Progressive Self-Learning for Domain Adaptation on Symbolic Regression of Integer Sequences

Yaohui Zhu[1], **Kaiming Sun**[2], **Zhengdong Luo**[3], **Lingfeng Wang**[1*]

[1]College of Information Science and Technology, Beijing University of Chemical Technology
[2]School of Computer Science, Shenyang Aerospace University
[3]Xinjiang Technical Institute of Physics and Chemistry, Chinese Academy of Sciences
{yaohui.zhu, lfwang}@buct.edu.cn, sunkaiming@stu.sau.edu.cn, luozhengdong21@mails.ucas.edu.cn

Abstract

Symbolic Regression of Integer Sequences (SRIS) aims to discover precise mathematical formulas from integer sequences. The neural machine translation-based method of SRIS trains the model using randomly generated data, and directly utilizes the trained model for inference on target sequences. However, the method often fails to effectively generalize to the target sequence, since the randomly generated data can not adequately cover the distributions of target data, i.e., there are distribution differences between them. In this work, we propose a progressive self-learning (PSL) method to explicitly capture sequence-formula distributions of the target domain. Specifically, a source domain dataset is generated by incorporating initial terms of the target domain to reduce the sequence distribution gap between the source domain and the target domain. Meanwhile, a self-learning loop strategy is adopted to improve the ability of the model to capture the sequence-formula distribution of the target domain. In this strategy, a neural machine translation model is used to learn the mappings from sequences to formulas in an end-to-end fashion. Then, this model is employed to explore candidate formulas of the target sequence using beam search. After verifying these candidate formula correctness, some of them are retained as training data for the next learning. Experimental results on OEIS datasets demonstrate that the proposed method surpasses current state-of-the-art methods in accuracy, and also discovers new formulas.

Introduction

Symbolic Regression of Integer Sequences (SRIS) aims to discover precise mathematical formulas from integer sequences by generating symbolic expressions. Compared with the method of using approximate nonsymbolic expressions (Lu et al. 2021), the SRIS task needs to produce explicit formulas, which provide strong interpretability. Currently, a larger number of integer sequence formulas have yet to be discovered, such as the prime sequence and the busy beaver sequence. Exploring these patterns advances mathematics and science. For example, studying the patterns of prime sequences not only helps solve mathematical problems such as Goldbach's conjecture (Wang 2022) and twin prime conjectures (Wang 2022), but also serves modern scientific and technological fields, such as cryptography and data compression. We can see, it is of far-reaching significance to the study of SRIS.

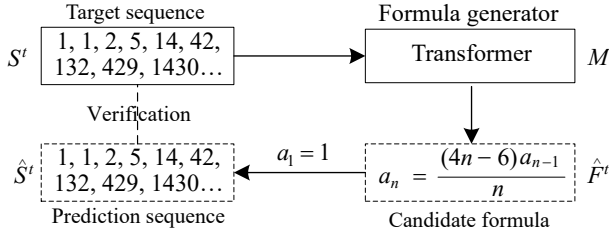

Figure 1: The process of formula verification. The correctness of $\hat{F}^t$ is checked by comparing S^t and $\hat{S}^t$ term by term.

The neural machine translation-based methods (Lample and Charton 2020; Biggio et al. 2021; d'Ascoli et al. 2022) exhibit great potential for SRIS. These methods train a translation model of transformer on randomly generated data, and directly utilize the model for inference on target sequences. However, the model struggles to generalize to the target sequences, since the generated data can not adequately cover the distributions of target data, i.e., there are distribution differences between them. For example, the model cannot discover the formula of sequence A214993, i.e., [11, 121, 1341, 14871,...], in OEIS (Sloane et al. 2018), while this formula is a simple linear recurrence $a_n = 12a_{n-1} - 10a_{n-2} - a_{n-3}$. Relying solely on source domain data may not be a promising approach. How to use both generated source data and target sequences to discover the formulas of target sequences is a significant problem, known as the Domain Adaptation on Symbolic Regression of Integer Sequences (DASRIS). However, this problem has barely been explored before.

Previous domain adaptation works (Yang et al. 2021; Chang et al. 2021; Karouzos, Paraskevopoulos, and Potamianos 2021) have been applied mainly in the fields of computer vision, focusing on aligning the data distribution of the source domain and the target domain. These domain adaptation methods have the same category space between the source domain and the target domain. Aligning the distributions of the source and target domains allows accurate semantic representations of target domain samples. In

*Corresponding Author.

the SRIS task, the formula space consists of various symbols. The flexible combination of symbols results in a wide range of formulas, which complicates the identification of the same formula space between the source and the target domains. Without a shared formula space, aligning data distributions between the source and target domains alone is insufficient for the DASRIS task.

Different from the image classification task, there is a close connection between the formula generated by the SRIS model and the input sequence. Deducing the input sequence with the generated formula can verify its correctness, as depicted in Figure 1. The correct formula can provide effective supervision information for the target domain, allowing the model to learn realistic distributions of the target domain.

