HSHSP Research Proposal

**Characterizing Sharpness in Genetic Programming**

**Razan Abdelrahman**

**Mentor: Nathan Haut**

**Department of Computational Mathematics, Science, and Engineering**

**Michigan State University**

**Introduction**

Sharpness aware minimization (SAM) is a methodology used to enhance the stability of deep neural network models by focusing on how sharp the error is [2]. Training while using SAM functions decreases fitness sharpness which improves model stability, which then improves the generalization ability of the models. SAM has recently been adapted to work with genetic programming (GP) [2]. By using SAM in genetic programming, we are able to find which solution best fits *and* is stable. Although, GP tends to find simpler equations as solutions, which can lead to stability, when there isn’t much data, there can be gaps between the data points which can make the response unstable when interpolating into these spaces. We can use sharpness metrics to identify models that make smoother predictions. Prior to using sharpness in GP, several different methods have been used to develop smoother, more stable models: order of nonlinearity [3], model curvature [4], random sampling technique (RST) [5], RelaxGP [6], overfit repulsors [7], and SAM [2]. Order of non-linearity and model curvature explore the intrinsic characteristics of the models to measure how likely they are to be overfitting. Random sampling techniques lessen overfitting by using multiple data subsets, which reduces exposure to the entire dataset at once. RelaxGP introduces a strategy to relax the model and penalize it for excessively close fits to response data; and overfit repulsors keep track of the models that had mistakes and construct new models different from the mistake prone ones [2]. This study investigates the sharpness metric from [2] to characterize the behavior when used to analyze models of multiple and different operators across different data ranges to further intuition into how the sharpness metric works.

**Materials and Methods**

In the study conducted, the authors utilized the StackGP system, accessible on GitHub [1], to investigate the sharpness of various different operators. Data ranges of [-1,1], [1,1000], and [100,105] were used to explore variations in operator sharpness across different data conditions. The [-1,1] range tested values near zero, [1,1000] covered a wider range of values, and [100,105] examined sharpness with large values within a narrow range. By incorporating these diverse data ranges, the authors gathered multiple results that were subsequently compared and depicted using bar graphs.

To assess operator sharpness, the study analyzed the sharpness of both single-layer and double-layer models. Single-layer evaluated the performance of individual operators, while double-layer examined how the sharpness of operators changed when paired with other operators.

Additionally, variations in percent and number of perturbations were adjusted to assess how sensitive the operators’ sharpness was to these parameters. Percent perturbation is the percent of the original value that is being changed. Changing the percent perturbation allows for managing how much noise is being introduced to the input data. Changing the number of perturbations value allows you to specify how many times the input values are changed. Being able to set an exact amount of perturbation can allow us to set a good boundary between accuracy and efficiency. How much you change the number value can impact the stability and accuracy of the sharpness estimate, the higher the value the more accurate the estimate at the expense of additional computation time. Number of perturbations included values of 5, 10, 50, and 100, while percent perturbations ranging from 0% to 100% were tested across all operators.

The study also delved into the sharpness characteristics of binary operators, namely *addition, subtraction, multiplication,* and *division.* As well, the authors explored the sharpness profiles across two layers within the three distinct ranges: [-1,1], [1,1000], and [100,105]. Each operator was evaluated when paired with various functions, and their respective sharpness measures were visually represented through bar graphs.

