TPSR: Transformer-based Planning for Symbolic Regression

Parshin Shojaee[*1] , **Kazem Meidani**[*2]
[1] Department of Computer Science, Virginia Tech
[2] Department of Mechanical Engineering, Carnegie Mellon University

Abstract

Symbolic regression (SR) is a machine learning task that finds a mathematical expression based on a function's values. Recent progress in SR highlights the efficiency of pre-trained transformer models in creating equations, benefitting from large-scale synthetic pre-training and faster inference time compared to GP-based methods. However, these models often ignore equation-specific objectives like accuracy and complexity, focusing mainly on NLP's token-matching optimization. To rectify this, we suggest a Transformer Planning strategy for Symbolic Regression, TPSR, which merges Monte Carlo Tree Search with transformer decoding. Unlike traditional decoding methods, TPSR uses fitting accuracy and complexity— non-differentiable feedback— as extra sources of knowledge in the equation generation process.

Motivation

Symbolic regression (SR) includes numerous methods across various categories. Classic techniques like Genetic Programming (GP) use heuristic search strategies where each item symbolizes a potential problem solution. Despite their capability to solve complex, nonlinear problems, GP algorithms are generally slow to converge and computationally intensive due to the immense functional search space. They are also known to overfit and are highly sensitive to parameter selection. Recent SR developments demonstrate promising outcomes with pre-trained transformers, creating equations as token sequences. These models use large-scale pre-training and generate equations quickly, resulting in faster inference times than GP-based methods. But, their focus is primarily on supervised pre-training objectives from text generation, often leading to a sub-optimal performance with respect to equation-specific objectives such as fitting accuracy and complexity. To address this, we propose TPSR, a Transformer Planning strategy for Symbolic Regression. It leverages a lookahead planning algorithm, using Monte Carlo Tree Search (MCTS) as a decoding strategy on top of pre-trained transformer-based SR models to guide equation sequence generation. TPSR significantly improves the performance of generated equations by considering feedback during the generation process and still remains faster than GP-based models which do not leverage the pre-training priors and learn each expression from scratch.

Preliminaries

Symbolic Regression (SR) endeavors to determine the symbolic form of an unknown function $f(\cdot)$, which transforms an d-dimensional input $\boldsymbol{x} \in \mathbb{R}^d$ into a target variable $y = f(\boldsymbol{x}) \in \mathbb{R}$. Using a dataset of n samples $\mathcal{D} = (\boldsymbol{x}_i, y_i)_{i=1}^n$, SR methods strive to generate an equation $\tilde{f}(\cdot)$ to approximate $y_i \approx \tilde{f}(\boldsymbol{x}_i)$ for every $i \in \mathbb{N}_n$. The resultant equation should ideally generalize well and balance between fitting precision and complexity. Transformer-based SR models are trained on an extensive dataset of equation instances $\{(\mathcal{D}_1, f_1(\cdot)); \ldots ; (\mathcal{D}_M, f_M(\cdot))\}$, with M being the dataset size. During the process, the model generates the equation $\tilde{f}(\cdot)$ sequentially in an autoregressive fashion. They typically use prefix notation for representing the equation expression tree as a sequence. These models first encode the input observations and then decode the equation sequence using the encoded data and masked tokens. Token-level cross-entropy loss is utilized for training the

*Contact: parshinshojaee@vt.edu, mmeidani@andrew.cmu.edu

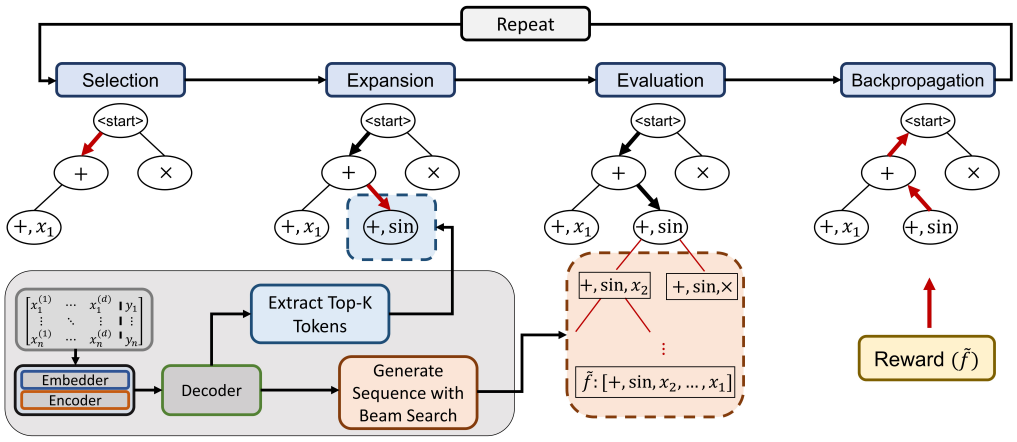

Figure 1: Overview of TPSR's key steps: Selection, Expansion, Evaluation, and Backpropagation. MCTS-guided decoding interacts with the pretrained transformer SR model in the expansion and evaluation steps employing the transformer *top-k* sampling and beam search, respectively. The designed reward is used to guide the backpropagation.

