Symbolic expression generation via Variational Auto-Encoder

Anonymous Authors[1]

Abstract

There are many problems in physics, biology, and other natural sciences in which symbolic regression can provide valuable insights and discover new laws of nature. A widespread Deep Neural Networks do not provide interpretable solutions. Meanwhile, symbolic expressions give us a clear relation between observations and the target variable. However, at the moment, there is no dominant solution for the symbolic regression task, and we aim to reduce this gap with our algorithm. In this work, we propose a novel deep learning framework for symbolic expression generation via variational autoencoder (VAE). In a nutshell, we suggest using a VAE to generate mathematical expressions, and our training strategy forces generated formulas to fit a given dataset. Our framework allows encoding apriori knowledge of the formulas into fast-check predicates that speed up the optimization process. We compare our method to modern symbolic regression benchmarks and show that our method outperforms the competitors on most of the tasks.

1. Introduction

The Discovery of new laws from experimental observations may seem to be an old and long-forgotten topic at first sight. Indeed, in the age of data-driven science, neural networks can easily fit a pretty complicated dependency. However, generalization and interpretation of those networks usually leave much to be desired. For such phenomena as a biotech reaction or molecular dynamics potentials or epidemic spread, a learning algorithm that represent the dependency between target property and dependent characteristic in a simplistic and human-conceivable, such as usual formulas, is invaluable. Such examples are everywhere. Also, several methods enhancing the so-called symbolic regression approach have been developed recently (Udrescu & Tegmark, 2020). Symbolic regression is akin to a simple regression, as it fits experimental data. However, symbolic regression tries to find suitable and functional feature transformations represented by a computational graph over the original feature vector. Such graphs impose additional complications while optimizing those using gradient descent-based approaches. Another difficulty such methods are regularly facing is due to the inherent noisiness of experimental measurements. Hence, each algorithm should ideally provide theoretical or empirical guarantees to the noise level robustness.

One of your key contribution is the design of a novel process of the symbolic regression training SEGVAE[1] that differs from the current state-of-the-art approaches: higher noise stability, higher data efficiency, and adjustability of the priors for the symbolic expression to the physics intuition of the user. Often scientists do know the frame of the searching formula (i.g limits and approximations) and laws that data have to follow. We introduce a predicate mechanism for formula search and the ability to implement known conversation laws. These features are essential for processing experimental data.

The structure of this paper is the following: section 2 contains an overview of prior art methods, section 3 contains a description of our method. All experiments with comparing performance are presented in section 4. Section 5 concludes the paper.

2. Related work

The goal of using artificial intelligence to help with discovering the scientific laws underlying experimental data has been pursued in several works. Some of these works assume prior knowledge of the mystery environments of interest. However, the ones most relevant to our study are the ones that minimize any assumptions.

The majority of traditional approaches generally exploit genetic programming (Searson et al., 2010). The most successful one of these is the commercial software Eureqa ((Schmidt & Lipson, 2009)), which was developed more than ten years ago and still holds one of the leading positions in the field.

There are several recent works dedicated to recovering physical laws in symbolic form. (Udrescu & Tegmark, 2020) introduce an AI Feynman algorithm and further improved in (Udrescu et al., 2020) AI Feynman 2.0, which

[1]The code will be released after the blind review.

uses a) physics-inspired deep learning strategies, b) dimensional analysis like search for symmetries, separability and alike, c) brute forces the simplified equation that recovers physical equations from experimental data. While this algorithm does a good job simplifying expressions, it struggles to recover expressions that could not be simplified enough.

Interesting and intuitively easy approach was demonstrated in (Martius & Lampert, 2016) later method have been updated in (Sahoo et al., 2018) and in it's latest version (Werner et al., 2021). The authors proposed architecture similar to multilayer perceptron (MLP) but the key difference is addition layer of set predefined functions from vocabulary. The authors claim the efficiency of such an approach over MPL neural network in out of domain region, thus good extrapolation capabilities have been demonstrated. But unfortunately, the authors did not present results of comparison with other existing methods on common datasets.

