MetaSymNet: A Tree-like Symbol Network with Adaptive Architecture and Activation Functions

Yanjie Li[1,2,3], **Weijun Li**[1,2,3,4*], **Lina Yu**[1*], **Min Wu**[1*], **Jingyi Liu**[1], **Shu Wei**[1,2], **Yusong Deng**[1,2]
Meilan Hao[1]

[1]AnnLab, Institute of Semiconductor, Chinese Academy of Sciences, Beijing, China
[2] School of Electronic, Electrical, and Communication Engineering, University of Chinese Academy of Sciences, Beijing, China
[3] Zhongguancun Academy, Beijing, China
[4]School of Integrated Circuits, University of Chinese Academy of Sciences, Beijing China
wjli@semi.ac.cn, yulina@semi.ac.cn, wumin@semi.ac.cn

Abstract

Mathematical formulas are the language of communication between humans and nature. Discovering latent formulas from observed data is an important challenge in artificial intelligence, commonly known as symbolic regression(SR). The current mainstream SR algorithms regard SR as a combinatorial optimization problem and use Genetic Programming (GP) or Reinforcement Learning (RL) to solve the SR problem. These methods perform well on simple problems, but poorly on slightly more complex tasks. In addition, this class of algorithms ignores an important aspect: in SR tasks, symbols have explicit numerical meaning. So can we take full advantage of this important property and try to solve the SR problem with more efficient numerical optimization methods? The Equation Learner (EQL) replaces activation functions in neural networks with basic symbols and sparsifies connections to derive a simplified expression from a large network. However, EQL's fixed network structure can not adapt to the complexity of different tasks, often resulting in redundancy or insufficient, limiting its effectiveness. Based on the above analysis, we propose MetaSymNet, a tree-like network that employs the PANGU meta-function as its activation function. PANGU meta-function can evolve into various candidate functions during training. The network structure can also be adaptively adjusted according to different tasks. Then the symbol network evolves into a concise, interpretable mathematical expression. To evaluate the performance of MetaSymNet and five baseline algorithms, we conducted experiments across more than ten datasets, including SRBench. The experimental results show that MetaSymNet has achieved relatively excellent results on various evaluation metrics.

Code — https://github.com/1716757342/MetSymNet

Introduction

Determining a comprehensible and succinct mathematical expression from datasets remains a crucial challenge in artificial intelligence research. Symbolic regression endeavors to identify an interpretable equation $Y = F(X)$ that precisely represents the relationship between the independent variable X and the dependent variable Y. Contemporary predominant approaches to symbolic regression often treat the problem as a combinatorial optimization challenge, wherein each mathematical operator is considered merely an 'action' with no inherent mathematical significance. These methodologies generally employ Genetic Programming (GP) or reinforcement learning techniques. While effective for simpler tasks, their performance deteriorates when addressing complex symbolic regression challenges due to the exponential growth of the search space. On the other hand, in contrast to traditional combinatorial optimization tasks, symbols used in symbolic regression possess inherent mathematical meanings. Consider the classic Traveling Salesman Problem (TSP), where the elements—four cities labeled [A, B, C, D]—lack additional implications beyond their identification. Conversely, in a symbolic regression scenario involving four symbols [+,×, sin,x], these are not merely labels. Each symbol carries a distinct mathematical significance. For instance, these symbols can function as activation functions within neural networks, facilitating processes such as forward or backpropagation.

The Equation Learner (EQL) modifies the architecture of the Multi-Layer Perceptron (MLP) by consistently incorporating a predetermined symbolic function as the activation function in each layer. Following this, a sparsification technique is applied to eliminate redundant connections within the network. This methodological framework allows EQL to derive a concise mathematical expression from complex data relationships. However, EQL has the following problems: **1) Sparsification Challenges:** Achieving effective sparsification within the fully connected network proves to be more problematic than anticipated. It is often difficult to reduce the network to only one or two connections while maintaining high fitting accuracy solely through L_1 regularization. As a result, many times the resulting expression does not fit the data well. **2) Fixed Network Structure:** The network structure is fixed and cannot be adjusted according to the complexity of the tasks. It is easy to cause a mismatch between network structure and task. For instance, an excessively large initialized network may produce overly complex expressions, while a network that is too small may compro-

*Corresponding author.

mise fitting performance.

In this paper, we introduce MetaSymNet, an innovative tree-based symbolic regression network. Its architecture is structured as an expression binary tree, featuring compact connections between nodes, which effectively eliminates the need for the sparsification of connection weights. Furthermore, we propose PANGU metafunctions and Variable metafunctions to serve as activation functions for internal nodes and leaf nodes, respectively. The PANGU function possesses the flexibility to evolve into any operational symbol during the training process, while Variable metafunctions enable the selection of the type of input variable. Notably, MetaSymNet is capable of adaptively modifying its structure, allowing it to grow or shrink in response to the complexity of the task at hand. Our contributions are summarized as follows:

- We introduce MetaSymNet, an innovative symbolic regression algorithm based on numerical optimization that features dynamic adaptation of node activation functions in response to specific task requirements. Additionally, MetaSymNet's network architecture is capable of real-time adaptive adjustments driven by gradient information, optimizing its structure to align closely with the complexities of the tasks at hand.
- We propose a novel activation function, PANGU metafunction, which can adaptively evolve into various candidate functions during the numerical optimization process.
- We propose a structural adjustment mechanism that utilizes the difference in the number of inputs required by unary and binary functions. This approach allows MetaSymNet to modify its tree network structure in real time, guided by gradient, progressively refining its topology toward an optimal configuration.
- We incorporate an entropy loss metric for each set of selected parameters into the loss function. This integration aims to augment MetaSymNet's training efficiency and precision.

Related Work

Based on Genetic algorithm The genetic algorithm (GA) (Mirjalili and Mirjalili 2019; Katoch, Chauhan, and Kumar 2021) is a classical algorithm that imitates biological evolution, and the algorithm that applies the GA algorithm to solve the problem of symbolic regression is the Genetic Programming (GP) (Espejo, Ventura, and Herrera 2009; Fortin et al. 2012; Augusto and Barbosa 2000) algorithm. GP represents each expression in the form of a binary tree, initializes an expression population, and then evolves a better population employing crossover, mutation, etc. Repeat the process until the termination condition is reached.

