Deep symbolic regression

Hongsup Oh (u1314063), Nolan Strauss (u1079968)

December 2022

1 Introduction

Traditional deep learning has demonstrated the ability to map features to labels effectively. Unfortunately, deep learning and other black-box machine learning approaches lack interpretability, which is critical to their applications in fields such as mechanical engineering. Genetic programming based symbolic regression (GPSR), an interpretable machine learning algorithm, has been suggested as an alternative approach. While GPSR has been shown to perform well on many test cases, it frequently prefers models that overfit to a given dataset and takes more time to obtain the solution. Additionally, GPSR may struggle to find a consistent solution across multiple runs due to the stochastic nature of the evolutionary algorithm. Recent work has been done towards combining the advantages of deep learning and GPSR to offset their shortcoming. In this project, we aim to mimic one of the best research papers in this field by recreating deep symbolic regression (DSR) [2]. The network architecture of DSR is a recurrent neural network (RNN) of which the classic implementation is used to compute the probability of the best word at a specific position. As the same approach of classic RNN, the RNN architecture in DSR calculates the probability of the best symbolic operator at the specific position. The expressions in DSR can be x (variable), c (constant), $+$, $-$, $\times$, $\div$, log, etc. In this work, we test the performance of DSR against five separate test cases in hopes that the best symbolic equation discovered by DSR is always the right answer. The work in this report is organized as follows. In Section 2, we explain how to design DSR and detail background knowledge. In Section 3, we demonstrate the performance of the developed method. We conclude in Section 4 with a work summary.

2 Recurrent neural network (RNN)

In GPSR, a symbolic expression is represented by an acyclic graph. The acyclic graph is composed of binary nodes (i.e.$+,-,\times,\div$) or unary nodes (i.e. sin,cos,exp). In this project, we only consider the binary nodes. Figure 1 is an example of an acyclic graph representing $c_0 * X + c_1$. In GPSR, a node in the acyclic graph is randomly selected based on uniform distribution. GPSR frequently overfits the given dataset because of its randomness. We try to solve this problem by

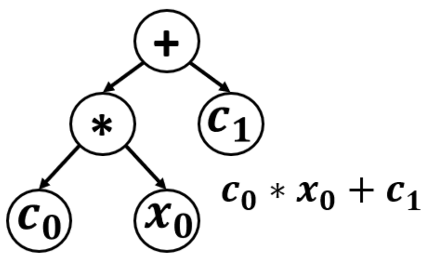


Figure 1: An example of acyclic graph

using a recurrent neural network (RNN) to build the acyclic graph. The RNN is used to sample the symbolic operator in the sequence of preorder transversal. According to the preorder traversal, the example acyclic graph in Figure 1 is expressed as $\tau = [+, \times, c, X, c]$. Figure 2 shows how RNN samples symbolic

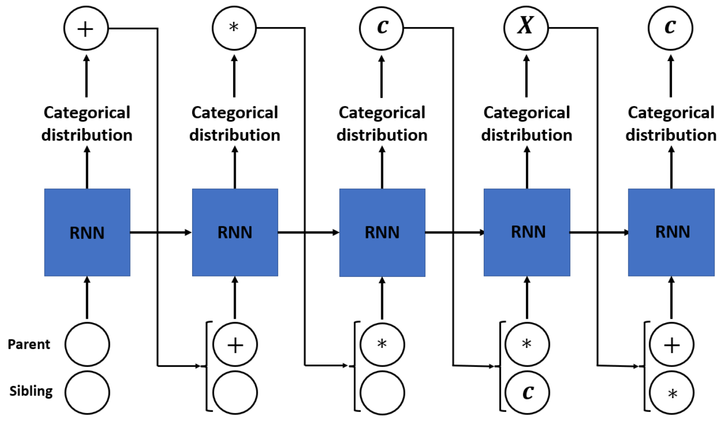


Figure 2: The framework of RNN

operators of the acyclic graph in Figure 1. At each RNN cell, the input data is both the parent and sibling nodes of the current node. Then, the categorical distribution returned by the RNN is created using the softmax function. The current node is sampled from this categorical distribution. This process can be mathematically represented by

$$\tau_i \sim p(\cdot|\tau_s, \tau_p, \theta), \tag{1}$$

where, $p(\cdot|\tau_s, \tau_p, \theta)$ is the categorical distribution and τ_i, τ_s, and τ_p are the current, sibling, and parent nodes. Each sampled operator is stored as the sequence of the preorder traversal. Our goal is to discover the symbolic expression having the smallest error with the numerical solution. To measure the error, we use the reward function:

