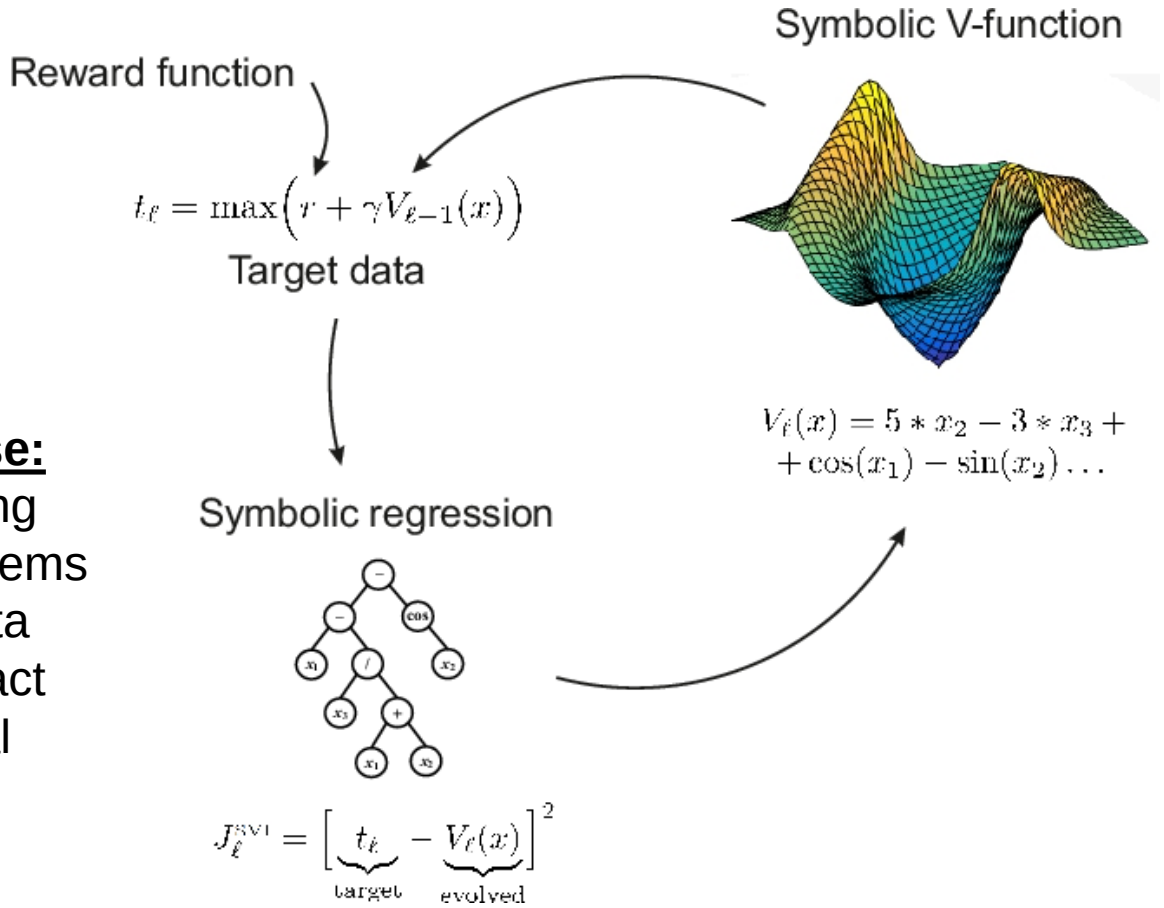
Deep Symbolic Regression (DSR) and Related Frameworks

Symbolic Regression

Background

Problem: Symbolic Regression

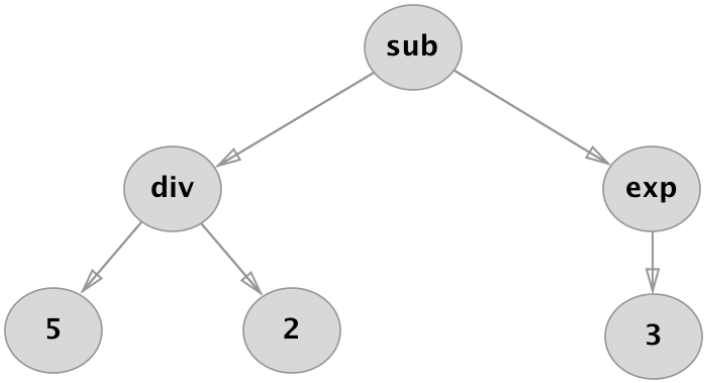


Key use case:
Understanding physical systems based on data with a compact mathematical expression.

Image from Kubalik et al., 2019

Key Concept in Symbolic Regression: Expression Tree

$$\frac{5}{2} - e^3$$



An expression tree is a syntax tree for mathematical expressions.

Unlike LNN's where the syntax tree is given and embedded into the NN, in DSO/DSR we learn the tree using an RNN in a RL framework.

Image Source: https://jankrepl.github.io/symbolic-regression/

Notes on Symbolic Regression

- Conjectured to be **NP hard**
- **Genetic programming** (GP) was the most popular approach
- GP has **scalability issues**

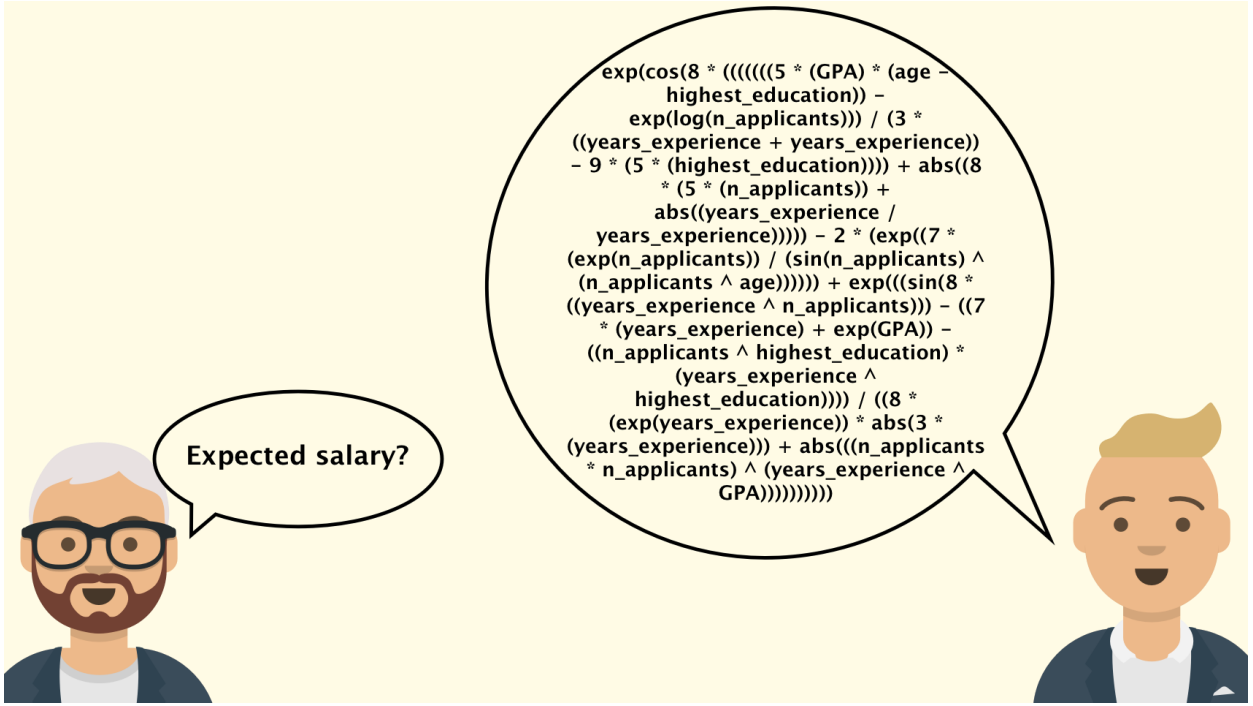


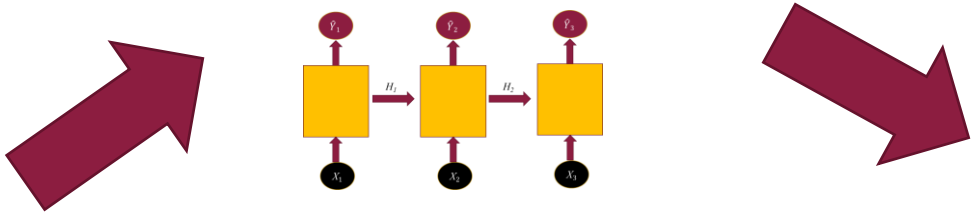
Image Source: https://jankrepl.github.io/symbolic-regression/

Deep Symbolic Regression

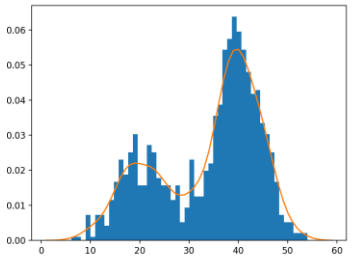
Overview

Overview:
Deep Symbolic Regression

Update RNN parameters



Generate distribution of expressions with an RNN



Compute gradient using top epsilon expressions via "risk-seeking" reward function

$$\nabla_\theta J_{\text{risk}}(\theta;\varepsilon) \approx \frac{1}{\varepsilon N}\sum_{i=1}^{N}\left[R(\tau^{(i)}) - \tilde{R}_\varepsilon(\theta)\right]\cdot \mathbf{1}_{R(\tau^{(i)})\geq \tilde{R}_\varepsilon(\theta)}\nabla_\theta \log p(\tau^{(i)}|\theta)$$

