Complexity-Aware Deep Symbolic Regression with Robust Risk-Seeking Policy Gradients

Zachary Bastiani [1,2]   **Robert M. Kirby** [1,2]   **Jacob Hochhalter** [3]   **Shandian Zhe** [1]

[1] Kahlert School of Computing   [2] Scientific Computing and Imaging Institute
[3] Department of Mechanical Engineering
University of Utah
{u0450013, jacob.hochhalter} @utah.edu
{kirby, zhe} @cs.utah.edu

Abstract

This paper proposes a novel deep symbolic regression approach to enhance the robustness and interpretability of data-driven mathematical expression discovery. Despite the success of the state-of-the-art method, DSR, it is built on recurrent neural networks, purely guided by data fitness, and potentially meet tail barriers, which can zero out the policy gradient and cause inefficient model updates. To overcome these limitations, we use transformers in conjunction with breadth-first-search to improve the learning performance. We use Bayesian information criterion (BIC) as the reward function to explicitly account for the expression complexity and optimize the trade-off between interpretability and data fitness. We propose a modified risk-seeking policy that not only ensures the unbiasness of the gradient, but also removes the tail barriers, thus ensuring effective updates from top performers. Through a series of benchmarks and systematic experiments, we demonstrate the advantages of our approach.

1   Introduction

Interpretability is an essential measure of machine learning models. Although large complex models, such as those with billions of parameters, have become standard practice in many fields, these models are typically black-box and provide little insight about the data. This can raise severe concerns regarding the reliability of deploying such models, particularly in scientific and engineering domains. Symbolic regression (SR) [6, 12] is an important research direction to achieve interpretability in machine learning. Given a dataset that measures the input and output of a complex system of interest, symbolic regression aims to find a simple, concise equation to reveal the underlying mechanism of the system as to improve the understanding of the system and ensure reliability of the model.

For a long time, Genetic programming (GP) [8, 11, 2] has been the mainstream approach for symbolic regression. However, GP is known to be computationally costly and time consuming due to the its evolutionary nature. The recent state-of-the-art, Deep Symbolic Regression (DSR) [10], instead uses a recurrent neural network (RNN) to generate expression trees, and employs a risk-seeking policy to train the RNN via reinforcement learning. While DSR is successful in many SR benchmarks, the RNN-based architecture might cause learning issues, such as vanishing gradients [4], especially in large tree structures. In addition, DSR uses data fitness as the reward, which can tend to generate complex expressions to overfit the data, especially with the presence of noise. Furthermore, due to the usage of the reward difference as the weights in the policy gradients, DSR takes the risk of meeting tail barriers, namely, zero policy gradients, which can render the learning to be only driven by the entropy term, leading to over-exploration and inefficient model updates.

To overcome these limitations, we propose CADSR, a novel deep symbolic regression approach. The major contributions of our work are summarized as follows.

- **Expression Model.** We use transformers [21] to build the expression generation model, which can fully leverage the contextual information of each node in the expression tree and overcome the vanishing gradient issues to improve the learning performance. To obtain a good representation, we design tree node encodings based on both the depth and horizontal position. We couple with breadth-first-search (BFS) to sample the expression tree, which is not only computationally efficient but also straightforward to implement.

- **Reward Design.** We design a Bayesian Information Criterion (BIC) as the reward function, for which we use the number of nodes and constant tokens in the expression tree to indicate the model complexity. In this way, our reward function considers not only the data fitness but also the expression complexity. The reinforcement learning can then seek to optimize the trade-off between the interpretability and data fitness of the expressions, avoiding producing over-complicated expressions that overfit the data, especially at present of noises.

- **Policy Gradient.** We propose a modified risk-seeking policy gradient for our BIC-based reward. Instead of using the reward difference as the weight to compute the gradient, we use a simple step reward mapping, which gives a constant weight. We show that this guarantees to prevent the learning from meeting any tail barrier or partial tail barrier. Accordingly, we can effectively leverage all the top performers for model updating, avoiding wild, less useful exploration.

- **Experiments.** We evaluated the performance of our method, CADSR, on the standard SR benchmark and ablation studies. In addition to DSR, we compared with fourteen other popular and/or state-of-the-art SR methods and several commonly used machine learning approaches. The performance of CADSR in both symbolic discovery and prediction accuracy is consistently among the best. In particular, the symbolic discovery rate of CADSR is the highest when data includes significant noises. In all the cases, CADSR generates the most interpretable expressions, while maintaining a high level of accuracy. CADSR outperforms the most comparable model, DSR, in all categories showing that it is a direct improvement.

2   Background

Given a set of input and output examples collected from the target system, denoted as $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, symbolic regression aim to identify a concise mathematical expression that characterizes the input-output relationship, such as $y = \sin(2\pi x_1) + \cos(2\pi x_2)$. Deep symbolic regression (DSR) [10] discovers equations via an RNN-based reinforcement learning approach [16], which can be broken down into four parts: environment, actor, reward, and policy.

The environment is designed to be the creation of an expression tree that represents a specific equation. Expression trees are directed trees where each node holds a token from the available list of operations and variables. For example, a common list of tokens would be $\{+, -, \times, /, \sin, \cos, x_1, 1\}$. Expression trees are built by selecting nodes in a preorder traversal of the tree with each tree starting, with only the root node. After a token for a node is selected, empty children will be added to node based on the token. Binary functions will have two empty children added, unary functions will only have one child, and variables will have no children. These trees are many-to-one mappings to the mathematical expressions, which causes an increase in the search space but prevents the generation of any invalid expression.

The actor is an RNN that predicts a categorical distribution of the available tokens for each node in the expression tree based on the hidden state of the RNN and the sibling and parent of the current node. Each token is randomly sampled from the categorical distribution. Additional rules are applied to the sampling process to prevent the selection of redundant operations or variables. Since the RNN builds the expression trees by in a preorder traversal (POT), there can be a significant delay between the prediction of sibling tokens.

The reward function and policy drive the actor to explore and exploit the complex environment. In DSR, the reward function is a direct measurement of the data fitness of the generated expression,

$$R(\tau) = \frac{1}{1 + \text{NRMSE}}, \tag{1}$$

where τ denotes the expression, NRMSE represents the normalized root-mean-square error, and is defined as NRMSE $= \frac{1}{\sigma_y}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \tau(\mathbf{x}_i))^2}$ where σ_y is the standard deviation of the training output in the dataset.