**Results**

**Figure 1:**

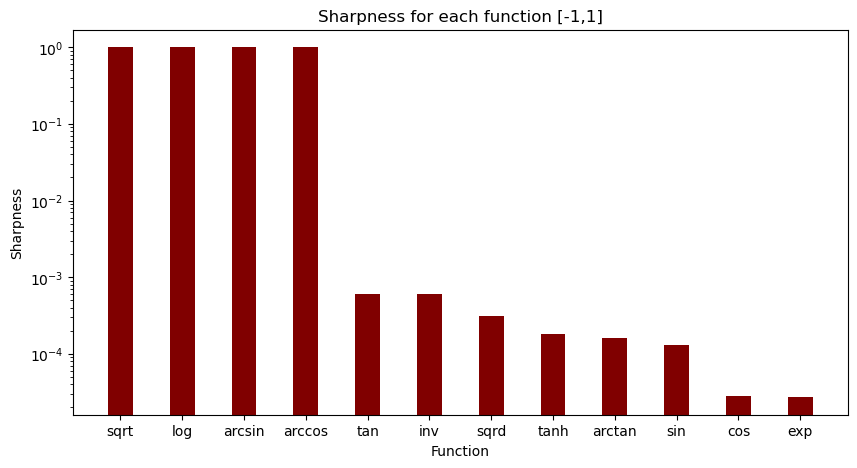
****

Figure 1 shows a bar graph representing the sharpness for the data range [-1,1]. It indicates that the square root, arcsin, arccos, and logarithmic operators are all nan (not a number) values. Square root and logarithmic operators give nan values because they can only function for positive real numbers. Arcsin and arccos are nan because they can only function within a specific range of [-1,1]. In this data range we observe that cos and exp are the most stable operators.

**Figure 1.1**

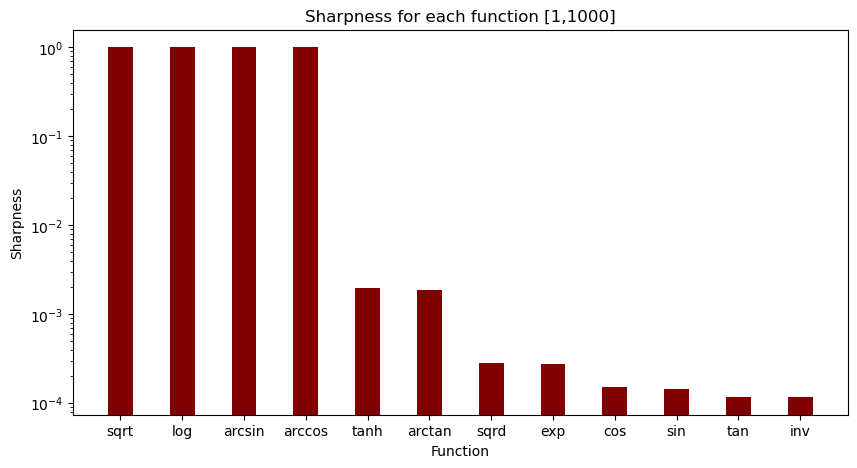
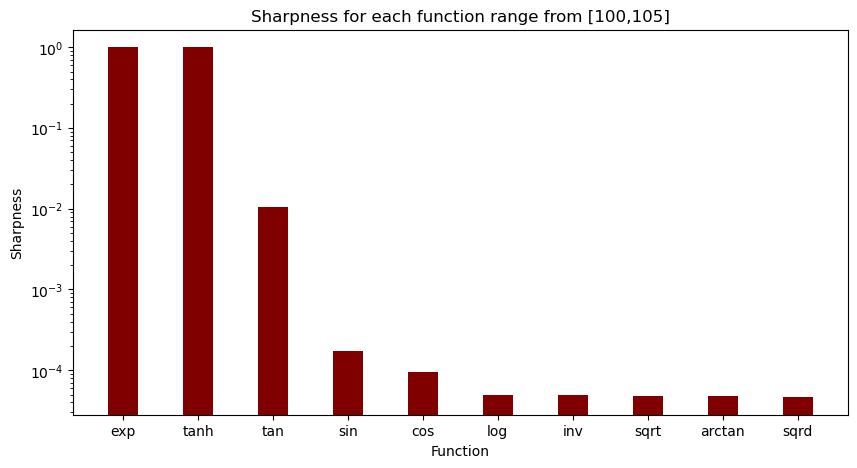
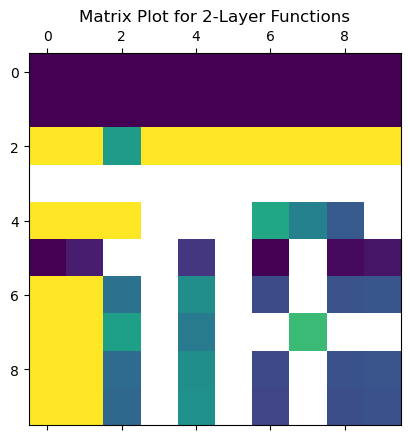
****

Figure 1.1 shows the sharpness across the data range [1,1000]. Similar to the sharpness behavior for the data range [-1,1], the square root and logarithmic operators both give nan values because they are unable to run with negative numbers. However, for this data range, the trigonometric operators (sin, cos, tan) have greater stability than the other operators, along with the inv function.

