Sparse Interpretable Deep Learning with LIES Networks for Symbolic Regression

Mansooreh Montazerin
University of Southern California

Majd Al Aawar
University of Southern California

Antonio Ortega
University of Southern California

Ajitesh Srivastava
University of Southern California

Abstract

Symbolic regression (SR) aims to recover interpretable closed-form expressions from data. Existing methods often rely on population-based search and autoregressive modeling which struggle with scalability and reliability. We introduce LIES (Logarithm, Identity, Exponential, Sine), a fixed neural network architecture with interpretable primitive activations optimized to model symbolic expressions. We develop a framework to extract compact formulae from LIES networks by training with an appropriate oversampling strategy and a tailored loss function to promote sparsity and to prevent gradient instability. After training, it applies additional pruning strategies to further simplify the learned expressions into compact formulae. Our experiments on SR benchmarks show that the LIES framework consistently produces sparse and accurate symbolic formulae outperforming all baselines. We also demonstrate the importance of each design component through ablation studies.

1 Introduction

Uncovering the underlying mathematical laws that govern complex systems is a fundamental goal in science and engineering. Symbolic regression (SR) provides a powerful means to achieve this by simultaneously discovering both the structure and parameters of equations, offering interpretability beyond conventional regression [6]. Its flexibility has made SR valuable across domains, from fluid mechanics to astrophysics and materials science [14, 18]. A key advantage of SR is its interpretability as resulting models are expressed in symbolic form that generalizes beyond training data [3]. However, uncovering meaningful equations is challenging due to the vast combinatorial search space where balancing accuracy, simplicity, and generalization is essential to avoid either overfitting or underfitting [8]

SR encompasses methods ranging from classical Genetic Programming (GP) to modern Deep Learning (DL) approaches such as Transformers, Graph Neural Networks, and generative models [2, 17, 5]. GP explores equations by iteratively refining candidates built from basic primitives, but its reliance on predefined operations and sequential search often yields overly complex, less interpretable expressions [2]. In contrast, DL-based methods [13, 3, 5] represent expressions as sequences, graphs, or latent samples, offering greater scalability and adaptability. However, they typically require large datasets, struggle with syntactic or semantic validity, and provide limited interpretability until symbolic decoding.

To address these challenges, we propose LIES, a feedforward neural architecture that embeds symbolic structure directly into its design while preserving interpretability. The model stacks layers of carefully chosen activation functions—logarithm (L), identity (I), exponential (E), and sine (S)—to capture

Machine Learning and the Physical Sciences Workshop, NeurIPS 2025.

fundamental operations found in natural laws. By combining sparsity-aware loss with pruning and symbolic simplification, LIES reduces symbolic regression to learning compact, sparse networks. Unlike GP-based methods, which rely on slow heuristic search and often generate overly complex equations, and DL-based methods, which require large datasets and reveal symbolic structure only after decoding, LIES embeds symbolic operations directly into its architecture, producing concise, interpretable, and data-efficient formulae.

Specifically, our main contributions are as follows: (1) We propose a novel architecture with specific symbolic activations and a modified loss function that helps avoid unstable gradients due to logarithm and exponential functions. (2) We introduce a pipeline that promotes sparsity and interpretability by combining a sparsity-aware loss function, pruning, and symbolic simplification. (3) We evaluate our method on 61 formulae from the AI Feynman dataset [16], showing strong symbolic and numerical performance, and provide an ablation study to isolate the contribution of each component in the pipeline.

Related work Symbolic regression (SR) seeks closed-form expressions that explain data, with genetic programming (GP) as the traditional backbone. Koza's pioneering work [7] established the foundation for GP-based SR, while PySR [2] represents a modern advancement with an evolve–simplify–optimize framework that yields concise and interpretable equations, especially for scientific applications. Nonetheless, GP-based methods remain computationally costly, hyperparameter-sensitive, and prone to overly complex or redundant expressions. In contrast to GP methods, DL-based SR employs neural networks for flexible expression generation, with examples including reinforcement learning with Recurrent Neural Networks [10], seq2seq models [1], and surrogate approaches such as Gaussian processes [11] and PINNs [4].

