





# Neuro-Symbolic Regression with Applications

Nour Makke<sup>(✉)</sup>  and Sanjay Chawla 

Qatar Computing Research Institute, HBKU, Doha, Qatar  
{nmakke, schawla}@hbku.edu.qa

**Abstract.** Discovering symbolic models is growing in popularity with the increasing interest in interpretable machine learning. Symbolic regression is the task of learning an analytical form of underlying models in data. Two machine learning techniques have proven their effectiveness: reinforce trick and transformer neural network. This paper discusses in detail the two techniques and presents the application of symbolic regression on a simulated data set that describes a high-energy physics process.

**Keywords:** Model discovery · Symbolic regression · Neural network · Transformer network · Physics data

## 1 Introduction

Model discovery in a data-driven manner is a standard task of machine learning (ML). Models learned from data often capture hidden patterns and can be used to make accurate predictions associated with the studied phenomenon. Based upon the technique adopted in the learning process, models are categorized as follows: uninterpretable or “BlackBox” models such as deep neural networks, where the relationship between the input and the output is neither transparent nor tractable, and interpretable or “Whitebox” models such as decision tree, where the input-output relationship is accessible, and allows for reasoning.

While there has been a lot of success in ML-based models in making highly accurate predictions, they remain uninterpretable and opaque. The increasing need for interpretable ML, especially in critical disciplines, motivates the development of ML-based methods that are predictive and interpretable. This is especially important for the application of ML in physical sciences, which is needed more than ever with the tremendous amount of data collected. For example, in high-energy physics experiments, a large amount of data is generated, and it is not humanly possible to carry out a manual examination to look for patterns. In such a scenario, a machine learning model effectively summarizes the data and can be used for making predictions. The model can then be introspected to elicit the prediction process if the predictions are accurate. More generally, physics is essentially described by mathematical equations, and one could ask if we can learn such equations directly from data. This is the symbolic regression. It was introduced back in 1970 [1]. It came back in 2009 with the commercial platform

Eureqa [2], which was publicized as a scientific discovery tool and developed using an evolutionary algorithm called genetic programming (GP). SR has since then been developed within the GP community. More recently, symbolic regression has been further tackled with deep learning-based tools. The last decade has witnessed the revolution of the deep learning field triggered by the development of ImageNet [3], with a recent review in [4]. The enormous growth of deep learning was based on the potential of deep neural networks, being universal function approximators particularly known for being end-to-end differentiable in their free parameters, predictive, and highly accurate.

The extensive use of deep neural networks over the last decade has unveiled the limitations of pure data-centric machine learning methods, ranging from adversarial attacks to explainability and fairness. These limits have triggered a trend to incorporate abstract concepts into the machine learning framework, such as graphs and symbolic representations, which help capture the complexity hidden in data while preserving interpretability. Recent machine learning methods aiming at avoiding out-of-distributions effects and promoting interpretability are growing in popularity. In particular, symbolic regression is attracting a wider research community within the machine learning field. In contrast, its adoption in scientific disciplines is at a very early stage, mainly due to immature developments made in this sub-field. Symbolic regression problems can be addressed using various approaches; see [5] for a full review. This paper discusses in detail and compares both the reinforce trick, which is used in reinforcement learning problems and the transformer network approach, which solves symbolic regression problems analogously to a language translation task. These approaches are of particular interest and have proven their effectiveness.

In the following, we present the definition of a function class (Sect. 2.1) and an expression tree representation (Sect. 2.2) within the problem statement section (Sect. 2). Furthermore, we discuss the symbolic regression problem within the zeroth-order optimization context (Sect. 3) and the attention mechanism context (Sect. 4). Finally, we present two state-of-the-art symbolic regression methods and compare their results on a physics application in Sect. 5. The conclusion is given in Sect. 6.

## 2 Problem Statement

Given a data set  $\mathcal{D} = \{x_i, y_i\}$ , where  $x_i \in \mathbb{R}^d$  and  $y_i \in \mathbb{R}$ , symbolic regression aims at finding the mapping ( $f^*$ ) in the class of mappings  $\mathcal{F} : (f : \mathbb{R}^d \rightarrow \mathbb{R})$  that minimizes the loss function as follows:

$$f^* = \arg \min_{f \in \mathcal{F}} l(f) \quad (1)$$

where  $f$  represents a non-linear, expressive, and parameterized function (e.g., neural network), and the loss function ( $l$ ) is defined by:

