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.

\[ t_k = \max \left( r + \gamma V_{k-1}(x) \right) \]

Target data

Symbolic V-function

\[ V_k(x) = 5 \times x_2 - 3 \times x_3 + \cos(x_1) - \sin(x_2) \ldots \]

Reward function

Symbolic regression

\[ J^{\text{INV}}_k = \left[ \frac{t_k}{\text{target}} - \frac{V_k(x)}{\text{evolved}} \right]^2 \]
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**
Deep Symbolic Regression

Overview
Overview: Deep Symbolic Regression

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

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

Generate distribution of expressions with an RNN

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 $\alpha$; entropy coefficient $\lambda_H$; risk factor $\varepsilon$; batch size $N$; reward function $R$

output Best fitting expression $\tau^*$

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

Sample expressions from the RNN
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

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

\begin{description}
\item[Input] learning rate $\alpha$; entropy coefficient $\lambda_{H}$; risk factor $\varepsilon$; batch size $N$; reward function $R$
\item[Output] Best fitting expression $\tau^*$
\end{description}

\begin{algorithm}
\begin{algorithmic}[1]
\State \textbf{repeat}
\State \textbf{for} $i = 1, \ldots, N$ \textbf{do}
\State $\tau^{(i)} \sim p(\cdot | \theta)$
\State \textbf{end for}
\State Sample $N$ expressions (Alg. 2 in Appendix A)
\State $\mathcal{T} \leftarrow \{\text{OptimizeConstants}(\tau^{(i)}, R)\}_{i=1}^{N}$ \Comment{Optimize constants w.r.t. reward function}
\State $\mathcal{R} \leftarrow \{R(\tau^{(i)})\}_{i=1}^{N}$ \Comment{Compute rewards}
\State $R_{\varepsilon} \leftarrow (1 - \varepsilon)$-quantile of $\mathcal{R}$ \Comment{Compute reward threshold}
\State $\mathcal{T}' \leftarrow \{\tau^{(i)} : R(\tau^{(i)}) \geq R_{\varepsilon}\}$ \Comment{Select subset of expressions above threshold}
\State $\mathcal{R}' \leftarrow \{R(\tau^{(i)}) : R(\tau^{(i)}) \geq R_{\varepsilon}\}$ \Comment{Select corresponding subset of rewards}
\State $g_1 \leftarrow \text{ReduceMean}((\mathcal{R} - R_{\varepsilon}) \nabla_{\theta} \log p(\mathcal{T} | \theta))$ \Comment{Compute risk-seeking policy gradient}
\State $g_2 \leftarrow \text{ReduceMean}(\lambda_{H} \nabla_{\theta} H(\mathcal{T} | \theta))$ \Comment{Compute entropy gradient}
\State $\theta \leftarrow \theta + \alpha (g_1 + g_2)$ \Comment{Apply gradients}
\State \textbf{if} $\max \mathcal{R} > R(\tau^*)$ \textbf{then} $\tau^* \leftarrow \tau(\arg \max \mathcal{R})$ \Comment{Update best expression}
\State \textbf{return} $\tau^*$
\end{algorithmic}
\end{algorithm}

\begin{algorithm}
\begin{algorithmic}[1]
\State $\text{Return best expression } \tau^*$
\end{algorithmic}
\end{algorithm}

Refinement
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

<table>
<thead>
<tr>
<th>input</th>
<th>Categorical probabilities $\psi$; corresponding library of tokens $L$; partially sampled traversal $\tau$</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>Adjusted categorical probabilities $\psi$</td>
</tr>
<tr>
<td>1:</td>
<td>$L \leftarrow</td>
</tr>
<tr>
<td>2:</td>
<td>for $i = 1, \ldots, L$ do</td>
</tr>
<tr>
<td>3:</td>
<td>if ViolatesConstraint($\tau, L_i$) then $\psi_i \leftarrow 0$</td>
</tr>
<tr>
<td>4:</td>
<td>$\psi \leftarrow \frac{\psi}{\sum_i \psi_i}$</td>
</tr>
<tr>
<td>5:</td>
<td>return $\psi$</td>
</tr>
</tbody>
</table>

|            | $\triangleright$ Length of library |
|            | $\triangleright$ If the token would violate a constraint, set its probability to 0 |
|            | $\triangleright$ Normalize probability vector back to 1 |