Once each expression tree has an associated reward, DSR applies a risk-seeking policy to update the actor according to its best predictions. Specifically, at each step, DSR samples a large batch of expressions, ranks their rewards, and selects the top $\alpha\%$ expressions to compute a policy gradient,

$$\nabla J_{\mathrm{risk}}(\theta; \alpha) = \frac{1}{\alpha B}\sum_{i=1}^{B}[R(\tau^{(i)}) - R_\alpha] \cdot \mathbf{1}_{R(\tau^{(i)}) \geq R_\alpha} \nabla_\theta \log(p(\tau^{(i)}|\theta)), \tag{2}$$

where B is the batch size, $\mathbf{1}_{(.)}$ is an indicator function, $\tau^{(i)}$ is the i-th expression in the batch, θ denotes the RNN parameters, $\log(p(\tau^{(i)}|\theta))$ is the probability of $\tau^{(i)}$ being sampled by the current RNN , R_α is the $1 - \alpha/100$ quantile of the rewards in the batch. Accordingly, all the equations below top $\alpha\%$ will not influence the update of the actor. Via such policy, the actor is able to generate worse-performing equations without it affecting the actor's overall performance — we only care about the top performed expressions. This allows for more unrestrained exploration and targeted exploitation of the top performers.

In addition to the policy gradient (2), DSR also introduces an "entropy bonus", to update the RNN actor parameters θ. The entropy bonus is the average entropy gradient for the token distributions associated with the top $\alpha\%$ expression trees. Denote the entropy bonus as $\nabla J_{\mathrm{entropy}}(\theta; \alpha)$, the RNN parameters θ are updated via

$$\theta \leftarrow \theta - l \cdot (\nabla J_{\mathrm{risk}}(\theta; \alpha) + \nabla J_{\mathrm{entropy}}(\theta; \alpha)), \tag{3}$$

where $l > 0$ is the learning rate.

3 Method

We now present CADSR, our new deep symbolic regression approach.

3.1 Transformer Actor

First, we design a transformer [21] as the actor in the reinforcement learning framework. RNNs rely on a single hidden state that summarizes the information across all the previous nodes, which can be insufficient and is known to cause the vanishing gradient issue [4] in larger structures. By contrast, the transformer explicitly integrate information of all the nodes to extract representations and make predictions, and hence can overcome the vanishing gradient issue and more effectively capture the nodes' relationships.

Specifically, to represent the expression tree nodes, we design a positional encoding (PE) to reflect their location information. Given a particular node, we consider both the depth d and horizontal position h in the tree. To align the horizontal positions of the nodes across different layers, we propose the following design. Denote the horizontal position of the parent node by h_p, if the node is the left child, we assign its horizontal position as $h = h_p - h_p/2^d$, otherwise we assign $h = h_p + h_p/2^d$. In this way, the parent node will be in between its two children horizontally, which naturally reflects the tree structure. The horizontal position for the root node is set to $1/2$. See Fig. 1 for an illustration. We develop a recursive algorithm to efficiently calculate the horizontal positions of all the nodes, as listed in Appendix Alogrithm 3. Given d and h, we then create a 2D-dimensional encoding,

$$\mathrm{PE}(d, h, 2i) = \sin(\frac{d}{10000^{(4i/D)}}) \tag{4}$$

$$\mathrm{PE}(d, h, 2i+1) = \cos(\frac{d}{10000^{(4i/D)}}) \tag{5}$$

$$\mathrm{PE}(d, h, 2j+D) = \sin(h) \tag{6}$$

$$\mathrm{PE}(d, h, 2j+1+D) = \cos(h), \tag{7}$$

where $0 \leq i, j \leq \lfloor D/2 \rfloor$. The embedding of each tree node is the positional encoding plus the one-hot encodings of the tokens associated with the parent and sibling nodes (if these nodes exist). By default, every tree node — when created — is assigned an empty token.

To predict a token distribution for each node (according to which we can sample new tokens to generate expressions), we use multiple self-attention layers across all the nodes in the tree, followed by a linear layer, and then apply softmax to the output to produce the token distribution for each node.

Expression Sampling. To generate an expression, we need to dynamically grow the expression tree and sequentially sample the token for each node, since we do not know the tree size and structure beforehand. DSR uses a preorder traversal to sample the token for each node. Since the order indexes of the nodes can vary with tree growth, one needs to re-vectorize the whole tree at every step, and hence it can be very inefficient. To overcome this problem, we use the breadth-first-search (BFS) order. Since the tree is expanding layer by layer, the ordering of the previous nodes will never change, and therefore we only need to perform vecotrization once before performing any sampling. This is not only efficient but straightforward to implement. Given the token distribution, we first fill out invalid tokens and then sample from the remaining. Whenever the tokens have formed an valid expression, we stop the sampling and return the expression. The expression tree sampling is summarized in Algorithm 2 of the Appendix.
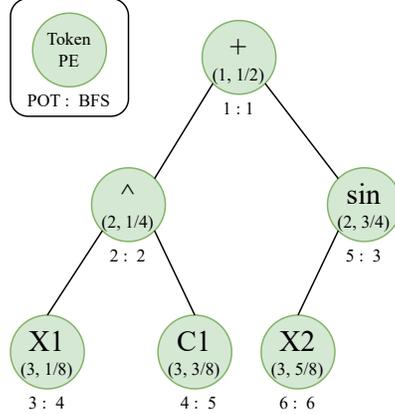

Figure 1: Expression tree for $y = x_1^{c_1} + \sin(x_2)$. POT and BFS denote the node order for preorder traversal and breadth-first-search, respectively.

3.2 BIC Reward Function

Interpretability is a key motivation for symbolic regression. However, if one uses data fitness as the reward to guide the learning, like in DSR, it will tend to learn lengthy, complex expressions to overfit the data, especially when data contains noise, which is common in practical applications. This can hurt the interpretability of the learned expressions. To address this problem, we use Bayesian information criterion (BIC) [14] to construct a new reward function. BIC introduces the number of parameters k to construct a penalty term to penalize over-complex models. Combined with the model likelihood, BIC enables a trade-off between the model complexity and data fitness, which is justified by Bayesian theorem and Laplace's method [3].

To reflect the complexity of a sampled expression, we set k to the number of nodes in the expression tree plus the number of constant tokens. The inclusion of the number of constant tokens is twofold: first, since the values of the constant tokens are unknown apriori — we need to estimate their values from data, they actually sever as unknown model parameters (like those in neural networks) and introduce extra degrees of freedoms, which increase the model complexity. Second, this can also prevent the actor from selecting an unnecessary number of constant tokens, which can cause a significant increase in runtime for optimizing their values. Our BIC reward is given below,

$$\text{BIC}(\tau) = k \log(N) - 2 \log(L(\tau)) \tag{8}$$

where $\log(L(\tau)) = \sum_{i=1}^{N} \log \mathcal{N}(y_i | \tau(\mathbf{x}_i), \sigma^2)$ is a (log) Gaussian likelihood, where σ^2 is the variance of the training outputs.