Based on reinforcement learning Deep Symbolic Regression (DSR) (Petersen et al. 2019) is a very good algorithm for symbolic regression using reinforcement learning. DSR uses a recurrent neural network as the strategy network of the algorithm, which takes as input the parent and sibling nodes to be generated, and outputs as the probability of selecting each symbol. DSR uses risk policy gradients to refine policy network parameters. DSO (Mundhenk et al. 2021) introduces the GP algorithm based on DSR. SPL (Sun et al. 2022) successfully applies MCTS to solve the problem of symbolic regression. In this algorithm, the author uses MCTS to explore the symbolic space and puts forward a modular concept to improve search efficiency. DySymNet (Li et al. 2023) uses reinforcement learning to guide the generation of symbol networks. RSRM (Xu, Liu, and Sun 2023) deeply combines MCTS, Double Q-learning block, and GP, and achieves good performance on many datasets.

Based on neural networks AI Feynman series algorithms are mainly divided into two versions, the main idea of this series of algorithms is to reduce complexity to simplicity. AI Feynman 1.0 (Udrescu and Tegmark 2020) first uses a neural network to fit the data and then uses the curve fitted by the neural network to analyze a series of properties in the data, such as symmetry and separability. Then the formula to be found is divided into simple units by these properties, and finally, the symbol of each unit is selected by random selection. The idea of AI Feynman 2.0 (Udrescu et al. 2020) and 1.0 is very similar, the biggest difference between the two is that version 2.0 introduces more properties so that the search expression can be divided into simpler units, improving the search efficiency. DGP (Zeng et al. 2023) works by normalizing the weight connections and then selecting the corresponding symbols. EQL (Martius and Lampert 2016; Kim et al. 2020) algorithm is an SR algorithm based on a neural network, which replaces the activation function in the fully connected neural network with basic operation symbols such as $[+, -, ..., sin...]$, then removes the excess connections through pruning, and extracts an expression from the network.

Based on Transformer The NeSymReS (Biggio et al. 2021) algorithm treats the symbolic regression problem as a translation problem in which the input $[x, y]$, and the output is a preorder traversal of the expressions. NeSymReS first generates several expressions and then uses the sampled data $[x, y]$ from these expressions as inputs and the backbone of the expressions as outputs to train a transformer (Vaswani et al. 2017), pre-training model. When predicting the data, $[x, y]$ is entered into the transformer, and then combined with the beam search, a pre-order traversal of the formula is generated in turn. The biggest difference between the end-to-end (Kamienny et al. 2022) algorithm and NeSymReS is that the end-to-end approach can directly predict a constant to a specific value. Instead of predicting a constant placeholder 'C'. (Shojaee et al. 2023) and (Kamienny et al. 2023), use the pre-trained model as a policy network to guide the search process of MCTS to improve the search efficiency. The SNIP (Meidani et al. 2023) uses contrastive learning to train the data feature extractor. Then, it trains a transformer to generate the expression skeleton in a self-supervised manner.

Methodology

MetaSymNet, a tree-like neural network, treats each node as a neuron with internal nodes using the PANGU metafunction as their activation function, and leaf nodes employing Variable metafunctions. During training, in addition to optimizing amplitude parameters $\mathcal{W}$ and bias term b like ordi-

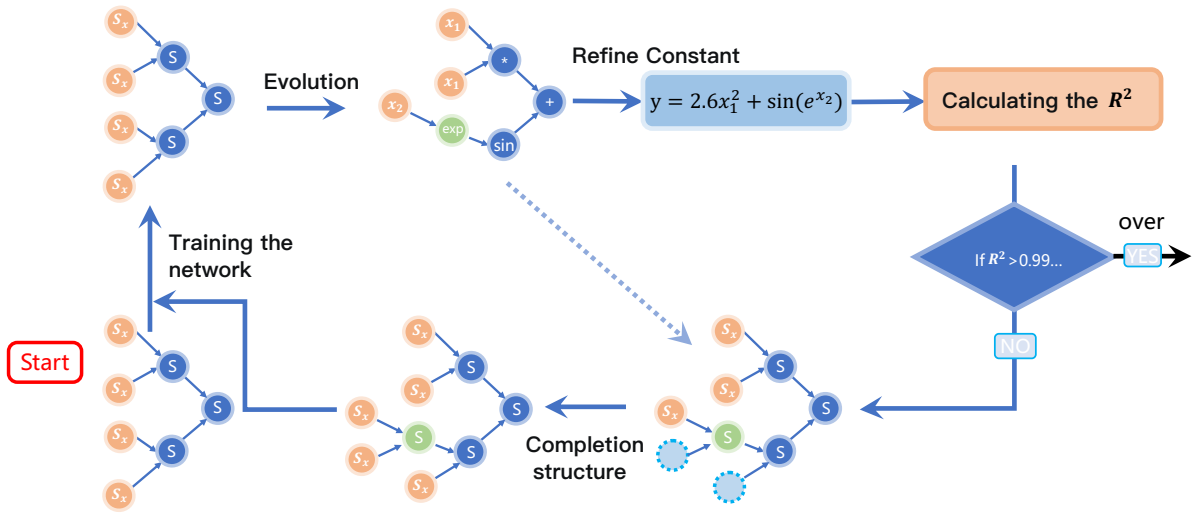


Figure 1: Flowchart of the MetaSymNet. (i) First randomly initialize a network, where the internal node S is the PANGU meta-function and the leaf node S_x is the Variable meta-function. (ii) A numerical optimization algorithm is used to optimize the parameters. In this process, the amplitude parameter $\mathcal{W}$ and the bias parameter $\mathcal{B}$ are optimized first, and then the selection parameters $\mathbb{Z}$ and $\mathbb{D}$ of the PANGU meta-functions and Variable meta-functions are optimized. Iterate several times. (iii) After parameter optimization, we determine the basic candidate symbols to which each PANGU metafunction and Variable metafunction should evolve based on the selection parameters $\mathbb{Z}$ and $\mathbb{D}$. (iv) When the network evolves into an expression, we further refine the constants of the expression and calculate the loss and R^2. The iteration stops when R^2 reaches the specified threshold. Otherwise, We replace the internal nodes (operation symbols) of the obtained expression binary tree with PANGU metafunctions and leaf nodes (variable symbols) with Variable metafunctions. The network structure is then supplemented with Variable meta-functions such that each PANGU meta-function has two children. The iteration continues.