Evaluate reward associated with expressions based on NRMSE to identify top epsilon expressions

$$\frac{1}{\sigma_y}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - f(X_i))^2}$$

NMSE / Reward function from Petersen et al., ICLR 2021
PDF image from https://machinelearningmastery.com/

Overall Pseudocode

Algorithm 1 Deep symbolic regression with risk-seeking policy gradient

input learning rate α; entropy coefficient $\lambda_{\mathcal{H}}$; risk factor ε; batch size N; reward function R
output Best fitting expression $\tau^\star$

1: Initialize RNN with parameters θ, defining distribution over expressions $p(\cdot|\theta)$
2: **Repeat**
3: $\quad \mathcal{T} \leftarrow \{\tau^{(i)} \sim p(\cdot|\theta)\}_{i=1}^{N}$ ▷ Sample N expressions (Alg. 2 in Appendix A)
4: $\quad \mathcal{T} \leftarrow \{\text{OptimizeConstants}(\tau^{(i)}, R)\}_{i=1}^{N}$ ▷ Optimize constants w.r.t. reward function
5: $\quad \mathcal{R} \leftarrow \{R(\tau^{(i)})\}_{i=1}^{N}$ ▷ Compute rewards
6: $\quad R_\varepsilon \leftarrow (1-\varepsilon)$-quantile of $\mathcal{R}$ ▷ Compute reward threshold
7: $\quad \mathcal{T} \leftarrow \{\tau^{(i)} : R(\tau^{(i)}) \geq R_\varepsilon\}$ ▷ Select subset of expressions above threshold
8: $\quad \mathcal{R} \leftarrow \{R(\tau^{(i)}) : R(\tau^{(i)}) \geq R_\varepsilon\}$ ▷ Select corresponding subset of rewards
9: $\quad \hat{g_1} \leftarrow \text{ReduceMean}((\mathcal{R} - R_\varepsilon)\nabla_\theta \log p(\mathcal{T}|\theta))$ ▷ Compute risk-seeking policy gradient
10: $\quad \hat{g_2} \leftarrow \text{ReduceMean}(\lambda_{\mathcal{H}} \nabla_\theta \mathcal{H}(\mathcal{T}|\theta))$ ▷ Compute entropy gradient
11: $\quad \theta \leftarrow \theta + \alpha(\hat{g_1} + \hat{g_2})$ ▷ Apply gradients
12: $\quad$ **if** $\max \mathcal{R} > R(\tau^\star)$ **then** $\tau^\star \leftarrow \tau^{(\arg\max \mathcal{R})}$ ▷ Update best expression
13: **return** $\tau^\star$

Sample expressions from the RNN

Algorithm from Petersen et al., ICLR 2021

Generating Expressions

- Candidate expressions are generated from an RNN
- The outline of the overall approach mirrors the GP approaches – except candidates are generated from an RNN as opposed to candidates from the previous generation

The RNN

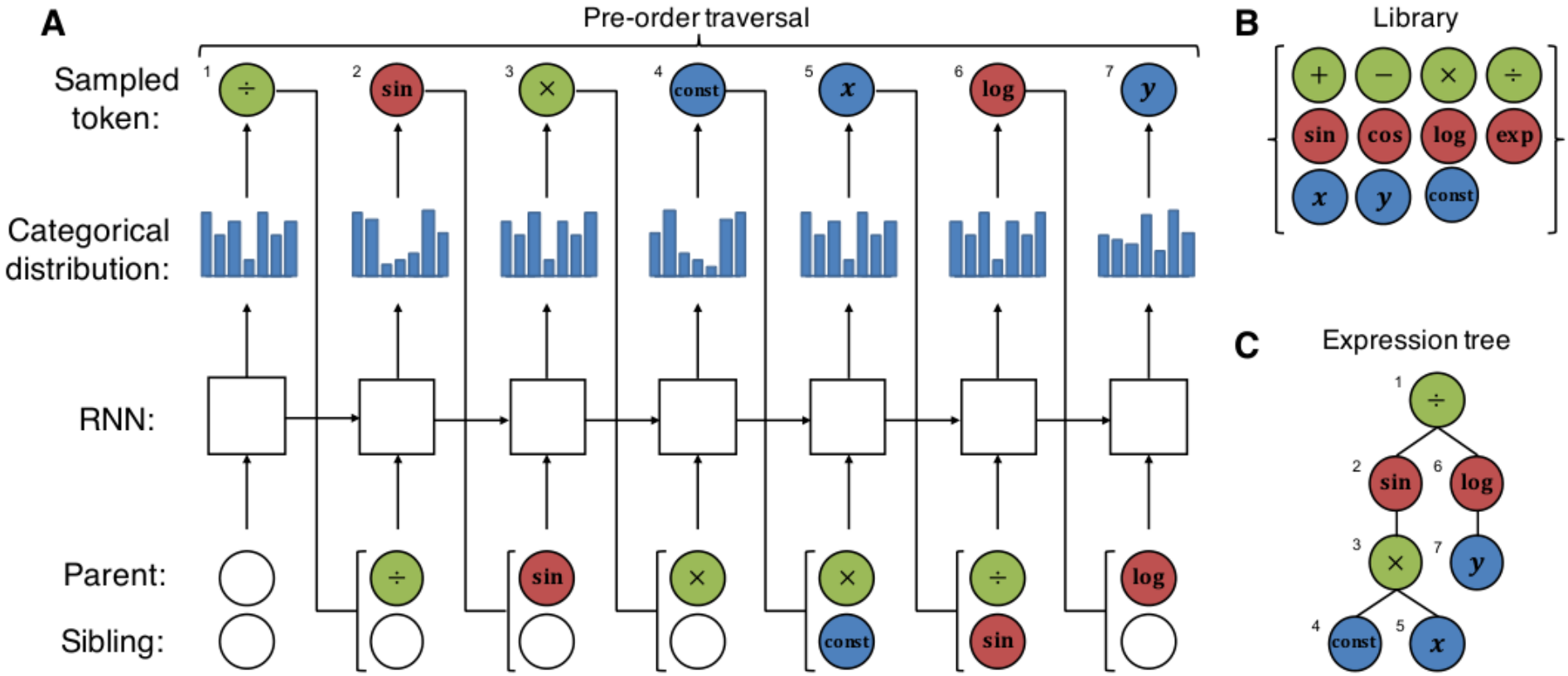
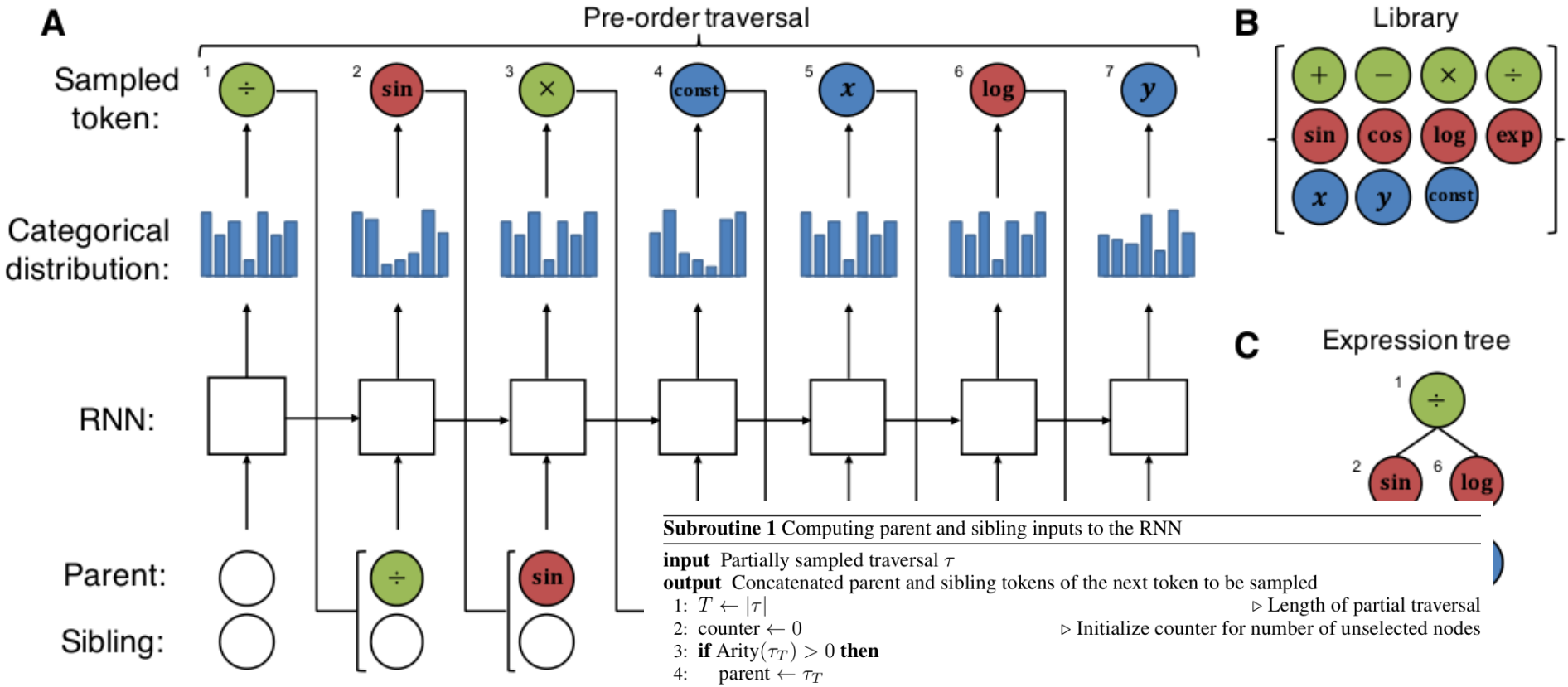