3.3 Robust Risk-Seeking Policy

While our BIC reward can account for expression complexity, applying it to the risk-seeking policy presents several challenges. The first challenge is that, because BIC is unbounded, directly incorporating it into equation (2) compromises unbiasedness.

Lemma 3.1. *When using the BIC reward* (8), *the risk-seeking policy gradient* (2) *is not guaranteed to be unbiased.*

We leave the analysis in Appendix Section 10.1. To overcome this problem, a commonly used strategy, which is also adopted in DSR, is to introduce a continuous mapping that maps the reward value to a bounded domain. For example, DSR uses the mapping $f(z) = \frac{1}{1+z}$ where $z \in [0, \infty]$,

4

Algorithm 1 Complexity-Aware Deep Symbolic Regression (CADSR)

---

input Learning rate l; risk factor α; batch size B; coefficients $\lambda > 0$ and $\lambda_{\mathcal{H}} > 0$
output The best equation τ^*

1: Initialize transformer with parameters θ
2: **while** $i < epochs$ **do**
3:    $\mathcal{T} \leftarrow \{\tau^{(i)} \sim p(\cdot|\theta)\}_{i=1}^B$ {Sample B expressions using the transformer}
4:    $\mathcal{T} \leftarrow \{\text{OptimizeConstants}(\tau^{(i)})\}_{i=1}^N$ {Optimize the values of the constant tokens}
5:    $\mathcal{R} \leftarrow \{\text{BIC}(\tau^{(i)})\}_{i=1}^N$ {Calculate the reward for each expression using (8)}
6:    $\mathcal{R}_\alpha \leftarrow (1 - \alpha/100)$-quantile of $\mathcal{R}$ {Find the reward that denotes the top α%}
7:    $\mathcal{T} \leftarrow \{\tau^{(i)} : \mathcal{R}(\tau^{(i)} \geq \mathcal{R}_\alpha\}$ {Reduce the sampled expressions to the top performers}
8:    $g_1 \leftarrow \frac{\lambda}{\alpha B} \sum_{i=1}^{\alpha B} \nabla_\theta \log p(\mathcal{T}|\theta)$ {Compute for top α% gradient}
9:    $g_2 \leftarrow \frac{\lambda_{\mathcal{H}}}{\alpha B} \sum_{i=1}^{\alpha B} \nabla_\theta \mathcal{H}(\mathcal{T}|\theta)$ {Compute entropy gradients}
10:    $\theta \leftarrow \theta + l(g_1 + g_2)$
11:    **if** $\max \mathcal{R} > \mathcal{R}(\tau^*)$ **then** $\tau^* \leftarrow \tau^{(\arg\max \mathcal{R})}$ {Update best equation}
12: **end while**
13: **return** τ^*

---

which maps the unbounded NRMSE (in $[0, \infty]$) to the range $(0, 1]$; see (1). Another choice can be the sigmoid function, $s(z) = \frac{1}{1+e^{-z}}$ that maps from $(-\infty, \infty)$ to $(0, 1)$. In doing so, we can apply Leibniz rule to interchange integration and differentiation to show the unbiasedness of the policy gradient. However, this strategy brings up the second challenge. That is, the reinforcement learning can encounter a *tail barrier*, defined as follows.

Definition 3.2 (Tail barrier). Let $\alpha \in [0, 1)$. A risk seeking policy meets a α-tail barrier if the top α% rewards of the sampled actions (*e.g.*, expression trees) are all equal to R_α.

Lemma 3.3. *Given any continuous mapping f that can map unbounded reward function values to a bounded domain (e.g., $(-\infty, \infty) \to [0, 1]$), suppose the reward function is continuous, there always exists a set of distinct rewards values that numerically create a tail barrier in the risk-seeking policy.*

The proof is given in Appendix Section 10.2. In practice, since we often use continuous reward functions (*e.g.*, NRMSE or Gaussian likelihood) and reward mappings, there is a risk of encountering the tail barrier. From (2), we can see that the tail barrier can incur *zero* policy gradient, since all the top α% rewards are identical to R_α, leading to a zero weight for every gradient. As a consequence, the actor model would not have any effective updates according to the feedback from the selected expressions (top performers). In DSR, the RNN model will be updated only from the entropy bonus (see (3)), and henceforth the learning starts to explore widely.

To address these challenges, we use a step function to perform a reward mapping,

$$f(z) = \begin{cases} \lambda, & \text{if } z > R_\alpha \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

where $\lambda > 0$ is a constant. Then the risk-seeking policy gradient is given by

$$\nabla J_{\text{risk}}(\theta; \alpha) \approx \frac{\lambda}{\alpha B} \sum_{i=1}^B \mathbf{1}_{\text{BIC}(\tau^{(i)}) > R_\alpha(\theta)} \nabla_\theta \log(p(\tau^{(i)}|\theta)). \tag{10}$$

Lemma 3.4. *By using the step function (9) for reward mapping, the policy gradient with our BIC reward, as shown in (10), is unbiased and will not encounter any tail barrier.*

The proof is given in Appendix Section 10.3. Our approach is summarized in Algorithm 1.

4 Related Work

DSR is a reinforcement learning method that uses an RNN actor [10]. The risk-seeking policy rewards the actor only for top-performing expressions, which allows the actor to make a diverse range of predictions without being punished for poor average performance.

Many symbolic regression models have started to use transformers, such as [7, 19, 20]. However, all of these models learn the mapping directly from the dataset to the expression trees in a supervised fashion. To achieve this end-to-end learning, a much larger model architecture is necessary, along with a substantial collection of datasets. By contrast, DSR and our method essentially perform unsupervised learning, and the dataset only corresponds to one (unknown) expression. Hence, the model size and data quantity is much smaller. Recently, Shojaee et al. [15] developed Monte Carlo tree search with transformers (TPSR), to improve the prediction of the best expressions. TPSR assumes a pre-trained transformer is given, and conducts a more efficient search procedure to identify well-performed expressions. On many benchmark tasks, this method has shown the best R^2 performance. However, TPSR typically finds very complex expressions, which reduces the interpretability and hence has the same limitations of other methods.