Based on the above analysis, we propose a progressive self-learning (PSL) method to directly capture the sequence-formula distribution of the target domain for DASRIS. Specifically, instead of randomly generating source domain data, the proposed method involves creating a source domain data that includes initial terms from target sequences to narrow the sequence distribution gap between the source domain and target domains. Moreover, a self-learning loop strategy is adopted to enhance the ability of the model to learn the sequence-formula distribution of the target domain. In this strategy, a neural machine translation model is used to learn the mappings from sequences to formulas in an end-to-end fashion. Then, this model uses beam search to explore candidate formulas of the target sequence, which are verified their correctness by checking the deduced sequences. The verified formula with the fewest operators or the lowest recurrence degree will be retained as training data for the next learning.

The proposed method achieves state-of-the-art performance on the OEIS easy dataset, proving the effectiveness of the method. In addition, a total of about 12k formulas on OEIS are discovered, including some new formulas that have not been discovered before. We hope that these new formulas are helpful for the integer sequences research.

Our main contributions can be summarized as follows.

- To the best of our knowledge, we are the first to explore integer sequences symbolic regression from the view of domain adaptation.
- We propose a progressive self-learning algorithm to automatically discover recurrence formulas for target sequences from domain adaptation on integer sequence symbolic regression.
- The proposed method achieves state-of-the-art performance on both OEIS Easy25 and OEIS Easy35, and also discovers some new formulas.

Related Work

Symbolic Regression

Traditional methods of symbolic regression are based on genetic programming (Koza 1994; Schmidt and Lipson 2009), which iteratively evolve a population of candidate symbolic expressions through mutation and recombination. Some studies (Martius and Lampert 2016; Petersen et al. 2020) leverage deep neural networks to handle the combinatorial challenge of symbolic regression. For example, Martius and Lampert (2016) use elementary functions (e.g. sin, log, etc.) instead of activation functions in a fully connected neural network to automatically explore formula expressions. These methods optimize the expression to approximate the input dataset using reinforcement learning. These methods need to evolve from scratch for new input data, which reduces the efficiency of expression discovery to a certain extent.

Recently, neural machine translation-based methods (Valipour et al. 2021; Biggio et al. 2021; d'Ascoli et al. 2022) train a translation model of transformer on randomly generated data, and directly utilize the trained model for inference on target sequences without evolving. More recently, Gauthier, Olšák, and Urban (2023) employ a neural machine translation model to learn mappings from sequences to programs, and propose a self-learning method to synthesize new programs. For symbolic regression on integer sequences, d'Ascoli et al. (2022) train a translation-based model on randomly generated data, and utilize the model to learn formulas for target sequences. However, the model's ability to generalize is limited, since the generated data fail to sufficiently cover the distributions of target data. In this work, we propose a progressive self-learning method to capture the target sequence-formula distribution.

Domain Adaptation

Domain adaptation aims to transfer generalization knowledge from the source domain to the target domain, which has various applications in computer vision (Yang et al. 2021; Karouzos, Paraskevopoulos, and Potamianos 2021). Current methods can be roughly divided into two types: distance-metric-based methods (Sohn et al. 2019) and adversarial learning-based methods (Xu et al. 2020). Distance-metric-based methods learn feature transformations by minimizing the distribution difference between the source domain and the target domain to achieve distribution alignment. Typical measures of distribution differences include Maximum Mean Discrepancy (Chang et al. 2021) and Wasserstein distance (Shen et al. 2018). Adversarial learning-based methods (Ganin and Lempitsky 2015; Long et al. 2018) reduce the distribution difference between the source and target domains through adversarial training, enabling a feature extractor to develop domain-independent representations.

Although current methods of domain adaptation are effective in the image domain, they struggle with the DASRIS task, because the huge differences of formula space between the source domain and the target domain prevent the transfer of knowledge from the source domain to the target domain. Instead of transferring knowledge, we propose a self-learning method to capture the sequence-formula distribution of the target domain.

Preliminary

Problem Definition

An integer sequence is a set of ordered integers, namely $S = \{a_i\}_{i=1}^{n}$, where S represents an integer sequence, a_i