Diagram from Petersen et al., ICLR 2021

The RNN



Subroutine 1 Computing parent and sibling inputs to the RNN

input Partially sampled traversal τ
output Concatenated parent and sibling tokens of the next token to be sampled

$T \leftarrow |\tau|$ ▷ Length of partial traversal
counter $\leftarrow 0$ ▷ Initialize counter for number of unselected nodes
if Arity$(\tau_T) > 0$ **then**
  parent $\leftarrow \tau_T$
  sibling $\leftarrow$ empty
  return parent$\|$sibling
for $i = T, \ldots, 1$ **do** ▷ Iterate backward
  counter $\leftarrow$ counter + Arity$(\tau_i) - 1$ ▷ Update number of unselected nodes
  if counter $= 0$ **then**
    parent $\leftarrow \tau_i$
    sibling $\leftarrow \tau_{i+1}$
    return parent$\|$sibling

Extra procedure to determine the parent and sibling of the next token to be generated.

Diagram and algorithm from Petersen et al., ICLR 2021

Overall Pseudocode

Algorithm 1 Deep symbolic regression with risk-seeking policy gradient

input learning rate α; entropy coefficient $\lambda_{\mathcal{H}}$; risk factor ε; batch size N; reward function R
output Best fitting expression $\tau^\star$

Initialize RNN with parameters θ, defining distribution over expressions $p(\cdot|\theta)$
repeat
$\quad \mathcal{T} \leftarrow \{\tau^{(i)} \sim p(\cdot|\theta)\}_{i=1}^{N}$ ▷ Sample N expressions (Alg. 2 in Appendix A)
$\quad \mathcal{T} \leftarrow \{\text{OptimizeConstants}(\tau^{(i)}, R)\}_{i=1}^{N}$ ▷ Optimize constants w.r.t. reward function
$\quad \mathcal{R} \leftarrow \{R(\tau^{(i)})\}_{i=1}^{N}$ ▷ Compute rewards
$\quad R_\varepsilon \leftarrow (1-\varepsilon)$-quantile of $\mathcal{R}$ ▷ Compute reward threshold
$\quad \mathcal{T} \leftarrow \{\tau^{(i)} : R(\tau^{(i)}) \geq R_\varepsilon\}$ ▷ Select subset of expressions above threshold
$\quad \mathcal{R} \leftarrow \{R(\tau^{(i)}) : R(\tau^{(i)}) \geq R_\varepsilon\}$ ▷ Select corresponding subset of rewards
$\quad \hat{g_1} \leftarrow \text{ReduceMean}((\mathcal{R} - R_\varepsilon)\nabla_\theta \log p(\mathcal{T}|\theta))$ ▷ Compute risk-seeking policy gradient
$\quad \hat{g_2} \leftarrow \text{ReduceMean}(\lambda_{\mathcal{H}} \nabla_\theta \mathcal{H}(\mathcal{T}|\theta))$ ▷ Compute entropy gradient
$\quad \theta \leftarrow \theta + \alpha(\hat{g_1} + \hat{g_2})$ ▷ Apply gradients
$\quad$ **if** $\max \mathcal{R} > R(\tau^\star)$ **then** $\tau^\star \leftarrow \tau^{(\arg\max \mathcal{R})}$ ▷ Update best expression
return $\tau^\star$

Refinement

Algorithm from Petersen et al., ICLR 2021

Refinements

As the output of the RNN is symbolic, it can be further refined based on the syntax of the symbols

Procedure for enforcing certain constraints:

- Min/max Length
- Children of an operator cannot all be constants
- Child of an operator cannot be its inverse (e.g. no *ln(exp(x))*)
- No nested trigonometric operators (very rarely occurring in science)

Subroutine 2 Applying generic constraints in situ when sampling from the RNN

input Categorical probabilities ψ; corresponding library of tokens $\mathcal{L}$; partially sampled traversal τ
output Adjusted categorical probabilities ψ

1: $L \leftarrow |\mathcal{L}|$ ▷ Length of library
2: **for** $i = 1, \ldots, L$ **do**
3: **if** ViolatesConstraint$(\tau, \mathcal{L}_i)$ **then** $\psi_i \leftarrow 0$ ▷ If the token would violate a constraint, set its probability to 0
4: $\psi \leftarrow \frac{\psi}{\sum_i \psi_i}$ ▷ Normalize probability vector back to 1
5: **return** ψ

Constant optimization

- Constants are optimized using a standard method from the greater symbolic regression literature

Subroutine 3 Optimizing the constants of an expression (inner optimization loop)

input Expression τ with placeholder constants ξ; reward function R
output Expression $\tau^\star$ with optimized constants $\xi^\star$

1: $\xi^\star \leftarrow \arg\max_\xi R(\tau; \xi)$ ▷ Maximize reward w.r.t. constants, e.g. with BFGS
2: $\tau^\star \leftarrow$ ReplaceConstants$(\tau, \xi^\star)$ ▷ Replace placeholder constants
3: **return** $\tau^\star$

Algorithms from Petersen et al., ICLR 2021

Overall Pseudocode

Algorithm 1 Deep symbolic regression with risk-seeking policy gradient

input learning rate α; entropy coefficient $\lambda_{\mathcal{H}}$; risk factor ε; batch size N; reward function R
output Best fitting expression $\tau^\star$

Initialize RNN with parameters θ, defining distribution over expressions $p(\cdot|\theta)$
repeat
  $\mathcal{T} \leftarrow \{\tau^{(i)} \sim p(\cdot|\theta)\}_{i=1}^N$ ▷ Sample N expressions (Alg. 2 in Appendix A)
  $\mathcal{T} \leftarrow \{\text{OptimizeConstants}(\tau^{(i)}, R)\}_{i=1}^N$ ▷ Optimize constants w.r.t. reward function
  $\mathcal{R} \leftarrow \{R(\tau^{(i)})\}_{i=1}^N$ ▷ Compute rewards
  $R_\varepsilon \leftarrow (1-\varepsilon)$-quantile of $\mathcal{R}$ ▷ Compute reward threshold
  $\mathcal{T} \leftarrow \{\tau^{(i)} : R(\tau^{(i)}) \geq R_\varepsilon\}$ ▷ Select subset of expressions above threshold
  $\mathcal{R} \leftarrow \{R(\tau^{(i)}) : R(\tau^{(i)}) \geq R_\varepsilon\}$ ▷ Select corresponding subset of rewards
  $\hat{g}_1 \leftarrow \text{ReduceMean}((\mathcal{R} - R_\varepsilon)\nabla_\theta \log p(\mathcal{T}|\theta))$ ▷ Compute risk-seeking policy gradient
  $\hat{g}_2 \leftarrow \text{ReduceMean}(\lambda_{\mathcal{H}} \nabla_\theta \mathcal{H}(\mathcal{T}|\theta))$ ▷ Compute entropy gradient
  $\theta \leftarrow \theta + \alpha(\hat{g}_1 + \hat{g}_2)$ ▷ Apply gradients
  if $\max \mathcal{R} > R(\tau^\star)$ **then** $\tau^\star \leftarrow \tau^{(\arg\max \mathcal{R})}$ ▷ Update best expression
return $\tau^\star$

Compute reward of each expression

Algorithm from Petersen et al., ICLR 2021

NRMSE

Normalized Root Mean Squared Error is the common metric used in symbolic regression to evaluate a learned expression f against n training samples; it is RMSE normalized by the standard deviation of the target values

$$\mathrm{NRMSE} = \frac{1}{\sigma_y}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - f(X_i))^2}$$

$$R(\tau) = 1/(1 + \mathrm{NRMSE}).$$

- The reward function R is created from this measure
- It is not differentiable, hence an RL framework is used.