Lastly, the recent work uDSR [1] combines several different methods together to improve DSR. One of the additions is a transformer trained using supervised learning or reinforcement learning to provide the RNN actor with additional information about the dataset. uDSR also incorporates genetic programming to each expression tree generation step, which allows for a larger variety of expression trees to be generated each epoch. Validating the performance of the different parts of uDSR required several ablation studies, where every combination of methods, including DSR, was tested in the uDSR paper [1]. The addition of a transformer to DSR showed minimal improvement and occasionally hindrance to uDSR during the ablation study. Furthermore, genetic programming was the single biggest influence on performance, showing that there needs to be an improvement to DSR. uDSR appeared on the Pareto frontier near the optimal mixture of accuracy and complexity.

5 Numerical Experiments

For evaluation, we examined CADSR in the well known comprehensive SRBench dataset [9], and then we studied each component of CADSR to confirm their efficacy.

5.1 Overall Perofrmance

In SRBench, we first tested on the 133 problems with known solutions. We ran eight trials for each problem at the four noise levels, 0%, 0.1%, 1%, and 10%. We ran CADSR in a large computer cluster, for which we set a time limit of 6 hours for each trial. We deployed the trials on 10 V100 GPUs with parallelization of 6 processes per GPU. The maximum number of epochs is set to 2000 without early termination. The architecture of the transformer is shown in Appendix Fig. 7. In each epoch, we sampled a batch of 1000 expressions to compute the policy gradient. We used ADAM optimization where the learning rate was set to 5E-4. The full list of hyperparameters of our method is provided in Appendix Table 2. We compared with 15 popular and/or state-of-the-art SR methods in terms of Symbolic Solution Rate (%), R^2 test, and Simplified Complexity, which are standard metrics for SR evaluation. These SR baselines include AIFeynman [18], AFP_FE [13], DSR [10], Bingo [11], etc. The majority of these are genetic programming methods with a few notable exceptions: DSR is deep reinforcement learning, BSR [5] is an MCMC method with a prior placed on the tree structure, and AIFeynmen is a divide-and-conquer method that breaks the problem apart by hyper-planes and fits with polynomials. The results of the competing methods are retrieved from the public SRBench report [9]. We show the results of all the methods in Fig. 2.

Overall, CADSR shows strong performance in symbolic discovery as measured by Symbolic Solute Rate. In particular, when data includes significant noise (1% and 10%), CADSR achieves the best solution rate, showing that our method is more robust to noise than all the competing methods. Meanwhile, the simplified complexity of our discovered expressions is among the lowest. This together shows that our method, with the BIC reward design, not only can find simpler and hence more interpretable expressions, but also is more resistant to data noise. The R^2 Test shows the prediction accuracy of the discovered expressions. As we can see, the R^2 Test obtained by CADSR is close to the best. The slightly better methods, such as AFP_FE and GP-GOMEA, however, generate lengthier and more complex expressions, which lack interpretability and are much far away from the ground truth expression. It is worth noting that CADSR outperforms DSR in both Symbolic Solution Rate (%) and R^2 Test, showing an improvement on both expression discovery and prediction accuracy. When data does not have noise, AIFeynman and Bingo shows better Symbolic Solution Rate than CADSR. This might be because AIFeynman tends to use polynomials to construct the
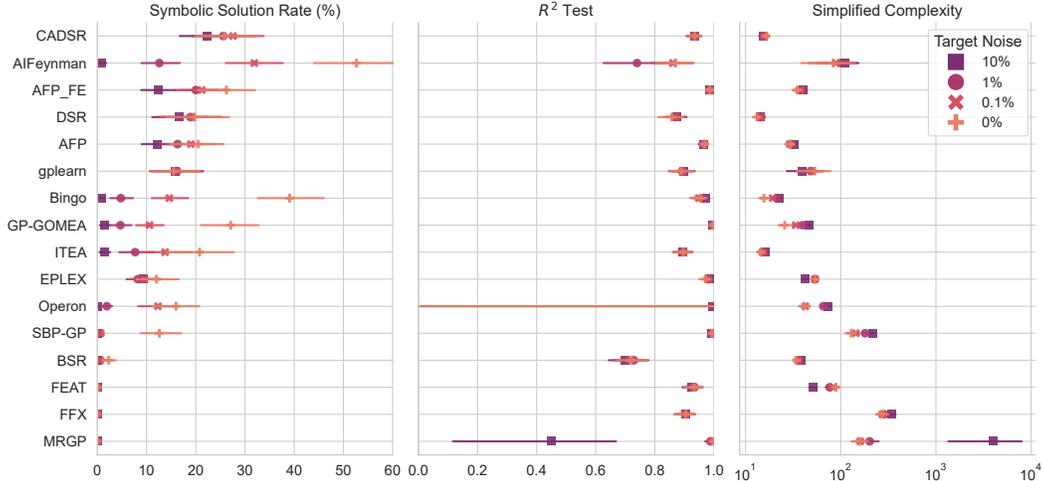
Figure 2: Symbolic regression performance on 133 problems from SRBench with known solutions.

expressions, which match most of the ground-truth; Bingo as a genetic programming approach, uses evolution operators to sample new expressions, which might explore more broadly.

Next, we tested with the 120 black-box problems in SRBench. Since the true solutions for these problems are unknown, we examined the Pareto front of all the methods in Model Size Rank *vs.* R^2 Test Rank. We also retrieved the results of several commonly used machine learning algorithms from the SRBench report [9] for comparison. These algorithms include Random Forest, AdaBoost, and Multi Layer Peceptron (MLP). The setting of these methods are given in [9]. As shown in Fig. 3, CADSR is at the most frontier, meaning CADSR is among the best in terms of the trade-off between the model size (expression complexity) and R^2 test (prediction accuracy). It is interesting to see that among those most frontier methods, CADSR tends to find the most concise expressions (ranked as the lowest in model size) while sacrificing the prediction accuracy to a certain degree. On the contrary, methods like Operon, typically generates way more complex expressions yet with smaller prediction error. It shows that our method can push the best trade-off toward more interpretability, which can be important in practical applications.
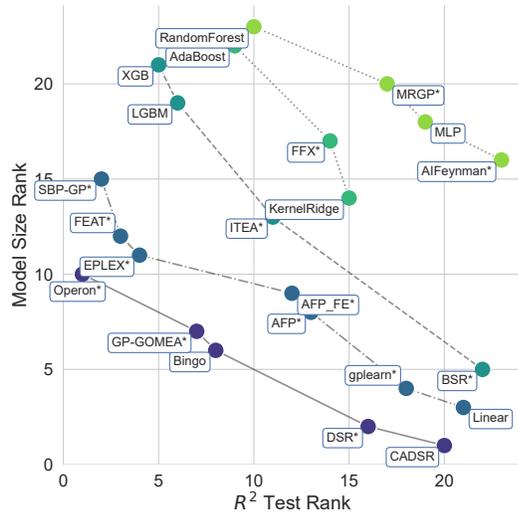

Figure 3: Pareto front of each method in 120 black-box problems of SRBench; the true solutions are unknown.