**Figure 1.2**

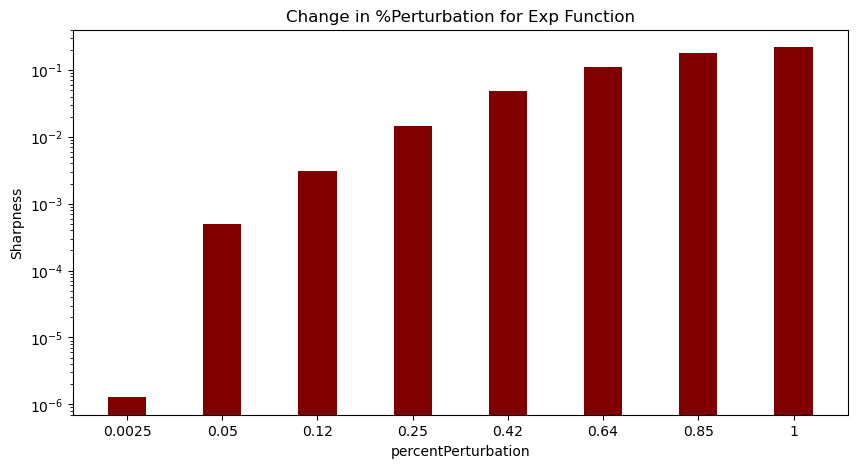
****Figure 1.2 bar graph illustrates the sharpness across the range [100,105]. Both the exponent and hyperbolic tangent operators diverge to infinity, which is why they are assigned the worst sharpness value of 1. In contrast, the trigonometric functions (tan, cos, sin) exhibit relatively stable behavior with low sharpness values, while all other operators also demonstrate stable and low sharpness characteristics.

| 1st Layer Function | sqrt | log | exp | sqrd | inv | cos | sin | tan | arctan | tanh |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| sqrt | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| log | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| exp | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| sqrd | 0.000029706 | 3.63161E-06 | 0.000046892 | 0.0000789 | 2.133E-06 | 6.861E-05 | 2.2975E-05 | 0.00006859 | 0.00001965 | 0.000018798 |
| inv | 1 | 1 | 1 | 2.1398E-06 | 7.71814E-05 | 3.1209E-05 | 0.0070232 | 0.001475 | 0.0007267 | 0.000475527 |
| cos | 0.0000318 | 0.000037243 | 1.9281E-05 | 1.8763E-05 | 5.3882E-05 | 0.000020363 | 3.175E-05 | 0.000016431 | 0.00033151 | 3.51718E-05 |
| sin | 1 | 1 | 0.0005032 | 1.8601E-05 | 0.0018892 | 0.00001742 | 0.0001905 | 0.00011725 | 0.0002182 | 0.0002279 |
| tan | 1 | 1 | 0.0046835 | 0.0000785 | 0.0012163 | 0.000057162 | 0.00008354 | 0.01914 | 0.00007503 | 0.000094817 |
| arctan | 1 | 1 | 0.00044311 | 0.0000177 | 0.0019235 | 0.00001656 | 0.000217 | 0.0000761 | 0.00024177 | 0.000256 |
| tanh | 1 | 1 | 0.00041541 | 0.0000157 | 0.002065 | 0.0000151 | 0.0002289 | 8.183E-05 | 0.000250766 | 0.000261726 |



This matrix plot visualizes the sharpness interactions across 2-layers from the table above, comparing each operator with itself and with others. Dark purple squares highlight operators that were initially and consistently broken when paired with others. Yellow squares indicate operators that became broken upon pairing. Green squares denote operators that increased in sharpness, while white squares represent those that decreased in sharpness.