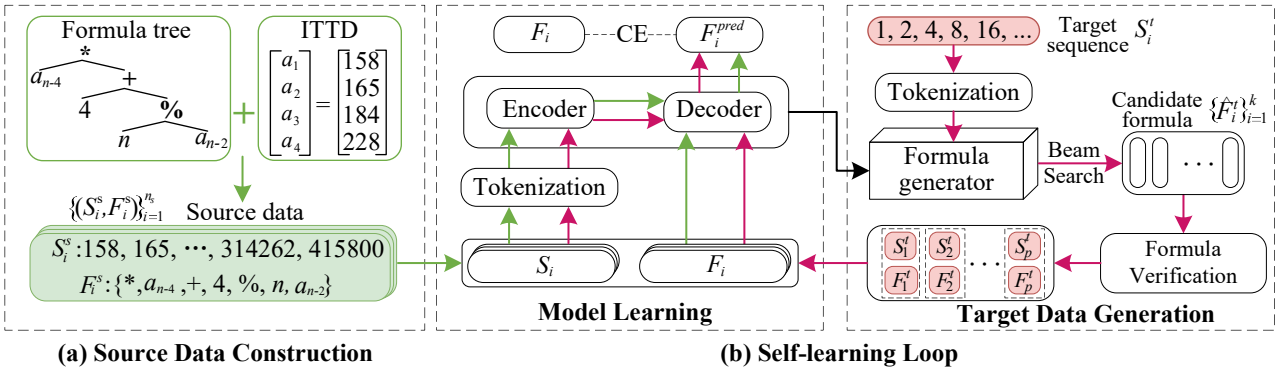

Figure 2: The pipeline of our method involves two phases: (a) Source data construction with the initial term of target domain (ITTD). (b) Self-learning Loop which consists model learning and target data generation. The green line represents the source domain data, and the red line represents the target domain data.

represents the i-th term, and n is the number of terms. The goal is to adapt a formula generator trained on source labeled data $\{(S_i^s, F_i^s)\}_{i=1}^{n_s}$ to unlabeled target domain $\{S_i^t\}_{i=1}^{n_t}$, where S_i^s and F_i^s respectively represent the i-th sequence of source domain and its corresponding formula, and S_i^t represents the i-th sequence of target domain. n_s and n_t represent the number of sequences in the source domain and the target domain, respectively. If a function F satisfies $a_i = F(i, a_{i-1}, ..., a_{i-d})$ $(d < i \leq n, d > 0)$, the function is a formula of S.

Unlike most symbolic regressions that use general formulas, we use recurrence formulas to express sequence rule. Recurrence formulas provide a more general framework for detailing the relationships among sequence terms. Additionally, some sequences may possess a recurrence formula without having a general formula. The recurrence degree is defined as the depth of the recurrence formula F, represented by the symbol d. For example, a recurrence degree of 2 means that a_n relies on both a_{n-1} and a_{n-2}. When d = 0, F represents a non-recurrence general formula.

Neural Machine Translation-based Method

Inspired by (Lample and Charton 2020; d'Ascoli et al. 2022), the sequence is considered as the source language and the formula is considered as the target language, and the translation model becomes a formula generator.

Tokenization. Different from language translation tasks, the integer term of the sequence is usually large, such as 10006556950807. If each term is coded directly by its integer value, the vocabulary of encoding term would be huge. For instance, if $1 \sim 1000$ need to be encoded, the size of vocabulary is 1000. To this end, we adopt a m-base encoding method. In this method, the integer is converted into m-base to obtain multiple digit symbols, each of which are encoded separately. For example, for a_n = 325, when the base is set to 10, i.e., $m = 10$, the encoding is $\{+, 3, 2, 5\}$. To represent the $1 \sim 1000$, the size of the vocabulary in 10-base encoding method is 10, which is far less than 1000. The m-base encoding makes integer encoding more efficient and avoids the problem of huge numerical vocabulary.

Moreover, the choice of m also needs to be discussed. For example, when the base is set to 30, the encoding of 325 becomes $\{10, 25\}$. The size of the numerical vocabulary becomes 30, while the token length of 325 is 2. When the base is set 10, the token length of 325 is 3. Choosing a value for m requires a trade-off between the input length of the sequence and the size of the numerical vocabulary.

Training Loss. The training loss is a cross-entropy loss, which is defined as follows:

$$L = -\frac{1}{n_s} \sum_{i=1}^{n_s} \sum_{j=1}^{c} y_j^i \log \hat{y}_j^i, \tag{1}$$

where n_s represents the number of symbols in the formula, and c represents the length of the whole symbol vocabulary, which includes both numerical symbols and mathematical symbols. y_j^i is a binary function that equals 1 if the true category of symbol i is j, and 0 otherwise. $\hat{y}_j^i$ represents the probability that symbol i belongs to category j.

Method

In this section, we present the proposed method PSL, designed to explicitly capture the sequence-formula distribution of target domain for DASRIS. As shown in Figure 2, the pipeline of our method involves source data construction and a self-learning loop.

Source Data Construction with ITTD

We use an automatic generation method to construct the sequence-formula data. The formula is randomly generated by constructing a binary tree and the generated formula is used to obtain the corresponding sequence.

Recurrence Formulas Generation. According to (Lample and Charton 2020; d'Ascoli et al. 2022), the binary tree of formula generation contains leaf nodes and non-leaf nodes. The non-leaf nodes are composed of both unary operators and binary operators. Unary operators include absolute, square, sign etc. Binary operators include add, sub, multiply etc. There are three types of leaf nodes: number, current index n and recurrence term $a_{n-i}, i \in [1, d]$, where d is the

recurrence degree. Specifically, the construction of the binary tree involves two steps:

1. The number of operator n_o is randomly sampled from [1, N_o], where N_o is the maximum value. The recurrence degree d is sampled from [1, D_r], where D_r is the maximum value.
2. According to recurrence degree d, the candidate sampled operators are set and sampling n_o operators act as non-leaf nodes. The leaf nodes are evenly sampled from the above three types.

