A Seq2Seq approach to Symbolic Regression

Luca Biggio[1,2], **Tommaso Bendinelli**[2], **Aurelien Lucchi**[1], **Giambattista Parascandolo**[1,3]
[1]Data Analytics Lab, ETH Zürich, Switzerland
[2]Automation and Machine Learning, CSEM SA, Switzerland
[3]MPI for Intelligent Systems, Tübingen, Germany
{lbiggio, parascag, aurelien.lucchi}@inf.ethz.ch, tbe@csem.ch

The results and methods presented in this paper have been extended and enhanced by the authors in "Neural Symbolic Regression that Scales" Biggio et al. [2021] (ICML 2021).

Abstract

Deep neural networks have proved to be powerful function approximators. The large hypothesis space they implicitly model allows them to fit very complicated black-box functions to the training data. However, often the data generating process is characterized by a concise and relatively simple functional form. This is especially true in natural sciences, where elegant physical laws govern the behaviour of the quantities of interest. In this work, we address this dichotomy from the perspective of Symbolic Regression (SR). In particular, we apply a fully-convolutional seq2seq model to map numerical data to the corresponding symbolic equations. We demonstrate the effectiveness of our approach on a large set of mathematical expressions by providing both a qualitative and a quantitative analysis of our results. Additionally, we release our new equation-generator Python library in order to facilitate benchmarking and stimulate new research on SR[1].

SR refers to the task of learning a mapping from a set of numerical input-output pairs $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{M}$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $\mathbf{y} \in \mathbb{R}^q$, to the corresponding symbolic expression f, implicitly defined by the mathematical relation $\mathbf{y} = f(\mathbf{x})$. While standard regression methods aim at fitting the training data as closely as possible, SR targets the underlying symbolic expression. Having access to such a compact representation of the data is of high interest for researchers in many scientific disciplines since it could lead to a deeper understanding of the physical system under consideration.

However, SR is hard. The space of mathematical expressions grows exponentially with the length of the equation, allowing current search methods to retrieve only limited-sized expressions. The strong combinatorial nature of SR has motivated the application of evolutionary algorithms and specifically genetic programming (GP) [Koza, 1993, Schmidt and Lipson, 2009] techniques. Despite their relatively good performances in several settings, including SR, these methods do not scale well to larger problems and are very sensitive to hyperparameter choices. A more recent line of research [Sahoo et al., 2018, Udrescu and Tegmark, 2020, Petersen, 2020] has started exploring the application of Deep Learning (DL) approaches to tackle the aforementioned combinatorial challenge. Despite some encouraging results, the performance of these approaches cannot improve with more data and compute. Specifically, solving more equations at training time does not change how the model works at test time, as the algorithms are hand-designed to work on one equation at the time.

Instead, in this work we propose a method that *learns* to do symbolic regression, and benefits from both data and compute [Sutton, 2019]. We adapt the convolutional seq2seq model proposed by Gehring et al. [2017] in the context of Natural Language Processing (NLP) to directly map numerical input-output pairs $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{M}$ to the corresponding symbolic expression f. We qualitatively show that our model can recover the exact form of the symbolic expression describing the data and provides

[1]https://github.com/SymposiumOrganization/EQLearner

1st Workshop on Learning Meets Combinatorial Algorithms @ NeurIPS 2020, Vancouver, Canada.

better generalization performances than a fully-connected deep neural network trained to map each $\mathbf{x}_i$ into the corresponding $\mathbf{y}_i$.