**Figure 2:**

Figure 2 represents the changes in percent perturbation within the exponent operator. Percent perturbation is the percent of the original value that is being changed. As you increase the percent perturbation, you are increasing the change from the original input data, which increases sharpness.

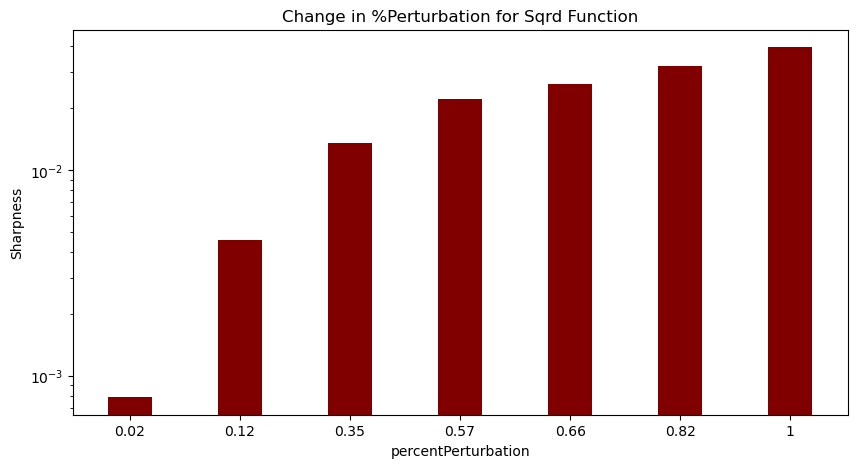
**Figure 2.2:**  

Figure 2.2 represents the different percent perturbations for the squared function. As you increase percent perturbation, you increase the amount of noise present, which increases sharpness of the analysis. This occurs for all of the operators.

**Figure 2.3**

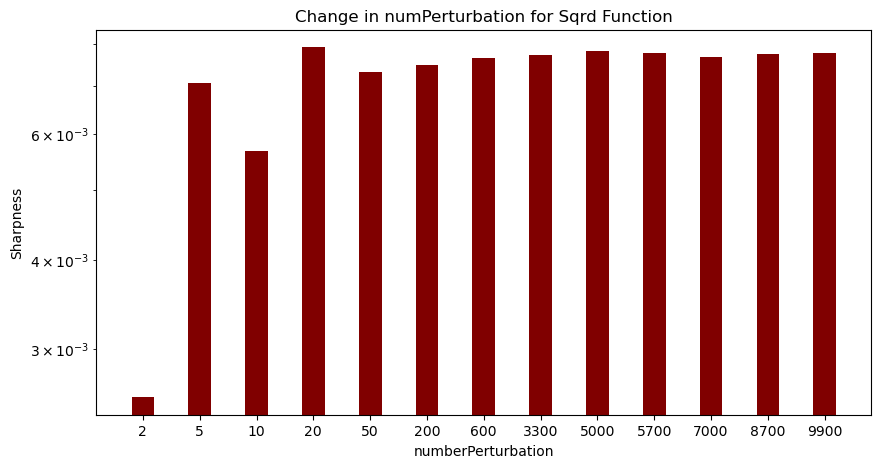


Figure 2.3 shows the different number perturbations for the squared operator. It follows an oscillating pattern within the first few numbers. When making large changes to the input values, the sharpness increases then becomes stable because there are a lot of samples, so the estimate becomes closer to the true sharpness.