nary fully connected neural networks, we also optimize the internal Selecting parameters $\mathbb{Z}$, $\mathbb{D}$ of the PANGU metafunctions and Variable metafunctions. The end-to-end training of MetaSymNet not only results in a high degree of data fit but also facilitates the evolution of the metafunctions into a variety of fundamental symbols, transforming the network into an analyzable and interpretable mathematical formula. MetaSymNet's algorithm schematic is shown in Figure 1. See Appendix 1 for the pseudocode

We begin by stochastically initializing a tree-like network whose architecture and neuronal count are arbitrarily defined. The activation functions of the network are the PANGU meta-functions and Variable metafunctions. In the beginning, each neuron has two inputs because our symbol library contains not only unary activation functions like $[sin, cos, exp, sqrt, log]$ but also binary activation functions like $[+, -, \times, \div]$, etc.

PANGU Meta-Function

In a standard neural network, the activation function is static, typically restricted to a single type such as ReLU or sigmoid. This makes neural networks ultimately a combination and nesting of multiple sets of activation functions and parameters, often a complex and hard-to-interpret 'black box'. Mathematical formulas in natural science are composed of basic operation symbols such as sin() and cos(). Can we design a meta-activation function that can automatically evolve into various basic operators during training? To realize this, we design a PANGU meta-function, shown in Fig. 2. The formula is as follows Eq.1:

$$\mathcal{OUT} = w * O\mathcal{E}^T + b \tag{1}$$

Here, OUT is the corresponding output of the PANGU meta-function, $O = [o_1, o_2, \ldots, o_n] = [x_l + x_r, \ldots, e^{x_l}, x_1, \ldots, x_n]$, and o_i is the output of the i^{th} candidate function in the library. The vector $\mathcal{Z} = [z_1, z_2, ..., z_n]$ is a set of selection parameters that can be optimized to control the probability of each activation function being selected. And all internal neuron function selection parameters $\mathcal{Z}$ form the set $\mathbb{Z} = [\mathcal{Z}_1, \mathcal{Z}_2, ..., \mathcal{Z}_n]$. Here n denotes the number of internal neurons. $\mathcal{E} = softmax(\mathcal{Z}) = [e_1, e_2, \ldots, e_n]$ (or $\mathcal{E} = softmax(\mathcal{Z} - max(\mathcal{Z}))$ (Stevens et al. 2021; Dong, Zhu, and Ma 2019)), $e_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{c*z_j}}$ is the i^{th} value in vector $\mathcal{E}$. And e_i can be the probability of selecting i^{th} candidate activation function.

Variables Meta-Function

At the leaf neurons of MetaSymNet, we will initialize a Variable meta-function, shown in Fig 2(bottom), but the function can only evolve into different variables $[x_1, x_2, ..., x_n]$. Each time, the top two most likely variables are selected. The specific evolution process is shown in the algorithm 2. Its expression is as follows 2:

$$\mathcal{OUT} = w * X\mathcal{E}^T + b \tag{2}$$

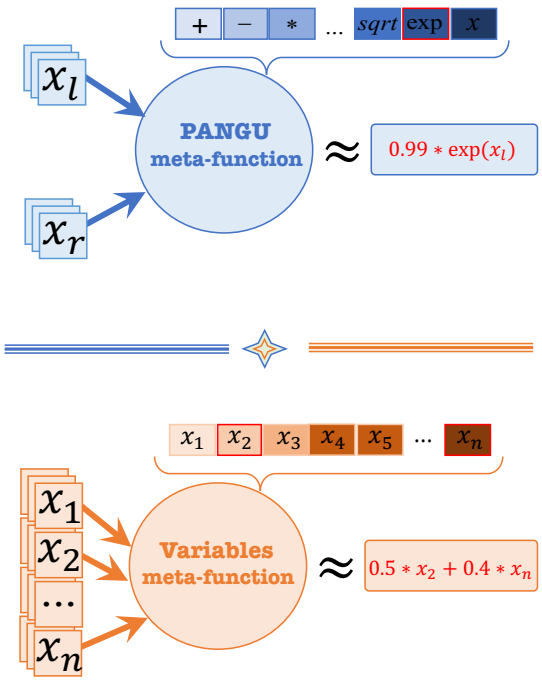


Figure 2: The structure of PANGU metafunctions and Variable metafunctions. The figure depicts the detailed internal structure diagram of the PANGU metafunction (top) and Variable metafunction (bottom). Note: The variable metafunctions are chosen from two variables at a time.

Here, $X = [x_1, x_2, ..., x_k]$, $\mathcal{E} = [\frac{e^{d_1}}{\sum_{i=1}^{k} e^{c*d_i}}, \frac{e^{d_2}}{\sum_{i=1}^{k} e^{c*d_i}} ... \frac{e^{d_n}}{\sum_{i=1}^{k} e^{c*d_i}}]$, $\mathcal{D} = [d_1, d_2, ..., d_k]$ is a set of variable selection parameters, k is the number of variables in the task, and q denotes the total number of leaf nodes. And w is the amplitude control parameter and b is the bias parameter. In MetaSymNet, the variable selection parameter $\mathbb{D} = [\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_q]$ and the activation function selection parameter $\mathbb{Z}$ are optimized together.

Parameters Optimization

Each neuron is assigned three types of parameters, an amplitude constant w, a bias constant b, and a set of selection parameters $\mathcal{Z} = [z_1, z_2, ..., z_n]$(The leaf neurons are $\mathcal{D} = [d_1, d_2, ..., d_n]$). Here, the amplitude constants w of all neurons form the parameter set $\mathcal{W}$. Similarly, all bias constants b form the set $\mathcal{B}$, and selection parameters $\mathcal{Z}$ for all internal neurons form the set $\mathbb{Z}$. All variable selection parameters form the set $\mathbb{D}$. We alternately optimize the above types of parameters with a numerical optimization algorithm (For example, SGD (Bottou 2012, 2010), BFGS (Dai 2002), L-BFGS (Liu and Nocedal 1989) etc). First, the parameters in the set W and B are optimized, and then the parameters in the set $\mathbb{Z}$ and $\mathbb{D}$ are optimized, next the activation functions of internal neurons and the variables of the leaf neurons are selected. Finally, after optimizing the parameters alternately for several rounds, we extract a formula from the network and further optimize W and B to achieve higher accuracy.