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

<table>
<thead>
<tr>
<th>input</th>
<th>Expression $\tau$ with placeholder constants $\xi$; reward function $R$</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>Expression $\tau^<em>$ with optimized constants $\xi^</em>$</td>
</tr>
<tr>
<td>1:</td>
<td>$\xi^* \leftarrow \arg \max_\xi R(\tau; \xi)$</td>
</tr>
<tr>
<td>2:</td>
<td>$\tau^* \leftarrow ReplaceConstants(\tau, \xi^*)$</td>
</tr>
<tr>
<td>3:</td>
<td>return $\tau^*$</td>
</tr>
</tbody>
</table>

| $\triangleright$ Maximize reward w.r.t. constants, e.g. with BFGS |
| $\triangleright$ Replace placeholder constants |
Overall Pseudocode

**Algorithm 1** Deep symbolic regression with risk-seeking policy gradient

**input** learning rate $\alpha$; entropy coefficient $\lambda_\mathcal{H}$; risk factor $\varepsilon$; batch size $N$; reward function $R$

**output** Best fitting expression $\tau^*$

1. Initialize RNN with parameters $\theta$, defining distribution over expressions $p(\cdot|\theta)$
2. repeat
   3. $\mathcal{T} \leftarrow \{\tau^{(i)} \sim p(\cdot|\theta)\}_{i=1}^N$
   4. $\mathcal{R} \leftarrow \{\text{OptimizeConstants}(\mathcal{T}; R)\}_{i=1}^N$
   5. $\mathcal{R} \leftarrow \{R(\tau^{(i)})\}_{i=1}^N$  $\triangleright$ Sample $N$ expressions (Alg. 2 in Appendix A)
   6. $\bar{R}_\varepsilon \leftarrow (1 - \varepsilon)$-quantile of $\mathcal{R}$  $\triangleright$ Compute reward threshold
   7. $\mathcal{T} \leftarrow \{\tau^{(i)} : R(\tau^{(i)}) \geq \bar{R}_\varepsilon\}$  $\triangleright$ Select subset of expressions above threshold
   8. $\mathcal{R} \leftarrow \{R(\tau^{(i)}) : \tau^{(i)} \in \mathcal{T}\}$  $\triangleright$ Select corresponding subset of rewards
   9. $\hat{g}_1 \leftarrow \text{ReduceMean}((\mathcal{R} - \bar{R}_\varepsilon) \nabla_\theta \log p(\mathcal{T}|\theta))$  $\triangleright$ Compute risk-seeking policy gradient
   10. $\hat{g}_2 \leftarrow \text{ReduceMean}(\lambda_\mathcal{H} \nabla_\theta \mathcal{H}(\mathcal{T}|\theta))$  $\triangleright$ Compute entropy gradient
   11. $\theta \leftarrow \theta + \alpha(\hat{g}_1 + \hat{g}_2)$  $\triangleright$ Apply gradients
   12. if $\max \mathcal{R} > R(\tau^*)$ then $\tau^* \leftarrow \tau^{(\arg \max \mathcal{R})}$  $\triangleright$ Update best expression
13. return $\tau^*$


---

Algorithm from Petersen et al., ICLR 2021
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

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

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

- 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 $\alpha$; entropy coefficient $\lambda_H$; risk factor $\varepsilon$; batch size $N$; reward function $R$

output Best fitting expression $\tau^*$

1: Initialize RNN with parameters $\theta$, defining distribution over expressions $p(\cdot|\theta)$
2: repeat
3: $\mathcal{T} \leftarrow \left\{ \tau^{(i)} \sim p(\cdot|\theta) \right\}^N_{i=1}$
4: $\mathcal{T} \leftarrow \{ \text{OptimizeConstants}(\tau^{(i)}, R) \}^N_{i=1}$
5: $\mathcal{R} \leftarrow \{ \text{R}(\tau^{(i)}) \}^N_{i=1}$
6: $\hat{\mathcal{R}}_\varepsilon \leftarrow (1 - \varepsilon)$-quantile of $\mathcal{R}$
7: $\mathcal{T} \leftarrow \{ \tau^{(i)} : R(\tau^{(i)}) \geq \hat{\mathcal{R}}_\varepsilon \}$
8: $\mathcal{R} \leftarrow \{ R(\tau^{(i)}) : R(\tau^{(i)}) > \hat{\mathcal{R}}_\varepsilon \}$
9: $\hat{g}_1 \leftarrow \text{ReduceMean}( (\mathcal{R} - \hat{\mathcal{R}}_\varepsilon) \nabla_\theta \log p(\mathcal{T}|\theta))$
10: $\hat{g}_2 \leftarrow \text{ReduceMean}(\lambda_H \nabla_\theta H(\mathcal{T}|\theta))$
11: $\theta \leftarrow \theta + \alpha(\hat{g}_1 + \hat{g}_2)$
12: if $\max \mathcal{R} > R(\tau^*)$ then $\tau^* \leftarrow \tau^{(\arg \max \mathcal{R})}$
13: return $\tau^*$

