvers and logic synthesis. arXiv preprint arXiv:2107.07116, 2021.
11. Franc ̧ois Charton, Amaury Hayat, and Guillaume Lample. Learning advanced mathematical computations from examples. arXiv preprint arXiv:2006.06462, 2020.
12. Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. arXiv preprint arXiv:2001.08361, 2020.
13. Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.
14. Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. A simple framework for contrastive learning of visual representations. In International conference on machine learning, pp. 1597–1607. PMLR, 2020a.
15. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), Advances in Neural Information Processing Systems, volume 30, pp. 5998–6008. Curran Associates, Inc., 2017. URL https://proceedings. neurips.cc/paper/2017/file/ 3f5ee243547dee91fbd053c1c4a845aa-Paper. pdf.
16. Lample, G. and Charton, F. Deep learning for symbolic mathematics. arXiv preprint arXiv:1912.01412, 2019.
17. Meurer, A., Smith, C. P., Paprocki, M., ˇ Cert ́ık, O., Kirpichev, S. B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J. K., Singh, S., Rathnayake, T., Vig, S., Granger, B. E., Muller, R. P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., Curry, M. J., Terrel, A. R., Rouˇ cka, v., Saboo, A., Fernando, I., Kulal, S., Cimrman, R., and Scopatz, A. Sympy: symbolic computing in python. PeerJ Computer Science, 3:e103, January 2017. ISSN 2376-5992. doi: 10.7717/peerj-cs.103. URL https: //doi.org/10.7717/peerj-cs.103.