$$r(y, \tilde{y}) = \frac{1}{1 + NRMSE(y, \tilde{y})}, \tag{2}$$

where $NRMSE(*)$ is the normalized root means square error (NRMSE) which is defined by the following equation,

$$NRMSE(y, \tilde{y}) = \frac{1}{\sigma_y} \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \tilde{y}_i)^2}, \tag{3}$$

where, y is the solution, $\tilde{y}$ is the output of the obtained equation, and σ_y is the standard deviation of y. As $\tilde{y}$ approaches y, the reward converges to 1. As we obtain the symbolic equation, we need to optimize the *const* operator in the equation by local optimization. Local optimization simply finds the numerical value of *const* maximizing the reward function, Equation 2:

$$c = \arg\max_{c} \left\{ r(y, \tilde{y}(c)) \right\}, \tag{4}$$

where c is a representation of the model constants in an array. For example, $c = [c_0, c_1]$ in Figure 1. With this, we will want to find the equation having the maximum reward value with the solution. Unfortunately, we cannot directly optimize the reward value between the discovered equation and the solution because the parameters of RNN (θ) are related to the categorical distribution in Equation 1. To solve this problem, we use the standard policy gradient. The standard policy is represented by

$$J(\theta) = E_{\tau \sim p(\tau|\theta)}[r(\tau)]. \tag{5}$$

Then, the gradient of the standard policy is expressed by

$$\nabla_\theta J(\theta) = \ E_{\tau \sim p(\tau|\theta)}[r(\tau) \nabla_\theta \ log(p(\tau|\theta)]. \tag{6}$$

Equation 6 can be approximated by

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \ r(\tau) \nabla_\theta \ log(p(\tau|\theta). \tag{7}$$

With this, we can find the optimized parameters of RNN using the standard policy gradient method. Additionally, this is a gradient ascent problem since we want to obtain the weight parameters maximizing Equation 2. To accomplish this we use the Adam algorithm in Pytorch [1]. The pseudo code of DSR is shown in Algorithm 1, where $\{X,y\}$ are the feature and label data, $\{W_{embed}, W_x, b, W_h, W_{affine}, b_{affine}\}$ are weight parameters of RNN, and T is the maximum limit of iteration. The main code is SymbolicRNN.py among submitted codes, and you can implement DSR using main.ipynb. We use $OperatorEmbedding(*)$, $rnn(*)$, and $affineLayer(*)$ from rnn.ipny of Homework 4. In addition to these codes we created $Agraph(*)$, $binaryNode(*)$, $reward(*)$, $NRMSE(*)$, and $localOptimizer(*)$.

Algorithm 1 Deep symbolic regression (DSR)

```
    Input {$X$,$y$},{$W_{embed}$,$W_x$,$b$,$W_h$,$W_{affine}$,$b_{affine}$},T
    Output $loss$, $r$
1:  $x_0 \leftarrow [0, 0]$                                  ▷ Initial set of parent and sibling nodes
2:  $h \in \mathbb{R}^{2\times H}$                           ▷ Initialize zero tensor $h$
3:  $\tau \leftarrow []$                                     ▷ Initialize empty traversal
4:  for $t = 1 : T$ do
5:      $x \leftarrow OperatorEmbedding(x_0, W_{embed})$     ▷ Embed $x_0$
6:      $h \leftarrow rnn(x, h, W_x, b, W_h)$                ▷ Find next $h$ using step RNN
7:      $z \leftarrow affineLayer(h, W_{affine}, b_{affine})$   ▷ Transform the vector space
8:      $p(\tau_i|\tau_s, \tau_p, \theta) \leftarrow SoftMax(z)$   ▷ Calculate categorical distribution
9:      $\tau_i \sim p(\tau_i|\tau_s, \tau_p, \theta)$       ▷ Sampling current node
10:     $\tau \leftarrow \tau||\tau_i$                       ▷ Append current node to traversal
11:     if $\tau_i$ is leaf node then
12:         $x_0 \leftarrow [\tau_i, 0]$                     ▷ Next set of parent and sibling nodes
13:     else
14:         $x_0 \leftarrow [\tau_p, \tau_i]$                ▷ Next set of parent and sibling nodes
15:     end if
16: end for
17: $c \leftarrow localOptimization(y, \tau)$               ▷ Find the optimized constants
18: $\tilde{y} \leftarrow evaluateAgraph(X, \tau, c)$       ▷ Evaluate the discovered symbolic equation
19: $r \leftarrow reward(y, \tilde{y})$                     ▷ Calculate reward
20: $loss \leftarrow policyGradient(y, \tilde{y})$          ▷ Calculate loss
```

3 Results

3.1 Problem introduction

To test our implementation of DSR we selected the five tests shown in Table 1. Each test is a single model that is used to generate data. With each generated data set we train DSR to find the most likely equation to fit the data and compare it to the true solution to determine if DSR was able to find the solution.