Activation Function Selection

There are many types of candidate activation functions. This article contains four binary activation functions $[+, -, \times, \div]$, five unary activation functions $[sin, cos, exp, log, sqrt]$, and variables $[x_1, x_2, ..., x_k]$.

PANGU Meta-Function Selection Process, We initialize a PANGU meta-function with an optimizable selection parameter vector $\mathcal{Z} = [z_1, z_2, ..., z_n]$, where n represents the class of candidate functions. We optimize $\mathcal{Z}$ by numerical optimization algorithm, Then the optimized $\mathcal{Z}_{new}$ each element minus the $Max(Z_{new})$, and then sent $softmax$ to get $\mathcal{E} = softmax(\mathcal{Z}_{new} - Max(\mathcal{Z}_{new})) = [e_1, e_2, ..., e_n]$. Each PANGU meta-function $\mathcal{S}$ has two inputs x_l and x_r (The output of the two child nodes), and then the candidate functions do the corresponding numerical operations respectively. We can obtain the vector $\mathcal{O} = [x_l + x_r, x_l - x_r, x_r * x_r, x_l/x_r, sin(x_l), cos(x_l), exp(x_l), log(x_l), sqrt(x_l), x_1, x_2, ..., x_n]$. Finally, we dot multiply the vectors $\mathcal{E}$ and $\mathcal{O}$, then multiply by a constant w, and add a bias term b to get the final output. After optimizing the vector $\mathcal{Z}$ multiple times in this way, we map $\mathcal{Z}$ to one-hot (Rodríguez et al. 2018) form and then dot multiply it with O. The final selected activation function is the symbol corresponding to the index equal to 1 in one-hot (Figure 1 top left, the evolution of PANGU metafunction S). Complete the activation function selection

Variable Meta-Function Selection Process, First, suppose that the output of the leaf neuron S_x is v_i after multiple optimizations. Here, $v_i = softmax(D_i) * X$, $X = [x_1, x_2, ..., x_n]$. Then the two variables with the highest probability are selected as x_l, x_r according to $softmax(D_i)$ (Fig. 1 Top left, the first step in the evolution of the Variable metafunction S_x, selecting two variables (red and pink boxes)). Finally, x_l and x_r are used as the input of each candidate activation function, and the $O_i = [x_l + x_r, x_l - x_r, ..., sin(x_l), ..., x1, ..., x_n]$ is obtained, then we choose the symbol corresponding to the value closest to v_i in vector O_i as the new activation function of the leaf neuron (Figure 1 top left, the second step in the evolution of Variable metafunction S_x). If the target expression is still not found, or R^2 is less than 0.9999. This process is repeated until a stopping condition is reached.

Network Structure Optimization

The network structure of traditional neural networks is fixed, which is very easy to cause network structure redundancy. Although there are many pruning methods to simplify the structure of neural networks by pruning unnecessary connections, it is not easy to extract a concise expression from a network (Hermundstad et al. 2011; Maksimenko et al. 2018). We propose an algorithm to optimize the MetaSymNet structure in real time under the guidance of gradient. From the above, we know that the activation function of MetaSymNet can be learned dynamically and each neuron has two inputs. We designed a method to realize the dynamic adjustment of the network structure when the activation function changes. The rules for adjusting the structure according to the activation function changes are as follows:

(1), Structural growth

Group	Dataset	BASELINES					
		MetaSymNet	**DSO**	**TPSR**	**SPL**	**NeSymReS**	**EQL**
Standards	Nguyen	$\mathbf{0.9999}_{\pm 0.000}$	$\mathbf{0.9999}_{\pm 0.001}$	$0.9948_{\pm 0.003}$	$0.9842_{\pm 0.001}$	$0.8468_{\pm 0.002}$	$0.9924_{\pm 0.005}$
	Keijzer	$\mathbf{0.9992}_{\pm 0.001}$	$0.9924_{\pm 0.001}$	$0.9828_{\pm 0.002}$	$0.8919_{\pm 0.002}$	$0.7992_{\pm 0.002}$	$0.9666_{\pm 0.003}$
	Korns	$\mathbf{0.9999}_{\pm 0.001}$	$0.9872_{\pm 0.000}$	$0.9325_{\pm 0.004}$	$0.8788_{\pm 0.001}$	$0.8011_{\pm 0.001}$	$0.9285_{\pm 0.004}$
	Constant	$\mathbf{0.9996}_{\pm 0.002}$	$0.9988_{\pm 0.003}$	$0.9319_{\pm 0.002}$	$0.8942_{\pm 0.002}$	$0.8444_{\pm 0.002}$	$0.9466_{\pm 0.003}$
	Livermore	$\mathbf{0.9924}_{\pm 0.003}$	$0.9746_{\pm 0.003}$	$0.8820_{\pm 0.004}$	$0.8728_{\pm 0.002}$	$0.7136_{\pm 0.004}$	$0.9037_{\pm 0.005}$
	Vladislavleva	$0.9826_{\pm 0.003}$	$\mathbf{0.9963}_{\pm 0.004}$	$0.9128_{\pm 0.005}$	$0.8433_{\pm 0.004}$	$0.6892_{\pm 0.004}$	$0.8926_{\pm 0.004}$
	R	$\mathbf{0.9921}_{\pm 0.002}$	$0.9744_{\pm 0.003}$	$0.9422_{\pm 0.001}$	$0.9122_{\pm 0.003}$	$0.8003_{\pm 0.004}$	$0.8637_{\pm 0.005}$
	Jin	$0.9896_{\pm 0.002}$	$\mathbf{0.9916}_{\pm 0.002}$	$0.9826_{\pm 0.004}$	$0.9211_{\pm 0.002}$	$0.8627_{\pm 0.002}$	$0.9677_{\pm 0.004}$
	Neat	$\mathbf{0.9953}_{\pm 0.004}$	$0.9827_{\pm 0.003}$	$0.9319_{\pm 0.004}$	$0.8828_{\pm 0.003}$	$0.7996_{\pm 0.005}$	$0.9631_{\pm 0.004}$
	Others	$\mathbf{0.9984}_{\pm 0.001}$	$0.9861_{\pm 0.002}$	$0.9667_{\pm 0.003}$	$0.9435_{\pm 0.003}$	$0.8226_{\pm 0.002}$	$0.9438_{\pm 0.004}$
SRBench	Feynman	$\mathbf{0.9960}_{\pm 0.002}$	$0.9610_{\pm 0.003}$	$0.8928_{\pm 0.003}$	$0.9284_{\pm 0.003}$	$0.7025_{\pm 0.003}$	$0.8725_{\pm 0.005}$
	Strogatz	$\mathbf{0.9424}_{\pm 0.004}$	$0.9313_{\pm 0.002}$	$0.8249_{\pm 0.003}$	$0.8411_{\pm 0.002}$	$0.6222_{\pm 0.002}$	$0.8844_{\pm 0.005}$
	Black-box	$\mathbf{0.9302}_{\pm 0.003}$	$0.9033_{\pm 0.004}$	$0.8753_{\pm 0.003}$	$0.9024_{\pm 0.002}$	$0.6825_{\pm 0.003}$	$0.7852_{\pm 0.005}$
	Average	**0.9859**	0.9749	0.9024	0.8997	0.7528	0.7852