1 Methodology

Our methodology is based on three core ideas. Primarily, we want to create a model that captures correlations across different symbolic expressions and does not need to be retrained for each new SR task. To this extent, we create programmatically a dataset, $\mathcal{D} = \{\mathbf{Y}_i\ f_i\}_{i=1}^{N}$, comprising N pairs of symbolic functions, f_i, and the corresponding numerical evaluations, $\mathbf{Y}_i = f_i(\mathbf{X}_i)$, on a fixed grid of M support points, $\mathbf{X_i} = \mathbf{X} = \{\mathbf{x}_j\}_{j=1}^{M}$. This method allows us to create arbitrarily large and diverse datasets to train and evaluate our model in a supervised fashion. As a result, our model takes as input a set of M numerical points, $\mathbf{Y}_i$, and outputs a string corresponding to the mathematical expression generating the data. The second idea stems from the observation that using a numerical loss, such as the RMSE, measuring the distance between the predicted equation and the ground truth one, often leads to numerical issues and overfitting. Indeed, given a model with enough capacity and an arbitrary underlying equation, training data points can often be approximated by multiple functions, with a different structure than the true one, up to a very small error. To cope with this aspect, we base our models on a loss function measuring how far our prediction stands from the true expression entirely at a symbolic level. We use the Cross Entropy Loss between the predicted tokens and the ground-truth ones as our objective function. The third idea, similarly to Petersen [2020] is to defer to a second independent step, the computation of the numerical constants. This design choice is motivated by the consideration that using tokenized values for representing numerical values is not possible. On the other end, it is simple to fit numerical constants with non-linear optimizers (e.g. BFGS Fletcher [1987]), given a symbolic skeleton. An illustration of the aforementioned ideas and an overview of our approach are shown in Fig. 1.

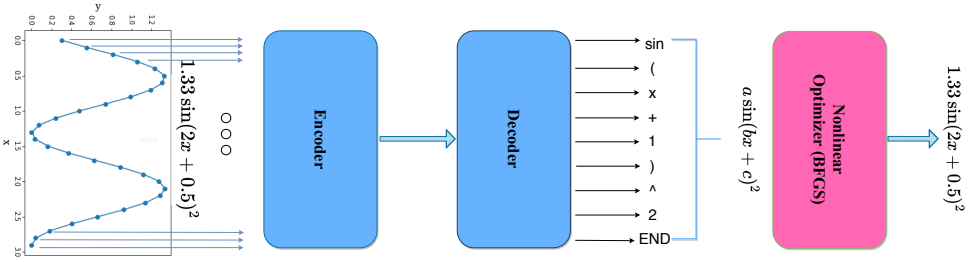

Figure 1: At each forward pass, a set of M numerical data points, $\mathbf{Y}_i = f_i(\mathbf{X}_i)$, described by a mathematical expression, f, are passed through an encoder. The encoder is responsible for processing the data and passing the resulting representation to the decoder. Finally, the decoder generates a sequence of integer tokens, each corresponding with a different symbol, making up the final output expression. In a second and independent step, we fit the numerical constants with a non-linear optimizer (BFGS).

The ideas exposed above present a number of advantages over existing SR approaches. First and foremost, it is possible to generate an arbitrarily large dataset of equations to train our model. This is in contrast with previous methods that have to be retrained from scratch for each new expression. Furthermore, it is very easy to impose specific inductive biases into the learning process by opportunely customizing the dictionary used for generating the symbolic equations. In contrast, previous works had to introduce specific design choices to make learning certain types of equations possible. Sahoo et al. [2018], for instance, had to modify their end-to-end architecture to include division activation functions and adapting the training strategy accordingly to avoid numerical instabilities. Udrescu and Tegmark [2020] instead, manually check whether a set of data is characterized by multiplicative separability in order to recursively restrict the search space for their algorithm. Our approach also differs from the technique proposed by Petersen [2020], since in their case, the network is not conditoned on the numerical input data and it is trained indirectly with the REINFORCE algorithm [Williams, 1992] using the RMSE loss as the reward function.

Seq2Seq Modelling. We choose to implement our approach by opportunely adapting the architecture proposed by Gehring et al. [2017] to the SR setting. This model is fully-convolutional and

incorporates gated recurrent units, residual connections and attention. CNNs are easier to parallelize during training than RNN-based models, thus allowing full exploitation of GPU hardware, resulting in significant run-time gains.