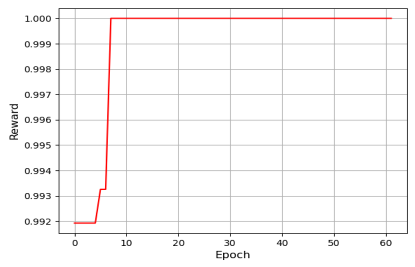
Table 1: **Test equations**

	Target analytical solution
Test 1	$1.452 \times X + 0.012 \times X$
Test 2	$-1.2 \times X^2 + 0.823 \times X + 2.1$
Test 3	$3.2 \times X^3 - 1.2 \times X^2 + 0.823 \times X + 2.1$
Test 4	$0.05 \times X^4 - 0.125 \times X^3 - 1.11 \times X^2 + 0.7053 \times X + 4.2$
Test 5	$2.01 \times X^4 + 3.2 \times X^2 + 4 \times X$

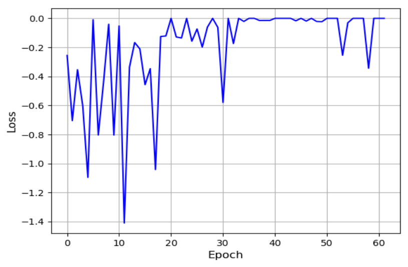
Tests are terminated when the reward, $r(y, \tilde{y})$, is larger than 0.99 and the probabilities of sampled nodes are larger than 95%, or if DSR reaches the maximum iteration limit.

3.2 Test 1

Test 1 represents the most simple test case for DSR. For this test, DSR is tasked with finding a simple linear equation, and, as expected, DSR finds the exact solution within 100 epochs. Through the first 100 epochs, trends can be seen in plots of the reward and loss vs. epochs. These plots can be seen below in Figure 3.



(a) Test 1: Best reward



(b) Test 1: Loss

Figure 3: Test 1: Best reward and Loss vs. epochs. These plots show the trend of DSR to find better models w.r.t. epochs while still continually exploring the model space after models with a reward of 1.0 are found. Variations in the loss plot are due to the stochastic nature of sampling models through epochs wherein the model is not comprised of only the most likely nodes but of nodes that are sampled from each pass through the RNN. This is trend is consistent across all subsequent tests.

After training DSR the best equation returned for test 1 in its unchanged form is

$$y = \left(\frac{X}{c_0}\right) \times c_1 + c_2 \tag{8}$$

where c_0, c_1, and c_2 are unknown constants that were fit by local optimization. After fitting the parameters and performing simplification, the equation returned was

$$y = 1.452 \times X + 0.01199 \tag{9}$$

of which is essentially the true solution. The only reason for minor discrepancies in the parameters is due to the local optimization. This simplification was performed for all tasks. As was discussed in previous sections, operators can be sampled from the posterior at each nodal value and can then be used to form an equation. An example of a sampled model along with the nodal probabilities can be seen in Table 2., where the operator for each node is displayed along with its associated probability of being sampled at that node given the previous selections. Upon sampling models from the estimated posterior, the nodes to develop the proposed solution obtain probabilities above 95%, showing that DSR is extremely confident in the proposed solution. This is shown here for test 1 but will not be replicated for subsequent tests due as it will not be beneficial to recreate. Finally, upon sampling 100 separate models from the posterior, these models simplify to the true solution **100%** of the time.

Table 2: $y = 1.452 \times X + 0.012$.