$$l(f) = \sum_{i=1}^b l(f(x_i), y_i) \quad (2)$$

## 2.1 Function Class

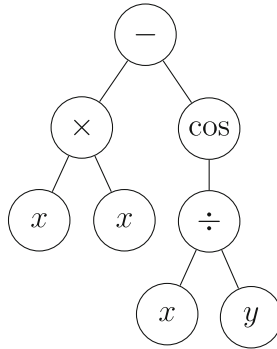
The class of function  $\mathcal{F}$  can be defined as the set of functions that can be obtained by the composition of mathematical operations and operands from a pre-defined library  $\mathcal{L}$ . The latter includes unary and binary mathematical operations, arithmetic operations, variables, and placeholders. Also, it can be extended to include as many operations as needed, as follows:

$$\mathcal{L} = \{+(\cdot, \cdot), -(\cdot, \cdot), \times(\cdot, \cdot), \div(\cdot, \cdot), \cos(\cdot), \sin(\cdot), \tan(\cdot), \exp(\cdot), \log(\cdot), \sqrt{(\cdot)}, x, \text{etc.}\}$$

The class of function  $\mathcal{F}$  defines the search space in the symbolic regression problem and is, by definition, of discrete nature.

## 2.2 Expression Tree Representation

Every mathematical equation can be represented as a unary-binary tree. The latter is a rooted tree in which internal nodes are mathematical operators (e.g.,  $\div$ ,  $\log$ ) and terminal nodes are operands, i.e., input variable or constant (e.g.,  $x$ ). Operators may be unary (one argument, e.g., cosine) or binary (two arguments, e.g., addition). For example, the function  $f(x) = x^2 - \cos(x/y)$  is represented by the tree illustrated in Fig. 1.



**Fig. 1.** Unary-binary tree-like structure of the mathematical function  $f(x, y) = x^2 - \cos(x/y)$ .

Furthermore, an expression tree can be represented as a unique sequence of symbolic representations, following the polish notation [7], by traversing the (binary) tree top to bottom and left to right in a depth-first manner.

-	×	x	x	cos	÷	x	y
---	---	---	---	-----	---	---	---

### 3 Zeroth-Order Optimization Problem

Symbolic regression methods aim to learn a mapping from a set of input-output pairs of numerical values to an analytical equation by minimizing a loss function. Therefore, SR learns the structure and the parameters of underlying models in data. Similarly to ML-based methods, the optimization task involves, through gradient-descent approaches, the computation of gradients of the loss function. However, the latter is not differentiable, which makes it a zeroth-order optimization problem [8].

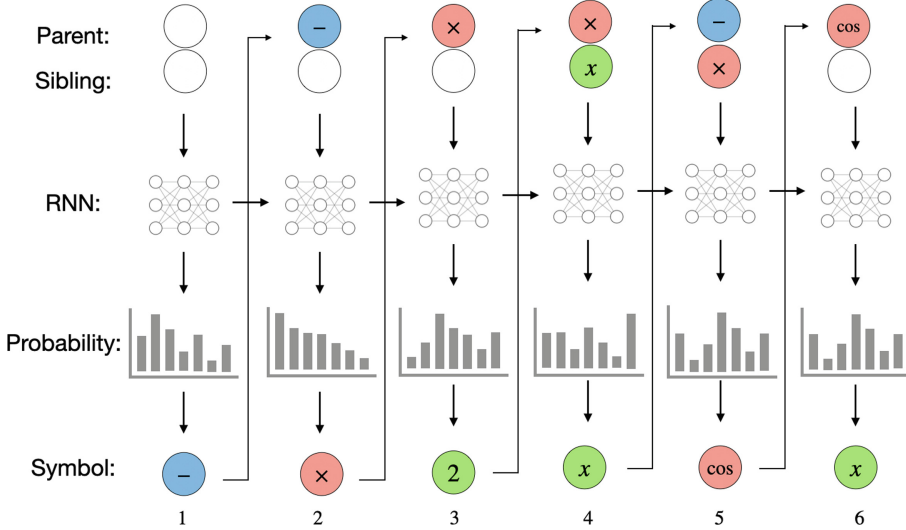
A state-of-the-art symbolic regression application is Deep Symbolic Regression (DSR) [9], in which mathematical equations are represented by symbolic expression trees. DSR uses a deep neural network to generate sequences and the “Reinforce trick” [6] to train it. The sequence generator is chosen to be a recurrent neural network (RNN). The latter is a parameterized distribution over mathematical expressions  $p(\tau|\theta)$  that allows backpropagation of a differentiable loss function with respect to parameters  $\theta$ . Symbols of a sequence  $\tau$  are generated one at a time, and each symbol  $\tau_i$  is sampled from a pre-defined library of mathematical operations, e.g.,  $\mathcal{L} = \{+, -, \times, \div, \sin, \cos, \log, x\}$ . For each symbol  $\tau_i$ , RNN takes as input the parent and sibling nodes of the symbol being sampled and outputs a probability distribution over  $\mathcal{L}$ , conditioned by the preceding symbols  $\tau_1, \dots, \tau_{(i-1)}$ , as illustrated in Fig. 2. RNN is trained using the reinforcement learning-based technique described in the following. Once the mathematical expression is sampled, it is evaluated with a reward function  $R(\tau)$  that is defined using the normalized root-mean-square error (RMS),  $R(\tau) = 1/(1 + \text{RMS})$ .