Despite these advances, GP methods still rely on heuristic search and produce overly complex formulas, while DL methods tend to operate as black boxes, requiring substantial data and offering limited symbolic interpretability or structure. To address this, we propose LIES, a feedforward network whose activations mirror symbolic operations from natural laws. By embedding symbolic structure and applying pruning with gradient-based sparsification, LIES avoids evolutionary search while yielding compact and interpretable expressions. EQL÷ [12] uses fixed symbolic activations but is limited in scope, restricting division to the output layer and failing to compactly capture common operations such as exponents, logarithms, and roots [10]. In contrast, our proposed LIES network, built solely from unary operators (L, I, E, S), acts as a universal approximator and represents a wider range of natural laws. Our framework includes sparsification, pruning, and symbolic simplification, which enable accurate learning of a large class of compact formulae.

2 Methodology

In SR, the objective is to discover a compact and interpretable expression $\hat{f}$ that approximates an unknown target function $f : \mathbb{R}^d \to \mathbb{R}$ from data samples $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ where $\hat{f}(\mathbf{x}_i) \approx y_i$. The learned expression should not only fit observed pairs $(\mathbf{x}_i, y_i)$, but also remain interpretable and generalize effectively to unseen data.

The proposed LIES architecture Unlike prior SR approaches that rely on population-based search or sequential modeling, we propose the LIES network, a fixed neural architecture designed to simplify naturally into symbolic expressions. The key hypothesis is that most natural laws can be expressed using a small set of primitives—logarithm (L), identity (I), exponential (E), and sine (S). Basic arithmetic and trigonometric functions can be represented or approximated through these primitives, enabling broad coverage of symbolic operations found in scientific laws.

The LIES architecture is a multilayer feedforward network with exactly four neurons per layer, each corresponding to one primitive activation. Fully dense residual connections are included to improve expressivity, and a constant all-1 input node substitutes for bias terms. A key property of the LIES architecture is its universal approximability(Theorem 1): by configuring its primitive activations, it can emulate standard nonlinearities (e.g., sigmoid) and replicate multilayer perceptrons, thus inheriting their approximation power (see Appendix A.1).

Theorem 1 (Universal Approximation)**.** *LIES Networks are Universal Approximators.*

The overall architecture and processing pipeline of our proposed method are illustrated in Figure 1.

Figure 1: End-to-end pipeline of the proposed framework.

Sparsity While LIES networks are easier to interpret due to the choice of activations, a dense network will have a large number of terms. Real-world formulas are compact, and therefore, we should encourage high sparsity in the network. We employ three strategies: (1) an Alternating Direction Method of Multipliers (ADMM)-based weight pruning framework, (2) node pruning to eliminate inactive neurons, and (3) gradient-based pruning guided by analyzing the sensitivity of the output to individual weights. ADMM pruning enforces sparsity by alternating weight updates and shrinkage steps that drive small weights to zero while preserving accuracy. Node pruning complements weight pruning by removing entire neurons, particularly logarithm and exponential units that may produce non-zero outputs even with zero inputs. Lastly, we perform "gradient-based pruning". We derive the following theorem that suggests that if the partial derivative of the network output with respect to a parameter remains small over an interval, then pruning it induces a negligible output change.

Theorem 2 (Gradient-based Pruning/Rounding)**.** *Let $f : \mathbb{R}^{p+d} \to \mathbb{R}$ be a function of p parameters and an input $\mathbf{x} \in \mathbb{R}^d$ (representing the variables), with continuous partial derivatives with respect to a parameter P_i. If $\left| h \cdot \frac{\partial f(P_i;\mathbf{x})}{\partial P_i} \right| < \epsilon \quad \forall P_i \in [\alpha, \alpha + h]$, $\mathbf{x}$, for some small $\epsilon > 0$, then $f(P_i = \alpha + h; \mathbf{x}) \approx f(P_i = \alpha; \mathbf{x})$.*

Using Theorem 2, we prune the weights whose effect on the output is negligible, retaining only parameters with meaningful impact. The same theorem also underpins gradient-based rounding where we replace learned constants with nearby rounded values when the predicted output change is negligible to improve readability of the derived formulae. The details are provided in Appendix A.2.

Oversampling and coefficient optimization To improve symbolic accuracy, we oversample regions of the input space where the model shows high error, encouraging refinement and better generalization. After pruning and rounding, we refine the remaining constants using least-squares regression, ensuring the final symbolic formula closely fits the data.