Another deep learning approach to symbolic regression is introduced by (Petersen et al., 2021). The authors present a gradient-based approach for symbolic regression based on reinforcement learning, which they call deep symbolic regression (DSR). DSR consists of a recurrent neural network that outputs a distribution over mathematical formulas. This network is used to sample equations, which afterward will be evaluated based on the given dataset, and then the evaluation result will be used to further improve the distribution over mathematical formulas making the better expressions more probable. Recently, the DSR method has been updated by introducing a genetic programming component, which gave significant improvements on several benchmark tests. The latest algorithm version performs better than others to the best of our knowledge. Thus, we are using it ((Mundhenk et al., 2021)) as a baseline known as DSO.

SciNet ((Iten et al., 2020)) approach is inspired by human thinking along a physical modeling process. Just like human physicists do not rely on actual observations but rather on their compressed representation to make some theoretical conclusions, SciNet encodes the experimental data to a latent representation that stores different physical parameters and uses this representation to answer specific questions about the underlying physical system. Undoubtedly SciNet is successful at learning relevant physical concepts. However, its goals are very different from ours: it does not recover the laws in symbolic form but uses a neural network to model them.

In paper on Neural-Symbolic Regression that Scales (Biggio et al., 2021) authors propose to use pre-trained transformers (Vaswani et al., 2017) to predict symbolic expression. The network consists of transformer encoder and transformer decoder trained on generated formulas and BFGS algorithm for constants optimization. They compare transformer results to DSR (previous version of DSO) as a baseline. Despite good evaluation time, results quality is lower than DSR on the Nguyen dataset.

Our method is akin to the work (Bowman et al., 2016). It adapts the variational autoencoder by using LSTM RNNs for both encoder and decoder. Thus, forming a sequence autoencoder with the Gaussian prior acting as a regularizer on the hidden code. The proposed generative model incorporates distributed latent representations of entire sentences. By examining paths through this latent space, it is possible to generate coherent novel sentences that interpolate between known sentences.

Algorithm 1 SEGVAE: Overall algorithm

Input: data (X, y), library of operators $\mathcal{L}$, $N_{epochs} > 0$
$F_{pre-train}$ = **GeneratePretrainFormulas**($\mathcal{L}$)
$F_{pre-train}$ = [f **if** $predicate(f)$ **for** f **in** $F_{pre-train}$]
vae.**train**($F_{pre-train}$)

for $i = 1$ **to** N_{epochs} **do**
  F_{train} = vae.**sample**($batch\_size$);
  F_{train} = [f **if** $predicate(f)$ **for** f **in** F_{train}]
  $\overline{mse} = \frac{1}{|X|}\sum_j (F_{train}(X_j) - y)^2$
  if $F_{train}(x)$ = NaN or out of Y domain **discard** $F_{train}(x)$
  if simplify F_{train} = $simplify(F_{train})$;
  bbf = bbf.**update**($F_{train}, \overline{mse}$)
  vae.**train**(bbf)
end for
Result: $bbf.pareto\_front$

3. SEGVAE

This section introduces the Symbolic expression generation via Variational Auto-Encoder (SEGVAE) algorithm. In a nutshell, our architecture is a Variational Autoencoder (Kingma & Welling, 2013) in which the encoder and decoder are based on recurrent neural networks. We describe our implementation of this architecture in section 3.1 and the training procedures in 3.2

3.1. Architecture

SEGVAE generates formulas in a sequential manner. It is possible due to the one-to-one correspondence between sequences of tokens and formulas. Each token can be one of three types: input variables, constants, and operators. In this paper our typical library of operators is ['add', 'sub', 'mul', 'div', 'sin', 'cos', 'log', 'exp']. The number of variables X and constants depends on a task. We discuss