A preordered walk through this binary tree give us a formula expression.

Integer Sequence Generation. A recurrence formula can generate infinite sequences by changing the initial terms. However, there is a large deviation between the sequence generated by randomly selecting the initial terms from a fixed interval and the target sequence. To alleviate this problem, we use the Initial Term of Target Domain (ITTD) to generate the sequence. ITTD is the first d terms of the sequence, where d is the recurrence degree of the formula.

To facilitate the selection of initial terms, we construct ITTD sets for different recurrence degrees. For example, the ITTD set of recurrence degree d = 4 is $\{(158, 165, 184, 228), (0, 4, 16, 52), ...\}$. When generating a sequence with a formula of the recurrence degree 4, some initial terms are randomly selected from the above ITTD set. After obtaining the initial terms, the next l terms of the sequence can be calculated by the formula, where l is uniformly sampled from [l_{min}, l_{max}], l_{min} and l_{max} represents minimum length and maximum length, respectively. The total length of the sequence is $d + l$. As shown in Figure 2 (a), the sequence $\{158, 165, ..., 314262, 415800\}$ and the prefix expression of the recurrence formula $\{*, a_{n-4}, +, 4, \%, n, a_{n-2}\}$ constitutes a training sample data in the source domain.

Self-learning Loop

This self-learning process gradually iterates between model learning and data updates to discover new formulas of target sequences. The detailed process is shown in Algorithm 1.

Model Learning. Generally speaking, a sequence may correspond to multiple matching formulas. According to Occam's razor principle (Domingos 1999), we prioritize simple formulas. We argue that simple formulas are composed of fewer operators and a lower recurrence degree. Therefore, we use the following selection strategies:

- Few-operation Priority. The formula with the fewest operators is selected, and if the counts are equal, the one with the smaller degree is retained. This strategy encourages the model to generate concise formulas.
- Little-recurrence Priority. The formula with the smallest degree is selected. When the counts are equal, the one with fewer operators is chosen. This strategy makes the model tend to produce a general formula.

If there are multiple formulas with the same number of operators and degree, each selection strategy selects at most 3 formulas. The two selection strategies are simultaneously used to pick up formulas for the next training (i.e., $\{(S_i^t, F_i^t)\}_{i=1}^{m_t}$, m_t is the number of selecting formulas).

Initially, only source domain data is used to train the model. Subsequently, both the source domain data and selecting data from target domain (i.e. $\{(S_i^t, F_i^t)\}_{i=1}^{m_t}$) are employed to train the model. To ensure that high-quality recurrence formulas are learned, a target domain sequence containing at least D_r terms is selected for training, and $1 \sim n_v$ terms are reserved for verifying the correctness of the generated formula. The relationship between the number of training sequence terms l^{tr} and the total number of terms in the sequence l_t is given by Eq. (2):

$$l^{tr} = \begin{cases} D_r, \ l_t \leq D_r + n_v \\ \min(l_t - n_v, 25), \ l_t \geq D_r + n_v \end{cases} \tag{2}$$

where l^{tr} is less than 25 and n_v=10.

Based on the generated data, a neural machine translation-based model can be trained with the tokenization and training loss in preliminary section. Then the trained translation-based model can be considered as a formula generator.

Target Data Generation. The above learned formula generator attempts to produce formulas for sequences of the target domain. This process involves candidate formula generation and formula verification.

Candidate Formula Generation. For all sequences on target domain $\{S_i^t\}_{i=1}^{n_t}$, the formula generator is utilized to generate candidate formulas. To obtain diverse formulas, we use the beam search decoding with width k. Thus the formula generator produces k candidate formulas for each target sequence, which is denoted as $\{\hat{F}_{i,j}^t | j = 1, ..., k\}$.

Formula Verification. As some generated candidate formulas $\hat{F}_{i,j}^t$ may not match their corresponding sequences, it is necessary to verify the correctness of these candidate formulas. For each candidate formula, the predicted terms are calculated by using initial terms of the formula. If the predicted terms are equal to the terms of the sequence except for the initial terms, the candidate formula is considered as a matching formula for the sequence.

To augment training data, we check our candidate formula with all target domain sequences using a limited form

Algorithm 1: The overview of our PSL.

Input: Target sequence $\{S_i^t\}_{i=1}^{n_t}$.
Output: Formula of target sequence $\{F_i^t\}_{i=1}^{n_t}$.

1: Randomly generating sequences and formulas as source domain data $\{(S_i^s, F_i^s)\}_{i=1}^{n_s}$ via binary trees and ITTD.
2: Initialization of training data $D := \{(S_i^s, F_i^s)\}_{i=1}^{n_s}$.
3: Training formula generator M on D.
4: Using the formula generator M to generate candidate formulas, and verifying these candidate formulas to form new data $(S_i^t, F_i^t)_{i=1}^{m_t}$.
5: Updating $D := (S_i^t, F_i^t)_{i=1}^{m_t}$.
6: If iteration completes, goes to next, else go to step 4.
7: Testing formula generator with $\{S_i^t\}_{i=1}^{n_t}$ to output $\{F_i^t\}_{i=1}^{n_t}$.