Compute top $1-\varepsilon$ 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 $\alpha$; entropy coefficient $\lambda_\mathcal{H}$; risk factor $\varepsilon$; batch size $N$; reward function $R$

**Output** Best fitting expression $\tau^*$

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

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

$$\nabla_\theta J_{risk}(\theta; \varepsilon) \approx \frac{1}{\varepsilon N} \sum_{i=1}^{N} \left[ R(\tau^{(i)}) - \tilde{R}_{\varepsilon}(\theta) \right] \cdot 1_{R(\tau^{(i)}) \geq \tilde{R}_{\varepsilon}(\theta)} \nabla_\theta \log p(\tau^{(i)} | \theta)$$
Algorithm 1 Deep symbolic regression with risk-seeking policy gradient

**input** learning rate $\alpha$; entropy coefficient $\lambda_{H}$; risk factor $\varepsilon$; batch size $N$; reward function $R$

**output** Best fitting expression $\tau^*$

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

*Entropy gradient points adds some extra performance improvement*

*Update parameters for RNN and optimal expression*
## Experiments

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

End-to-end symbolic regression with transformers

Chris1978

Abstract
Symbolic regression is a classical problem in machine learning that is important for understanding the behavior of complex systems. In this paper, we propose a new approach to symbolic regression using transformers, a popular language model architecture. Our approach is based on end-to-end training, which allows us to learn both the symbolic representation and the underlying mathematical functions simultaneously. We show that our method achieves state-of-the-art performance on a variety of benchmark datasets.

Introduction
Symbolic regression is a classical problem in machine learning that is important for understanding the behavior of complex systems. Existing methods for symbolic regression include genetic programming, neural networks, and decision trees, but they have limitations such as difficulty in handling high-dimensional data and inability to explain the learned models. In this paper, we propose a new approach to symbolic regression using transformers, a popular language model architecture.

Results
We evaluate our method on a variety of benchmark datasets, including the well-known symbolic regression datasets and real-world datasets from different domains. Our method achieves state-of-the-art performance on all datasets, with relative errors ranging from 1% to 5%.

Conclusion
In conclusion, we have presented a new approach to symbolic regression using transformers, an end-to-end learning method that can learn both the symbolic representation and the underlying mathematical functions simultaneously. Our method achieves state-of-the-art performance on a variety of benchmark datasets and shows promise for a wide range of applications.

Acknowledgments
This work was supported by the National Science Foundation under grant number BCS-1935156.

References

Chris1978

1 Chris1978 is a fictional author.
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 $[\cos, \text{mul}, +, 2424, \text{E-3}, x]$

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

Intuition:
- State space is continuous (\( s_i \) variables in the expression)
- Actions are continuous but have \textbf{multiple dimensions} (the example on this slide has 2 dimensions)
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.

$$R(\tau) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T(i)} r^{(i)}_t$$

- Number of episodes
- Time steps in episode $i$
- Reward for time $t$ in episode $i$
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 $1,...,i-1$ and the anchor (non-symbolic NN-learned) actions for $i+1,...,n$. 
Adding Multiple Dimensions

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

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

Input: Learning rate $\alpha$, risk factor $\epsilon$, entropy weight $\eta$, entropy decay $\gamma$, environment $E$ with observation space $S \subset \mathbb{R}^m$ and action space $A \subset \mathbb{R}^n$, pre-trained “anchor” policy $\Psi$ (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

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

$\tau^* \leftarrow \text{null}$  // 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

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

\[ \mathcal{T}_e = \{\tau^{(i)} : R(\tau) \geq R_e\} \]  // Select subset of expressions above threshold

\[ \hat{g}_1 = \frac{1}{|\mathcal{T}_e|} \sum_{\tau \in \mathcal{T}_e} ((R(\tau) - R_e) \nabla_{\theta} \log p(\tau | \theta)) \]  // Risk-seeking policy gradient