In this approach, the optimization problem reduces to maximize the reward function. For this goal, the standard policy gradient objective defined by the expectation of the reward is considered, i.e.,  $J(\theta) = \mathbb{E}_{\tau \sim p(\tau|\theta)}[R(\tau)]$ . The optimization problem is thus formulated as:

$$\theta^* = \arg \max_{\theta} J(\theta) \quad (3)$$

This optimization task (Eq. 3) is challenging because the reward function  $R(\tau)$  is not differentiable with respect to learnable parameters  $\theta$ . Here the “Reinforce trick”, i.e., REINFORCE, originally introduced in the reinforcement learning community [6] is used. It transforms the gradient of the reward  $\nabla_{\theta} R(\tau)$  to the gradient of the logarithm of the policy  $\log(p(\tau, \theta))$  as follows:

$$\begin{aligned} \nabla_{\theta} \mathbb{E}_{p(\tau, \theta)}[R(\tau)] &= \nabla_{\theta} \int R(\tau) p(\tau, \theta) d\theta \\ &= \int R(\tau) \nabla_{\theta} p(\tau, \theta) d\theta \\ &= \int R(\tau) \frac{\nabla_{\theta} p(\tau, \theta)}{p(\tau, \theta)} p(\tau, \theta) d\theta \\ &= \int R(\tau) \nabla_{\theta} \log(p(\tau, \theta)) p(\tau, \theta) d\theta \\ &= \mathbb{E}_{\tau \sim p(\tau, \theta)} [R(\tau) \nabla_{\theta} \log p(\tau, \theta)] \end{aligned} \quad (4)$$



**Fig. 2.** An example of generating the mathematical expression  $f(x) = 2x - \cos(x)$  from RNN. For the first node of the (binary) tree, empty symbols are given to the RNN as input because the tree node does not have a parent or sibling. Following the pre-order traversal of the tree, symbols are autoregressively sampled until the tree is completed.

The importance of this result is that  $\nabla_{\theta} J(\theta)$  can be evaluated by computing the sample mean over a batch of  $N$  sampled expressions as follows:

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N R(\tau^{(i)}) \nabla_{\theta} \log p(\tau^{(i)} | \theta) \quad (5)$$

This, in turn, can be optimized using gradient ascent:

$$\theta \leftarrow \theta + \alpha R(\tau) \sum_i \nabla_{\theta} [\log p(\tau, \theta)] \quad (6)$$

A key technique that boosts the performance of DSR is the use of “risk-seeking policy gradient”, i.e., to optimize the best-case performance of a policy instead of optimizing its average performance. For this goal, the top-performing  $\epsilon$  fraction of expressions found during training are selected, and a new learning objective is defined by:

$$J(\theta, \epsilon) = \mathbb{E}_{\tau \sim p(\tau | \theta)} [R(\tau) \mid R(\tau) \geq R_{\epsilon}(\theta)] \quad (7)$$

The Reinforce trick is used to estimate the new objective function where only the top  $\epsilon$  fraction of expressions from each batch are used in the gradient computation.

## 4 Transformer Neural Network for Symbolic Regression

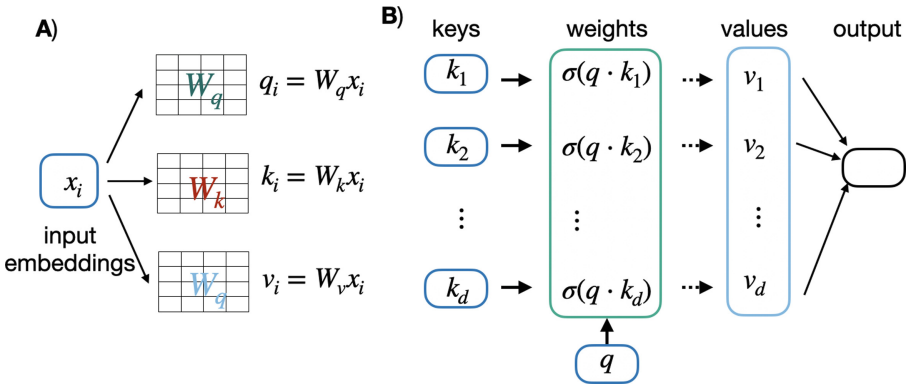
A transformer neural network (TNN) is a novel neural network architecture designed to treat sequential data (e.g., translation tasks). It is a sequence-to-sequence model that was developed in Natural Language Processing (NLP) [11] community to capture and model long-range dependencies in sequential data based on the attention mechanism. Consider the English-to-French translation of the two following sentences (sequences of words):