	Operator	**Probability**
Node 1	$-$	99.06%
Node 2	$-$	98.71%
Node 3	$-$	99.84%
Node 4	$\div$	96.61%
Node 5	$\times$	99.75%
Node 6	c	99.97%
Node 7	x	100.0%
Node 8	c	99.99%
Node 9	c	99.31%
Node 10	c	99.99%
Node 11	c	97.1%

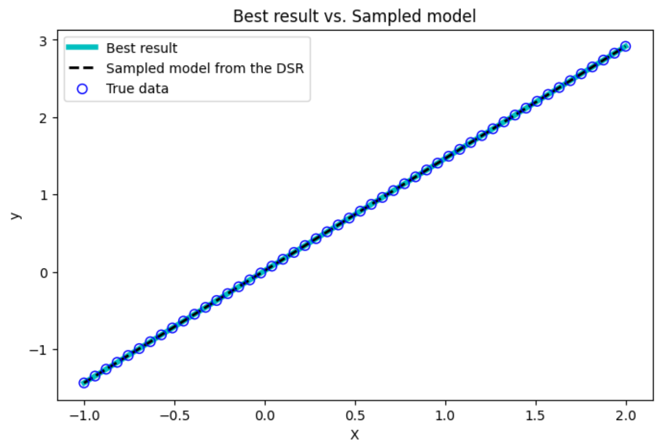
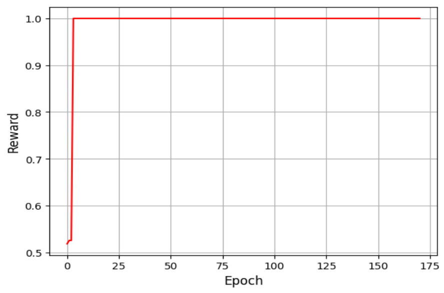


Figure 4: Plot of the best model from training DSR vs. sampled model from posterior estimated by DSR along with data used for training.

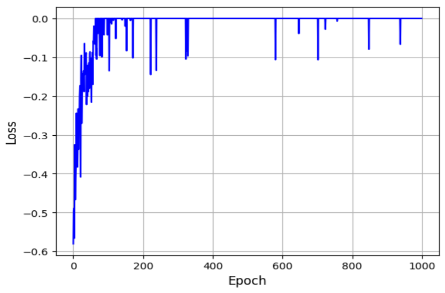
3.3 Test 2

Test 2 represents a slightly more complex test case for DSR in contrast to Test 1. For this test, DSR is tasked with finding the equation $-1.2 \times X^2 + 0.823 \times X + 2.1$. For Test 2, DSR finds the exact solution within 1000 epochs. Through the first 1000 epochs, trends can be seen in plots of the reward and loss vs. epochs.

These plots can be seen below in Figure 5. Note, trends are seen in these plots that match those seen in the plots from Test 1, specifically in the apparent randomness of loss values at later epochs. Upon sampling 100 separate models from the posterior, these models simplify to the true solution **99%** of the time.



(a) Test 1: Best reward



(b) Test 1: Loss

Figure 5: Test 2: Best reward and Loss vs. epochs.

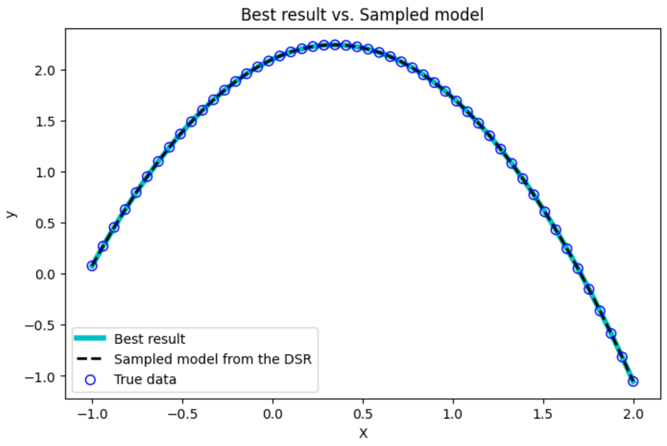
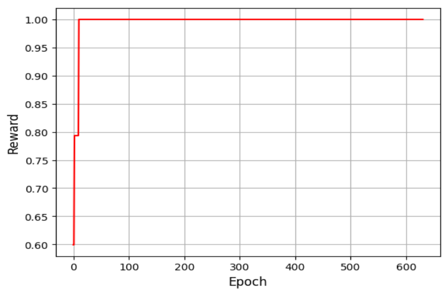


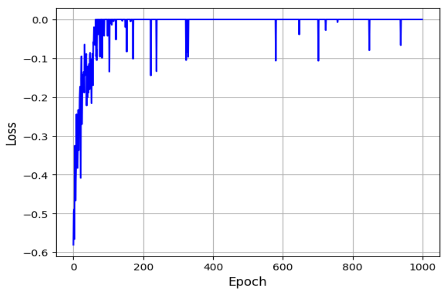
Figure 6: Plot of the best model from training DSR vs. sampled model from posterior estimated by DSR along with data used for training.