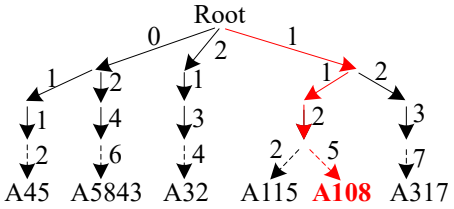

Figure 3: Sequence tree of target domain.

of hindsight experience replay (Andrychowicz et al. 2017). These target domain sequences are organized into a tree of sequence, as shown in Figure 3. Once the predicted sequence reaches a leaf in the tree or gets a branch that does not exist in the tree, the verification process stops. If the predicted sequence reaches a leaf in the tree, it indicates that the formula matches the sequence.

Experiments

Datasets and Experimental Setups

Source Domain Data. As mentioned in subsection of recurrence formulas generation, ordinary recurrence formulas are generated according to the following parameters: N_o = 10, D_r = 12, l_{min} = 13, l_{max} = 30. In the construction process of ordinary recurrence formula, the probability of generating nonlinear recurrence formulas, e.g., $a_n = (n+1)a_{n-2} - a_{n-3}$, is much greater than that of generating linear recurrence formulas, e.g., $a_n = 5a_{n-1} + 2a_{n-2} - 3a_{n-3}$. In this case, the random generated ordinary recurrence formulas contain extremely few linear recurrence formulas. However, the linear recurrence formula is a common form in the target domain. To this end, we also construct linear recurrence formulas to improve the model's ability for identifying them. In the construction process of linear recurrence formula, the operator list of linear recurrence only contains $\{+, -, *\}$, and the leaf node does not contain index n.

Target Domain Sequence. The Online Encyclopedia of Integer Sequences (OEIS) (Sloane et al. 2018) is an online database with more than 360,000 integer sequences. However, many OEIS sequences do not have closed-form recursive relationships such as the stops on the New York City Broadway Line subway (A000053), which naturally leads to difficulties in discovering formulas for these sequences. Fortunately, OEIS sequences, labeled as "easy", mean that there is a logic to find the next terms on most cases. We select 'easy' sequences with no less than 25 terms as target domain sequences, obtaining more than 60,000 target sequences.

To ensure fair comparisons with existing works (d'Ascoli et al. 2022), we create two test sets with the keyword "easy":

- OEIS Easy25 is collected from first 10,000 sequences of OEIS with no less than 25 terms.
- OEIS Easy35 is collected from first 10,000 sequences with no less than 35 terms.

Implementation Details. Similar to (Lample and Charton 2020), we employ an encoder-decoder transformer-based architecture (Vaswani et al. 2017) as translation model. Both encoder and decoder include 6 hidden layers with 8 attention heads and 512 embedding dimensions. The Adam optimizer is utilized with a learning rate from 10^{-7} to $4 * 10^{-4}$ in the first 200 steps, and then decaying according to the reciprocal square root of the number of steps.

During tokenization, we choose 10000-base for integer sequences encoding to ensure effective representation without excessively enlarging the vocabulary. During inference, $k = 32$ candidate formulas are generated for each target sequence through beam search, and the value of k is determined experimentally. In our experiments, the total number of iterations is 50. Since the training data for subsequent iterations after the first iteration differ in order of magnitude, we set different epochs accordingly. For the first iteration (Iter1), we train 10 epochs on randomly generated source data, which contain 25 million ordinary recurrence sequence-formulas pairs and 25 million linear recurrence sequence-formulas pairs. For other iterations (Iter2-Iter50), we train on the target domain data for 100 epochs, with batch size of 256. The experiments are performed on four Tesla T4 GPUs with 16G memory, and the time for one iteration is about 40 minutes.

Evaluation Metric. Since there are no formulas for many target sequences, or the formulas have multiple expression forms, it is impossible to directly determine whether the generated candidate formulas are correct by symbolic comparisons. In this paper, the correctness of the candidate formula is judged by comparing the predicted sequence $\{\hat{a}_j\}$ and the ground-truth sequence $\{a_j\}$. If each term of $\{\hat{a}_j\}$ is equal to $\{a_j\}$, then the candidate formula is considered as a matching formula for the input sequence. For the sequence of the target domain S^t, if there is at least one matching formula among the generated k candidate formulas, it is considered that the formula of the sequence has been found.

The evaluation metric is calculated with Eq. (3).

$$acc(n_p) = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}(\max_{n_i \le j \le \text{len}} |\hat{a}_{ij} - a_{ij}| = 0), \quad (3)$$

where N is number of testing sequences, $len = n_i + n_p$, n_i is the number of sequence terms in the input model, n_p is the number of terms in the predicted sequence, a_{ij} represents the j-th term of the i-th sequence in the test set, $\hat{a}_{ij}$ represents the j-th term of the predicted sequence of the i-th sequence in the test set.