Overall Pseudocode

Algorithm 1 Deep symbolic regression with risk-seeking policy gradient

input learning rate α; entropy coefficient $\lambda_{\mathcal{H}}$; risk factor ε; batch size N; reward function R
output Best fitting expression $\tau^\star$

Initialize RNN with parameters θ, defining distribution over expressions $p(\cdot|\theta)$
repeat
  $\mathcal{T} \leftarrow \{\tau^{(i)} \sim p(\cdot|\theta)\}_{i=1}^N$ ▷ Sample N expressions (Alg. 2 in Appendix A)
  $\mathcal{T} \leftarrow \{\text{OptimizeConstants}(\tau^{(i)}, R)\}_{i=1}^N$ ▷ Optimize constants w.r.t. reward function
  $\mathcal{R} \leftarrow \{R(\tau^{(i)})\}_{i=1}^N$ ▷ Compute rewards
  $R_\varepsilon \leftarrow (1-\varepsilon)$-quantile of $\mathcal{R}$ ▷ Compute reward threshold
  $\mathcal{T} \leftarrow \{\tau^{(i)} : R(\tau^{(i)}) \geq R_\varepsilon\}$ ▷ Select subset of expressions above threshold
  $\mathcal{R} \leftarrow \{R(\tau^{(i)}) : R(\tau^{(i)}) > R_\varepsilon\}$ ▷ Select corresponding subset of rewards
  $\hat{g}_1 \leftarrow \text{ReduceMean}((\mathcal{R} - R_\varepsilon)\nabla_\theta \log p(\mathcal{T}|\theta))$ ▷ Compute risk-seeking policy gradient
  $\hat{g}_2 \leftarrow \text{ReduceMean}(\lambda_{\mathcal{H}} \nabla_\theta \mathcal{H}(\mathcal{T}|\theta))$ ▷ Compute entropy gradient
  $\theta \leftarrow \theta + \alpha(\hat{g}_1 + \hat{g}_2)$ ▷ Apply gradients
  if $\max \mathcal{R} > R(\tau^\star)$ **then** $\tau^\star \leftarrow \tau^{(\arg\max \mathcal{R})}$ ▷ Update best expression
return $\tau^\star$

Compute top 1-ε quantile of expressions

Algorithm from Petersen et al., ICLR 2021

Risk Seeking Reward Function

- Standard policy gradient is based on an overall expected value

- CVaR policy gradient considers only the lowest risk candidates in a given sample (Tamar et al., 2014)

- This work uses a **risk-seeking policy gradient** that is looking at the highest-reward expressions

Overall Pseudocode

Algorithm 1 Deep symbolic regression with risk-seeking policy gradient

input learning rate α; entropy coefficient $\lambda_{\mathcal{H}}$; risk factor ε; batch size N; reward function R
output Best fitting expression $\tau^\star$

Initialize RNN with parameters θ, defining distribution over expressions $p(\cdot|\theta)$
repeat
  $\mathcal{T} \leftarrow \{\tau^{(i)} \sim p(\cdot|\theta)\}_{i=1}^N$ ▷ Sample N expressions (Alg. 2 in Appendix A)
  $\mathcal{T} \leftarrow \{\text{OptimizeConstants}(\tau^{(i)}, R)\}_{i=1}^N$ ▷ Optimize constants w.r.t. reward function
  $\mathcal{R} \leftarrow \{R(\tau^{(i)})\}_{i=1}^N$ ▷ Compute rewards
  $R_\varepsilon \leftarrow (1-\varepsilon)$-quantile of $\mathcal{R}$ ▷ Compute reward threshold
  $\mathcal{T} \leftarrow \{\tau^{(i)} : R(\tau^{(i)}) > R_\varepsilon\}$ ▷ Select subset of expressions above threshold
  $\mathcal{R} \leftarrow \{R(\tau^{(i)}) : R(\tau^{(i)}) > R_\varepsilon\}$ ▷ Select corresponding subset of rewards
  $\hat{g}_1 \leftarrow \text{ReduceMean}((\mathcal{R} - R_\varepsilon)\nabla_\theta \log p(\mathcal{T}|\theta))$ ▷ Compute risk-seeking policy gradient
  $\hat{g}_2 \leftarrow \text{ReduceMean}(\lambda_{\mathcal{H}} \nabla_\theta \mathcal{H}(\mathcal{T}|\theta))$ ▷ Compute entropy gradient
  $\theta \leftarrow \theta + \alpha(\hat{g}_1 + \hat{g}_2)$ ▷ Apply gradients
  if $\max \mathcal{R} > R(\tau^\star)$ **then** $\tau^\star \leftarrow \tau^{(\arg\max \mathcal{R})}$ ▷ Update best expression
return $\tau^\star$

Compute risk-seeking policy gradient

Algorithm from Petersen et al., ICLR 2021

Gradient of Risk-Seeking Policy Gradient

Proof of gradient follows intuitions from risk-avoiding policy gradient of CVaR.

Proposition 1. *Let $J_{risk}(\theta;\varepsilon)$ denote the conditional expectation of rewards above the $(1-\varepsilon)$-quantile, as in Equation (1). Then the gradient of $J_{risk}(\theta;\varepsilon)$ is given by:*

$$\nabla_\theta J_{risk}(\theta;\varepsilon) = \mathbb{E}_{\tau\sim p(\tau|\theta)}[(R(\tau) - R_\varepsilon(\theta)) \cdot \nabla_\theta \log p(\tau|\theta) \mid R(\tau) \geq R_\varepsilon(\theta)]$$

Approximation is performed via MC sampling

$$\nabla_\theta J_{\text{risk}}(\theta;\varepsilon) \approx \frac{1}{\varepsilon N} \sum_{i=1}^{N} \left[R(\tau^{(i)}) - \tilde{R}_\varepsilon(\theta)\right] \cdot \mathbf{1}_{R(\tau^{(i)})\geq \tilde{R}_\varepsilon(\theta)} \nabla_\theta \log p(\tau^{(i)}|\theta)$$

Overall Pseudocode

Algorithm 1 Deep symbolic regression with risk-seeking policy gradient

input learning rate α; entropy coefficient $\lambda_{\mathcal{H}}$; risk factor ε; batch size N; reward function R
output Best fitting expression $\tau^\star$

Initialize RNN with parameters θ, defining distribution over expressions $p(\cdot|\theta)$
repeat
  $\mathcal{T} \leftarrow \{\tau^{(i)} \sim p(\cdot|\theta)\}_{i=1}^N$ ▷ Sample N expressions (Alg. 2 in Appendix A)
  $\mathcal{T} \leftarrow \{\text{OptimizeConstants}(\tau^{(i)}, R)\}_{i=1}^N$ ▷ Optimize constants w.r.t. reward function
  $\mathcal{R} \leftarrow \{R(\tau^{(i)})\}_{i=1}^N$ ▷ Compute rewards
  $R_\varepsilon \leftarrow (1-\varepsilon)$-quantile of $\mathcal{R}$ ▷ Compute reward threshold
  $\mathcal{T} \leftarrow \{\tau^{(i)} : R(\tau^{(i)}) \geq R_\varepsilon\}$ ▷ Select subset of expressions above threshold
  $\mathcal{R} \leftarrow \{R(\tau^{(i)}) : R(\tau^{(i)}) \geq R_\varepsilon\}$ ▷ Select corresponding subset of rewards
  $\hat{g}_1 \leftarrow \text{ReduceMean}((\mathcal{R} - R_\varepsilon)\nabla_\theta \log p(\mathcal{T}|\theta))$ ▷ Compute risk-seeking policy gradient
  $\hat{g}_2 \leftarrow \text{ReduceMean}(\lambda_{\mathcal{H}} \nabla_\theta \mathcal{H}(\mathcal{T}|\theta))$ ▷ Compute entropy gradient
  $\theta \leftarrow \theta + \alpha(\hat{g}_1 + \hat{g}_2)$ ▷ Apply gradients
  if $\max \mathcal{R} > R(\tau^\star)$ **then** $\tau^\star \leftarrow \tau^{(\arg\max \mathcal{R})}$ ▷ Update best expression
return $\tau^\star$

Entropy gradient points adds some extra performance improvement

Update parameters for RNN and optimal expression

Algorithm from Petersen et al., ICLR 2021

Experiments

Benchmark	Expression	DSR	PQT	VPG	GP	Eureqa	Wolfram
Nguyen-1	$x^3 + x^2 + x$	100%	100%	96%	100%	100%	100%
Nguyen-2	$x^4 + x^3 + x^2 + x$	100%	99%	47%	97%	100%	100%
Nguyen-3	$x^5 + x^4 + x^3 + x^2 + x$	100%	86%	4%	100%	95%	100%
Nguyen-4	$x^6 + x^5 + x^4 + x^3 + x^2 + x$	100%	93%	1%	100%	70%	100%
Nguyen-5	$\sin(x^2)\cos(x) - 1$	72%	73%	5%	45%	73%	2%
Nguyen-6	$\sin(x) + \sin(x + x^2)$	100%	98%	100%	91%	100%	1%
Nguyen-7	$\log(x + 1) + \log(x^2 + 1)$	35%	41%	3%	0%	85%	0%
Nguyen-8	$\sqrt{x}$	96%	21%	5%	5%	0%	71%
Nguyen-9	$\sin(x) + \sin(y^2)$	100%	100%	100%	100%	100%	–
Nguyen-10	$2\sin(x)\cos(y)$	100%	91%	99%	76%	64%	–
Nguyen-11	x^y	100%	100%	100%	7%	100%	–
Nguyen-12	$x^4 - x^3 + \frac{1}{2}y^2 - y$	0%	0%	0%	0%	0%	–
	Average	**83.6%**	75.2%	46.7%	60.1%	73.9%	–

End to End Symbolic Regression with Transformers

Overview

After hearing about DSR, you may be wondering…

- Why didn't they just replace NRMSE with something like cross-entropy and train with supervised technique?