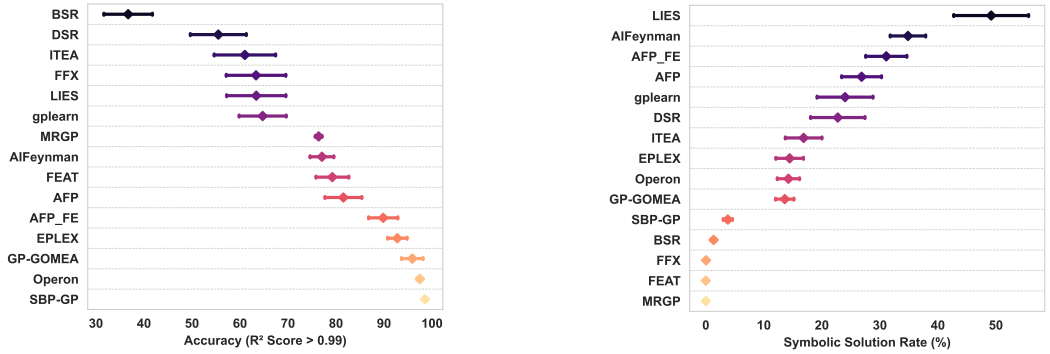
Loss function and training Our training objective combines four components: (1) exponential loss, which resembles the Mean Absolute Error (MAE) but operates in the exponential domain for log-transformed input data(2) sparsity-inducing terms from ADMM and ℓ_1 regularization to promote interpretability, (3) activation-specific penalties to handle the instability of logarithm and exponential functions near their cutoffs, and (4) a masking loss to keep activations within valid domains. Together, these terms ensure stable training, enforce sparsity, and guide the model toward interpretable symbolic expressions.

3 Experiments and results

We evaluate LIES on the AI Feynman dataset, which contains 100 physics-based equations with 1–7 variables. We focus on 61 formulas with four or fewer variables because higher-dimensional cases require deeper networks that hinder effective pruning, and some trigonometric formulas with negative inputs are unstable under the log-space transformation. For each formula, the LIES network

is configured with $n + 1$ layers (where n is the number of variables) and trained on 10% of the available data per task. All experiments are repeated three times to ensure consistency. We evaluate performance using two metrics: the R^2 score, which measures numerical accuracy of predictions, and the Symbolic Solution Rate (SSR), which reports how often the model exactly recovers a formula that is symbolically equivalent to the ground truth. All our code is publicly available at [1].

We compare LIES against the SRBench baselines [8] on the subset of 61 Feynman equations, reporting both the frequency with which models achieve $R^2 > 0.99$ (Figure 2a) and the Symbolic Solution Rate (SSR, Figure 2b). Results show that LIES consistently outperforms both GP- and DL-based methods in terms of SSR, highlighting its ability to identify correct symbolic structures rather than overfit to data. However, LIES sometimes attains slightly lower R^2 accuracy, largely due to small deviations in recovered constants—cases where the symbolic form is correct but minor constant offsets reduce test accuracy below the 0.99 threshold.



(a) Accuracy ($R^2 > 0.99$).

(b) Symbolic Solution Rate (SSR).

Figure 2: Performance of LIES vs. SRBench baselines on 61 Feynman equations.

We assess the contribution of each module in the LIES pipeline—oversampling, gradient pruning, gradient rounding, and coefficient optimization—by systematically removing them and tracking performance changes. As summarized in Table 1, every component plays a critical role: removing oversampling reduces accuracy, disabling gradient pruning leads to runtime failures, omitting rounding severely degrades sparsity and SSR, and skipping coefficient optimization causes the largest overall drop. More details are provided in the Appendix.

Table 1: Performance comparison of LIES with different configurations within a fixed deadline. Values are percentages averaged across trials.

Components	Sym Sol True	Sym Sol False	OOT	$R^2 > 0.99$
Main Pipeline	98.9	0	1.1	88.9
w/o Oversampling	70	20	10	66.7
w/o Gradient Pruning	0	0	100	N/A
w/o Gradient Rounding	34.4	54.5	11.1	27.8
w/o Optimization	20	76.7	3.3	23.3

Note. "Sym Sol True" = Correct symbolic solution, "Sym Sol False" = Incorrect symbolic solution, "OOT" = out-of-time error.