En: The mouse did not eat the cheese because it was sick.  
 Fr: *La souris n'a pas mangé le fromage parce qu'elle était malade.*

En: The mouse did not eat the cheese because it was expired.  
 Fr: *La souris n'a pas mangé le fromage parce qu'il était périmé.*

The translation of the word “it” is different because of the different contexts in the two sentences. However, they are almost identical. The only difference between the two sentences is in the last word, which refers to the mouse in the first sentence (i.e., “sick”), whereas it refers to the cheese in the second sentence (i.e., “expired”). This is what the attention mechanism is about. It pays particular attention to the terms (of the sequence) with high weights. In this example, the noun that the adjective of each sentence refers to has a significant weight and is therefore considered for translating the word “it”. Technically, an embedding  $x_i$  is assigned to each element of the input sequence, and a set of  $m$  key-value pairs is defined, i.e.,  $\mathcal{S} = \{(k_1, v_1), \dots, (k_m, v_m)\}$ . For each query, the attention mechanism computes a linear combination of values  $\sum_j \omega_j v_j$ , where the attention weights ( $\omega_j \propto q \cdot k_j$ ) are derived using the dot product between the query ( $q$ ) and all keys ( $k_j$ ), as follows:

$$\text{Attention}(q, \mathcal{S}) = \sum_j \sigma(q \cdot k_j) v_j \quad (8)$$



**Fig. 3.** **A)** The original embeddings (queries  $\{q_i\}$ , keys  $\{k_i\}$ , values  $\{v_i\}$ ) computed from the embeddings of the input sequence  $\{x_i\}$ . **B)** Evaluation of  $\text{Attention}(q, \mathcal{S})$  (Eq. 8) for a query  $q$ .

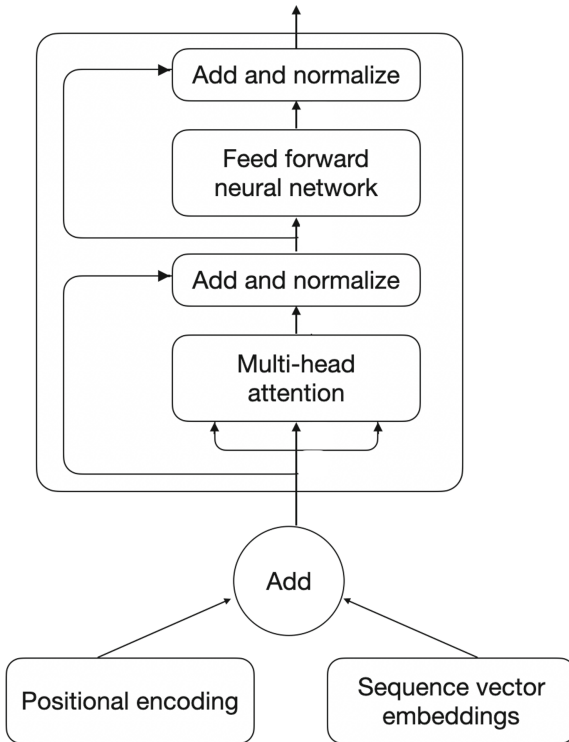
Here,  $q = xW_q$  is a query,  $k_i = x_iW_k$  is a key,  $v_i = x_iW_v$  is a value, and  $W_q, W_k, W_v$  are learnable parameters. The architecture of the self-attention mechanism is illustrated in Fig. 3.

For symbolic regression tasks, mathematical equations are regarded as sequences of symbolic representations, and TNN is used as a set-to-sequence model. Consider the function  $f(x, y) = x^2 - \cos(x/y)$  whose tree is illustrated in Fig. 1. Its sequence of embeddings is given by:

$$x_1 : - \quad x_2 : \times \quad x_3 : x \quad x_4 : x \quad x_5 : \cos \quad x_6 : \div \quad x_7 : x \quad x_8 : y$$

In this example, for the prediction of the query  $(x_8 : y)$ , the attention mechanism will compute a higher weight for the binary division operator  $(x_6 : \div)$  rather than for the subtraction operator  $(x_1 : -)$  or the variables  $(x_3 : x)$  or  $(x_4 : x)$ .

Transformers have an encoder-decoder structure. The structure of each block mainly comprises an attention layer and a feed-forward neural network. As an example, the encoder block of TNN is illustrated in Fig. 4. TNN takes the sequence of embeddings  $\{x_i\}$  and outputs a “context-dependent” sequence of embeddings  $\{y_i\}$ , through a latent representation. The output can have a multipurpose use (e.g., translation task, classification task, etc.).



**Fig. 4.** Structure of the encoder in a transformer neural network. It comprises an attention layer and a feed-forward neural network [13].