**Figure 3**

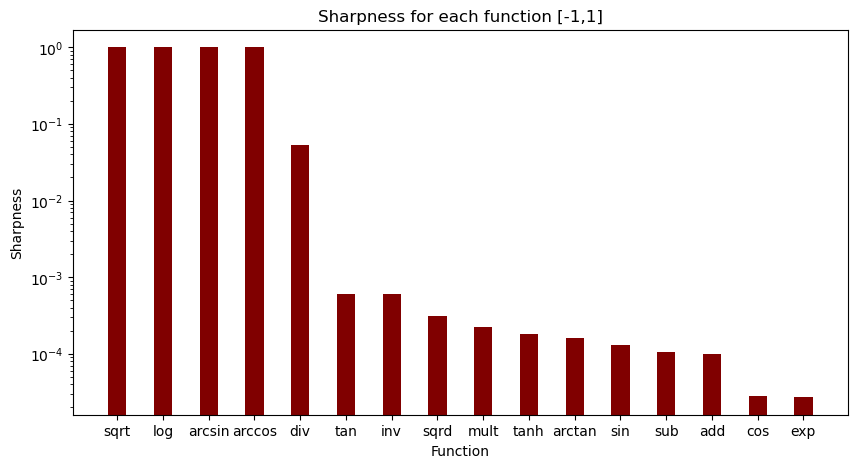


Figure 3 shows the sharpness for each function within a range of [-1,1] along with the binary operators (addition, subtraction, multiplication, division). The division operator has a relatively high sharpness level because in the narrow range of [-1,1], the outcome being NaN is very probable because the denominator has a high possibility of being 0. Multiplication is relatively stable and subtraction and addition are stable probably because of their simplicity.

**Figure 3.1**

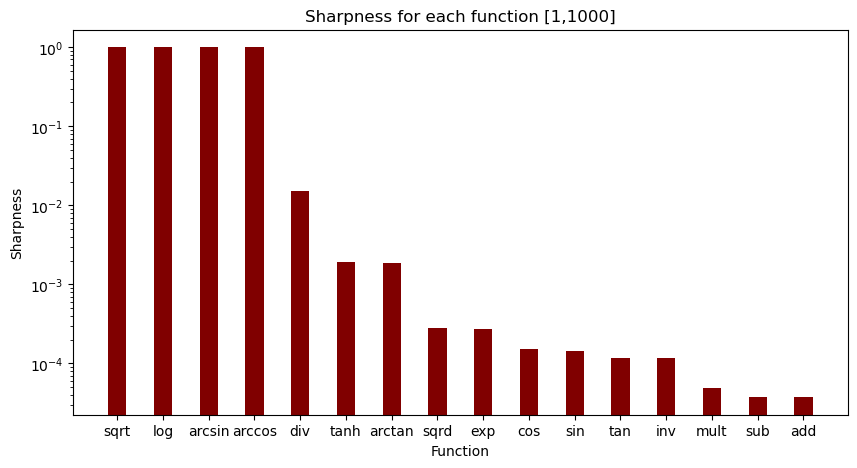


Figure 3.1 shows the sharpness for both unary and binary operators within the range of [1,1000]. Like figure 2, the division operator’s sharpness level is high and unstable, and the other operators have low sharpnesses and are stable.

**Figure 3.2**

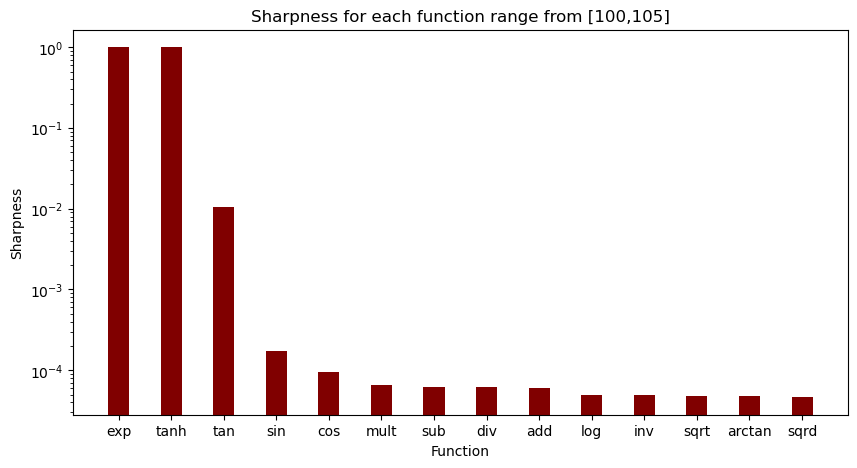


Figure 3.2 illustrates the sharpness for both unary and binary operators across the range [100,105]. Unlike the two other ranges, the [100,105] range depicted the division operator on the low end of the sharpness level. This is likely due to the range being so far from 0, so it probably won’t break (nan). However, now all of the operators seem to be relatively stable with around the same sharpness levels.