2 Experimental Results

We evaluate our approach on a dataset, $\mathcal{D} = \{\mathbf{Y}_i\ f_i\}_{i=1}^{N}$, of $N = 50.000$ equations created with our library. Each function, f_i contains randomly generated numerical constants and we instantiate the input of our model, $\mathbf{Y}_i$, by evaluating f_i on a fixed set of $M = 30$ equally-spaced support points, $\mathbf{X}$, into the interval $[0.1, 3.1]$. Our seq2seq model is trained to map each $\mathbf{Y}_i$ into an expression, g_i, which has the same skeleton as f_i but does not contain any numerical constants. The final function f_i can then be retrieved by running the BFGS optimizer on the family of equations spanned by g_i (see Fig. 1). Since every function contains different constants, all the elements of the dataset are characterized by different numerical inputs $\mathbf{Y}_i$. On the other hand, multiple instances of the same symbolic skeleton might be present. Further details about the dataset can be found in the Appendix.

Visualizations. Fig. 2 shows a comparison between our model and a three-layer fully-connected neural network with 100 units per layer, trained to approximate three different nonlinear functions. For each plot, the green region indicates the training domain, whereas the red area shows the extrapolation region. This example shows that our network (orange) successfully retrieves the underlying symbolic equation (green), while the fully-connected neural network (blue) fails to extrapolate outside the interval. Note that the equations shown in this example appear in the training set with different internal and external constants than those used here.

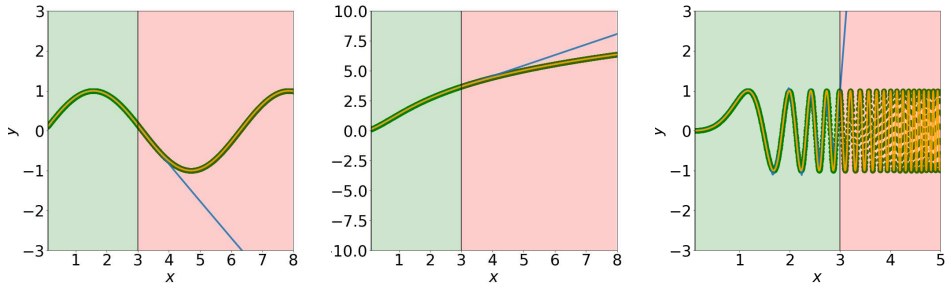

Figure 2: Comparison of the performance of our seq2seq model (orange) and a fully-connected neural network (blue) on the approximation of three different functions (green): (left) $y = \sin(x)$; (middle) $y = \log(x^3 + x^2 + x + 1)$; (right) $y = \sin(x^3)$

Fig. 3 illustrates the behaviour of our model when equations comprising out-of-dictionary symbols are used as input. These equations include tokens that our network has never seen during training (e.g. "cos", "cosh" and the division operator "/").

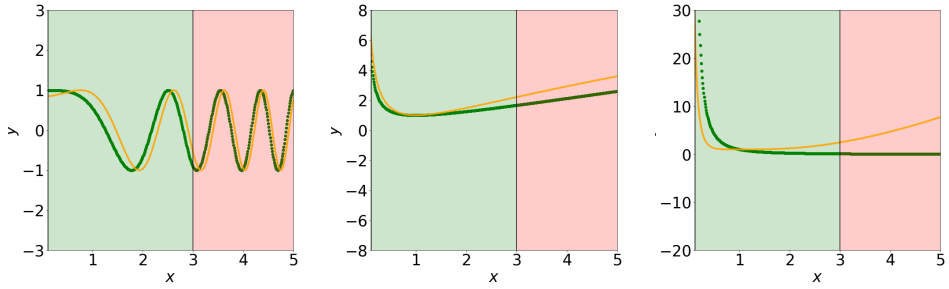

Figure 3: Performance of our seq2seq model (orange) on the approximation of three different equations (green) comprising out-of-dictionary symbols: (left) $y = \cos(x^2)$; (middle) $y = \cosh(\log(x))$; (right) $y = 1/x^2$