“Neural Symbolic Regression that Scales” (NeSymReS) [12] is a recently developed TNN-based symbolic regression method. It builds on and combines two approaches that have proven to be highly efficient. The first is pre-training large models on large datasets, and the second is using transformers for tasks that involve symbolic operations. Whereas existing TNN-based methods use transformers in a pure symbolic domain (e.g., symbolic integration, solving differential equations, etc.), this approach aims at mapping numerical values to symbolic equations. Therefore it represents a set-to-sequence model that predicts an equation (expression tree) from a set of input-output pairs of numerical values  $(X, Y)$ .

NeSymReS comprises two phases. First is the pre-training phase. A large amount of training data is generated using computers, which can generate an infinite amount of data with perfect accuracy at almost no cost. A training example is defined by  $\{x_i, f(x_i)\}_{i=1}^n$  and an equation  $(\tau)$  that represents the mapping  $f : \mathbb{R}^{d_x} \rightarrow \mathbb{R}^{d_y}$ . In the pre-training phase, the encoder maps each sequence  $\{x, y\} \in (X, Y)$  into a latent representation  $z$ . The decoder then generates a sequence of symbols  $\tau = \{\tau_1, \dots, \tau_{|\tau|}\}$ , where  $|\tau|$  is the length of the sequence (traversal of the expression tree). The decoder outputs a probability distribution  $P(\tau_i | \tau_{1:(i-1)}, z)$  given the latent representation  $z$  and the previously sampled symbols. Only mathematical operators (in the sequence) are sampled, i.e., numerical constants are replaced by a placeholder symbol ( $\Delta$ ), and will be fit at a later stage. This is called the skeleton. Each placeholder will be treated as an independent parameter in case of occurrence.

An example of sampling the mathematical expression  $f(x) = 2x - \cos(x)$  in NeSymReS is illustrated in Fig. 5. This method first predicts the sequence of symbols as shown below, and then fits the numerical values of existing constants.

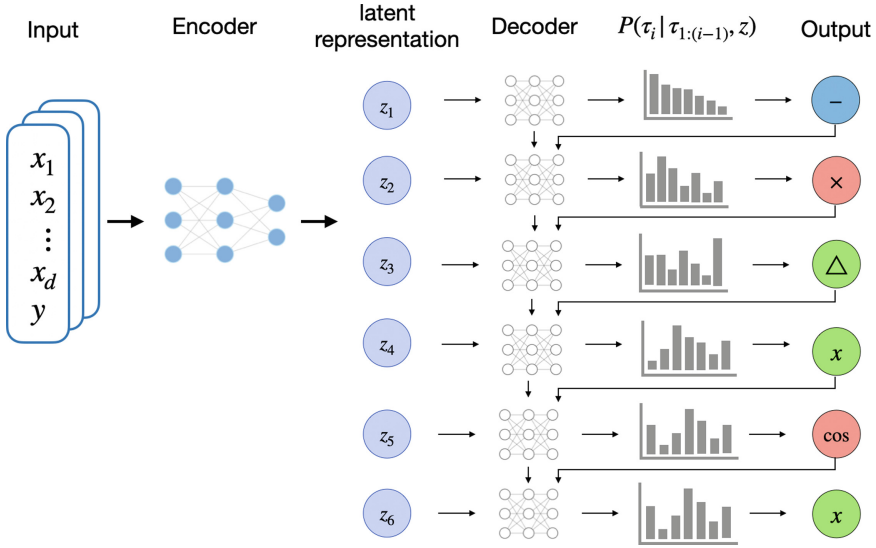
$$x_1 : - \quad x_2 : \times \quad x_3 : \Delta \quad x_4 : x \quad x_5 : \cos \quad x_6 : x$$

The second phase is the test time, in which the efficiency is measured on the test data set. A key factor in this approach is that it improves over time with data, and the (encoder-decoder) networks do not have to be retrained from scratch for each new experiment (equation), in contrast with all SR approaches, in particular, DSR.

## 5 Physics Application

This section presents the application of SR approaches to high-energy physics. A problem of interest is the so-called “hadronization mechanism” [14], which describes the formation of hadrons (such as protons and neutrons) from elementary particles (the so-called “quarks”). This process is studied using experimental measurements of particle distributions, which are observed to follow an exponential functional form. Experimental measurements are commonly performed in a multidimensional scheme for an in-depth understanding of the target phenomena. In our discussion, we will only consider two dimensions for simplicity, i.e.,  $\mathcal{D}(x, y)$ .  $\mathcal{D}$  represents the distributions of particles as a function of one variable