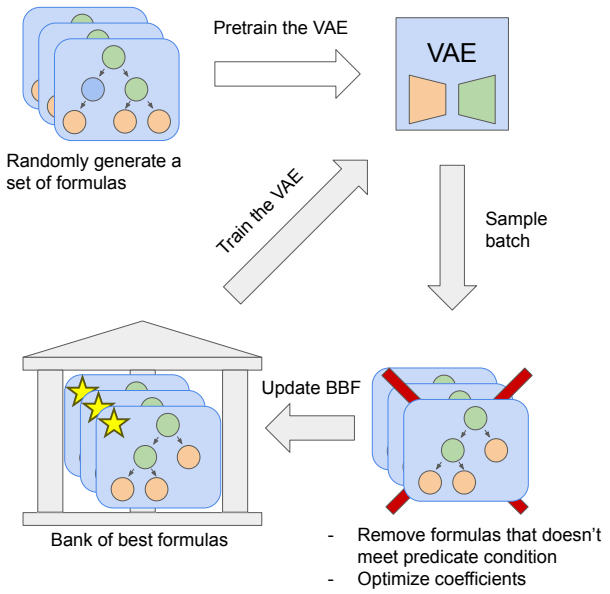

Figure 1. The SEGVAE algorithm training scheme. Pre-training stage and main training cycle.

the way to deal with constants in the subsection below. In order to represent formulas as sequences, we use normal Polish notation in which operators precede their operands. The main advantage of Polish notation over the conventional one is that Polish representation is unambiguous and does not require brackets.

Variational autoencoder. VAE is a generative encoder-decoder based latent variable model. Given an observation space $\mathcal{X}$ with a distribution $p(x)$ the model's encoder maps it into a latent space $\mathcal{Z}$ with a distribution $p(z)$, and the model's decoder maps $\mathcal{Z}$ back into the observation space $\mathcal{X}$. Let m - dimensionality of the latent space, μ_i, σ_i^2 - the i^{th} components of vectors $\mu(x), \sigma^2(x)$. If $p(z) = \mathcal{N}(0, 1)$ and $q_{\theta_E}(z|x) = \mathcal{N}(\mu(x), \sigma^2(x))$, then the objective takes the following form:

$$-KL(q_{\theta_E}(z|x)||p(z)) + \mathbb{E}_{q_{\theta_E}(z|x)} \log p_{\theta_D}(x|z)$$

This approach allows the model to decode plausible equations from every point in the latent space that has a reasonable probability under the prior.

3.2. Training

Here we summarize the details of our training protocol, which consists of two steps pre-training and actual training.

Pre-training. This step allows the model to memorize the general formula structure and generate valid formulas afterward. Firstly, we randomly sample sequences of tokens from given library $\mathcal{L}$. However, uniformly sampling expressions with n internal nodes is not a simple task. Naive algorithms tend to favor specific kinds of expressions. So, we follow the data generator technique introduced in (Lample & Charton, 2020). Secondly, we choose only statements that meet our predicate conditions and thus create a pre-train dataset. Also, we add to the pre-train dataset a set of generalized formulas that commonly appear in natural science. Then the variational autoencoder is trained on these formulas. As a result of training, the vast majority of generated formulas in the next steps of the algorithm are valid formulas, so for the sake of simplicity, we can safely ignore invalid formulas in the following stages. Note that this step does not depend on the target task at all, so we do the pre-training once for each library $\mathcal{L}$.

Training. When the model can generate valid formulas, it is time to teach it to generate valid formulas which describe a given environment or system under exploration. Firstly, a batch of formulas is sampled using the variational autoencoder for each epoch. Then all the duplicates and invalid formulas are removed. Secondly, each formula f is evaluated on the dataset $\mathcal{D} = (X_d, y_d)$ in terms of mean squared error between $f(X_d)$ and Y_d:

$$error(f) = \frac{1}{|\mathcal{D}|} \sum_{x,y \in \mathcal{D}} (f(x) - y)^2$$

Then the P percent of the formulas with the smallest mean squared error are candidates to be saved to the bank of the best formulas (BBF). First, those candidates needs to be checked on correct domain of definition and values, by default X domain is (X_{min}, X_{max}) and Y domain is (Y_{min}, Y_{max}) presented in the dataset. To do this we evaluate function f results on points sampled from uniform distribution on X domain, if the result is NaN or Y value is out of domain region we discard this formula. Second, if needed, formulas can be simplified using $sympy$ library prior to save it in BBF. This bank keeps formulas from the last N epochs. Our typical value for both hyperparameters P and N is 20 and 5. Finally, the VAE is fine-tuned on formulas from the BBF.