4 Conclusion and future work

In this work, we introduced LIES, a neural framework for SR that combines interpretable activations with structured pruning to recover compact and accurate expressions. Experiments on the AI Feynman dataset showed strong symbolic and numerical performance, with ablation studies confirming the importance of each component in the framework. Current limitations lie in handling trigonometric and high-dimensional formulae. Future work will explore complex-domain training, improved pruning,

[1] `https://github.com/MansoorehMontazerin/LIES`

and architecture search to scale LIES to more challenging settings. We will explore the application of LIES networks on real-world datasets.

References

[1] Luca Biggio, Tommaso Bendinelli, Aurelien Lucchi, and Giambattista Parascandolo. A seq2seq approach to symbolic regression. *Learning Meets Combinatorial Algorithms at NeurIPS2020*, 2020.

[2] Miles Cranmer. Interpretable machine learning for science with pysr and symbolicregression. jl. *arXiv preprint arXiv:2305.01582*, 2023.

[3] Miles Cranmer, Alvaro Sanchez Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. Discovering symbolic models from deep learning with inductive biases. *Advances in neural information processing systems*, 33:17429–17442, 2020.

[4] Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli. Scientific machine learning through physics–informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, 92(3):88, 2022.

[5] Pierre-Alexandre Kamienny, Guillaume Lample, Sylvain Lamprier, and Marco Virgolin. Deep generative symbolic regression with monte-carlo-tree-search. In *International Conference on Machine Learning*, pages 15655–15668. PMLR, 2023.

[6] Liron Simon Keren, Alex Liberzon, and Teddy Lazebnik. A computational framework for physics-informed symbolic regression with straightforward integration of domain knowledge. *Scientific Reports*, 13(1):1249, 2023.

[7] John R Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4:87–112, 1994.

[8] William La Cava, Bogdan Burlacu, Marco Virgolin, Michael Kommenda, Patryk Orzechowski, Fabrício Olivetti de França, Ying Jin, and Jason H Moore. Contemporary symbolic regression methods and their relative performance. *Advances in neural information processing systems*, 2021(DB1):1, 2021.

[9] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[10] Brenden K Petersen, Mikel Landajuela, T Nathan Mundhenk, Claudio P Santiago, Soo K Kim, and Joanne T Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. *arXiv preprint arXiv:1912.04871*, 2019.

[11] C Rasmussen and C Williams. Gaussian processes for machine learning.,(mit press: Cambridge, ma), 2006.

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

[13] Parshin Shojaee, Kazem Meidani, Amir Barati Farimani, and Chandan Reddy. Transformer-based planning for symbolic regression. *Advances in Neural Information Processing Systems*, 36:45907–45919, 2023.

[14] Wassim Tenachi, Rodrigo Ibata, and Foivos I Diakogiannis. Deep symbolic regression for physics guided by units constraints: toward the automated discovery of physical laws. *The Astrophysical Journal*, 959(2):99, 2023.

[15] Yuan Tian, Wenqi Zhou, Michele Viscione, Hao Dong, David S Kammer, and Olga Fink. Interactive symbolic regression with co-design mechanism through offline reinforcement learning. *Nature Communications*, 16(1):3930, 2025.

[16] Silviu-Marian Udrescu and Max Tegmark. Ai feynman: A physics-inspired method for symbolic regression. *Science advances*, 6(16):eaay2631, 2020.

[17] Martin Vastl, Jonáš Kulhánek, Jiří Kubalík, Erik Derner, and Robert Babuška. Symformer: End-to-end symbolic regression using transformer-based architecture. *IEEE Access*, 2024.

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

A Proofs

A.1 Proof of Theorem 1

Proof. We establish the universality of the LIES network by showing that it can emulate any standard multilayer perceptron (MLP) with one or more hidden layers of arbitrary width. Specifically, Figure 3a illustrates how the LIES network can be configured to approximate a sigmoid activation function (Sigmoid block). Figure 3b further demonstrates how an L-layer MLP can be systematically transformed into an equivalent LIES network. Additionally, we can prove the universal approximability by showing, through a similar argument, that LIES networks can replicate any polynomial function. □