model and learning the distribution of the next token prediction. However, generating accurate constants in the equation for optimal fitting performance can be challenging. A solution involves optimizing the constants in the produced skeleton or equation using nonlinear techniques, like the Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS). Previous studies have combined constant optimization with transformer decoding via beam search and sampling strategies to suggest multiple candidate equations. These are then evaluated using fitting metrics such as R^2 to rank them and select the best-performing equation. Although transformer models using beam search or sampling decoding can generate several high-likelihood equation sequences, they are dependent on logits derived from pretrained model parameters via token-matching loss. Consequently, they lack the ability to optimize equation generation for specific objectives, such as equation fitting or complexity.

MCTS-Guided Equation Generation

To create better-fitting and simpler equations, we use Monte Carlo Tree Search (MCTS) during inference to infuse feedback into equation generation, steering the decoder towards optimal solutions (Fig. **??**(c)). This MCTS-guided transformer decoding explores multiple possibilities to find the most promising paths. In this method, we treat the SR equation creation as a Markov Decision Process (MDP), with a state s symbolizing the current sequence at the generation phase t. If s hasn't reached the end (i.e., the <EOS> token), we select the next token as action a, updating state s' by combining s and a. When we hit the end, the reward r is calculated and used to update the decoding model. During the generation phase, MCTS operates iteratively to plan ahead and determine the next token. But this broad search-space necessitates integrating MCTS with the pre-trained transformer model to generate high-quality equations. Therefore, we incorporate next-token probabilities from the pre-trained transformer SR models into the MCTS planning process, improving the search process. The key MCTS stages for transformer decoding in SR models (Fig. 1) are:

Selection. The Upper Confidence Bound for Trees (UCT) criterion is employed to select actions (i.e., next tokens) for fully extended nodes in the search tree, balancing exploration and exploitation. We use the P-UCB heuristic as

$$\mathrm{UCT}(s,a) = Q(s,a) + \beta(s) \cdot P_\theta(a|s) \cdot \sqrt{\frac{\ln\left(N(s)\right)}{1+N(s')}}, \tag{1}$$

where $Q(s,a)$ is the maximum return for action a in state s across all simulations, promoting the exploitation of the optimal child node. The second term encourages exploration of less-visited children, with $N(s)$ as state s's visit count and s' as the subsequent state. $P_\theta(a|s)$ is the probability of the next token a given the partial sequence state s from pretrained transformer model parameterized by θ. The exploration-exploitation trade-off is adjusted by $\beta(s)$, which depends on state s's visit count. Lastly, the next token action maximizes the UCT: $\mathrm{Select}(s) = \arg\max_a \mathrm{UCT}(s,a)$.

Expansion. In the expansion stage, after selecting a node that is not fully expanded, a new child (next token) for the current state is explored. Random expansion of the node from the vocabulary, however, might result in an invalid equation (that does not comply with the prefix notation) and makes the process very time-consuming. Therefore, given partial equations, only *top-k* most likely choices of the next token are considered as the possible children of the node for expansion. In other words, we are restricting the actions to be only from the *top-k* high-likelihood options which are retrieved from the pretrained transformer SR model's logits. These options are then ordered to determine the sequence in which the children will be expanded.

Evaluation. To evaluate the newly expanded nodes, we perform simulations to complete the equation sequence. This is necessary because the new state may still be a partial equation and performance feedback can only be obtained at the end of the sequence when the equation generation is completed. In MCTS, it is common to employ random actions during the simulation stage. Nevertheless, random action selection for equation generation, much like during expansion, suffers from certain drawbacks in terms of time and the possibility of generating invalid equations. Consequently, the pretrained transformer SR model is invoked again, this time utilizing beam search with a beam size of b, to generate complete equation candidates based on the current state. The beam size b determines the number of complete equations to be generated from the current partial equation. Following the simulations, the highest reward among all the candidates is assigned to the new node value.