Constants. There are two ways of dealing with constants in resulting formulae. The first method supposes that all constants are incorporated in a library $\mathcal{L}$. In this case, constants are regular tokens, and we do not need any modifications to our algorithm. The main drawback of this approach is the lack of expressiveness. However, this significantly helps the algorithm to avoid overfitting to noisy data.

The second method is generating placeholders for future constants by including token $'const'$ to the library $\mathcal{L}$. Then, after the sampling stage, each placeholder is replaced by a parameter which we minimize the mean squared error between $f(X_d, consts)$ and Y_d by BFGS optimization algorithm ((Fletcher, 1987)).

Predicates. It is relatively straightforward to incorporate

some prior knowledge about the target formula in the proposed algorithm framework. At the training stage, while we choose the best-sampled formulas by VAE, we ignore formulas that do not meet prior conditions regardless of their metrics. This technique reduces the search space, which significantly helps SEGVAE find correct equations, as demonstrated in the experiment section below.

Imagine we know that our physical system is described by the relation of polynomials, and the task is to determine that exact dependency. One way of helping the model is to reduce search space by shrinking the library of operators, e.g., excluding trigonometrical operations. However, one can go further, creating a condition on possible positions for remaining tokens. In our case ['div'] operator should be located only in the first place. Eventually model will learn not to generate formulas that violate prior conditions. We will demonstrate algorithm work results on chosen list of formulas and predicates in this paper below. The list of formulas and related predicates are listed in table 3.

3.3. Inference

Once the VAE model is trained, one can sample a batch of candidate expressions that are supposed to fit the given dataset. Overwhelmed formulas that describe the dataset best in terms of mean squared error could be overfitted to the noise in the data. So, there is a trade-off between error and complexity. In order to choose final expressions, we evaluate the complexity of each equation as:

$$C(t) = \sum_{i}^{T} c(\gamma_i)$$

where c is complexity of given token, which is equal to one for all input variables, constants and operators ['add', 'mul', 'sub'], c is two for ['div'], three for ['sin', 'cos'] and finally four for ['log', 'exp']. Then we use this C and error values to identify Pareto-front and allow user to pick those formulas that satisfy her needs.

4. Results and discussion

This section reports comparison results between our approach and state-of-the-art symbolic regression packages. We took the bests of our knowledge algorithms, namely Deep Symbolic Optimization (DSO)[1]. We have used Nguyen 1-12 formulas (table 1) and formulas listed in table 3 to generate datasets. We compare SEGVAE results with outputs of DSO algorithm as available from github repository.

Datasets. Each symbolic regression task corresponds to a table of numbers, those rows are of the form $x_1, ..., x_n, y$, where $y = f(x_1, ..., x_n)$. The task is to discover the correct symbolic expression f.

- **Nguyen.** Nguyen benchmark is commonly used as symbolic regression benchmark. Nguyen dataset consists of 12 formulas. We used the same dataset as in Deep Symbolic Regression (DSR) paper and its updates, in two variations of 20 and 200 points (see table 1). To check the method's robustness, we add Gaussian noise proportional to y.
- **Livermore.** New benchmark dataset from the same group of authors (DSR). Livermore consists of 22 formulas, but we do not take all of them as a benchmark. The reason is the large formulas' fraction similarity in this dataset to the Nguyen dataset.