Table 1: Comparison of the coefficient of determination (R^2) between MetaSymNet and five baseline. Bold values indicate state-of-the-art (SOTA) performance. The confidence level is 0.95.

- When S_x evolves into a unary activation function. At this point, S_x evolves into the chosen unary operator symbol, and, a variable (leaf node) is selected as an input, (For example, the green symbol exp in Fig. 1). Finally, in Figure 1 'Completion structure' stage, the unary operation symbol is changed into PANGU meta-function S, and two variable meta-functions S_x are added as child nodes. Realize the growth of network structure.
- When S_x evolves into a binary activation function(e.g. +, *). At this point, S_x evolves into the chosen binary operator symbol, and two variable leaf nodes are selected as input. Finally, in Figure 1 'Completion structure' stage, the binary operation symbol is changed into PANGU meta-function S, and two variable meta-functions S_x are added as child nodes. Realize the growth of network structure.

(2), Structural reduction

- When S evolves into a variable symbol (e.g. x_1,x_n). At this point, S evolves into the chosen variable symbol, and all child nodes following this node are clipped off. Finally, in Figure 1 'Completion structure' stage, the variable symbol is changed into a Variable meta-function S_x. Realize the reduction of network structure.

Loss Function

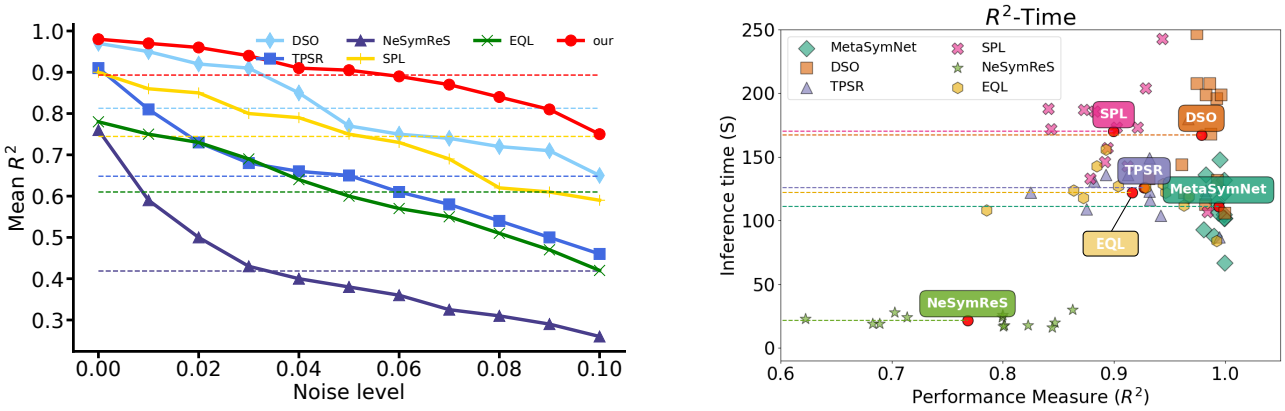
In the process of MetaSymNet training, the loss function has a crucial position, because it directly determines the direction of neural network optimization. In this study, we introduce a new loss function that aims to further optimize the performance of MetaSymNet. For each PANGU meta-function, it has a selection parameter $\mathcal{Z}$, and $\mathcal{Z}$ gets $\mathcal{E}$ after passing through the softmax function, where $\mathcal{E} = [e_1, e_2, ..., e_n]$. We introduce the entropy (Wehrl 1978; Rényi 1961) of $\mathcal{E}$ for all PANGU metafunctions as part of the loss. Our goal is for the largest element of each vector $\mathcal{E}$ to be significantly higher than the others while maintaining a high fitting accuracy. We only have to choose one symbol for the PANGU meta function. To facilitate the evolution of the PANGU meta-function. Specifically, the expression for the loss function is as follows 3:

$$\begin{aligned}\mathcal{L} &= L_{MSE} + L_{Entr} \\ &= \frac{1}{\mathcal{N}}\sum_{n=1}^{\mathcal{N}}(y_i - \hat{y}_i)^2 - \lambda\frac{1}{\mathcal{M}}\sum_{j=1}^{\mathcal{M}} log(max(\mathcal{E}_j))\end{aligned} \quad (3)$$

Where $\mathcal{N}$ is the number of sample points, y is the true y value, and $\hat{y}$ is the predicted y value. $\mathcal{M}$ is the number of PANGU meta-functions under the current network, and $\mathcal{E}_j$ is the value of the selection parameter of the j^{th} PANGU meta function after passing softmax. λ is the entropy loss regulation coefficient.

Results

In order to test the performance of MetaSymNet, we compare the performance of our algorithm with five state-of-the-art symbolic regression algorithms (**DSO (Mundhenk et al. 2021)**, **NeSymReS (Biggio et al. 2021)**, **SPL (Sun et al. 2022)**, **EQL (Martius and Lampert 2016)**, and **TPSR (Shojaee et al. 2023)**) on ten public datasets. Specific formula details are given in Table tables 10 to 14, in the appendix. Fig. 5 shows the Pareto plots(time and accuracy) of MetaSymNet and baselines(in SRBench) on black-box data and Feynman.