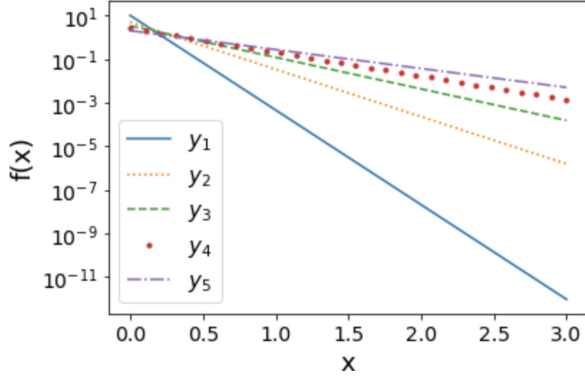
**Fig. 5.** An example of sampling the mathematical expression  $f(x) = 2x - \cos(x)$  in “Neural Symbolic Regression that Scales” application based on transformers. The triangle symbol represents a numerical constant that will be fit at a later stage.  $P(\tau_i | \tau_{1:(i-1)}, z)$  represents the probability distribution over the symbol  $\tau_i$  given previous symbols  $\tau_{1:(i-1)}$  and the latent representation  $z$ .

$x$ , in ranges of another variable  $y$ . The goal of such measurements is to learn the analytical form of the underlying mechanism in terms of  $x$  and  $y$ . This goal is traditionally achieved by fitting data points using a pre-defined functional form with unknown parameters, which is not an ideal solution simply because the underlying model is unknown. This application aims at applying symbolic regression to such data points, and we expect that the symbolic models extracted in the multiple ranges of  $y$  will be the same, however, numerical values of the parameters are expected to change from one range of  $y$  to another.

In this application, we will use simulated data points instead of experimental ones. Assume that experimental measurements follow the functional form:

$$f(x, y) = \frac{1}{a(y)} \exp\left(\frac{-x}{a(y)}\right) \tag{9}$$

Here  $x$  and  $y$  represent two physical properties of the particles (e.g., energy and momentum), and  $a$  is a fit parameter. The function  $f(x)$  exhibits an implicit dependence upon  $y$  via its free parameter  $a$ . To imitate experimental data, we simulate five data sets, each corresponding to a different range of  $y$ , i.e., to a different value of  $a$  in the simulation. Figure 6 illustrates the function  $f$  as a function of  $x$  ( $\in [0, 3]$ ) in intervals of  $y$ . The slopes of the curves significantly change for different intervals of  $y$ , up to six orders of magnitude difference at



**Fig. 6.** Mathematical expression  $f(x, y) = 1/a(y) \exp(-x/a(y))$  as a function of  $x$  in different ranges of  $y$ . Each range of  $y$  corresponds to a different value of  $a$ :  $y_1$  ( $a = 0.1$ , solid curve),  $y_2$  ( $a = 0.2$ , dotted curve),  $y_3$  ( $a = 0.3$ , dashed curve),  $y_4$  ( $a = 0.4$ , loosely dotted curve) and  $y_5$  ( $a = 0.5$ , dash-dotted curve).

high values of  $x$  between the smallest and the largest intervals of  $y$ , i.e., between the smallest and the largest values of  $a$  given that data points are simulated.