5.2 Ablation Study

Next, we performed ablation studies to confirm the efficacy of each component of our method.

Transformer actor. We first looked into the transformer actor of our method. We replaced the RNN actor in DSR with our transformer actor for expression sampling. The training follows the same way as in DSR, namely, the risk-seeking policy gradient plus entropy bonus (see (2) and (3)). We used the *Strogatz_lv2* problem in SRBench for comparison. We first ran the original DSR for eight trials, and at best DSR achieved 12.5% Symbolic Solution Rate (*i.e.,* discovery rate) and 0.47 R^2 Test score. After replacing the RNN actor with our transformer actor, DSR then achieved 100% discover rate and 1.0 R^2 Test score across all five trials, showing a substantial performance lift. We list the discovered

7

Trial	Full Expression	R^2	Symbolically Accurate
1	$(((((x_1 + c_0) - (c_1 c_2)) - ((c_3 - c_4)/(c_5 + c_6)))$ $/(1((\cos(c_7) - 1))) - x_0)(x_0/(x_0 * c_8)))x_1$	1.0	Yes
2	$(((1((c_0((\sqrt{c_1}) - ((x_1 c_2) - c_3)))$ $+(((e^{c_4})x_0)1)))c_5)(c_6(x_1 c_7)))$	1.0	Yes
3	$((c_0 x_1)(x_1 + (x_0 - (\log(c_1)))))$	1.0	Yes
4	$c_0 x_1 - x_0 x_1 - x_1^2$	1.0	Yes
5	$(x_1 x_0) + \log(((1 - c_0)(((c_1$ $+c_2)^{c_3 + ((x_0 - (c_4(c_5 - x_1)))((c_6 + c_7) + x_1))})c_8)))$	1.0	Yes

Table 1: Running DSR on *Strogatz_lv2* problem with the transformer actor. The true solution is $y = 2x_1 - x_0 x_1 - x_1^2$

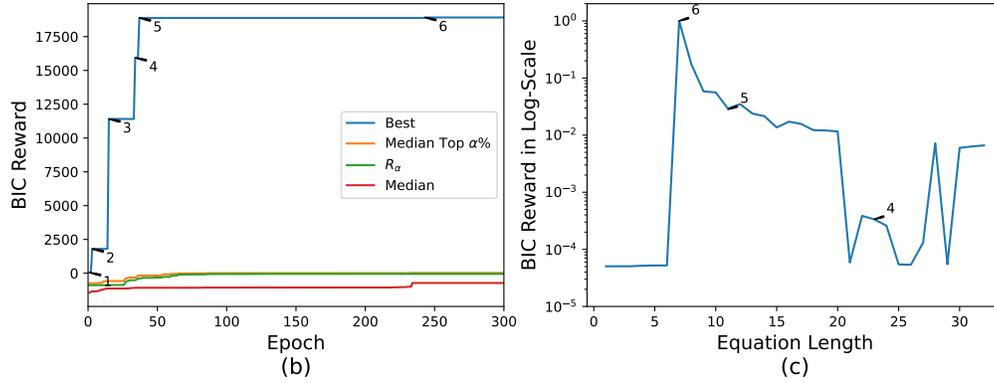| Label	Epoch	R^2	Expression
1	0	0.9771	$((x_1 * ((x_1^{c_0})/(\sqrt{(c_1)}))) * c_2)) * \ldots$
2	2	0.9999	$(c_0 - (\sqrt{((c_1 - (c_2 * (((((c_3 * x_0) \ldots$
3	14	1.0	$(\log(((c_0^{((((((x_1/x_0)/(x_1/c_1))^{c_2}} \ldots$
4	33	1.0	$(((x_1 * (c_0^{c_1})) * ((\cos(c_2)) * ((((\sqrt{(c_3)}) + 1) - \ldots$
5	36	1.0	$((x_1 * ((x_0 + (x_1 - c_0))/c_1)) * c_2)$
6	242	1.0	$c_0 x_1 - x_0 x_1 - x_1^2$

(a)

(b)

(c)

Figure 4: Learning progress with BIC reward function on *Strogatz_lv2* problem where the true solution is $y = 2x_1 - x_0 x_1 - x_1^2$

expressions after each trial in Table 1. Although these expressions seem lengthy at the first sight, after automatic simplification (*e.g.*, via SymPy), they can all reduce to the true solution.

BIC reward. On the other hand, the lengthy expressions found in Table 1 indicate that the DSR reward (1), which is purely based on data fitness, does not seek to reduce model complexity to enhance the inpretability. We further replace the DSR reward by our BIC reward, and re-run the learning process. While across five runs, the discovery rate and R^2 test scores remain 100% and 1.0 respectively, the discovered expressions are much simpler and closer to the true solution; see Table 3 of the Appendix. In Fig. 4, we further showcased the learning progress of one trial with our BIC reward function. It can be seen clearly from Fig. 4a, that along with more epochs, the learned expressions are increasingly concise, and finally exactly recover the true solution. Meanwhile, from Fig. 4b, we can see that those increasingly simpler expressions receive increasingly bigger rewards while Fig. 4c further confirms that shorter expressions do receive larger rewards. All these demonstrate our BIC reward indeed guides the learning toward more concise and hence more intepretable expressions in addition to fitting the data well.

Robust risk-seeking policy. Finally, we examined the learning behavior with our robust risk-seeking policy. To this end, we tested on the *Strogatz predprey1* problem in SRBench. We ran our method for one trial and then replaced our policy with the risk-seeking policy as used in DSR (see (2)) to