(a) Noise robustness

(b) R^2-time Pareto plot

Figure 3: Analysis on performance. a demonstrates the trends of R^2 values between MetaSymNet and five baseline methods across varying noise levels. b Provides a R^2-time Pareto plot of the various algorithms on individual datasets, showing optimization performance.

Fit Ability Test

When doing comparative experiments, we strictly control the experimental variables and ensure that all the other conditions are the same except for the algorithm type to ensure the fairness and credibility of the experiment. Specifically, (1) At test time, on the same test expression, we sample x_i on the same interval for different algorithms. To ensure the consistency of the test data. See Appendix tables 10 to 14, for the specific sampling range of each expression. (2) For some of the 'Constraints' used in our algorithm, we also use the same 'Constraints' in other algorithms. To ensure that the performance of the individual algorithms is not affected by the difference in the 'Constraints'. In the experiment, we use the coefficient of determination (R^2) (Nagelkerke et al. 1991; Ozer 1985) to judge how well each algorithm fits the expression. The results are shown in Table 1, from which we see that MetSymNet achieves the performance of SOTA on multiple datasets. The expression for R^2 is given in the following: $\mathcal{R}^2 = 1 - \frac{\sum_{i=1}^{\mathcal{N}}(y-\hat{y})^2}{\sum_{i=1}^{\mathcal{N}}(y-\overline{y})^2}$; Where $\hat{y}$ is the predicted y value and $\overline{y}$ is the mean of the true y.

Noise Robustness Test

In the real world, data often contains noise and uncertainty. This noise can come from measurement errors, interference during data acquisition, or other unavoidable sources (Berglund, Hassmen, and Job 1996; Tam et al. 2008; Beall and Lowe 2007). The Noise robustness test can simulate the noise situation in the real environment, and help to evaluate the performance of the symbolic regression algorithm in the case of imperfect data, to help us understand its reliability and robustness in practical applications. These trials can guide the tuning and selection of the model, ensuring that it will work robustly in real scenarios. (Ziyadinov and Tereshonok 2022; Gao et al. 2020). We generate noisy data y_{noise} in the following way. To simulate different levels of noise in the real world, we divide the noise into ten levels. First, the noisy data $Data_{noise}$ is obtained by randomly sampling on the interval $[-\mathcal{L} * Span, +\mathcal{L} * Span]$. Here, $\mathcal{L} = [0.00.01.0.02, ..., 0.1]$ is the level of noise. $Span = abs|max(y) - min(y)|$ is the Span of y. $y_{noise} = y + Data_{noise}$. We employed the datasets to assess the algorithm's noise robustness. For each mathematical expression, we conducted 20 times at varying noise levels. Subsequently, we computed the R^2 between the curve derived from each trial and the original curve, serving as a metric to quantify noise resilience. The outcome was determined by averaging the results from the 20 trials. The average R^2 of both MetaSymNet and five symbolic regression baselines were compared across different noise levels, and the results are depicted in Figure 6a. The comprehensive analysis demonstrates that MetaSymNet exhibits superior noise immunity performance in contrast to the other five symbolic regression baselines.

Inference Time Test

In order to evaluate a symbolic regression algorithm, in addition to fitting ability and anti-noise ability, the inference speed of the algorithm is also an extremely important index. Therefore, in order to test the inference efficiency of MetaSymNet and the various baselines, we picked the part of the expression data in all test datasets where the various algorithms could achieve $R^2 > 0.99$. These selected expressions make up dataset $\mathbb{A}$. We then test each expression in dataset $\mathbb{A}$ 10 times with each algorithm. We plot the $R^2 - time$ coordinate plot as shown in Fig 6, For this plot, each color corresponds to an algorithm, and each scatter corresponds to the average R^2 and average inference time for all expressions in one test dataset. The red dot indicates the center of gravity of each algorithm, and the closer it is to the bottom-right corner of the figure, the better the overall performance of the algorithm. The red star part represents our algorithm. From the figure, we can see that MetaSymNet achieves a good balance between efficiency and accuracy. In addition, Fig5 shows the

Dataset	MetaSymNet	DSO	SPL	TPSR	NeSymReS	EQL
Nguyen	16.5	20.3	26.0	**16.0**	18.2	22.4
Keijzer	**18.0**	18.4	28.4	20.6	21.3	20.8
Constant	24.4	26.6	33.5	**22.9**	24.1	32.9
Livermore	34.8	38.2	47.3	35.3	**32.9**	41.5
Vladislavleva	42.4	46.3	59.4	38.2	**36.2**	49.2
R	28.5	31.3	38.2	**24.6**	27.3	36.9
Jin	20.6	22.0	32.0	**16.2**	19.9	28.4
Others	**28.3**	33.2	39.5	29.5	32.2	37.4
Neat	19.5	22.7	28.7	**16.4**	20.6	27.2
Korns	25.8	26.8	32.4	**22.5**	23.5	32.4
Feynman	23.1	24.1	34.2	**21.3**	22.4	26.6
Strogatz	**21.4**	27.2	32.3	24.4	28.1	31.9
Black-box	**25.5**	32.9	39.2	29.3	33.9	35.3
Average	25.3	28.5	33.2	**24.4**	26.2	32.5

Table 2: Comparison of the average number of symbols (complexity) of the resulting expressions between MetaSymNet and the other five baselines.