- Why was an RNN used and not a transformer architecture?

Research at Meta AI asked the same questions – and wrote this paper

- Transformer, language-model style architecture
- Trained model with cross-entropy loss
- Generates training data using other techniques from SR
- Faster than DSR and other approaches
- Not yet published – currently an arXiv preprint

arXiv:2204.10532v1 [cs.LG] 22 Apr 2022

End-to-end symbolic regression with transformers

Pierre-Alexandre Kamienny[†1,2], Stéphane d'Ascoli[†1,3], Guillaume Lample[1] and François Charton[1]

[1]Meta AI, Paris
[2]ISIR MLIA, Sorbonne Université, Paris
[3]Department of Physics, Ecole Normale Supérieure, Paris

Abstract

Symbolic regression, the task of predicting the mathematical expression of a function from the observation of its values, is a difficult task which usually involves a two-step procedure: predicting the "skeleton" of the expression up to the choice of numerical constants, then fitting the constants by optimizing a non-convex loss function. The dominant approach is genetic programming, which evolves candidates by iterating this subroutine a large number of times. Neural networks have recently been tasked to predict the correct skeleton in a single try, but remain much less powerful. In this paper, we challenge this two-step procedure, and task a Transformer to directly predict the full mathematical expression, constants included. One can subsequently refine the predicted constants by feeding them to the non-convex optimizer as an informed initialization. We present ablations to show that this end-to-end approach yields better results, sometimes even without the refinement step. We evaluate our model on problems from the SRBench benchmark and show that our model approaches the performance of state-of-the-art genetic programming with several orders of magnitude faster inference.

Introduction

Inferring mathematical laws from experimental data is a central problem in natural science; having observed a variable y at n points $\{x_i\}_{i\in\mathbb{N}_n}$, it implies finding a function f such that $y_i \approx f(x_i)$ for all $i \in \mathbb{N}_n$. Two types of approaches exist to solve this problem. In *parametric statistics* (PS), the function f is defined by a small number of parameters that can directly be estimated from the data. On the other hand, *machine learning* (ML) techniques such as decision trees and neural networks select f from large families of non-linear functions by minimizing a loss over the data. The latter relax the assumptions about the underlying law, but their solutions are more difficult to interpret, and tend to overfit small experimental data sets, yielding poor extrapolation performance.

Symbolic regression (SR) stands as a middle ground between PS and ML approaches: f is selected from a large family of functions, but is required to be defined by an interpretable analytical expression. It has already proved extremely useful in a variety of tasks such as inferring physical laws [1, 2].

SR is usually performed in two steps. First, predicting a "skeleton", a parametric function using a pre-defined list of operators – typically, the basic operations $(+, \times, \div)$ and functions (sqrt, exp, sin, etc.). It determines the general shape of the law up to a choice of constants, e.g. $f(x) = \cos(ax + b)$. Then, the constants in the skeleton (a, b) are estimated using optimization techniques, typically the Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS).

[†]pakamienny@fb.com
[†]stephane.dascoli@gmail.com
[†]Equal contribution.

Meta's SR Approach

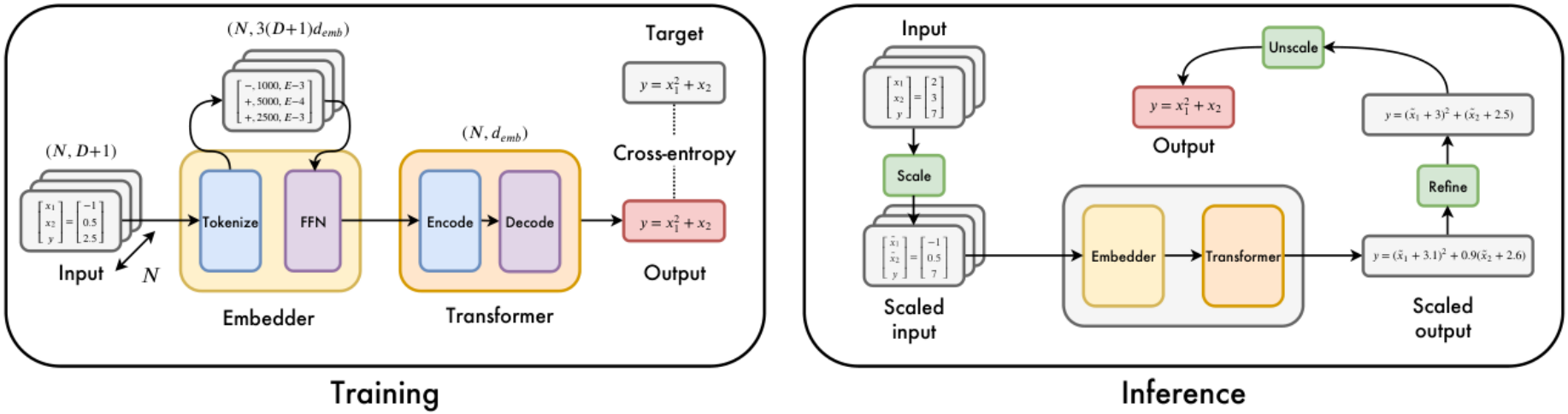


Figure from Kamienny et al., 2022.

Supervised Approach

To take a supervised approach, the authors generate training data

- Generate functions (methodology using prior work on SR) – each function is a sample
- Generate sample data (methodology they propose seeding sample data with random centroids)
- Translate functions into direct Polish notation

expression $f(x) = \cos(2.4242x)$ is encoded as `[cos,mul,+,2424,E-3,x]`

- Training samples consist of up to D input numbers and 1 output number. Each number is represented by 3 tokens (sign, mantissa, and exponent). This results in each of N *input values* (per function) having 3(D+1) tokens
  - N varies but is usually 100-200

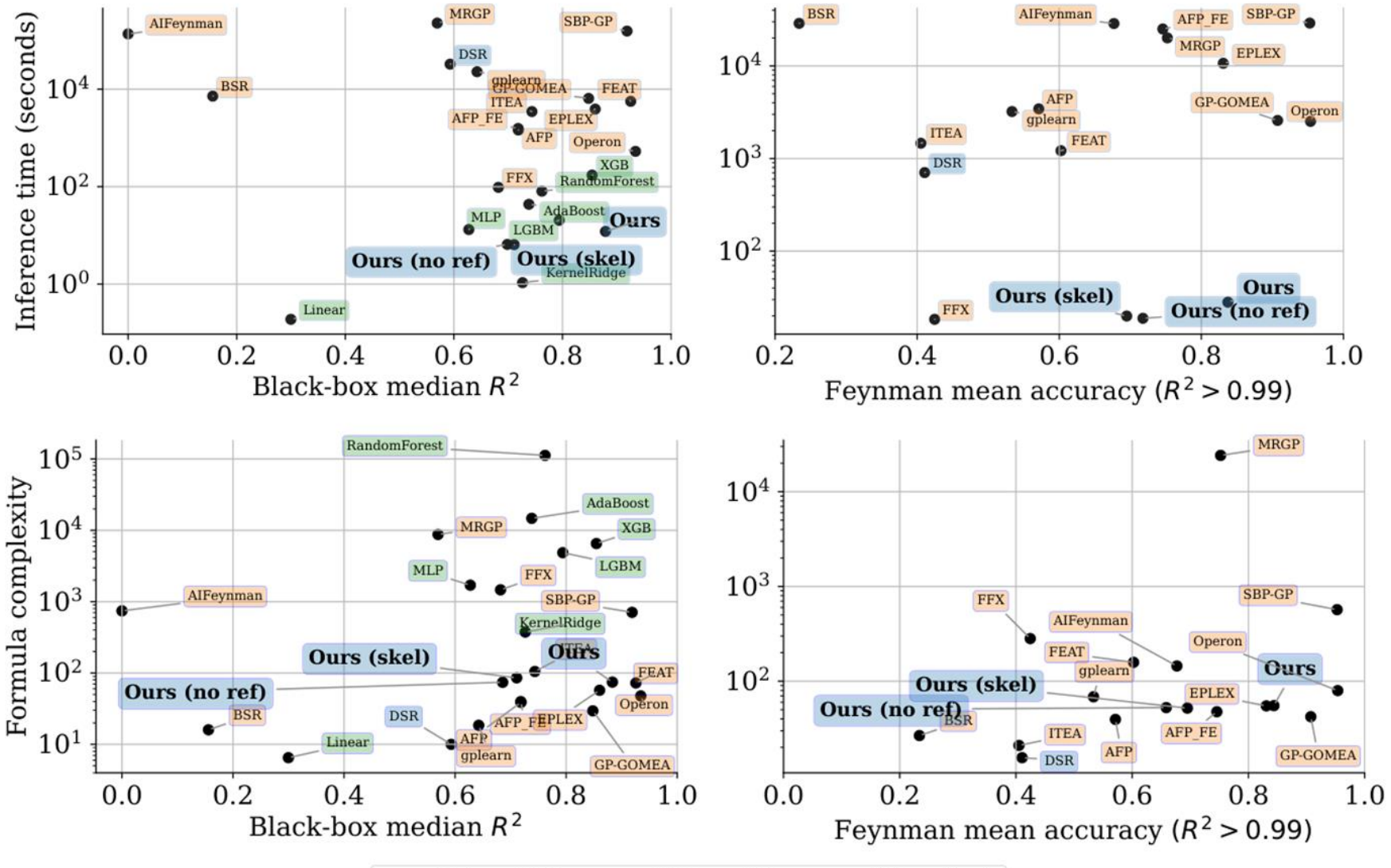
Notation from Kamienny et al., 2022.