**Figure 4**

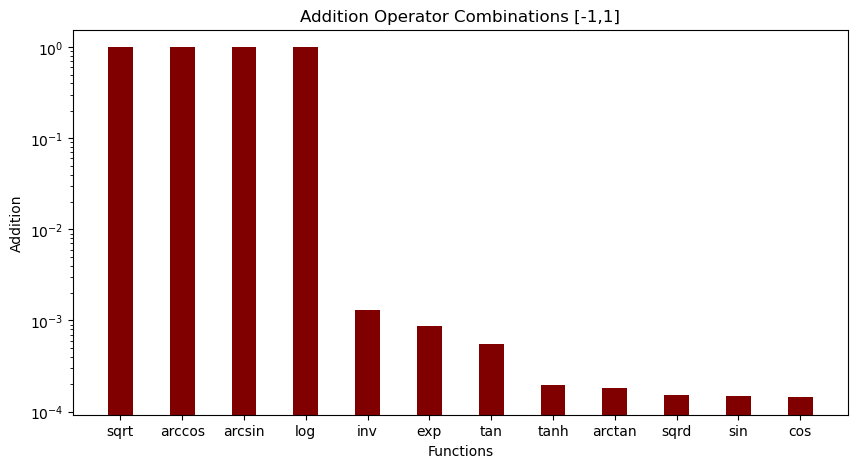


Figure 4 represents the unary operators combined with the addition operator within the range of [-1,1]. Sin and cos seem to be stable and square root, arccos, arcsin, and log are all broken. Sin and cos are stable because their boundaries are predictable since the range is [-1,1].

**Figure 4.1**

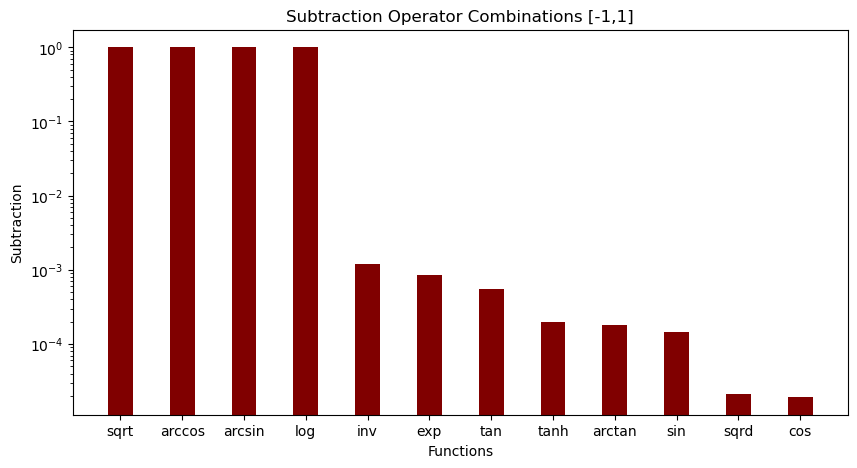
****

Figure 4.1 shows the sharpness for unary operators combined with the subtraction operator. Like the addition combination graph, sin and cos are stable and the first four operators are broken. Squared is probably stable because when you square a number between [-1,1], the result is always positive and stays within the same range, so it most likely keeps it stable.