Our model outputs expressions with known symbols whose numerical values resemble those associated with the ground-truth equations. In the left panel, the output of our model is "$y = \sin(x^2 + 1)$", which captures the sinusoidal nature of the ground-truth function. In the middle, our network outputs "$y = \log(x)^2 + 1$", whereas for the equation to the right, its output is "$y = \log(x)^4 + 1$". In both cases, the network's output diverges at $x = 0$, in accordance with the ground-truth. In one case (middle), the prediction follows the same trend as the ground truth as $x \to \infty$, while for the other (right), the network fails to model the decreasing trend of the underlying function in the extrapolation region. We argue that this behaviour results from the lack of training examples characterized by such decreasing trend and could thus be resolved by enlarging and diversifying our training set.

Quantitative Analysis. Here, we propose a novel approach to evaluate SR models, with a particular emphasis on generalization. We consider two test sets of 1.000 equations each. The first test set includes skeleton expressions appearing in the training set with different numerical constants. The second dataset includes more complex novel symbolic expressions that our model has never seen during training. For each function f_i, we feed the network with the corresponding evaluation points $f(\mathbf{X}) = \mathbf{Y}_i$, where $\mathbf{X}$ is a vector of 30 equally spaced points into the interval $\mathcal{I} = [0.1, 3.1]$. Then, the resulting output expression is evaluated on a set of 30 points into the extrapolation region $\mathcal{E} = [3.1, 6.1]$. Finally, the RMSE is calculated between prediction and ground-truth on $\mathcal{E}$. We compare its performances with a 3-layer feed-forward neural network (FFNN3 for short) trained on $\mathcal{I}$ and tested on $\mathcal{E}$. Fig. 4 illustrates the extrapolation performance of the two methods.

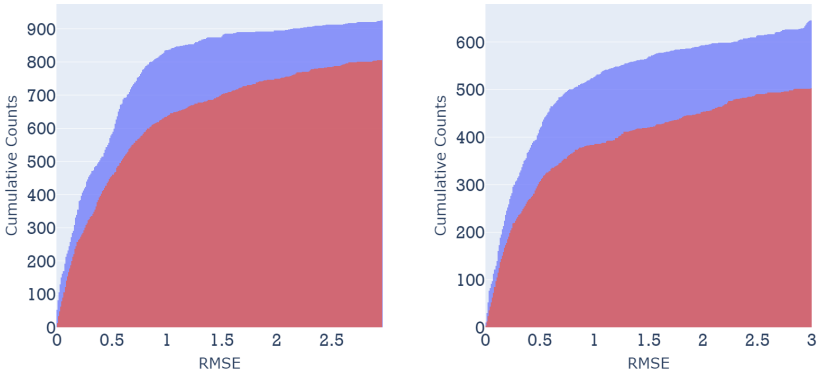

Figure 4: RMSE cumulative histograms for (left) first and (right) second test sets. Our model is shown in blue, while FFNN3 in red.

As shown above, our model outperforms the neural network on both datasets. The conventional feed-forward model must be trained separately for each new expression disregarding information of previously seen equations. Our model can instead leverage knowledge from the entire training set.

3 Discussion

This work proposes a novel approach to perform SR tasks in an end-to-end manner, i.e., from numbers directly to symbolic expressions. The results show that our seq2seq model, originally proposed in the context of NLP applications, can also be effective in the domain of SR. We also introduce a new library to programmatically generate datasets of equations and facilitate benchmarking. Future work will be mainly focused on four research lines: **1**) Extending our method to handle multivariate equations and not relying on fixed support points introduces scalability issues. To cope with them, a possible approach could be to cast our method into a set-to-sequence framework and leverage recent advances in this domain [Choy et al., 2019, 2020]. **2**) Recursively refining the network's output according to the signal obtained from a suitable combination of symbolic and numerical losses could improve predictive performances. A Reinforcement Learning approach could be helpful to this extent; **3**) Leveraging more recent advances in the NLP literature, such as Transformers [Vaswani et al., 2017], could also provide additional benefits. **4**) Exploiting our freedom to generate arbitrarily large datasets will likely improve the performance of our method.