Embedding and Transformer

- For large *D, N,* the *N* samples of *3(D+1)* tokens becomes large for the transformer architecture (which has quadratic complexity)
- An embedder feeds the $3(D+1)d_{emb}$ vector into two fully connected layers to project down to d_{emb} dimensions
- Transformer has 16 attention heads, embedding dimension of 512, 86M parameters (roughly 4 times larger than the first AlphaFold)
- Note that the N input points are permutation invariant, so the positional encoding is not included
- Despite being an end-to-end approach, refinement is still use, in particular to optimize the constants (but the authors note the optimization is initialized with their results)

Notation from Kamienny et al., 2022.

Experimental Results



From Kamienny et al., 2022.

Comparison of Approaches

- DSR takes in one set of N samples that are to be explained by a single mathematical expression and trains an RNN to generate candidate expressions that fit to the N samples.
- Meta's transformer-based approach to SR takes in 3M examples (each with N samples, and the number of samples per example varies) to create a model that takes as input N samples to produce a mathematical expression

Deep Symbolic Policies

The RL approach to SR, revisited

- Are there other reasons why we would want to leverage an RL approach for SR?

- Yes, we may want to create policies around control.

National Ignition Facility:
Controlling Nuclear Fusion Reactions with Lasers

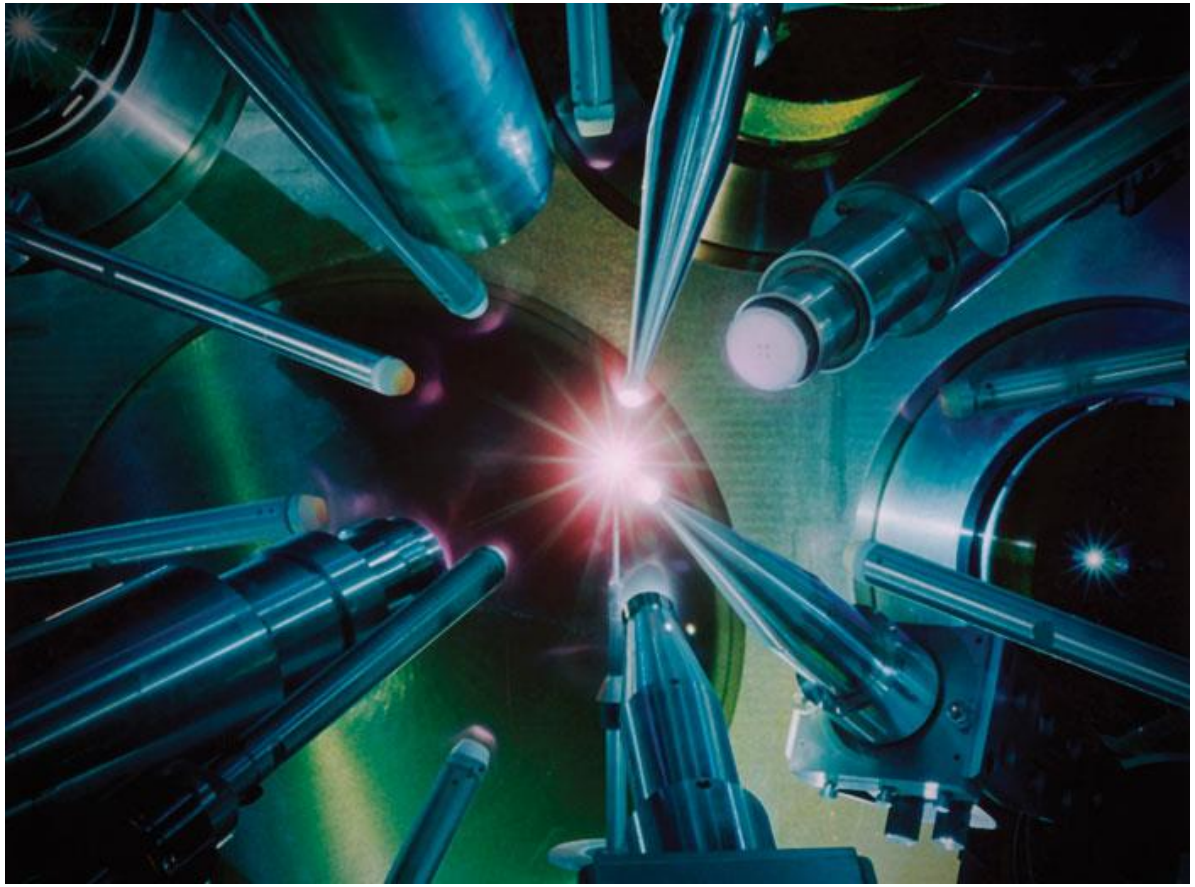


Image from https://www.americansecurityproject.org/inertial-confinement-fusion-at-the-national-ignition-facility/

National Ignition Facility (actual photo)

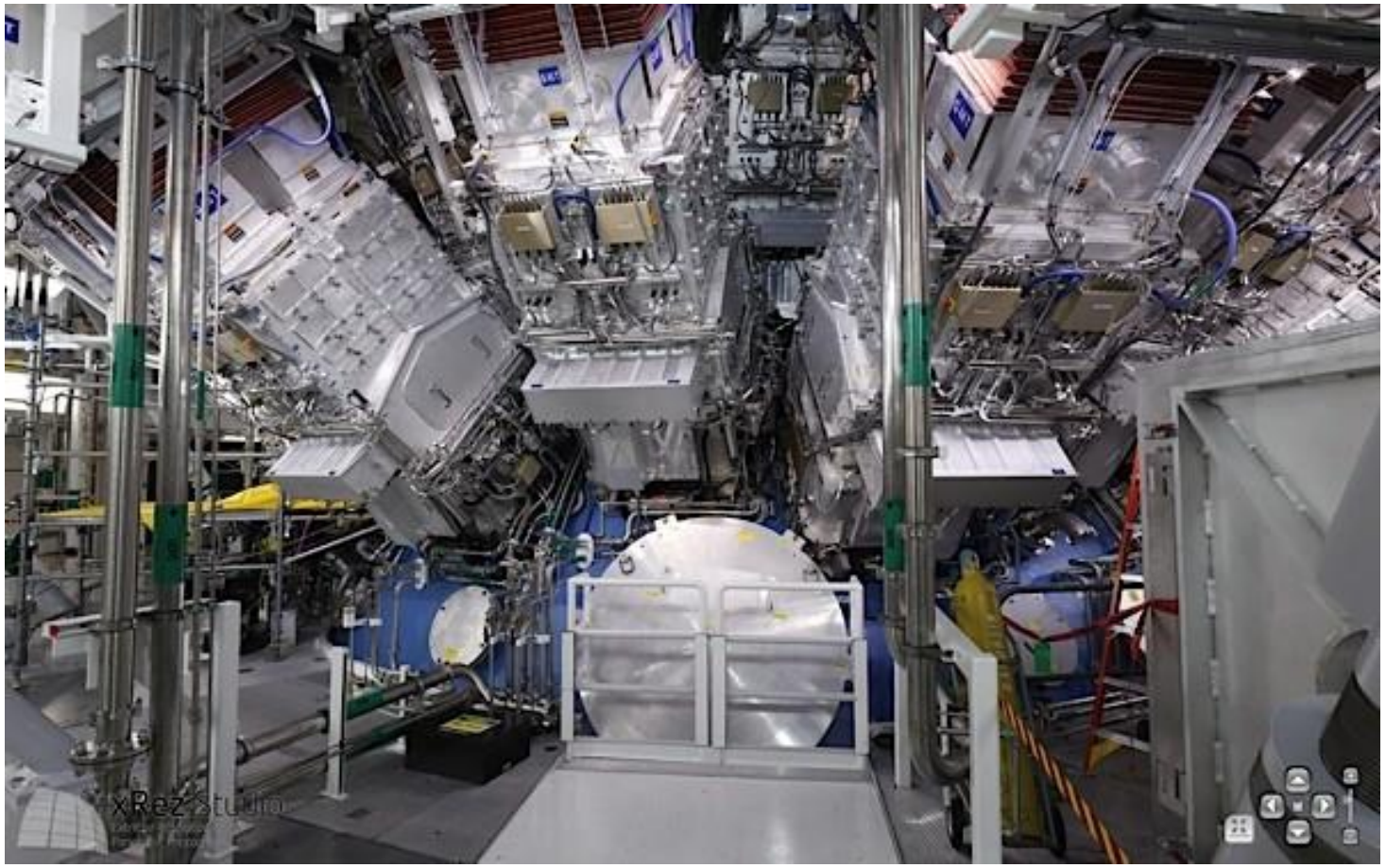


Image source https://energyvulture.com/2015/03/06/national-ignition-facility-one-step-closer-to-fusion-power/