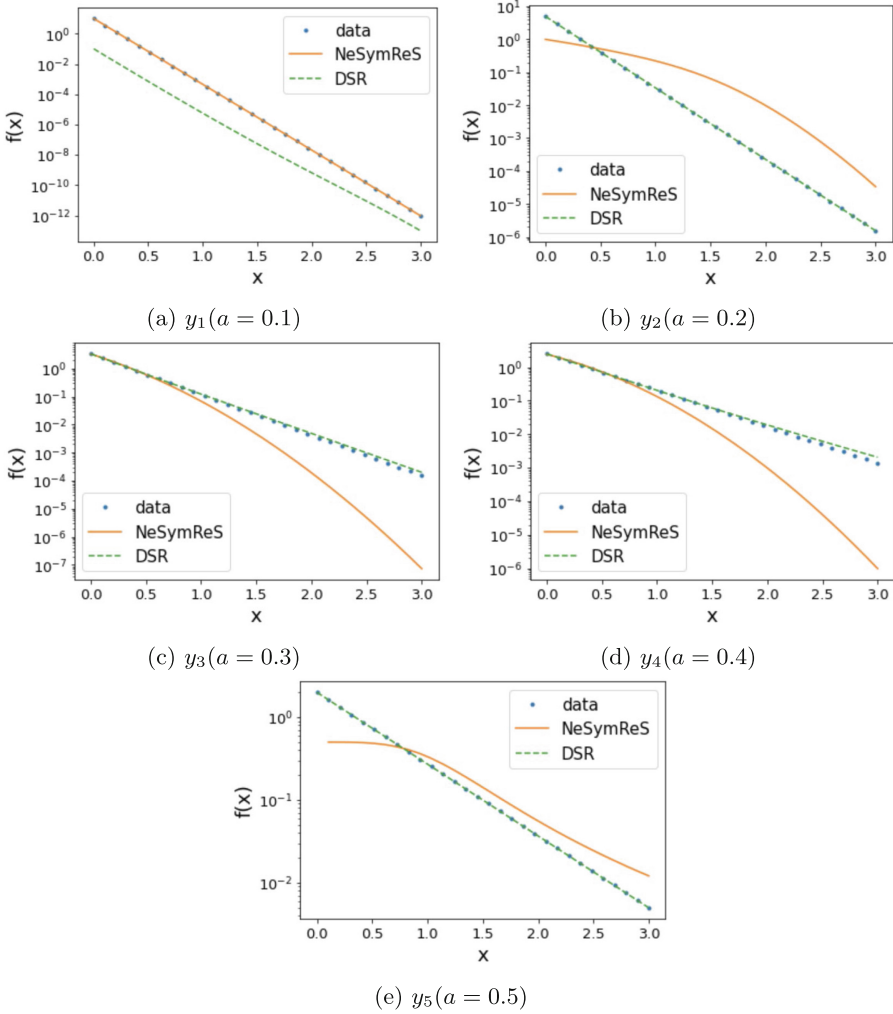
Simulated data sets, i.e.,  $\mathcal{D} = \{x_i, f(x_i)\}_{i=1}^n$ , each consisting of 30 input-output pairs, are generated with perfect accuracy, and no noise is added. In this application, we will apply the SR approaches DSR and NeSymReS, and compare their results. Ideally, we expect to learn the same underlying model, i.e.,  $f(x) = 1/a \exp(-x/a)$ , in the five intervals of  $y$ , while obtaining different values of  $a$ . We use a pre-trained version of NeSymReS, whereas DSR is trained for each new experiment (data set), as previously mentioned in the text. A total of 20 runs is performed on each experiment using NeSymReS and 100 runs using DSR.

Figure 7 shows simulated data points (solid markers) and the results obtained by applying both SR approaches to simulated data sets (curves). The resulting equations and recovery rates (how many times the exact mathematical equation is covered) are presented in Table 1 for each experiment. Results obtained using NeSymReS show a significant discrepancy between the predicted expressions and the ground-truth function for all intervals of  $y$  except one. An in-depth investigation shows that, in a successful experiment, either the skeleton is correctly predicted ( $c_1 * \exp(c_2 * x)$  in this case), or it is not accurately predicted, but, it reduces to the correct equation after fitting the constants. For example, for the only successful case in this application, i.e.,  $y_1$  ( $a = 0.1$ ), NeSymReS predicts the following skeleton:

$$c_1 * (c_2 * x + \exp(c_3 * x))^{\text{pow}}$$

where  $c_1$ ,  $c_2$ , and  $c_3$  are fit constants, and pow is a power coefficient. They have the following values:

$$c_1 \approx 10 \quad c_2 = 2.075 \times 10^{-6} \quad c_3 \approx 10 \quad \text{pow} = -1$$



**Fig. 7.** Results of DSR and NeSymReS applied to simulated data. The functional form  $f(x, y) = 1/a(y) \exp(-x/a(y))$  is used to generate five data sets with  $x \in [0, 3]$  in five ranges of the variable  $y$ .

The predicted expression becomes:

$$10 * (0 * x + \exp(10 * x))^{-1} = 10 * \exp(-10 * x)$$

Therefore the predicted equation is equal to the ground-truth function. Consider one of the failed experiments, i.e.,  $y_2(a = 0.2)$ . NeSymReS predicts the following expression:

$$(x + \exp(x)^2)^{-1}$$

In this case, the NeSymReS predicts a skeleton that does not match the ground-truth function and does not include any constant to fit, in contrast with the successful case. In conclusion, the main reason for failure, in this method, is the incorrect skeleton prediction rather than at the level of fitting the numerical values of the constants in the predicted skeleton.

DSR successfully predicts mathematical equations for two (out of five) data sets ( $a = 0.2$  and  $a = 0.5$ ), and it produces an accurate fit for the other ranges of  $y$  with an  $R$ -coefficient that is greater than 0.99. Consider one successful experiment, for example,  $y = 0.5$ . DSR predicts, in this case, a sequence of 14 symbols given by:

$$\div + \cos - x x \cos - x x \exp + x x$$

which is equivalent to:

$$\frac{\cos(x - x) + \cos(x - x)}{\exp(x + x)} = \frac{2}{\exp(2 * x)} = 2 * \exp(2 * x)^{-1}$$

Therefore, equations predicted by DST do not necessarily match the exact sequence of the ground-truth function.