Experimental Results

Comparative Methods. We compare several advanced methods, which include Mathematica's built-in function FindSequenceFunction (FSF) (d'Ascoli et al. 2022), FindLinearRecurrence (FLR) (d'Ascoli et al. 2022), and the deep symbolic regression (DSR) method proposed by (d'Ascoli et al. 2022). The FSF is used to discover non-recurrence formulas, while the FLR is used to discover linear recurrence formulas. The DSR only trains the model (i.e., formula generator) on random generation data, thus it can be considered as our baseline method. To the best of our knowledge, there is no relevant work designed for Symbolic Regression of Integer Sequences yet.

Model	OEIS Easy25 ($n_i = 15$)		OEIS Easy35 ($n_i = 25$)	
	$n_p = 1$	$n_p = 10$	$n_p = 1$	$n_p = 10$
FSF	17.1	12.0	8.1	7.2
FLR	17.4	14.8	21.2	19.5
DSR	33.4	19.2	34.5	21.3
PSL (ours)	**53.1**	**27.4**	**54.9**	**29.5**

Table 1: Accuracy (%) of different methods on the OEIS Easy25 and OEIS Easy35.

Parameter group (PG)	Ordinary recurrence	Linear recurrence	Recurrence degree	Initial terms	OEIS Easy35 ($n_i = 25$)	
					$n_p = 1$ (%)	$n_p = 10$ (%)
PG1	5 million	0	d = 6	RIT	17.2	10.8
PG2	5 million	0	d = 6	ITTD	22.7	14.7
PG3	5 million	0	d = 12	ITTD	25.8	18.7
PG4	3 million	2 million	d = 12	ITTD	25.0	19.0
PG5	5 million	2 million	d = 12	ITTD	25.5	**19.5**

Table 2: The effect of different hyperparameters on accuracy. RIT: Random Initial Terms. ITTD: Initial Term of Target Domain.

Results. The experimental results on OEIS Easy25 and OEIS Easy35 are shown in Table 1. It can be observed that our proposed method PSL achieves the best performance. Compared with the function-based methods FSF and FLR, our method obtains over 30% gains under $acc(n_{p=1})$ and over 10% gains under $acc(n_{p=10})$ on both OEIS Easy25 and OEIS Easy35. FSF and FLR both employ the function approximation manner to handle continuous real numbers, while they struggle to deal with discrete integers. The method of function approximation may not be suitable for symbolic regression of integer sequences. Compared to DSR, our method achieves about 20% improvements under $acc(n_{p=1})$ and over 8% improvements under $acc(n_{p=10})$ on both OEIS Easy25 and OEIS Easy35. DSR only uses randomly generated data and struggles to generalize to the target sequences. In contrast, our method uses both randomly generated data and target sequences, and progressively captures the rule of target sequences via a self-learning manner. These significant performance improvements show the superiority of our method.

Comparing the performance between $acc(n_{p=1})$ and $acc(n_{p=10})$, the performance of $acc(n_{p=1})$ is obviously higher than that of $acc(n_{p=10})$. This is because predicting 10 terms is much harder than predicting one. When comparing the performance on OEIS Easy25, both the DSR and our method perform better on OEIS Easy35. The underlying meaning is that using longer sequences allows the model to efficiently capture the rule of sequence. This seems to suggest that it is a good choice for using long sequences to learn their formulas in practical application.

Experimental Analysis and Discussions

To further explore the impact of different factors, we design comparative experiments, as shown Table 2. These impact factors include initial terms (RIT or ITTD), recurrence degree d, data size and data formation. All models are trained with 10 epochs on 5 million or 7 million sequence-formula pairs, and they are tested under $acc(n_{p=1})$ and $acc(n_{p=10})$ evaluation metric on OEIS Easy35.

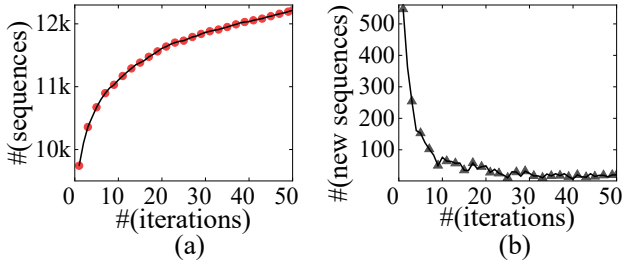

Figure 4: The number of discovery formulas with the training iterations on OEIS. #(X) represents the number of X.

Effectiveness of ITTD. Compared with PG1, PG2 changes the initial terms with ITTD, obtaining 5.5% and 3.9% improvements under $n_p = 1$ and $n_p = 10$, respectively. The domain gaps can be reduced with ITTD, which contribute to learn a suitable model for target domain.