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

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

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

\[ \tilde{f}_i \leftarrow \text{Instantiate}(\tau^*) \]  // Set fixed sub-policy for next action dimension

end

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

return $f$

Note this follows the same steps as DSR.
Adding Multiple Dimensions

Algorithm 2 Policy Evaluator, used to compute reward for symbolic policy $\tau$

Function PolicyEvaluator($\tau, \bar{f}_1, \ldots, \bar{f}_{i-1}, \Psi, \mathcal{E}, N$)

Input: Symbolic policy being evaluated $\tau$, previously learned fixed symbolic expressions $\bar{f}_1, \ldots, \bar{f}_{i-1}$, pre-trained “anchor” policy $\Psi$, environment $\mathcal{E}$, number of training episodes $N$

Output: Reward $R$ for symbolic policy $\tau$

$R \leftarrow 0$

$f \leftarrow$ Instantiate($\tau$)

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

$R \leftarrow \frac{R}{N}$

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.
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

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

Table from Landajuela et al, ICML 2021
Comparison to Baselines

Table from Landajuela et al, ICML 2021

<table>
<thead>
<tr>
<th>Environment</th>
<th>DSP</th>
<th>DSP₀</th>
<th>Regression</th>
<th>DDPG</th>
<th>TRPO</th>
<th>A2C</th>
<th>PPO</th>
<th>ACKTR</th>
<th>SAC</th>
<th>TD3</th>
<th>Zoo</th>
</tr>
</thead>
<tbody>
<tr>
<td>CartPole</td>
<td>999.59</td>
<td>1000</td>
<td>211.82</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>993.94</td>
<td>1000</td>
<td>971.78</td>
<td>997.98</td>
<td>994.81</td>
</tr>
<tr>
<td>MountainCar</td>
<td>99.09</td>
<td>99.11</td>
<td>95.16 †</td>
<td>91.77</td>
<td>93.95</td>
<td>93.63</td>
<td>92.56</td>
<td>93.77</td>
<td>90.40</td>
<td>93.93</td>
<td>92.86</td>
</tr>
<tr>
<td>Pendulum</td>
<td>-160.5</td>
<td>-155.4</td>
<td>-1206.9</td>
<td>-169.0</td>
<td>-147.6</td>
<td>-162.2</td>
<td>-154.8</td>
<td>-211.2</td>
<td>-159.3</td>
<td>-147.1</td>
<td>-164.4</td>
</tr>
<tr>
<td>InvDoublePend</td>
<td>9148.2</td>
<td>9149.9</td>
<td>637.2</td>
<td>8855.1</td>
<td>8854.8</td>
<td>8951.9</td>
<td>9225.5</td>
<td>7554.1</td>
<td>9313.8</td>
<td>9357.8</td>
<td>8873.3</td>
</tr>
<tr>
<td>InvPendSwingup</td>
<td>891.84</td>
<td>891.90</td>
<td>-19.21</td>
<td>891.34</td>
<td>892.51</td>
<td>67.52</td>
<td>853.38</td>
<td>890.34</td>
<td>891.47</td>
<td>889.33</td>
<td>767.98</td>
</tr>
<tr>
<td>LunarLander</td>
<td>251.66</td>
<td>261.36</td>
<td>56.08</td>
<td>246.24</td>
<td>168.79</td>
<td>227.08</td>
<td>225.12</td>
<td>245.39</td>
<td>272.65</td>
<td>225.35</td>
<td>230.93</td>
</tr>
<tr>
<td>Hopper</td>
<td>2090.2</td>
<td>2122.4</td>
<td>47.35</td>
<td>1632.7</td>
<td>2583.4</td>
<td>1925.1</td>
<td>2439.7</td>
<td>2456.7</td>
<td>2455.0</td>
<td>2741.9</td>
<td>2319.2</td>
</tr>
<tr>
<td>BipedalWalker</td>
<td>264.39</td>
<td>311.78</td>
<td>-110.77</td>
<td>94.21</td>
<td>311.08</td>
<td>241.01</td>
<td>286.20</td>
<td>299.32</td>
<td>307.26</td>
<td>310.19</td>
<td>264.18</td>
</tr>
</tbody>
</table>

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 $\bar{R}_{ep}$| 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 DSP₀ = DSP with constant optimization)
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 works to obtain the goal

Image from Xu and Ferki, arXiv, 2021.