MetaSymNet	With entropy loss			Without entropy loss		
	$R^2 \uparrow$	Nodes $\downarrow$	Time(s)$\downarrow$	$R^2 \uparrow$	Nodes $\downarrow$	Time(s) $\downarrow$
Nguyen	0.9999	16.5	96	0.9998	18.7	118
Keijzer	0.9992	18.0	124	0.9977	23.8	135
Korns	0.9999	24.4	103	0.9924	27.3	132
Constant	0.9996	34.8	109	0.9872	38.3	124
Livermore	0.9924	42.4	104	0.9834	49.2	122
Vladislavleva	0.9826	28.5	122	0.9807	32.2	132
R	0.9921	20.6	100	0.9829	26.4	111
Jin	0.9896	28.3	112	0.9744	32.9	125
Neat	0.9953	19.5	104	0.9881	24.7	131
Others	0.9984	25.8	121	0.9904	31.2	136
Feynman	0.9960	23.1	128	0.9763	27.4	142
Strogatz	0.9424	21.4	132	0.9114	28.2	151
Black-box	0.9302	25.5	142	0.9006	26.3	164
Average	0.9859	25.3	115	0.9743	29.7	133

Table 3: Ablation experiments on whether to introduce entropy loss in the loss function for MetaSymNet.

graph of MetaSymNet compared with 20 other baselines under the SRBench standard. We believe that the reason for MetaSymNet's efficiency is that the symbolic regression is changed from a combinatorial optimization problem to a numerical optimization problem while retaining the binary tree representation of the expression. As we know, in the current field of machine learning, the efficiency of numerical optimization algorithms is higher than that of reinforcement learning and evolutionary computation algorithms. Therefore, compared with the traditional symbolic regression algorithm, MetaSymNet is more efficient.

Result Complexity Test

In symbolic regression, our ultimate goal is to get a relatively concise expression to fit the observed data. If the resulting expression is too complex, then its interpretability is greatly reduced. Therefore, we set up the following experiment to compare the complexity (the number of nodes in the binary tree) of the resulting expressions obtained by each algorithm. we selected the expression in which all methods in the database can achieve $R^2 > 0.999$ as the test set to compare the complexity (number of nodes) of the expression obtained by different algorithms when $R^2 > 0.999$. Each expression was run 20 times and then averaged. The maximum length is set to 80 for all algorithms. The specific statistical results are shown in Table 2. As we can see from the table, MetaSymNet has the second-lowest average number of nodes after TPSR.

Ablation Experiments with Entropy Loss

For each PANGU metafunction, we have a set of selection $\mathcal{Z}$ passes through $softmax()$ to obtain $\mathcal{E} = [e_1, e_2, ..., e_n]$, here $\sum_{i=1}^{i=n} e_i = 1$. To improve the efficiency and performance of MetaSymNet, we introduce the entropy of $\mathcal{E}$ of all PANGU meta-functions in the network as part of the loss function. Specifically $L_{Entr} = \frac{1}{\mathcal{M}} \sum_{j=1}^{\mathcal{M}} log(max(\mathcal{E}_j))$, here, $\mathcal{M}$ denotes the number of PANGU metafunctions in the network. To demonstrate the effectiveness of entropy loss, we perform ablation experiments on it. The specific experimental results are shown in Table 3. We think that the introduction of entropy loss can promote the values in $\mathcal{E}$ to appear as 'big is bigger, small is smaller'. Makes One of its values significantly larger than the others, which is closer to the one-hot form. It promotes a more efficient and accurate evolution of the PANGU meta-function to different activation functions.

Discussion and Conclusion

In this paper, we propose MetaSymNet, which treats the SR as a numerical optimization problem rather than a combinatorial optimization problem. MetaSymNet's structure is a tree-like network that is dynamically adjusted during training and can be expanded or reduced. Compared with the baselines, MetaSymNet has a better fitting ability, noise robustness, and complexity. We propose a PANGU metafunction as the activation function of MetaSymNet. The function can autonomously evolve into various candidate functions under the control of selection parameters. In addition, we present variable metafunctions that can be used to select variables. Furthermore, the final result of MetaSymNet is a concise, interpretable expression. This characteristic enhances the credibility of MetaSymNet and presents significant potential for application in fields that involve high-risk decision-making, such as finance, medicine, and law. In such domains, where decisions can profoundly impact people's lives, people must understand and trust the algorithm's decision-making process. Despite MetaSymNet yielding satisfactory results, it has its limitations. For instance, tuning certain hyperparameters, such as the λ in the loss function, proves to be challenging. Additionally, the method can occasionally become trapped in local optima, resulting in approximate rather than exact expressions. Next, we plan to alter the evolution process of PANGU metafunctions. Specifically, instead of relying on the greedy strategy for function selection, we intend to explore a variety of search methods, including beam search, Monte Carlo Tree Search, and others, to enhance the algorithm's performance.

Acknowledgements

This work was supported in part by the National Natural Science Foundation of China under Grant 92370117, in part by CAS Project for Young Scientists in Basic Research under Grant YSBR-090

References

Augusto, D. A.; and Barbosa, H. J. 2000. Symbolic regression via genetic programming. In *Proceedings. Vol. 1. Sixth Brazilian symposium on neural networks*, 173–178. IEEE.

Beall, E. B.; and Lowe, M. J. 2007. Isolating physiologic noise sources with independently determined spatial measures. *Neuroimage*, 37(4): 1286–1300.

Berglund, B.; Hassmen, P.; and Job, R. S. 1996. Sources and effects of low-frequency noise. *The Journal of the Acoustical Society of America*, 99(5): 2985–3002.

Biggio, L.; Bendinelli, T.; Neitz, A.; Lucchi, A.; and Parascandolo, G. 2021. Neural symbolic regression that scales. In *International Conference on Machine Learning*, 936–945. PMLR.

Bottou, L. 2010. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers*, 177–186. Springer.

Bottou, L. 2012. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade: Second Edition*, 421–436. Springer.

Castellano, G.; Fanelli, A. M.; and Pelillo, M. 1997. An iterative pruning algorithm for feedforward neural networks. *IEEE transactions on Neural networks*, 8(3): 519–531.

Dai, Y.-H. 2002. Convergence properties of the BFGS algoritm. *SIAM Journal on Optimization*, 13(3): 693–701.

Dong, X.; Zhu, X.; and Ma, D. 2019. Hardware implementation of softmax function based on piecewise LUT. In *2019 IEEE International Workshop on Future Computing (IWOFC*, 1–3. IEEE.

Drucker, H.; Burges, C. J.; Kaufman, L.; Smola, A.; and Vapnik, V. 1996. Support vector regression machines. *Advances in neural information processing systems*, 9.

Espejo, P. G.; Ventura, S.; and Herrera, F. 2009. A survey on the application of genetic programming to classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 40(2): 121–144.