Effect of Recurrence Degree. Comparing PG2 and PG3, we observe that increasing recurrence degree d in the source data can obtain accuracy improvements, e.g., 3.1% gains under $n_p = 1$ and 4.0% gains under $n_p = 10$. Big recurrence degree helps build more complex sequences and formulas, which are beneficial for model learning.

Effect of Data Size and Data Formation. Compared with PG3, PG5 with increasing the number of linear recurrence sequences obtains 0.8% improvements under $n_{p=10}$. Analogously, compared with PG4, PG5 with increasing the number of ordinary recurrence sequences obtains 0.5% improvements under $n_{p=10}$. Moreover, comparing PG3 and PG4, the model with more linear recurrence sequences obtains stable sequence predictions. Increasing the amount of data can slightly improve the ability of the model, and increasing linear recurrence is a good choice for stability.

OEIS	Sequences	Discovery formulas	Keyword	Formulas type
A158525	38, 134, 462, 1582, 5406, 18462	$a_n = 4a_{n-1} - 2a_{n-2} + 2$	easy	linear recurrence
A026476	1, 3, 4, 9, 12, 23, 26, 37, 40, 51	$a_n = a_{n-2} + 13 + sign(a_{n-5})$	easy	linear recurrence
A014642	0, 8, 40, 96, 176, 280, 408, 560	$a_n = a_{n-1} + 24n - 40$ $a_n = (6n-7)^2//3$	easy	linear recurrence
A264041	1, 3, 6, 10, 16, 21, 29, 36, 46, 55	$a_n = n^2 - a_{n-1} + (n+1)//6$	more	nonlinear recurrence
A273331	1, 5, 17, 69, 281, 1129, 4521	$a_n = 1 + 4 * (a_{n-1} + 1\%a_{n-3})$	more	nonlinear recurrence
A134342*	0, 2, 5, 9, 15, 24, 38, 59, 90, 137	$a_n = a_{n-1} + (a_{n-1} + 4)//2$	hard	nonlinear recurrence
A213685*	1, 3, 6, 9, 12, 17, 22, 28, 33, 41	$a_n = (3n-2)//2 + a_{n-4} + n$	hard	nonlinear recurrence
A157795*	1, 2, 4, 6, 9, 12, 15, 18, 22, 26, 31	$a_n = n + (n^2 + 1 - n\%5)//6$	hard	nonlinear recurrence

Table 3: Discovery Formulas. $*$ means the sequence without a known formula in OEIS.

Iterative Analysis. We record the number of discovery formulas on target domain sequences during the self-learning process. As shown in Figure 4 (a), the model learned from the source domain discovers over 9500 formulas. As training goes on, more formulas are discovered with culminating over 12k formulas.

As shown in Figure 4 (b), the number of newly discovered formulas decreases as training progresses. This is because, in the early iterations, most sequence formulas containing relatively simple rules are discovered. Consequently, subsequent iterations can only discover more complex sequence formulas, which are not easily discovered correctly. The above observations illustrate the strong capability of the continuous iterative self-learning method in discovering simple formulas, as well as its potential to uncover complex ones. This suggests that our method is well-suited for progressive learning from simple to complex formulas.

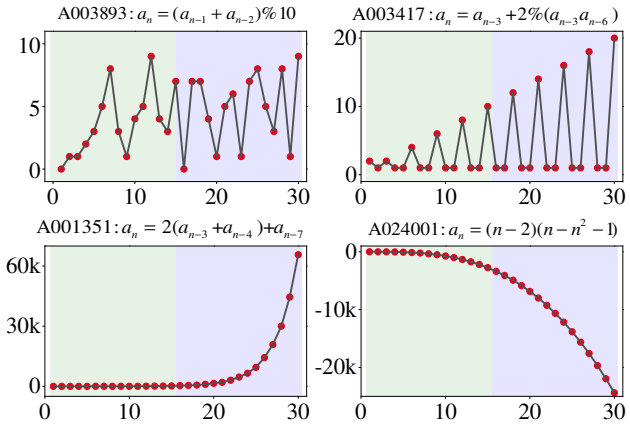

Figure 5: The function curve of discovery formulas.

Formula Discovery and Analysis

We have found many formulas for OEIS sequences that cannot be found before. Table 3 shows some sequences and corresponding discovery formulas, which include labeled 'easy' formulas, labeled 'more' formulas, labeled 'hard' formulas, linear recurrence formulas and nonlinear recurrence formulas. In linear recurrence formulas, the proposed method discovers the formulas for A158525, which has significantly varying initial items and discovers the formulas for A026476 with an uncommon symbol $sign(a_{n-5})$. Our method also find different types of formulas for A014642, which contain both the recurrence formula and the general formula. In nonlinear recurrence formulas, the symbolic expressions contain some nonlinear operations such as division $//$, modulus $\%$. The formula of A157795 contains both $//$ and $\%$ symbol, and the formula of A27331 contains an unexpected symbolic expression $1\%a_{n-3}$. These findings highlight the robustness of our model in formula discovery.