Ablation studies. As was described above, SEGVAE has many parameters such as number of layers, sampled formulas in pretrained step, number of an epoch, and a maximum length of generated expression. To find optimal parameters, we have performed ablation study. As a baseline to tune SEGVAE parameters, we have used a subset of the Ngyuen dataset, namely Ngyuen-4,5,9,10. First, we concluded that a maximum expression length of 30 is perfectly balanced in terms of model expressibility and model stability. Secondly, we found that hidden dimensionality in the range of 64 to 256 does not affect the recovery rate of 50%. By recovery here, we mean exact symbolic equivalence of the suggested expression to the original formula. For example, we initialize the algorithm 100 times with different random seeds for a given dataset generated by some formula. Out of 100 runs, 80 correctly represented the initial formula. In this case recovery rate is 80%. Nevertheless, the recovery rate depends on latent space, as presented in table 2. We have found that latent configuration space of size 128 with 128 hidden units to be optimal for this study. Another critical observation is the SEGVAE's recovery rate dependence on the Library selection. We checked the dependence of the recovery rate on the number of tokens in the library for the DSO and SEGVAE algorithms. Small library size may be why the algorithm cannot find a correct formula. It is simply because not enough tokens are available to describe a formula. On the other hand, an over-inflated library exponentially increases algorithm search space for limited search iteration numbers. Thus, choosing excessive library contents may be a reason for a miserable formula reconstruction. A scientist has some prior knowledge about unknown yet dependency in an actual research process. Thus, we can use both predicates and a task-related library to simulate an actual searching formula situation.

Noise in data. Noisy data were created by adding Gaussian noise with zero mean and standard deviation propor-

[1] https://github.com/brendenpetersen/deep-symbolic-optimization

Name	Expression	Dataset I	Dataset II	Library
Nguyen-1	$x_1^3 + x_1^2 + x_1$	$U(-1, 1, 20)$	$U(-1, 1, 200)$	$L_0 + [0.5, 1, -1, 2]$
Nguyen-2	$x_1^4 + x_1^3 + x_1^2 + x_1$	$U(-1, 1, 20)$	$U(-1, 1, 200)$	$L_0 + [0.5, 1, -1, 2]$
Nguyen-3	$x_1^5 + x_1^4 + x_1^3 + x_1^2 + x_1$	$U(-1, 1, 20)$	$U(-1, 1, 200)$	$L_0 + [0.5, 1, -1, 2]$
Nguyen-4	$x_1^6 + x_1^5 + x_1^4 + x_1^3 + x_1^2 + x_1$	$U(-1, 1, 20)$	$U(-1, 1, 200)$	$L_0 + [0.5, 1, -1, 2]$
Nguyen-5	$sin(x_1^2)cos(x_1) - 1$	$U(-1, 1, 20)$	$U(-1, 1, 200)$	$L_0 + [0.5, 1, -1, 2]$
Nguyen-6	$sin(x_1) + sin(x_1 + x_1^2)$	$U(-1, 1, 20)$	$U(-1, 1, 200)$	$L_0 + [0.5, 1, -1, 2]$
Nguyen-7	$log(x_1 + 1) + log(x_1^2 + 1)$	$U(0, 2, 20)$	$U(0, 2, 200)$	$L_0 + [0.5, 1, -1, 2]$
Nguyen-8	$sqrt(x_1)$	$U(0, 4, 20)$	$U(0, 4, 200)$	$L_0 + [0.5, 1, -1, 2]$
Nguyen-9	$sin(x_1) + sin(x_2^2)$	$U(0, 1, 20)$	$U(0, 1, 200)$	$L_0 + [0.5, 1, -1, 2]$
Nguyen-10	$2sin(x_1)cos(x_2)$	$U(0, 1, 20)$	$U(0, 1, 200)$	$L_0 + [0.5, 1, -1, 2]$
Nguyen-11	$x_1^{x_2}$	$U(0, 1, 20)$	$U(0, 1, 200)$	$L_0 + [0.5, 1, -1, 2]$
Nguyen-12	$x_1^4 - x_1^3 + 0.5x_2^2 - x_2$	$U(0, 1, 20)$	$U(0, 1, 200)$	$L_0 + [pow(x, N), 0.5, N, e, pi]$

Table 1. Nguyen Dataset. Variables are denoted as x_1 and x_2. Variables are uniformly sampled, U(a, b, c) denotes c times sampling between a and b for each input variable, N natural numbers, $L_0 = [add, sub, mul, div, exp, ln, sin, cos]$.