References

L. Biggio, T. Bendinelli, A. Neitz, A. Lucchi, and G. Parascandolo. Neural symbolic regression that scales. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 936–945. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/biggio21a.html`.

C. Choy, J. Gwak, and S. Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3075–3084, 2019.

C. Choy, J. Lee, R. Ranftl, J. Park, and V. Koltun. High-dimensional convolutional networks for geometric pattern recognition, 2020.

R. Fletcher. *Practical Methods of Optimization; (2nd Ed.)*. Wiley-Interscience, USA, 1987. ISBN 0471915475.

J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin. Convolutional sequence to sequence learning, 2017.

J. Koza. Genetic programming - on the programming of computers by means of natural selection. In *Complex adaptive systems*, 1993.

B. K. Petersen. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients, 2020.

S. S. Sahoo, C. H. Lampert, and G. Martius. Learning equations for extrapolation and control, 2018.

M. D. Schmidt and H. Lipson. Distilling free-form natural laws from experimental data. *Science*, 324:81 – 85, 2009.

R. Sutton. The bitter lesson. *Incomplete Ideas (blog), March*, 13:12, 2019.

S.-M. Udrescu and M. Tegmark. Ai feynman: a physics-inspired method for symbolic regression, 2020.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

4 Appendix

4.1 Expression generation

By the term "expression", we refer to a finite sequence of symbols with a well-defined meaning from a mathematical point of view. Syntactically meaningful expressions are generated programmatically by our developed library. Tables 1 shows all the symbols making up the vocabulary we used in our experiments.

Symbol	Integer Id	Symbol	Integer Id
x	1	Start	12
sin	2	End	13
exp	3	1	14
log	4	2	15
pow	5	3	16
+	6	4	17
/	7	5	18
*	8	6	19
(	9	7	20
)	10	8	21
e	11	9	22

Table 1: Symbols available for expression generation and their corresponding integer tokens

It is important to note that there might be multiple ways to write the same mathematical expression. Consider, for example, "$\sin(x)+\cos(x)^2+\cos(x)$": one can reverse the order of the terms in the sum (e.g. "$\cos(x)^2+\sin(x)+\cos(x)$") or group some terms together (e.g. "$\sin(x)+\cos(x)(\cos(x)+1)$") and the resulting equations will have exactly the same mathematical information content. Furthermore, one can exploit trigonometric identities and rewrite the original equation as "$\sin(x)+1-\sin(x)^2+\cos(x)$". These one-to-many relationships hinder our neural network performance by introducing ambiguities into the mapping between numbers and symbols. To counteract this issue, when generating a batch of expressions used for training, we adopt the following additional rules:

1. We generate expressions that cannot be simplified by summation (i.e. library won't generate "$2\sin(x)+3\sin(x)$" , while it may "$5\sin(x)$").

2. We generate exclusively fully-expanded equations, where no elements are collected (e.g. "$\sin(x)(\sin(x)+1)$" is not generated, while "$\sin(x)^2+\sin(x)$" is).

3. We keep order consistent by alphabetically sorting operands of summations and multiplications.

4.2 Training dataset

In this section, we provide some additional details about our training dataset. Table 2 shows four data instances extracted form the dataset. We generate the data starting from an expression, f_i, containing numerical constants. We evaluate such expression on a fixed set of numerical support points, $\mathbf{X}$, to obtain the input of our model, $\mathbf{Y}_i$. The goal of the network is to predict the symbolic skeleton, g_i, associated with the original expression f_i. Note that there might be several instances of the same symbolic skeletons, each associated with different evaluation points. This aspect is illustrated in Fig. 5, were multiple instances of the same symbolic skeleton ($\sin(x^3)$) are shown. Overall, our training set contains 50.000 numerically different inputs $\mathbf{Y}_i$ and ~ 17.000 unique symbolic skeletons, g_i.