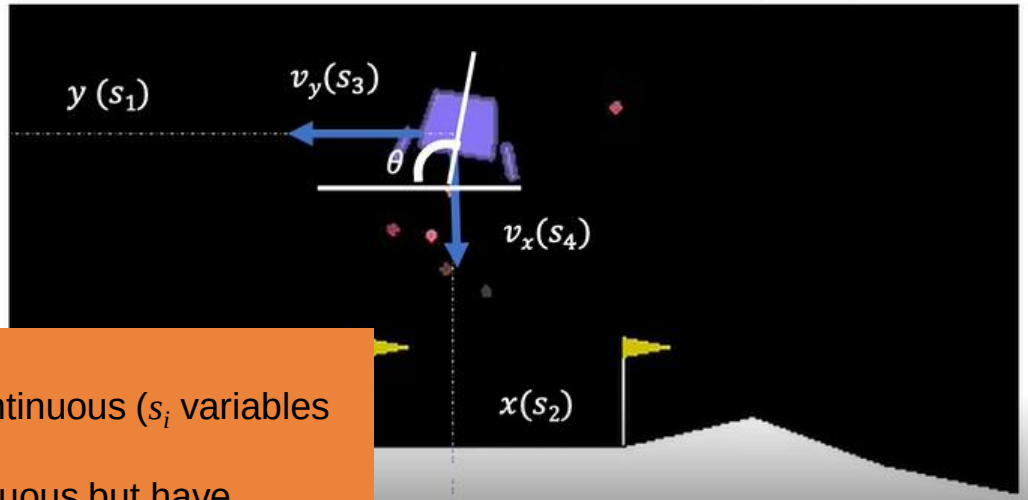
Concept: Learn Symbolic Policies for Control

6. LunarLander

(Gym: LunarLanderContinuous-v2)

Discovered symbolic policy

$$a_1 = -10s_2 + \sin(s_3) - 14s_4 - 1.99 \qquad a_2 = -5.79\frac{s_4}{s_6 - s_3}$$



States

s_1	y
s_2	x
s_3	v_y
s_4	v_x
s_5	θ
s_6	$\mathrm{d}\theta/dt$
s_7	First leg contact?
s_8	Second leg contact?

Actions

a_1	Fire main engine
a_2	Fire left (-) or right (+) engine

Intuition:

- State space is continuous (s_i variables in the expression)
- Actions are continuous but have ***multiple dimensions*** (the example on this slide has 2 dimensions)

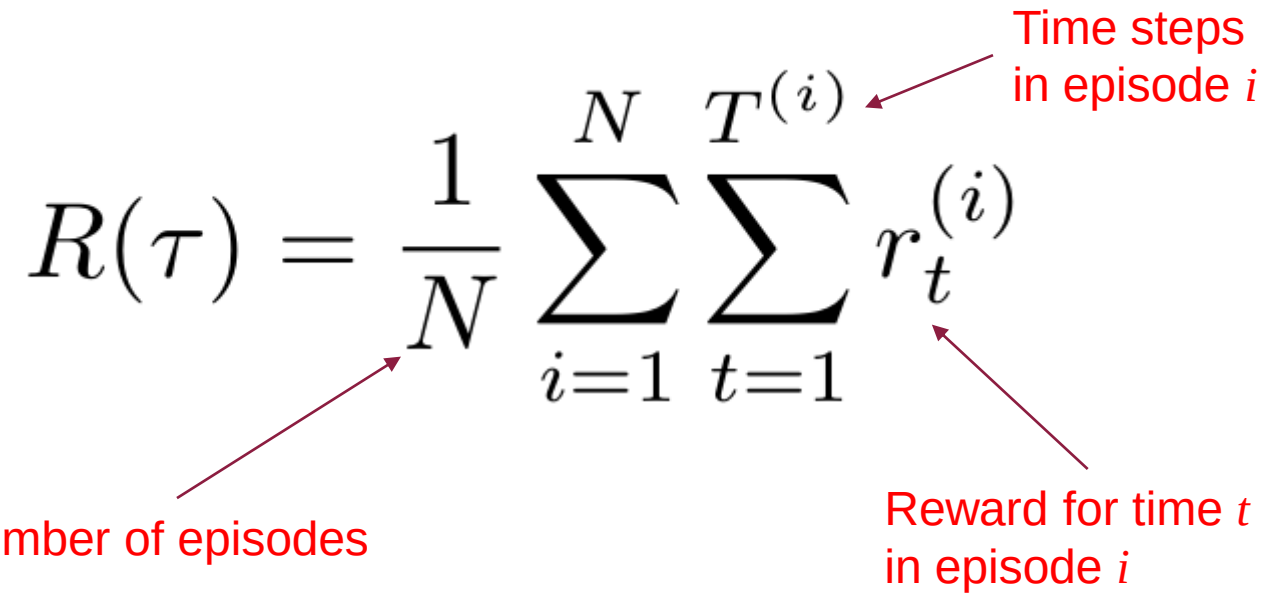
Image from Supplemental Material of Landajuela et al., ICML 2021.

Why Symbolic Policies for Control?

- Traditional control theory and mathematical physics approaches for control result in simple but effective models
- Further, these models are mathematical equations that are simple (hence regularized), easily understood, and can be efficient to implement
- Prior work on RL for control results in black-box models that do not have these features

Why Not Apply SR Directly?

- The authors cite “objective function mismatch” meaning that the policy should be trained on actual reward
- They show that using standard SR optimization criteria leads to catastrophic failure



Expression from Landajuela et al., ICML 2021.

Adding Multiple Dimensions

- The authors note that multiple action dimensions leads to a combinatorial explosion
- They overcome the problem using a non-symbolic “anchor model”
- The intuition is that each action dimension is learned sequentially.
  - When action dimension i is learned, the algorithm uses previously learned symbolic actions $1,\ldots,i\text{-}1$ and the anchor (non-symbolic NN-learned) actions for $i+1,\ldots,n$.

Adding Multiple Dimensions

Algorithm 1 Deep symbolic policy with "anchoring" for multi-action environments

Function DeepSymbolicPolicy($\alpha, \epsilon, \eta, \gamma, \mathcal{E}, \Psi, M, N$)

Input: Learning rate α, risk factor ϵ, entropy weight η, entropy decay γ, environment $\mathcal{E}$ with observation space $\mathcal{S} \subset \mathbb{R}^m$ and action space $\mathcal{A} \subset \mathbb{R}^n$, pre-trained "anchor" policy Ψ (if $n > 1$), maximum number of iterations M, number of training episodes N

Output: Fully symbolic policy f

for *action dimension* $i = 1, \ldots, n$ **do**

Initialize RNN with parameters θ, defining distribution over expressions $p(\cdot|\theta)$

$R(\cdot) \leftarrow \text{PolicyEvaluator}(\cdot, \bar{f}_1, \ldots, \bar{f}_{i-1}, \Psi, \mathcal{E}, N)$ // Define closure for Policy Evaluator

$\tau^\star \leftarrow \text{null}$ // Initialize best expression

for *iteration* $= 1, \ldots, M$ **do**

$\mathcal{T} = \{\tau^{(i)} \sim p(\cdot|\theta)\}_{i=1,\ldots,N}$ // Sample expressions via Policy Generator

$\mathcal{R} = \{R(\tau^{(i)})\}_{i=1,\ldots,N}$ // Compute rewards

$R_\epsilon = (1-\epsilon)$ percentile of $\mathcal{R}$ // Compute reward threshold

$\mathcal{T}_\epsilon = \{\tau^{(i)} : R(\tau) \geq R_\epsilon\}$ // Select subset of expressions above threshold

$\hat{g}_1 = \frac{1}{|\mathcal{T}_\epsilon|} \sum_{\tau \in \mathcal{T}_\epsilon} ((R(\tau) - R_\epsilon) \nabla_\theta \log p(\tau|\theta))$ // Risk-seeking policy gradient

$\hat{g}_2 = \frac{1}{|\mathcal{T}_\epsilon|} \sum_{\tau \in \mathcal{T}_\epsilon} \left(\eta \sum_{i=1}^{|\tau|} \gamma^{i-1} \nabla_\theta H[p(\tau_i|\tau_{1:(i-1)}; \theta)] \right)$ // Hierarchical entropy gradient

$\theta \leftarrow \theta + \alpha(\hat{g}_1 + \hat{g}_2)$ // Apply gradients

if $\max \mathcal{R} > R(\tau^\star)$ **then** $\tau^\star \leftarrow \tau^{(\arg\max \mathcal{R})}$ // Update best expression

end

$\bar{f}_i \leftarrow \text{Instantiate}(\tau^\star)$ // Set fixed sub-policy for next action dimension

end

$f \leftarrow \langle \bar{f}_1, \ldots \bar{f}_n \rangle$ // Final policy is fully symbolic

return f

Note this follows the same steps as DSR.

Pseudocode from Supplemental Material of Landajuela et al., ICML 2021.