Backpropagation. After generating a complete equation $\tilde{f}(\cdot)$, the corresponding reward $r(\tilde{f}(\cdot))$ can be computed. The highest reward among all simulations is then assigned to the new node, which recursively backpropagates its estimated value to its parents until it reaches the root of the tree. This update process involves updating the Q values of all state-action pairs, denoted as s' and a', along the trajectory in the tree to reach the root. Specifically, for each state-action pair, the Q value is updated by taking the maximum of the current Q value and the new value r: $Q(s', a') \leftarrow \max(Q(s', a'), r)$.

Reward Definition

We define a numerical reward $r \in \mathbb{R}$ to evaluate complete equation candidate $\tilde{f}(\cdot)$, promoting fitting accuracy and regulating complexity. After optimizing constants in the complete sequence, we compute the reward. We first calculate the normalized mean squared error (NMSE) between ground-truth target variable y and predicted target variable $\tilde{y} = \tilde{f}(\boldsymbol{x})$, and formulate the reward as:

$$r(\tilde{f}(\cdot)|\boldsymbol{x}, y) = \frac{1}{1 + \text{NMSE}(y, \tilde{f}(\boldsymbol{x}))} + \lambda \exp(-\frac{l(\tilde{f}(\cdot))}{L}), \tag{2}$$

where l represents equation complexity as the sequence length in prefix notation; L denotes the model's maximum sequence length; and λ is a hyperparameter balancing fitting and complexity reward. Higher λ values favor less complex equations, encouraging best-fitting and penalizing non-parsimonious solutions. NMSE is calculated as $(\frac{1}{n}\|y - \tilde{f}(\boldsymbol{x})\|_2^2)/(\frac{1}{n}\|y\|_2^2 + \epsilon)$, where ϵ is a small constant to prevent numerical instability.

Algorithm Pseudo-code

Algorithm 1 outlines the specifics of the MCTS-Guided decoding strategy in SR. The segments highlighted in blue correlate with the use of reward and selection functions as defined in Eqs. (2) and (1). These equations are crucial in guiding the MCTS-based Transformer Decoding technique for SR, guaranteeing effective probing and optimal use of the search space. The steps highlighted in red in the algorithm mark the points where the pre-trained transformer SR model is invoked to retrieve the *top-k* next tokens and equation candidate beams. These collected tokens and beams are used in the MCTS algorithm's expansion and evaluation stages, respectively. The integration of the pre-trained transformer SR model enables the MCTS-Guided decoding strategy to effectively employ the model's intrinsic semantic knowledge acquired during extensive pre-training, improving the quality of equation candidates and enhancing the overall performance of the SR method.

Algorithm 1: MCTS-Guided Decoding for Symbolic Regression

Input : r_{max}: maximum number of rollouts, k_{max}: number of children of nodes used for *top-k* next token selection, b: beam size, c: P-UCB exploration parameter

```
while r < r_max do
    node ← root;
    1) Selection
    while |node.children| > 0 do
        node ← SELECT(node.children, c);
    end
    2) Expansion
    next tokens ← TOP_K(node, k_max);
    for action ∈ next tokens do
        next state ← CONCAT(node, action);
        Add next state as a child of node;
    end
    3) Evaluation
    Equation ← BEAM_SEARCH(node, b);
    reward ← GET_REWARD(Equation);
    Save (Equation, reward) pair in a dictionary. ;
    4) Backpropagation
    Update values on the trajectory given the reward ;
end
Return Equation with the highest reward ;
```

Implementation Details

Our model implementation builds upon the open-source End-to-End Transformer (E2ET) model [**?**], chosen as our pre-trained transformer SR backbone due to its publicly accessible architecture, weight parameters, and logits in the repository [1]. The above algorithm outlines our model's steps, and the experiment implementation code along with setup details for replicability can be found in the competition repo [2]. For our experiments, we stipulate the maximum sequence length of the model to be $L = 200$. The constant safeguarding numerical stability, ϵ, used in the NMSE computation $(\frac{1}{n}|y - \tilde{f}(\boldsymbol{x})|_2^2)/(\frac{1}{n}|y|2^2 + \epsilon)$, is assigned a value of $1e - 9$. The best found models for the first, second, and third datasets were determined by setting the maximum number of node expansions (k_{max}) to 5, 3, and 10 respectively. Correspondingly, the beam size for simulations (b) was configured at 3 for the first dataset, 1 for the second, and 3 for the third. Furthermore, the number of rollouts (r) was standardized at 9 for both the first and third datasets, while it was set to 3 for the second dataset. Following hyperparameter optimization, we set the complexity-controlling parameter (λ) to 0.4 for the implementation of the framework over all three datasets. For our testing, we divided the three provided datasets into training and testing subsets at a 75%/25% ratio.

[1] `https://github.com/facebookresearch/symbolicregression`

[2] `https://github.com/ufabc-bcc/srbench-competition-2023-track-1-pksm/tree/main/`