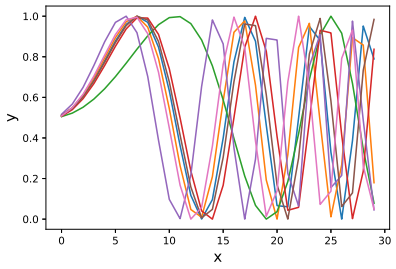

Figure 5: Several instances of the expression $y = sin(x^2)$ corresponding to various internal constant samples.

		Input	Output
Equation with constants (f_i)	Support ($\mathbf{X}$)	Evaluation Points ($\mathbf{Y}_i$)	Symbolic Equation (g_i)
$5.4x^2 + 3.2\log(1.4x)$	$0.1, 0.2, ..., 2.9, 3$	$-6.24, -3.86, ..., 46.71, 49.49$	$x^2 + \log(x)$
$\exp(1.4\sin(0.9x^3))$	$0.1, 0.2, ..., 2.9, 3$	$1, 1.01, ..., 3.01, 1.06$	$\exp(\sin(x^3)$
$4.3x^6 + 1.1x^5 + 0.32x^4$	$0.1, 0.2, ..., 2.9, 3$	$0, 0.01, ..., 2309, 2837$	$x^6 + x^5 + x^4$
$2.1x^2 + 0.8\log(1.6x)$	$0.1, 0.2, ..., 2.9, 3$	$-1.45, 0.83, ..., 17.66, 18.89$	$x^2 + \log(x)$

Table 2: The network is fed with a 30-dimension vector, $\mathbf{Y}_i$ of evaluation points. The symbolic skeleton, g_i, is the target output. The first and last row represent two training point with same symbolic skeleton but with different evaluation points. In total we have 50.000 evaluation inputs and ~ 17.000 unique symbolic equations.

4.3 Additional Results

In this section, we report further experimental results not included in the main body of the manuscript due to space constraints. First, we show how the BFGS optmizer acts in combination with our seq2seq neural network to successfully retrieve the constants present in the original expression. Then, we report additional results related to the quantitative analysis performed in section 2.

Visualizations Fig. 6 provides three examples where the combination of our model and the BFGS optimizer manages to retrieve the exact expression of three equations with internal and external constants.

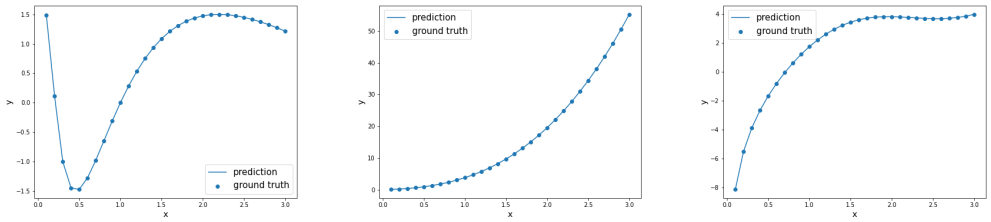

Figure 6: (left) $y = 1.5\sin(2\log(x))$; (middle) $1.33x^3 + 2x^2 + 0.44x$; (right) $2\sin(x + 0.2)^2 + 1.2\log(x^3)$

In all the above examples, the neural network outputs the correct symbolic skeleton and the BFGS optimizer successfully finds the numerical constants.

Quantitative Analysis Table 3 shows the percentage of expressions for which the RMSE extrapolation error is below three threshold levels for our seq2seq model and FFNN3.

	First Dataset			Second Dataset		
	0.05 Error	0.1 Error	0.2 Error	0.05 Error	0.1 Error	0.2 Error
SeqToSeq Model	**15.3%**	**22.34%**	**37.09%**	**8.70%**	**14.31%**	**27.60%**
FFNN3	8.80%	15.80%	26.30%	4.43%	10.34%	19.09%

Table 3: For each dataset, performances are divided into three categories: percentage of equation under (i) 0.05 RMSE, (ii) 0.1 RMSE and (iii) 0.2 RMSE.

As expected, for both models the percentages increase as the threshold grows. However, the seq2seq model consistently outperforms FFNN3 on both datasets.