**Figure 4.2**

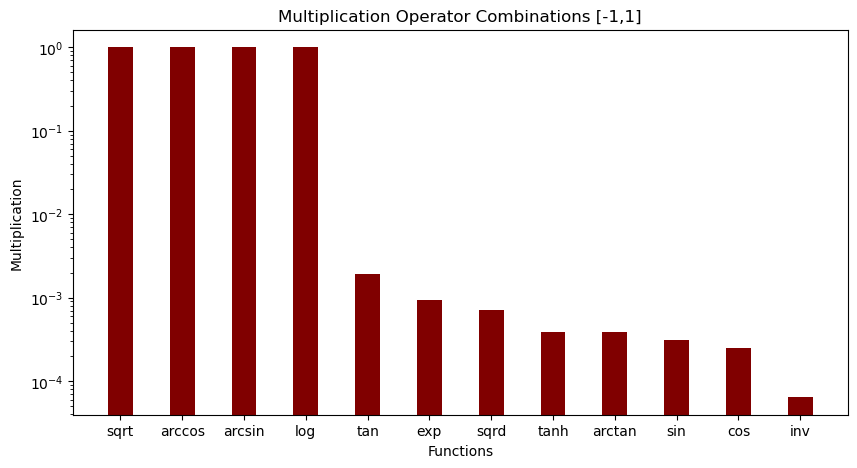
****

Figure 4.2 shows the multiplication operator combined with the unary operators within the range [-1,1]. Unlike the addition and subtraction operators, the multiplication operator shows the inverse function as being the most stable with the lowest sharpness level.

**Figure 4.3**

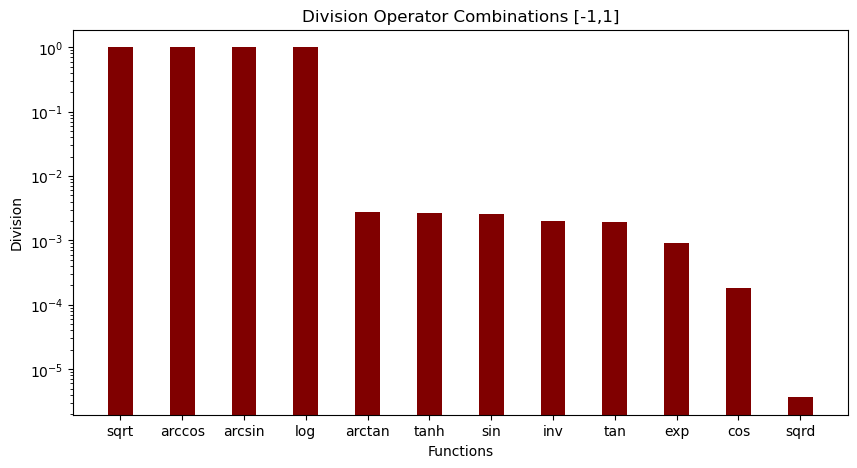
****

Figure 4.3 displays the division operator combinations with the unary operators. The only function that seems to be stable is the squared function. This is probably due to the fact that the results don’t grow too big or small, they are within a reasonable range.