In summary, the success rates for both methods are less than 50% in this application. However, the predictions made by DSR fit very well the data for the whole range of  $x$  in all ranges of  $y$ , even in cases when the function is not correctly predicted, in contrast to NeSymReS, although the latter has been shown to outperform DSR on various data benchmarks in terms of test-time compute and out-of-distribution prediction [12].

**Table 1.** Equations and recovery rates obtained by the application of SR methods, DSR and NeSymReS, on simulated data sets. The equation predicted in  $y_1$ -range is

$$f(x) = e^{-10*x + e^{-\cos(x)} - e^{-\sin(e^{-4e^{-6e^{-1e^x}}})}}$$

$$f(x, y_3) = e^{-3*x + e^{-e^{-x}} + e^{-e^{x1 - e^{\cos(e^{-x})}}}}$$

$y$ -range $\downarrow$	DSR			NeSymReS		
	Fit	$f(x)$	Rate (%)	Fit	$f(x)$	Rate (%)
$y_1(a = 0.1)$	✗	In caption	0	✓	$10e^{-10x}$	100
$y_2(a = 0.2)$	✓	$5e^{-5x}$	5	✗	$(x + e^{2x})^{-1.15}$	0
$y_3(a = 0.3)$	✗	In caption	0	✗	$3.3 e^{(-x^2 - 2.8x)}$	0
$y_4(a = 0.4)$	✗	$e^{-2x - \cos(e^{-3/8x})}$	0	✗	$2.4 e^{(-x^2 - 1.9x)}$	0
$y_5(a = 0.5)$	✓	$2e^{-2x}$	100	✗	$\frac{x}{(x^5 + 2x)}$	0

## 6 Conclusion

Learning interpretable models is growing in interest and in popularity in the last decade. Symbolic regression represents a key method to learn interpretable models in a purely data-driven manner. Recent developments in the symbolic regression field have shown that the use of deep neural networks boosts the performance of these methods. In this paper, we presented the application of two symbolic regression methods on simulated data towards applying them to experimentally measured data, and we showed that state-of-the-art symbolic regression methods do not necessarily produce consistent results, as one expects.

## References

1. Langley, P.: Data-driven discovery of physical laws. *Cogn. Sci.* **5**(1), 31–54 (1981). [https://doi.org/10.1016/S0364-0213\(81\)80025-0](https://doi.org/10.1016/S0364-0213(81)80025-0). Conference 2016
2. Dubcakova, R.: Eureka: software review. *Genet. Program Evolvable Mach.* **12**(2), 173–178 (2011). <https://doi.org/10.1007/s10710-010-9124-z>
3. Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, L.: ImageNet: a large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 248–255 (2009). <https://doi.org/10.1109/CVPR.2009.5206848>
4. Chawla, S., et al.: Ten years after ImageNet: a 360° perspective on AI. [arXiv:2210.01797](https://arxiv.org/abs/2210.01797)
5. Makke, N., Chawla, S.: Interpretable scientific discovery with symbolic regression: a review. [arXiv:2211.10873](https://arxiv.org/abs/2211.10873)
6. Ronald, W.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.* **8**, 229–256 (1992). <https://doi.org/10.1007/BF00992696>
7. Lukasiewicz, J.: Aristotle’s Syllogistic from the Standpoint of Modern Formal Logic, Second Edition Enlarged, pp. xvi 222. Clarendon Press, Oxford (1957). Cloth, 305.net
8. Liu, S., Chen, P., Kailkhura, B., Zhang, G., Hero III, A.O., Varshney, P.: A primer on zeroth-order optimization in signal processing and machine (2020). [arXiv:2006.06224](https://arxiv.org/abs/2006.06224)
9. Petersen, B.: Deep symbolic regression: recovering mathematical expressions from data via policy gradients. *CoRR* (2019). [arXiv:1912.04871](https://arxiv.org/abs/1912.04871)
10. Mundhenk, T.N., Landajuela, M., Glatt, R., Santiago, C.P., Faissol, D.M., Petersen, P.K.: Symbolic regression via neural-guided genetic programming population seeding. *CoRR* (2021). [arXiv:2111.00053](https://arxiv.org/abs/2111.00053)
11. Vaswani, A., et al.: Attention is all you need. *CoRR* (2017). [arXiv:1706.03762](https://arxiv.org/abs/1706.03762)
12. Biggio, L., Bendinelli, T., Neitz, A., Lucchi, A., Parascandolo, G.: Neural symbolic regression that scales (2021). [arXiv:2106.06427](https://arxiv.org/abs/2106.06427)
13. Wood, T.: Transformer neural network. <https://deeptai.org/machine-learning-glossary-and-terms/transformer-neural-network>
14. Andersson, B., Gustafson, G., Soderberg, B.: A general model for jet fragmentation. *Z. Phys. C* **20**, 317 (1983). <https://doi.org/10.1007/BF01407824>