3.4 Test 3

Test 3 challenged DSR to find the equation $3.2 \times X^3 - 1.2 \times X^2 + 0.823 \times X + 2.1$, a slightly altered version of the equation from Test 2 that now includes the term $3.2 \times X^3$. For Test 3, DSR finds the exact solution within 1000 epochs. Through the first 1000 epochs, trends can be seen in the plots of the reward and loss vs. epochs that closely match the results of Test 1 and Test 2. Upon sampling 100 separate models from the posterior, these models simplify to the true solution **96%** of the time.



(a) Test 1: Best reward



(b) Test 1: Loss

Figure 7: Test 1: Best reward and Loss vs. epochs.

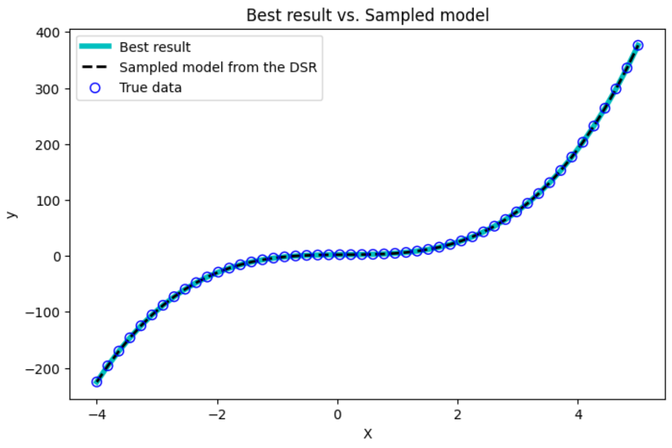
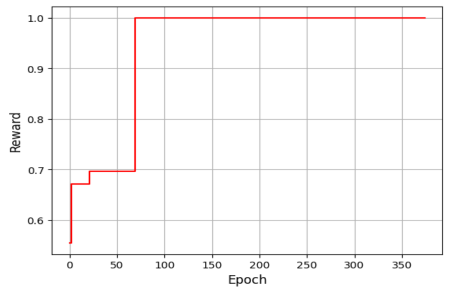


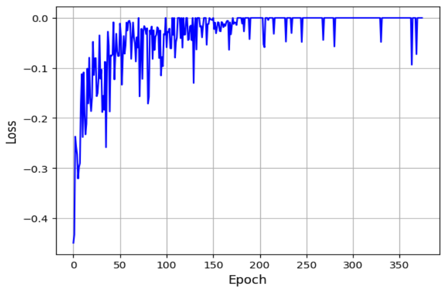
Figure 8: Plot of best model from training DSR vs. sampled model from posterior estimated by DSR along with data used for training.

3.5 Test 4

Test 4 challenged DSR to find the equation $0.05 \times X^4 - 0.125 \times X^3 - 1.11 \times X^2 + 0.7053 \times X + 4.2$.For Test 4, DSR finds the exact solution within 1000 epochs. The plots of reward vs. epochs and loss vs. epochs show the first test where DSR struggles to converge towards the solution within the first 100 epochs and shows the aggressive tendency of DSR to test out more different models. Upon sampling 100 separate models from the posterior, these models simplify to the true solution **100%** of the time.



(a) Test 4: Best reward



(b) Test 4: Loss

Figure 9: Test 4: Best reward and Loss vs. epochs.

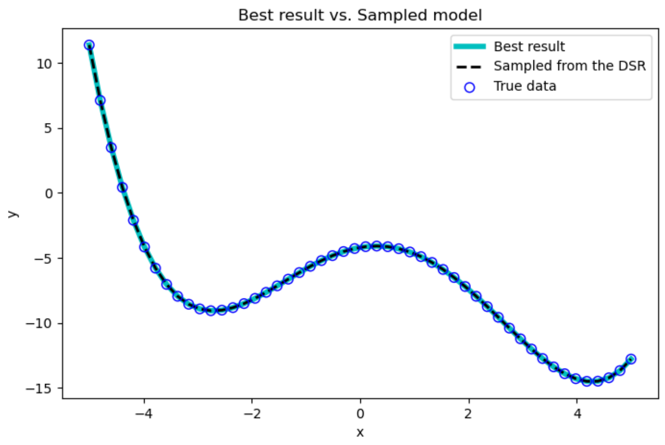
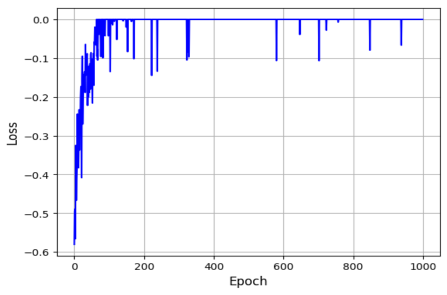