Latent space dimension	8	16	32	64	128	256
Mean recovery rate	50	58	60	65	75	75

Table 2. Mean recovery rate dependence from latent space dimension with fixed hidden dimensions on sub Nguyen Dataset.

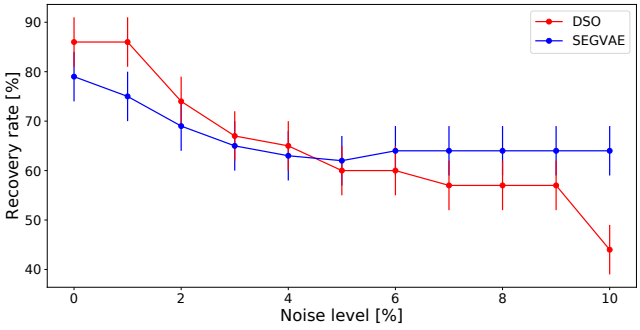

Figure 2. Average recovery rates of SEGVAE (blue) and DSO (red) algorithms on Ngyuen dataset (dataset I) with error bars.

tional to the root-mean-square of the dependent variable y. To check models' robustness, we present averaged recovery rates as a function of the noise from 0% (noiseless) to the maximum 10%. The main difficulty of regression with noise is that the model tendency to overfit the data. Sometimes increasing the number of points in a dataset may help.

Experiments. We have evaluated the SEGVAE algorithm on the most commonly used Nguyen benchmark, consisting of 12 formulas. We compare and evaluate our models on two variants of Nguyen datasets. For the first ($Dataset\ I$) variant, we sample only 20 uniformly distributed points, as shown in table 1. For the second ($Dataset\ II$) variant, we sample 200 uniformly distributed points. We have used the same hyperparameters in SEGVAE for all runs. The number of examined expressions is set to 2 million per run, the same as in the DSR paper ((Petersen et al., 2021)).

Comparison of DSO and SEGVAE on Dataset I is presented in Figure 1 as the dependence of averaged recovery rates over noise level. Red and blue colors denote outputs of DSO and SEGVAE algorithms correspondingly. Essential to reiterate that by recovery, we mean at least one exact match with the wanted formula in Pareto front output.

SEGVAE and DSO show similar average recovery rates (ARR) with small noise levels. We average the recovery rate overall Nguyen formulas at a given nose level to compute ARR. With increasing noise levels, DSO recovery rates slowly go down. On the other hand, SEGVAE demonstrates good recovery rates stability until the largest noise level of 10% with a recovery rate of 70%. The difference becomes visible at high noise levels where SEGVAE slightly outperforms DSO, 70% vs. 45% at maximum noise level. SEGVAE algorithm demonstrates higher noise stability on this dataset even without prior knowledge.

In general, recovery rates improve with increasing dataset size. Both algorithms demonstrate great Evaluation results on "large" Dataset II. DSO shows on the first 11 Nguyen formulas but still can not solve Nguyen 12 while SEGVAE demonstrates recovery rare of 50%. We went further in algorithms comparison and selected formulas from the original DSO paper, where the DSO algorithm shows daunting results. These formulas are presented in table 3. This time we used specific predicates and token libraries to reveal these formulas and, at the same time to demonstrate the power of our approach. The results of this comparison are summarised in the same way through the ARR in Figure 3. The detailed results on noiseless data are presented in table 4.