Fortin, F.-A.; De Rainville, F.-M.; Gardner, M.-A. G.; Parizeau, M.; and Gagné, C. 2012. DEAP: Evolutionary algorithms made easy. *The Journal of Machine Learning Research*, 13(1): 2171–2175.

Gao, X.; Saha, R. K.; Prasad, M. R.; and Roychoudhury, A. 2020. Fuzz testing based data augmentation to improve robustness of deep neural networks. In *Proceedings of the acm/ieee 42nd international conference on software engineering*, 1147–1158.

Hermundstad, A. M.; Brown, K. S.; Bassett, D. S.; and Carlson, J. M. 2011. Learning, memory, and the role of neural network architecture. *PLoS computational biology*, 7(6): e1002063.

Kamienny, P.-A.; d'Ascoli, S.; Lample, G.; and Charton, F. 2022. End-to-end symbolic regression with transformers. *Advances in Neural Information Processing Systems*, 35: 10269–10281.

Kamienny, P.-A.; Lample, G.; Lamprier, S.; and Virgolin, M. 2023. Deep Generative Symbolic Regression with Monte-Carlo-Tree-Search. *arXiv preprint arXiv:2302.11223*.

Katoch, S.; Chauhan, S. S.; and Kumar, V. 2021. A review on genetic algorithm: past, present, and future. *Multimedia tools and applications*, 80: 8091–8126.

Kim, S.; Lu, P. Y.; Mukherjee, S.; Gilbert, M.; Jing, L.; Čeperić, V.; and Soljačić, M. 2020. Integration of neural network-based symbolic regression in deep learning for scientific discovery. *IEEE transactions on neural networks and learning systems*, 32(9): 4166–4177.

Li, W.; Li, W.; Yu, L.; Wu, M.; Liu, J.; and Li, Y. 2023. A Neural-Guided Dynamic Symbolic Network for Exploring Mathematical Expressions from Data. *arXiv preprint arXiv:2309.13705*.

Liu, D. C.; and Nocedal, J. 1989. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3): 503–528.

Maksimenko, V. A.; Kurkin, S. A.; Pitsik, E. N.; Musatov, V. Y.; Runnova, A. E.; Efremova, T. Y.; Hramov, A. E.; and Pisarchik, A. N. 2018. Artificial neural network classification of motor-related eeg: An increase in classification accuracy by reducing signal complexity. *Complexity*, 2018.

Martius, G.; and Lampert, C. H. 2016. Extrapolation and learning equations. *arXiv preprint arXiv:1610.02995*.

Matsubara, Y.; Chiba, N.; Igarashi, R.; and Ushiku, Y. 2022. Rethinking symbolic regression datasets and benchmarks for scientific discovery. *arXiv preprint arXiv:2206.10540*.

Meidani, K.; Shojaee, P.; Reddy, C. K.; and Farimani, A. B. 2023. SNIP: Bridging Mathematical Symbolic and Numeric Realms with Unified Pre-training. *arXiv preprint arXiv:2310.02227*.

Mirjalili, S.; and Mirjalili, S. 2019. Genetic algorithm. *Evolutionary Algorithms and Neural Networks: Theory and Applications*, 43–55.

Mundhenk, T. N.; Landajuela, M.; Glatt, R.; Santiago, C. P.; Faissol, D. M.; and Petersen, B. K. 2021. Symbolic regression via neural-guided genetic programming population seeding. *arXiv preprint arXiv:2111.00053*.

Nagelkerke, N. J.; et al. 1991. A note on a general definition of the coefficient of determination. *biometrika*, 78(3): 691–692.

Ozer, D. J. 1985. Correlation and the coefficient of determination. *Psychological bulletin*, 97(2): 307.

Petersen, B. K.; Landajuela, M.; Mundhenk, T. N.; Santiago, C. P.; Kim, S. K.; and Kim, J. T. 2019. Deep symbolic regression: Recovering mathematical expressions

from data via risk-seeking policy gradients. *arXiv preprint arXiv:1912.04871*.

Rényi, A. 1961. On measures of entropy and information. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, volume 4, 547–562. University of California Press.

Rodríguez, P.; Bautista, M. A.; Gonzalez, J.; and Escalera, S. 2018. Beyond one-hot encoding: Lower dimensional target embedding. *Image and Vision Computing*, 75: 21–31.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1986. Learning representations by back-propagating errors. *nature*, 323(6088): 533–536.

Shojaee, P.; Meidani, K.; Farimani, A. B.; and Reddy, C. K. 2023. Transformer-based Planning for Symbolic Regression. *arXiv preprint arXiv:2303.06833*.

Stevens, J. R.; Venkatesan, R.; Dai, S.; Khailany, B.; and Raghunathan, A. 2021. Softermax: Hardware/software co-design of an efficient softmax for transformers. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 469–474. IEEE.

Sun, F.; Liu, Y.; Wang, J.-X.; and Sun, H. 2022. Symbolic physics learner: Discovering governing equations via monte carlo tree search. *arXiv preprint arXiv:2205.13134*.

Tam, C. K.; Viswanathan, K.; Ahuja, K.; and Panda, J. 2008. The sources of jet noise: experimental evidence. *Journal of Fluid Mechanics*, 615: 253–292.

Udrescu, S.-M.; Tan, A.; Feng, J.; Neto, O.; Wu, T.; and Tegmark, M. 2020. AI Feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity. *Advances in Neural Information Processing Systems*, 33: 4860–4871.

Udrescu, S.-M.; and Tegmark, M. 2020. AI Feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16): eaay2631.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Wehrl, A. 1978. General properties of entropy. *Reviews of Modern Physics*, 50(2): 221.

Xu, Y.; Liu, Y.; and Sun, H. 2023. RSRM: Reinforcement Symbolic Regression Machine. *arXiv preprint arXiv:2305.14656*.

Zeng, P.; Song, X.; Lensen, A.; Ou, Y.; Sun, Y.; Zhang, M.; and Lv, J. 2023. Differentiable Genetic Programming for High-dimensional Symbolic Regression. *arXiv preprint arXiv:2304.08915*.

Ziyadinov, V.; and Tereshonok, M. 2022. Noise immunity and robustness study of image recognition using a convolutional neural network. *Sensors*, 22(3): 1241.