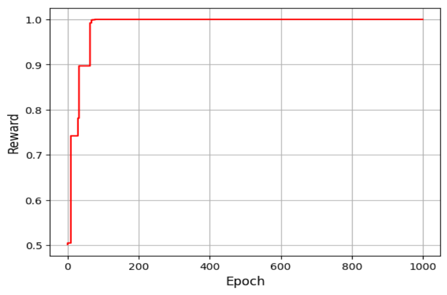
Figure 10: Plot of best model from training DSR vs. sampled model from posterior estimated by DSR along with data used for training.

3.6 Test 5

Test 5 delivered the most difficult challenge for DSR to find the equation $2.01 \times X^4 + 4.2 \times X^2 + 4.0 \times X$. In Test 5, DSR finds the exact solution within 1000 epochs. However, upon sampling 100 separate models from the posterior, we only found the true model or a model that would simplify to the true model 3 times, i.e. 3%. While this is quite troublesome, the models selected form the posterior are numerically identical to the true solution. To further expand upon this, while the true model is found during the training of DSR, models sampled from the estimated posterior space for model nodes tend to not match the true solution, but these sampled models do numerically match the output of the true model. To show this, the following plot shows the data from the best model which happens to be the true solution in cyan and the most commonly sampled model from the estimated posterior distribution returned by DSR. This sampled model is $2.011 \times X^4 + 0.0097 \times X^3 + 3.012 \times X + 3.27 \times X + 4.09$.



(a) Test 5: Best reward



(b) Test 5: Loss

Figure 11: Test 4: Best reward and Loss vs. epochs. These plots show the trend of DSR to find better models w.r.t. epochs while still continually exploring the model space after models with a reward of 1.0 are found.

In the following plot we can view that while the models do not match each other in form, they do match in output over the given domain of input data. It is believed that DSR prefers to select this alternate model as it obtains more expressivity than the true model due to it including additional terms.

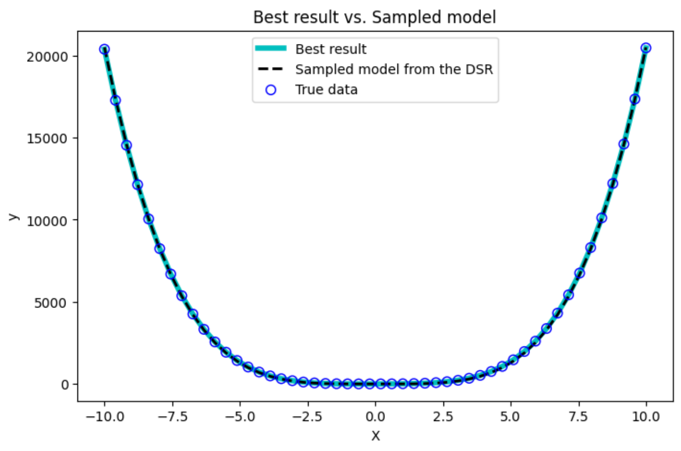


Figure 12: Plot of best model from training DSR vs. sampled model from posterior estimated by DSR along with data used for training.

4 Conclusion

To conclude, DSR is a very promising alternative to GPSR and other white-box machine learning algorithms. In four of our five proposed cases DSR is able to both find the true solution during training, where the model is stored as the best model, as well as estimate posterior distributions that allow for the user to sample the correct model with a probability over 95%. While the final test, Test 5, fails at performing both of these tasks, it is capable of finding the true solution while training, though it highlights a potential issue with DSR. This issue is with the consistent nature of DSR to prefer more expressive models to simpler models by filling in terms that may not be present in the true solution. Additionally, this issue shines light on the heavy dependence of the selection of hyperparameters used in DSR as one can assume that through subtle tweaking the results of DSR may become better.

References

[1] Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Jonathan Reynolds, Alexander Melnikov, Natalia Lunova, and Orion Reblitz-Richardson. Pytorch captum. https://github.com/pytorch/captum, 2019.

[2] Brenden K Petersen, Mikel Landajuela Larma, T Nathan Mundhenk, Claudio P Santiago, Soo K Kim, and Joanne T Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. *arXiv preprint arXiv:1912.04871*, 2019.