Adding Multiple Dimensions

Algorithm 2 Policy Evaluator, used to compute reward for symbolic policy τ

Function PolicyEvaluator($\tau, \bar{f}_1, \ldots, \bar{f}_{i-1}, \Psi, \mathcal{E}, N$)
Input: Symbolic policy being evaluated τ, previously learned fixed symbolic expressions $\bar{f}_1, \ldots, \bar{f}_{i-1}$, pre-trained "anchor" policy Ψ, environment $\mathcal{E}$, number of training episodes N
Output: Reward R for symbolic policy τ

$R \leftarrow 0$
$f \leftarrow$ Instantiate(τ)
for *episode* $= 1, \ldots, N$ **do**
    $s \leftarrow$ Reset($\mathcal{E}$) // Sample new starting state
    while $\mathcal{E}$ *is non-terminal* **do**
        $a \leftarrow \langle \bar{f}_1(s), \ldots, \bar{f}_{i-1}(s), f(s), \Psi_{i+1}(s), \ldots, \Psi_n(s) \rangle$ // Compute action
        $s, r \leftarrow$ Execute($\mathcal{E}, a$) // Step the environment
        $R \leftarrow R + r$
    end
end
$R \leftarrow \frac{R}{N}$ // Average episodic rewards
return R

Note the evaluation uses previously learned symbolic policies up to dimension i-1, the current policy being evaluated for i and the anchor for the remaining dimensions.

Pseudocode from Supplemental Material of Landajuela et al., ICML 2021.

Additional Optimizations Introduced

Noted that these techniques are applicable to other combinatorial RL problems

- Hierarchical entropy regularization in loss function
- Soft length prior to lead to more variety of sequence length

Example Results

Environment	DSP	DSP$^{\text{o}}$
CartPole	$a_1 = 10s_3 + s_4$	$a_1 = 10.03s_3 + 0.45s_4$
MountainCar	$a_1 = -\frac{0.62}{\log(s_2)}$	$a_1 = -\frac{0.601}{\log(s_2)}$
Pendulum	$a_1 = -2s_2 - \frac{8s_2+2s_3}{s_1}$	$a_1 = -7.08s_2 - \frac{13.39s_2+3.12s_3}{s_1} + 0.27$
InvDoublePend	$a_1 = -12.2s_8$	$a_1 = -12.23s_8$
InvPendSwingup	$a_1 = s_1 + 5s_4 + s_5 + s_6 + \sin(s_2) + \sin(s_2 + s_4 + s_5) + 0.19$	$a_1 = s_1 + 5s_4 + s_5 + s_6 + \sin(s_2) + \sin(s_2 + s_4 + s_5) + 0.21$
LunarLander	$a_1 = -10s_2 + \sin(s_3) - 14s_4 - 1.99$ $a_2 = -5.79\frac{s_4}{s_6 - s_3}$	$a_1 = -5.99s_2 + 0.76\sin(s_3) - 9.80s_4 - 1.35$ $a_2 = -3.49\frac{s_4}{s_6 - s_3}$
Hopper	$a_1 = \exp\left(\frac{s_6}{s_{10}}\sec\left(\frac{s_1 s_2 s_{14}}{s_1 + s_4 - s_8}\right)\right)$ $a_2 = -5s_4 - 2s_6 - 6s_8 - s_{11} + \cos(s_4)$ $a_3 = \frac{\cos(s_{13})}{s_{11} + \sin(s_1)}$	$a_1 = 1.09\exp\left(\frac{s_6}{s_{10}}\sec\left(\frac{s_1 s_2 s_{14}}{s_1 + s_4 - s_8}\right)\right) - 0.02$ $a_2 = -6.22s_4 - 2.49s_6 - 7.47s_8 - 1.24s_{11} + 1.24\cos(s_4) - 0.03$ $a_3 = 1.03\frac{\cos(s_{13})}{s_{11} + \sin(s_1)}$
BipedalWalker	$a_1 = s_1 - s_8 - s_9 - 2s_{11} + s_{24}$ $a_2 = \frac{\sin(2s_3 - s_7)}{s_{22} - \sin(s_3) + \cos(s_7)} - s_7$ $a_3 = s_3 - s_{10} - \sin(s_{12}) - \cos(2s_{10})$ $a_4 = \frac{s_{10}}{s_2}\left(s_2^2 - \frac{e}{s_2} + \log(s_2) + 5\right)$	$a_1 = 0.03s_1 - 0.03s_8 - 0.03s_9 - 0.1s_{11} + 0.03s_{24} + 0.11$ $a_2 = \frac{0.9\sin(3.12s_3 - s_7)}{s_{22} - \sin(s_3) + \cos(s_7)} - 0.9s_7 - 0.4$ $a_3 = 1.14s_3 - 1.14s_{10} - 1.14\sin(s_{12}) - 1.14\cos(1.95s_{10}) - 0.22$ $a_4 = 0.18\frac{s_{10}}{s_2}\left(s_2^2 - \frac{e}{s_2} + \log(s_2) + 3.28\right) + 3.24$

Table from Landajuela et al, ICML 2021

Comparison to Baselines

Environment	DSP	DSP^{O}	Regression	DDPG	TRPO	A2C	PPO	ACKTR	SAC	TD3	$\overline{\text{Zoo}}$
CartPole	999.59	1000	211.82	1000	1000	1000	993.94	1000	971.78	997.98	994.81
MountainCar	99.09	99.11	95.16†	91.77	93.95	93.63	92.56	93.77	90.40	93.93	92.86
Pendulum	-160.5	-155.4	-1206.9	-169.0	-147.6	-162.2	-154.8	-211.2	-159.3	-147.1	-164.4
InvDoublePend	9148.2	9149.9	637.2	8855.1	8854.8	8951.9	9225.5	7554.1	9313.8	9357.8	8873.3
InvPendSwingup	891.84	891.90	-19.21	891.34	892.51	67.52	853.38	890.34	891.47	889.33	767.98
LunarLander	251.66	261.36	56.08	246.24	168.79	227.08	225.12	245.39	272.65	225.35	230.09
Hopper	2090.2	2122.4	47.35	1632.7	2583.4	1925.1	2439.7	2456.7	2455.0	2741.9	2319.2
BipedalWalker	264.39	311.78	-110.77	94.21	311.08	241.01	286.20	299.32	307.26	310.19	264.18
Average rank	**2.63**		8.13	5.63	3.25	5.63	5.63	4.88	4.50	3.50	
Worst rank	**6**		9	8	8	8	7	8	9	**6**	
Average $\bar{R}_{\text{ep}}$	**0.96**		0.07	0.75	0.85	0.72	0.85	0.86	0.85	0.90	

Standard SR (Regression = DSR) performs poorly
DSP generally outperforms standard RL methods
(Note DSP^{O} = DSP with constant optimization)

Table from Landajuela et al, ICML 2021

Beyond SR / DSR / DSP

Xu and Ferki, 2021 (preprint) propose a hierarchical RL problem for robot navigation

- Higher level MDP involves a symbolic transition function learned using dILP
- The higher-level solver selects a subgoal
- Lower-level solver work to obtain the goal

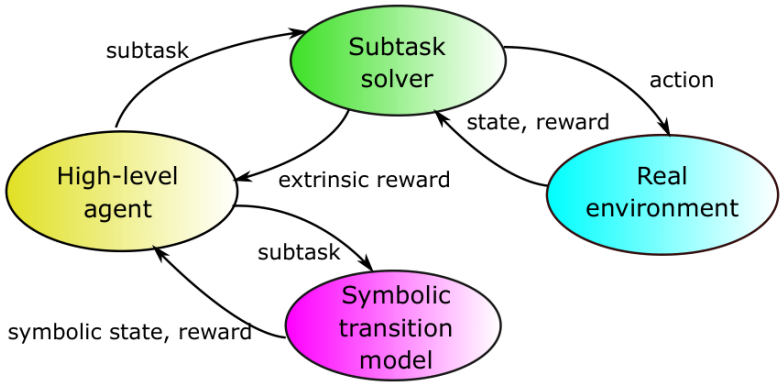
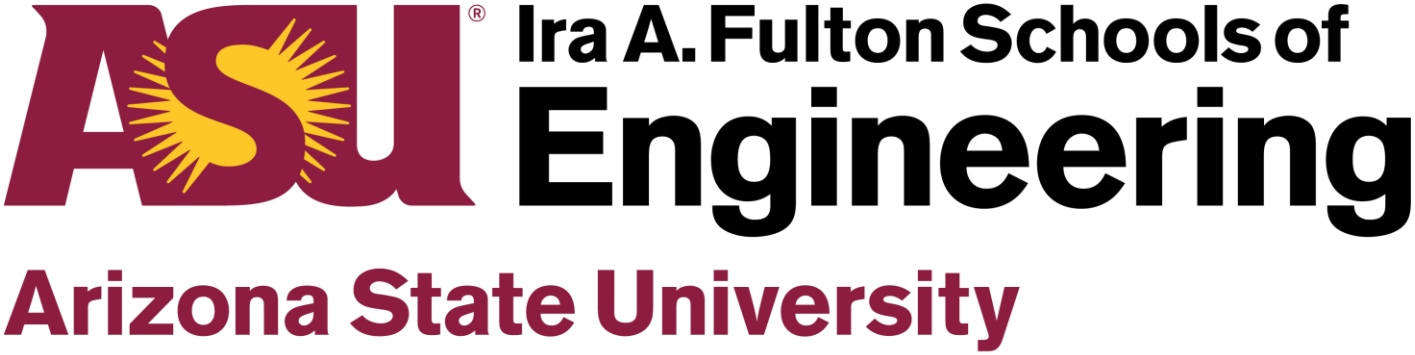


Image from Xu and Ferki, arXiv, 2021.


Ira A. Fulton Schools of
Engineering
Arizona State University