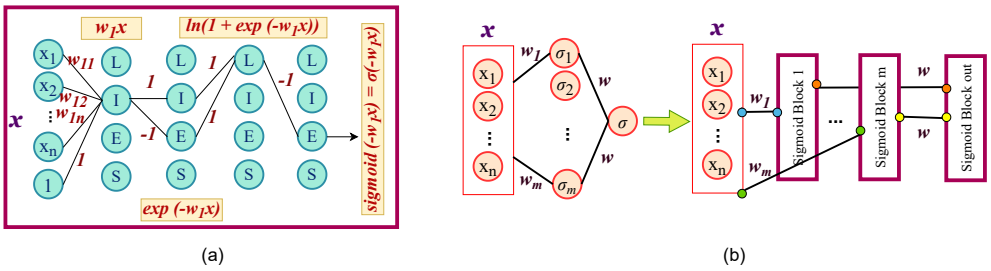


Figure 3: (a) Approximation of the Sigmoid function using a LIES-based configuration. (b) Transformation of an L-layer MLP into an equivalent LIES network using stacked LIES layers and Sigmoid blocks.

A.2 Proof of Theorem 2

Proof. Let $g_{\mathbf{x}}(y) = f(P_i = y; \mathbf{x})$. Then, by the mean value theorem, there exists $\alpha_0 \in [\alpha, \alpha + h]$ such that

$$g_{\mathbf{x}}(\alpha + h) - g_{\mathbf{x}}(\alpha) = h \cdot g'_{\mathbf{x}}(\alpha_0).$$

As a result, we have the bound

$$|g_{\mathbf{x}}(\alpha + h) - g_{\mathbf{x}}(\alpha)| = |h \cdot g'_{\mathbf{x}}(\alpha_0)| \leq h \cdot \max_{y \in [\alpha, \alpha+h]} |g'_{\mathbf{x}}(y)|,$$

where the maximum exists due to the continuity of $g'_{\mathbf{x}}$ on the closed interval. If this upper bound is smaller than a predefined threshold τ, i.e.,

$$h \cdot \max_{y \in [\alpha, \alpha+h]} |g'_{\mathbf{x}}(y)| < \tau,$$

then the change in the function value is considered negligible, and we may approximate:

$$g_{\mathbf{x}}(\alpha + h) \approx g_{\mathbf{x}}(\alpha).$$

Moreover, if the above condition holds for all $\mathbf{x} \in S$, where S denotes the set of all input samples to the network, then we approximate:

$$f(\alpha + h) \approx f(\alpha).$$

□

To quantify the effect of individual parameters, let w_i denote a weight in the trained model. We define its importance as the change in the network's output when it is set to zero for a given input $\mathbf{x}$:

$$|\Delta O(w_i, \mathbf{x})| = |O(w_i = 0, \mathbf{x}) - O(w_i, \mathbf{x})|$$

Here, $O(w_i, \mathbf{x})$ denotes the output of the network when w_i is active, and $O(w_i = 0, \mathbf{x})$ denotes the output when w_i is pruned. By Theorem 2, this change is bounded as:

$$|\Delta O(w_i, \mathbf{x})| \leq |w_i| \cdot \max_{s \in [0, w_i]} \left| \frac{\partial O(s, \mathbf{x})}{\partial w_i} \right|.$$

Since the gradient $\frac{\partial O}{\partial w_i}(w_i, \mathbf{x})$ is already computed during backpropagation, we avoid additional evaluations by approximating the maximum with the gradient at the current weight value:

$$\left|\frac{\partial O}{\partial w_i}(s, \mathbf{x})\right| \approx \left|\frac{\partial O}{\partial w_i}(w_i, \mathbf{x})\right|.$$

Thus, if the product $|w_i \cdot \frac{\partial O}{\partial w_i}(w_i, \mathbf{x})|$ is smaller than a predefined threshold for all input samples $\mathbf{x}$, we prune w_i as its influence on the output is considered negligible.