run another trial. The random seeds for the two runs were set to be the same. In Fig. 5a and 6a, we report how the best reward (among the sampled expressions at each step), the median reward, R_α, and the median of the top $\alpha\%$ rewards (denoted as Median Top $\alpha\%$), varied along with the training epochs. Obviously, when the median of the top $\alpha\%$ rewards are identical to R_α, many weights (around 50%) in the risk-seeking policy (2) become zero (since $R(\tau^{(i)}) = R_\alpha$), resulting zero gradients for the corresponding expressions. This can be viewed a "partial" tail barrier. To examine how many top $\alpha\%$ expressions are pruned due to their gradients being roughly zeroed out, we used a threshold 10^{-3} and report the portion of such expressions at each epoch in Fig. 5b and 6b. From Fig. 5a, we can see during the running of original risk-seeking policy, we encountered three significant events at epochs 250, 975, and 1750. In these events, the median of the top $\alpha\%$ rewards almost overlap with R_α, causing a large portion of the top $\alpha\%$ expressions pruned due to that their gradients were nearly zeroed out. Accordingly, the learning afterwards can be more dominated by entropy bonus (see (3)) rather than those best-performed expressions. By contrast, while during the course of our method, there are also quite a few events when the median of the top $\alpha\%$ almost overlap with R_α, no expression in the top $\alpha\%$ have their gradients zeroed out, since every such expression is assigned a constant reward λ; see (9) and (10). Therefore, the update of the model parameters will always leverage the information from all top $\alpha\%$ expressions, and hence can effectively avoid over-exploration.
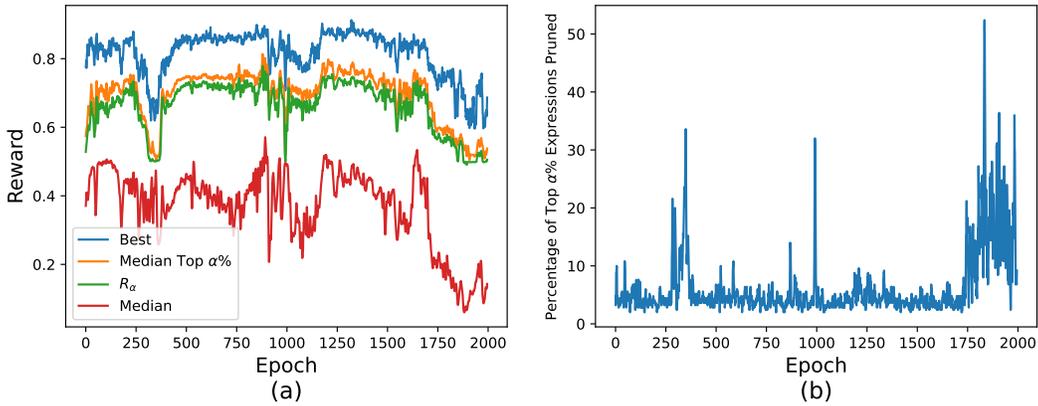

Figure 5: Running CADSR with risk-seeking policy (2) on *Strogatz predprey1* problem. The top-performed expressions can be pruned from being used for model updating, due to nearly zeroed out gradients.
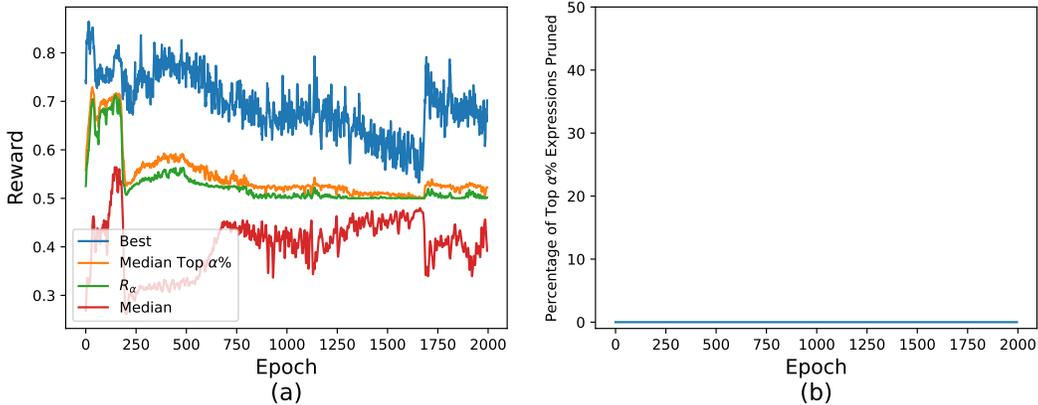

Figure 6: Running CADSR with robust risk-seeking policy (10) on *Strogatz predprey1* problem.

6 Conclusion

We have presented CADSR, a new symbolic regression approach based on reinforcement learning. On standard SR benchmark problems, CADSR shows promising performance. The ablation study

9

confirms the effectiveness of each component of our method. Nonetheless, our current work has two limitations. First, the implementation is inefficient, especially for the expression sampling coupled with validation rules. This leads to a slow training process. Second, we lack an early stopping mechanism to further reduce the training cost and prevent useless exploration. In the future, we plan to address these limitations and use our method in more practical applications.

References

[1] Bomarito, G., Leser, P., Strauss, N., Garbrecht, K., and Hochhalter, J. (2023). Automated learning of interpretable models with quantified uncertainty. Computer Methods in Applied Mechanics and Engineering, 403:115732.

[2] Burlacu, B., Kronberger, G., and Kommenda, M. (2020). Operon C++: an efficient genetic programming framework for symbolic regression. In Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, GECCO '20, pages 1562–1570, New York, NY, USA. Association for Computing Machinery.

[3] Ghosh, J. K., Delampady, M., and Samanta, T. (2006). An introduction to Bayesian analysis: theory and methods, volume 725. Springer.

[4] Hochreiter, S. (1998). The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 06(02):107–116. Publisher: World Scientific Publishing Co.

[5] Jin, Y., Fu, W., Kang, J., Guo, J., and Guo, J. (2020). Bayesian symbolic regression.

[6] Jobin, A., Ienca, M., and Vayena, E. (2019). The global landscape of AI ethics guidelines. Nature Machine Intelligence, 1(9):389–399. Publisher: Nature Publishing Group.

[7] Kamienny, P.-A., d'Ascoli, S., Lample, G., and Charton, F. (2022). End-to-end Symbolic Regression with Transformers.

[8] Koza, J. R. (1994). Genetic programming as a means for programming computers by natural selection. Statistics and Computing, 4(2):87–112.

[9] La Cava, W., Orzechowski, P., Burlacu, B., de Franca, F. O., Virgolin, M., Jin, Y., Kommenda, M., and Moore, J. H. (2021). Contemporary Symbolic Regression Methods and their Relative Performance. arXiv:2107.14351 [cs].

[10] 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.

[11] Randall, D. L., Townsend, T. S., Hochhalter, J. D., and Bomarito, G. F. (2022). Bingo: a customizable framework for symbolic regression with genetic programming. In Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '22, pages 2282–2288, New York, NY, USA. Association for Computing Machinery.

[12] Rudin, C. (2019). Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead. arXiv:1811.10154 [cs, stat].

[13] Schmidt, M. and Lipson, H. (2009). Distilling Free-Form Natural Laws from Experimental Data. Science, 324(5923):81–85. Publisher: American Association for the Advancement of Science.

[14] Schwarz, G. (1978). Estimating the dimension of a model. The annals of statistics, pages 461–464.

[15] Shojaee, P., Meidani, K., Farimani, A. B., and Reddy, C. K. (2023). Transformer-based Planning for Symbolic Regression.