Formula name	Formula	Predicate	Dataset	Library
Nguyen-12	$x_1^4 - x_1^3 + 0.5x_2^2 - x_2$	$f(x_1) + g(x_2)$	$U(0, 10, 200)$	$L_0 + [pow(x, N), 0.5, N, e, pi]$
Neat-8	$exp(-(x_1 - 1)^2)/(1.2 + (x_2 - 2.5)^2)$	$exp(f(x_1))/(g(x_2))$	$U(0.3, 4, 100)$	$L_0 + [0.5, 1, -1, 2]$
Neat-9	$1/(1 + x_1^4) + 1/(1 + x_2^4)$	$1/f(x_1) + 1/g(x_2)$	$U(-5, 5, 21)$	$L_0 + [0.5, 1, -1, 2]$
Livermore-5	$x_1^4 - x_1^3 + x_1^2 - x_2$	$f(x_1) + g(x_2)$	$U(0, 1, 20)$	$L_0 + [0.5, 1, -1, 2]$
Livermore-7	$0.5exp(x_1) - 0.5exp(-x_1)$	$f(x_1) - 1/g(x_1)$	$U(-1, 1, 20)$	$L_0 + [0.5, 1, -1, 2]$
Livermore-8	$0.5exp(x_1) + 0.5exp(-x_1)$	$f(x_1) + 1/g(x_1)$	$U(-1, 1, 20)$	$L_0 + [0.5, 1, -1, 2]$
Livermore-10	$6sin(x_1)cos(x_2)$	$const * f(x_1) * g(x_2)$	$U(0, 1, 20)$	$L_0 + [pow(x, N), 0.5, N, e, pi]$
Livermore-17	$4sin(x_1)cos(x_2)$	$const * f(x_1) * g(x_2)$	$U(0, 1, 20)$	$L_0 + [pow(x, N), 0.5, N, e, pi]$
Livermore-22	$exp(-0.5x_1^2)$	$exp(f(x_1))$	$U(0, 1, 20)$	$L_0 + [0.5, 1, -1, 2]$
R-2*	$(x_1 + 1)^3/(x_1^2 - x_1 + 1)$	$f(x_1)/g(x_1)$	$U(-10, 10, 20)$	$L_0 + [0.5, 1, -1, 2]$

Table 3. List of formulas and used predicates. Variables are denoted as x and y. Variables are uniformly sampled, U(a, b, c) denotes c times sampling between a and b for each input variable, N natural numbers, $L_0 = [add, sub, mul, div, exp, ln, sin, cos]$.

Formula name	$N-12$	$Neat-8$	$Neat-9$	$L-5$	$L-7$	$L-8$	$L-10$	$L-17$	$L-22$	$R-2*$
SEGVAE	100%	0%	0%	60%	20%	0%	100%	100%	100%	0%
DSR	0%	0%	0%	80%	0%	0%	63%	57%	84%	4%

Table 4. SEGVAE and DSO algorithms recovery rates comparison on selected noiseless dataset.

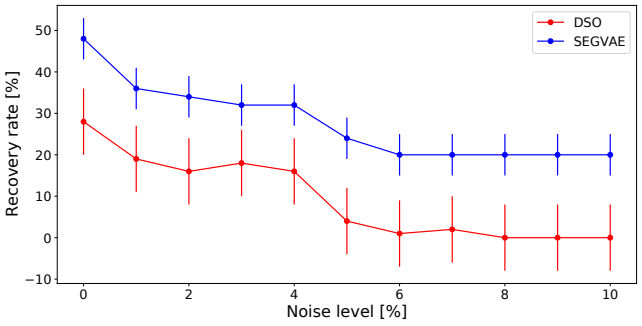

Figure 3. Average recovery rates of SEGVAE (blue) and DSO (red) algorithms base on formulas listed in table 3 with error bars.

It often appears that scientists already know the general form of the object of interest. Thus, it would be natural to add predicates to let SEGVAE search formulas in a specific domain. We have seen that our VAE based algorithm gives a similar result to the recent DSO results on the Nguyen dataset without any prior knowledge. However, there are many equations in which DSO is not accurate enough. To demonstrate the power of predicates in SEGVAE, we took predicates as listed in Table 3 and compared our results to DSO. The comparison results on noiseless data are presented in Table 4. We see that the SEGVAE approach with predicates demonstrates a superior recovery rate than DSO, especially on noiseless data. More detailed results on noiseless are presented in table 4. The proper predicates and optimal library play a crucial role in this case. In reality, scientists do not know the exact functional form of the studied effect. We can not use the recovery rate as a benchmark in this case. The only benchmark we can trust is MSE and formula shape or predicates that carry common scientific sense.