Gradient-based rounding After all pruning steps are complete, we backtrack through the remaining network weights to recover the symbolic formula. To further simplify the resulting expression, we apply a rounding step to the constants c_i within the symbolic formula. By Theorem 2, the change in the symbolic function f due to modifying a constant is bounded by a first-order approximation. Specifically, we estimate the effect of replacing each c_i with a rounded value r_i (e.g., rounding to the closest first decimal place) as:

$$|f(c_i) - f(r_i)| \leq |c_i - r_i| \cdot \max_{s \in [r_i, c_i]} \left|\frac{\partial f}{\partial c_i}(s)\right|.$$

To avoid computing gradients at multiple points, we approximate the maximum by evaluating the derivative at the rounded value r_i, yielding:

$$|f(c_i) - f(r_i)| \approx |c_i - r_i| \cdot \left|\frac{\partial f}{\partial c_i}(r_i)\right|.$$

If this estimated change is smaller than a predefined threshold for all input samples, we replace c_i with the rounded value r_i, as its influence on the function is negligible.

We apply gradient-based rounding twice during the final stages of formula extraction. First, we perform an initial rounding to zero before coefficient optimization to eliminate unnecessary terms, enhance sparsity, and reduce expression complexity, which in turn improves the efficiency of the optimization step. After optimization, we apply a second gradient-based rounding pass to check if any constants can be rounded to at most one decimal place, to obtain a cleaner symbolic formula.

B Experimental Setup

B.1 Dataset and model

We evaluate the proposed LIES architecture on the AI Feynman dataset. We focus on the formulae with four or fewer input variables. Higher-dimensional formulae require deeper networks with more parameters, which makes pruning more challenging and hinders the recovery of sparse symbolic expressions [15]. We also exclude formulae involving trigonometric functions that have negative inputs, as our use of log-space transformations can lead to instability in such cases. Please refer to subsection B.2 for a detailed discussion. Therefore, we conduct our experiments on 61 formulae in the AI Feynman dataset. Each experiment is conducted across three independent trials to account for the stochastic nature of training and ensure the consistency and robustness of the recovered expressions. Each of the weak ADMM training parts is run for 20 epochs and the strong ADMM is run for 30 epochs. The whole pipeline takes around 10 minutes to run on a single NVIDIA RTX A5000 GPU for a single trial. We use the RMSprop optimization algorithm with a learning rate of 1.5×10^{-2}. In the weak ADMM training phases, we set the penalty parameter $\rho = 0.5$ and the regularization coefficient $\lambda = 5 \times 10^{-4}$. During the strong ADMM training phase, we use a smaller penalty parameter $\rho = 0.005$ and adapt the regularization coefficient as $\lambda = 5 \times 10^{-(n-1)}$, where n is the number of input variables in the target expression. For the gradient-based pruning step, we apply a sensitivity threshold of 0.01 and in the gradient-based rounding step, we use a threshold of 0.1 to simplify small coefficients while maintaining functional fidelity.

B.2 Discussion

We proposed LIES, an SR framework that trains a fixed neural network with interpretable activations and applies structured pruning to extract accurate, compact, and human-readable formulae. We

evaluated LIES on 61 expressions from the AI Feynman dataset, demonstrating strong performance both symbolically and numerically, and we conducted ablation studies to assess the impact of each component in the pipeline.

The main limitations of our current approach lie in handling more complex formulae in terms of the number of variables and the use of trigonometric functions. Below, we discuss potential directions to address these limitations in the future.

B.2.1 Dealing with Trigonometric Functions

To explore the potential of LIES for trigonometric expressions, we conducted a targeted experiment using a synthetic dataset with inputs x and outputs $\cos(x)$. The framework successfully represented $\cos(x)$ as $\sin(\pi/2 - x)$, confirming the utility of the sine activation function for handling such expressions. However, long formulae including trigonometric functions require deeper LIES networks. Creating multiplications in such deep networks requires going through logarithms. The logarithm of negative outputs of the trigonometric function is currently cut off in our training to ensure real gradients. This makes it difficult to learn formulae involving the multiplication of trigonometric functions with other functions. In future work, we will explore training the network in the complex domain to support the logarithm of negative quantities, and thus support longer formulae with trigonometric functions.

B.2.2 Further Improvements

Handling formulae with a large number of variables remains challenging for the LIES network, often requiring more effective pruning and sparsification strategies. Particularly, formulae requiring functions of the sum of products (e.g., $\sqrt{x^2 + y^2 + z^2}$) require deep networks, making them hard to prune and learn. We observed that such formulae are highly sensitive to parameter tuning and prone to unstable training. In future work, we aim to develop improved pruning techniques, investigate architecture search [9], and enhance optimization strategies to enable the inclusion of larger and more complex symbolic formulae.