[16] Sutton, R. S. and Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

[17] Tamar, A., Glassner, Y., and Mannor, S. (2014). Policy Gradients Beyond Expectations: Conditional Value-at-Risk.

[18] Udrescu, S.-M. and Tegmark, M. (2020). AI Feynman: a Physics-Inspired Method for Symbolic Regression. arXiv:1905.11481 [hep-th, physics:physics].

[19] Valipour, M., You, B., Panju, M., and Ghodsi, A. (2021). SymbolicGPT: A Generative Transformer Model for Symbolic Regression. arXiv:2106.14131 [cs].

[20] Vastl, M., Kulhánek, J., Kubalík, J., Derner, E., and Babuvska, R. (2022). SymFormer: End-to-end symbolic regression using transformer-based architecture. arXiv:2205.15764 [cs].

[21] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2023). Attention Is All You Need. arXiv:1706.03762 [cs].

7 Algorithms

Below, we show the algorithms for expression sampling and positional encoding generation. Note that for expression sampling, we over-sample expressions so that we can return a high number of unique ones. If not enough unique expressions exist, then we begin to allow duplicates to fill out our batch size requirements.

Algorithm 2 Expression Tree Sampling

---

input Number of expressions to sample B; oversampling scalar $\gamma > 1$, maximum tree-node number ν
output A set of expressions $\mathcal{T}$

1: $\mathcal{T} \leftarrow$ ExpressionTrees(γB) {Creates γB empty expression trees}
2: **while** $i < \nu$ **do**
3:     $V_\mathcal{T} \leftarrow$ Inputs($\mathcal{T}$) {Fetching the input embeddings of all the expression trees}
4:     $S \leftarrow p(V_\mathcal{T}|\theta)$ {Predicting categorical distributions from the transformer}
5:     $S \leftarrow R(S)$ {Applying rules to each distribution}
6:     $K \leftarrow P(\cdot|S)$ {Sampling from the categorical distribution to obtain tokens}
7:     $\mathcal{T}_i \leftarrow K$ {Adding the new tokens into the expression trees}
8: **end while**
9: $\mathcal{T} =$ Unique($\mathcal{T}, B$) {Take the first B Unique expression trees}
10: **return** $\mathcal{T}$

Algorithm 3 Positional Encoding Generation

---

input An expression tree τ

1: τ.root_node.depth = 1
2: τ.root_node.horizontal = 1/2
3: PositionEncodingInformation(τ.root_node)

Algorithm 4 PositionEncodingInformation

---

input Current node

1: **if** node has left **then**
2:     node.left.depth = node.depth + 1
3:     node.left.horizontal = node.horizontal - $1/(2^{\text{node.left.depth}})$
4:     PositionEncodingInformation(node.left)
5: **end if**
6: **if** node has right **then**
7:     node.right.depth = node.depth + 1
8:     node.right.horizontal = node.horizontal + $1/(2^{\text{node.right.depth}})$
9:     PositionEncodingInformation(node.right)
10: **end if**

---

8 Model Details

Table 2 and Fig. 7 show the comprehensive hyperparameter settings and the architecture of the transformer used in our method. Note that we use the Levenberg–Marquardt algorithm to optimize constant tokens for each discovered equation.

9 Additional Results

Table 3 shows the discovered expressions across different trials for *Strogatz_lv2* problem in SRBench.

10 Theoretical Results

10.1 Proof of Lemma 3.1

Proof. The value of the BIC reward function is essentially a random variable, since it is determined by the random expression sampled by the transformer actor. Let us denote this random variable
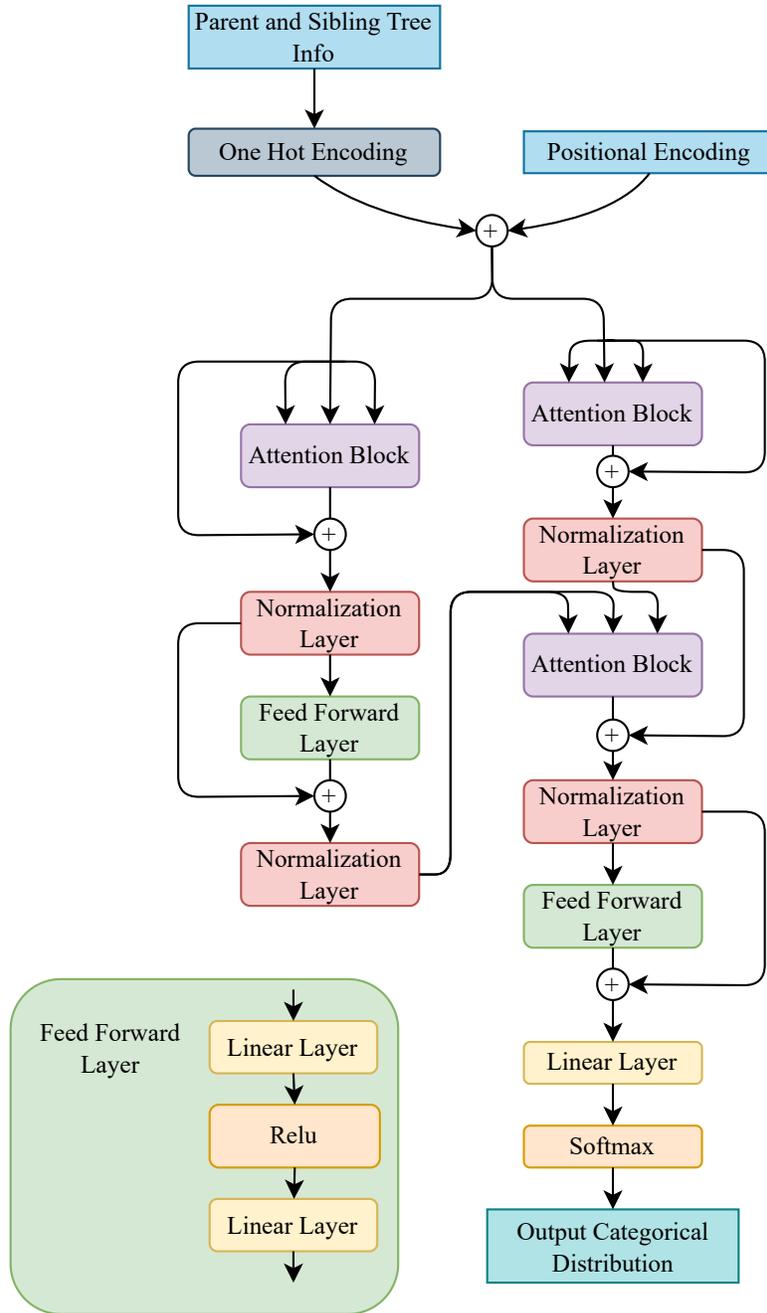
Figure 7: The architecture of the transformer actor in CADSR.