5. Conclusion

We introduce a novel algorithm, SEGVAE, for searching symbolic representation of functional dependence from dependent variables based on the VAE generative model. As a benchmark, we have used a set of formulas introduced in the DSO paper, namely the Nguyen and the Livermore datasets. Our approach has the flexibility of formulating a priory physical knowledge in the form of a) library of functions, b) pre-training functions, and c) selection predicates used for pruning incorrectly generated expressions. Besides the sole accuracy of the method, we have focused on the algorithm's performance in realistic noisy environments. Symbolic regression approaches excel significantly compared to deep learning models where interpretability is paramount. Thus, the DSO - already a successful approach, can be easily applied for many cases in such scientific branches as material sciences, biotechnologies, and astrophysics, to mention a few.

We have systematically compared SEGVAE with DSO and shown superior performance of our approach, thus outperforming Eureqa, Wolfram, and alike, which DSO has dominated before. For the scarce-data regime and high-noise regimes, the SEGVAE significantly outperforms the competitors.

We have pointed out the importance of the library size and showed that SEGVAE discovered formulas unreachable for DSO thanks to using predicates. Since experimental data usually contains noise and some prior knowledge on the

functional dependency is usually available, the SEGVAE benefits can be easily seen from an application point of view. Therefore, our model can come in handy in practical cases where interpretable symbolic solutions are needed to understand processes underlying experimental observations.

References

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

Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A., Jozefowicz, R., and Bengio, S. Generating sentences from a continuous space. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pp. 10–21, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/K16-1002. URL https://aclanthology.org/K16-1002.

Fletcher, R. *Practical Methods of Optimization*. John Wiley & Sons, Ltd, 1987. ISBN 9781118723203. doi: https://doi.org/10.1002/9781118723203.fmatter.

Iten, R., Metger, T., Wilming, H., del Rio, L., and Renner, R. Discovering physical concepts with neural networks. *Phys. Rev. Lett.*, 124:010508, Jan 2020. doi: 10.1103/PhysRevLett.124.010508. URL https://link.aps.org/doi/10.1103/PhysRevLett.124.010508.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Lample, G. and Charton, F. Deep learning for symbolic mathematics. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1eZYeHFDS.

Martius, G. and Lampert, C. H. Extrapolation and learning equations. *CoRR*, abs/1610.02995, 2016. URL http://arxiv.org/abs/1610.02995.

Mundhenk, T. N., Landajuela, M., Glatt, R., Santiago, C. P., Faissol, D. M., and Petersen, B. K. Symbolic regression via neural-guided genetic programming population seeding. In *35th Conference on Neural Information Processing Systems (NeurIPS 2021)*, 2021.

Petersen, B. K., Larma, M. L., Mundhenk, T. N., Santiago, C. P., Kim, S. K., and Kim, J. T. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=m5Qsh0kBQG.

Sahoo, S., Lampert, C., and Martius, G. Learning equations for extrapolation and control. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4442–4450. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/sahoo18a.html.

Schmidt, M. and Lipson, H. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009. doi: 10.1126/science.1165893. URL https://www.science.org/doi/abs/10.1126/science.1165893.

Searson, D. P., Leahy, D. E., and Willis, M. J. Gptips: An open source genetic programming toolbox for multigene symbolic regression. 2010.

Udrescu, S.-M. and Tegmark, M. Ai feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16), 2020. doi: 10.1126/sciadv.aay2631. URL https://advances.sciencemag.org/content/6/16/eaay2631.

Udrescu, S.-M., Tan, A., Feng, J., Neto, O., Wu, T., and Tegmark, M. Ai feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity. In *Advances in Neural Information Processing Systems 33 pre-proceedings (NeurIPS 2020)*, 12/2020 2020.

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. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Werner, M., Junginger, A., Hennig, P., and Martius, G. Informed equation learning, 2021. URL https://arxiv.org/abs/2105.06331.