Figure 5 shows the function curve of several discovery formulas. The 15 front terms of the sequence are input, and the 15 back terms are predicted. It can be seen that our model can find formulas for both monotonic sequences, e.g., A001351, A024001, and difficult nonmonotonic sequences, e.g., A003893, A003417. In the nonmonotonic sequences A003417, the first 15 terms and the last 15 terms show the similar change tends, while the last 15 terms have larger magnitude changes. In the monotonic sequences A001351 and A024001, although the first 15 terms and the last 15 terms have completely different change tends, our method can still capture them. These prove the effectiveness of our method in formulas discovery of both monotonic sequences and nonmonotonic sequences.

Conclusion

In this paper, we propose a progressive self-learning (PSL) method to explicitly capture sequence-formula distributions of the target domain to discover formulas of target sequences. The PSL involves a manner of source data construction with ITTD and a self-learning loop strategy. The construction of the source data aims to narrow the distribution gap between the source domain and the target domain. The self-learning loop strategy aims to enhance the model's capability by explicitly learning the sequence-formula distribution of the target domain. Experimental results on OEIS dataset exhibit the proposed method achieves state-of-the-

art performance and also discovers some new formulas. Our method uses a single model for entire sequences, making it challenging to capture various formula patterns. In future work, we will investigate a mixture of expert models to identify potential formula patterns.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant NO.62202058, NO.62376017, N0.62272009, and Fundamental Research Funds for the Central Universities (buctrc202221).

References

Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Pieter Abbeel, O.; and Zaremba, W. 2017. Hindsight Experience Replay. In *NeurIPS*.

Biggio, L.; Bendinelli, T.; Neitz, A.; Lucchi, A.; and Parascandolo, G. 2021. Neural symbolic regression that scales. In *ICML*, 936–945.

Chang, J.; Li, J.; Kang, Y.; Lv, W.; Xu, T.; Li, Z.; Xing Zheng, W.; Han, H.; and Liu, H. 2021. Unsupervised domain adaptation using maximum mean discrepancy optimization for lithology identification. *Geophysics*, 86(2): ID19–ID30.

d'Ascoli, S.; Kamienny, P.-A.; Lample, G.; and Charton, F. 2022. Deep symbolic regression for recurrence prediction. In *ICML*, 4520–4536.

Domingos, P. 1999. The role of Occam's razor in knowledge discovery. *Data mining and knowledge discovery*, 3: 409–425.

Ganin, Y.; and Lempitsky, V. 2015. Unsupervised domain adaptation by backpropagation. In *ICML*, 1180–1189.

Gauthier, T.; Olšák, M.; and Urban, J. 2023. Alien coding. *arXiv preprint arXiv: 2301. 11479*.

Karouzos, C.; Paraskevopoulos, G.; and Potamianos, A. 2021. UDALM: Unsupervised domain adaptation through language modeling. *arXiv preprint arXiv:2104.07078*.

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

Lample, G.; and Charton, F. 2020. Deep Learning For Symbolic Mathematics. In *ICLR*.

Long, M.; Cao, Z.; Wang, J.; and Jordan, M. I. 2018. Conditional adversarial domain adaptation. *NeurIPS*.

Lu, L.; Jin, P.; Pang, G.; Zhang, Z.; and Karniadakis, G. E. 2021. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature machine intelligence*, 3(3): 218–229.

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

Petersen, B. K.; Larma, M. L.; Mundhenk, T. N.; Santiago, C. P.; Kim, S. K.; and Kim, J. T. 2020. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *ICLR*.

Schmidt, M.; and Lipson, H. 2009. Distilling free-form natural laws from experimental data. *science*, 324(5923): 81–85.

Shen, J.; Qu, Y.; Zhang, W.; and Yu, Y. 2018. Wasserstein distance guided representation learning for domain adaptation. In *AAAI*, volume 32.

Sloane, N. J.; et al. 2018. The on-line encyclopedia of integer sequences. *Published electronically at https://oeis.org*.

Sohn, K.; Shang, W.; Yu, X.; and Chandraker, M. 2019. Unsupervised Domain Adaptation for Distance Metric Learning. In *ICLR*.

Valipour, M.; You, B.; Panju, M.; and Ghodsi, A. 2021. Symbolicgpt: A generative transformer model for symbolic regression. *arXiv preprint arXiv:2106.14131*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *NeurIPS*.

Wang, Y. 2022. A Proof of Goldbach Conjecture by Mirror-Prime Decomposition. *WSEAS Transactions on Mathematics*, 21: 563–571.

Xu, M.; Zhang, J.; Ni, B.; Li, T.; Wang, C.; Tian, Q.; and Zhang, W. 2020. Adversarial domain adaptation with domain mixup. In *AAAI*, 6502–6509.

Yang, J.; Shi, S.; Wang, Z.; Li, H.; and Qi, X. 2021. St3d: Self-training for unsupervised domain adaptation on 3d object detection. In *CVPR*, 10368–10378.