Hyperparameter	Setting
Variables	$\{1, c \text{ (Constant Token)}, x_i\}$
Unary Functions	$\{\sin, \cos, \log, \sqrt{(\cdot)}, \exp\}$
Binary Functions	$\{+, -, *, /, \hat{}\}$
Batch Size	1000
Risk Seeking Percent (α)	0.05
Learning Rate	5E-4
Max Depth	32
Oversampling	1.5 (Ideally 3)
Number of Epochs	2000
Policy	BIC
Entropy Coefficient $\lambda_{\mathcal{H}}$	0.005
Policy Coefficient λ	0.04

Table 2: Hyperparameter setting of CADSR.

Trial	Full Expression	R^2	Symbolically Accurate
1	$((2.0 - x_0) - x_1)x_1$	1.0	Yes
2	$(2.0 - (x_0 + x_1))x_1$	1.0	Yes
3	$(x_1 * (3.0 - (((x_0/x_0) + x_0) + x_1)))$	1.0	Yes
4	$((1 - ((x_1 + x_0) + -1)) * x_1)$	1.0	Yes
6	$(((\log(7.389)) - x_0) - x_1)x_1$	1.0	Yes

Table 3: Trials of using BIC reward on *Strogatz_lv2* problem, where the true solution is $y = 2x_1 - x_0 x_1 - x_1^2$.

by Z and its probability density by $p(z|\theta)$ where θ denotes the transformer parameters. Obviously, $Z \in (-\infty, \infty)$. The risk-seeking policy aims to approximate the gradient of the expectation over the truncated random variable $Z_\alpha = Z \cdot \mathbf{1}_{Z \geq s_\alpha}$, where s_α is the $1 - \frac{\alpha}{100}$ quantile of the distribution of Z,

$$s_\alpha = \inf\{z : \text{CDF}(z) \geq 1 - \frac{\alpha}{100}\}. \tag{11}$$

The probability density of Z_α is given by

$$p(z_\alpha|\theta) = \frac{1}{\alpha} p(z|\theta) \mathbf{1}_{z \geq s_\alpha}. \tag{12}$$

The gradient of the expectation is computed as

$$\mathbb{E}[Z_\alpha] = \frac{1}{\alpha} \int_{s_\alpha}^{\infty} z p(z|\theta) \mathrm{d}z, \tag{13}$$

$$\nabla_\theta \mathbb{E}[Z_\alpha] = \frac{1}{\alpha} \cdot \nabla_\theta \int_{s_\alpha}^{\infty} z p(z|\theta) \mathrm{d}z. \tag{14}$$

Since the integration upper-bound is ∞, we cannot apply Leibnize rule to equivalently interchange the gradient and the integration. In other words, if we switch the order of the expectation (integration) and the differentiation, the result can be changed. Therefore, the policy gradient in the form of (2) is no longer guaranteed to be unbiased.

$\square$

10.2 Proof of Lemma 3.3

Proof. Pick up any reward value z_0. Due to the continuity of the mapping f, for arbitrary $\epsilon > 0$, there exists $\delta > 0$ such that for all $z \neq z_0$, if $|z - z_0| < \delta$, then $|f(z) - f(z_0)| < \epsilon$. Let us take $\epsilon = \frac{1}{2}s$, where s is the machine precision (*e.g.*, 2^{-32}). Since the reward function is continuous, we can find a set of distinct reward values $z_1, \dots, z_M$ from $B(z_0, \delta(\epsilon)) = \{z \in \text{dom } f, |z - z_0| < \delta(\epsilon)\}$. Let us look at the mapped rewards, $f(z_1), \dots, f(z_M)$. For any $1 \leq i, j \leq M$, we have

$$|f(z_i) - f(z_j)| = |f(z_i) - f(z_0) + f(z_0) - f(z_j)| \leq |f(z_i) - f(z_0)| + |f(z_0) - f(z_j)| < \frac{s}{2} + \frac{s}{2} = s.$$

Therefore, there are no numerical difference among these mapped rewards, and they can create a tail barrier.

$\square$

10.3 Proof of Lemma 3.4

Proof. For any set of BIC reward values, $\mathcal{S} = \{R(\tau^{(1)}), \ldots, R(\tau^{(B)})\}$, we denote the mapped reward values by $\widehat{\mathcal{S}} = \{\widehat{R}_1, \ldots, \widehat{R}_B\}$, where each $\widehat{R}_j = f(R(\tau^{(j)}))$ $(1 \le j \le B)$. We know that each

$$\widehat{R}_j = \begin{cases} 1 & R(\tau^{(j)}) > R_\alpha \\ 0 & R(\tau^{(j)}) \le R_\alpha \end{cases} \tag{15}$$

where R_α is the $1 - \frac{\alpha}{100}$ quantile of the rewards in $\mathcal{S}$. Therefore, the $1 - \frac{\alpha}{100}$ quantile of the mapped reward values $\widehat{\mathcal{S}}$, namely, $\widehat{R}_\alpha = 0$. Since all the top $\alpha\%$ rewards in $\widehat{\mathcal{S}}$ take the value λ, which is strictly bigger than $\widehat{R}_\alpha = 0$, the tail barrier in the risk-seeking policy will never appear. Furthermore, every expression in the top $\alpha\%$ will not have their gradient zeroed out (see (2)), since the weight of the gradient is the constant λ (see (10)). That means, we even will not meet a partial tail barrier.

To show the unbiasedness, we need to replace R_α by the $1 - \alpha/100$ quantile of the distribution of the BIC reward R. Denote the mapped reward by $\widehat{R}$. Since $\widehat{R}$ is bounded, namely, $\widehat{R} \in [0, 1]$, we can follow exactly the same steps in the proof of the original risk-adverse gradient paper [17] to show the unbiasedness of (10). Note that, with the step mapping f, the expectation, $\mathbb{E}[\widehat{R}\mathbf{1}_{\widehat{R} > Q_\alpha}]$ where Q_α is the $1 - \alpha/100$ quantile of the distribution of $\widehat{R}$, is actually a constant, namely $\lambda\alpha$. Hence the gradient of the expectation is zero. That means, our new risk-seeking policy gradient (10) is asymptotically converging to zero (with $B \to \infty$). However, in practice, B is always finite, and our policy gradient is rarely close to zero. Rather, it will leverage all the top performers to conduct efficient model updates.

$\square$