**Figures 5**

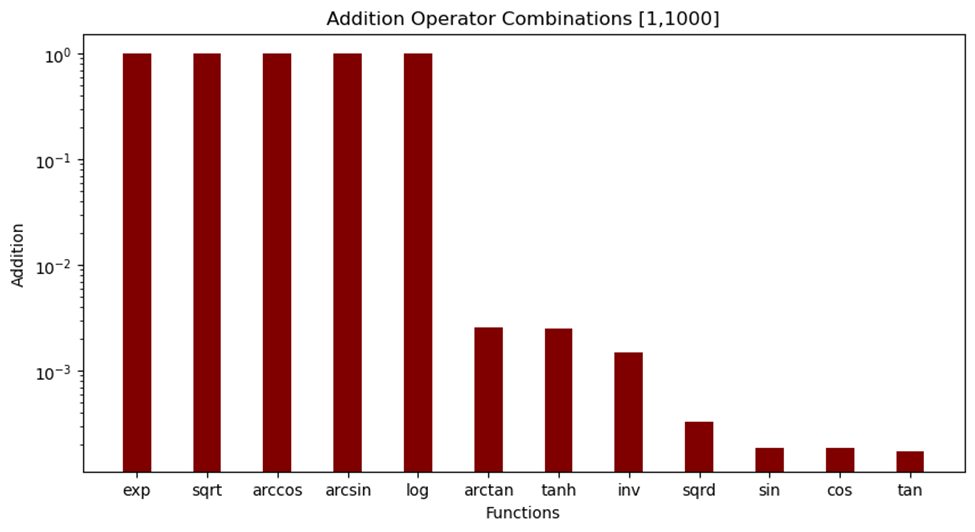


Figure 5 illustrates the sharpness of the unary operators combined with the addition operator for the range [1,1000]. Across all of the models, the exp, sqrt, arccos, arcsin, and log operators are broken, and the trigonometric operators seem to be on the stable side of the graph. The trig operators are most likely stable because of their intrinsic smoothness.

**Figure 5.1**

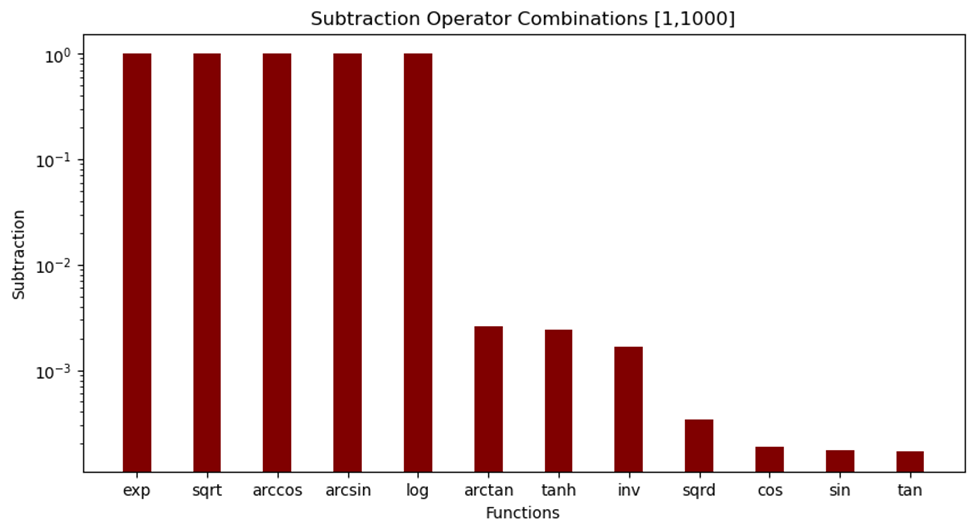
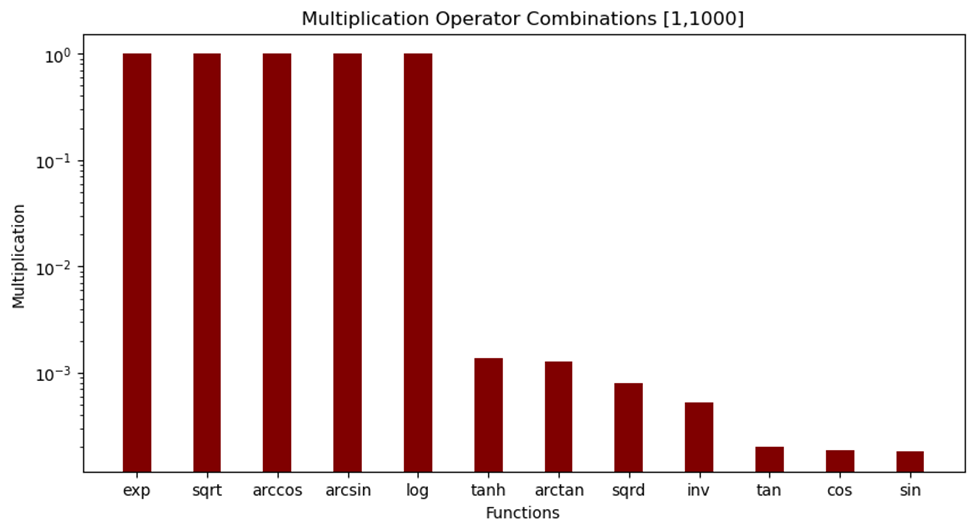
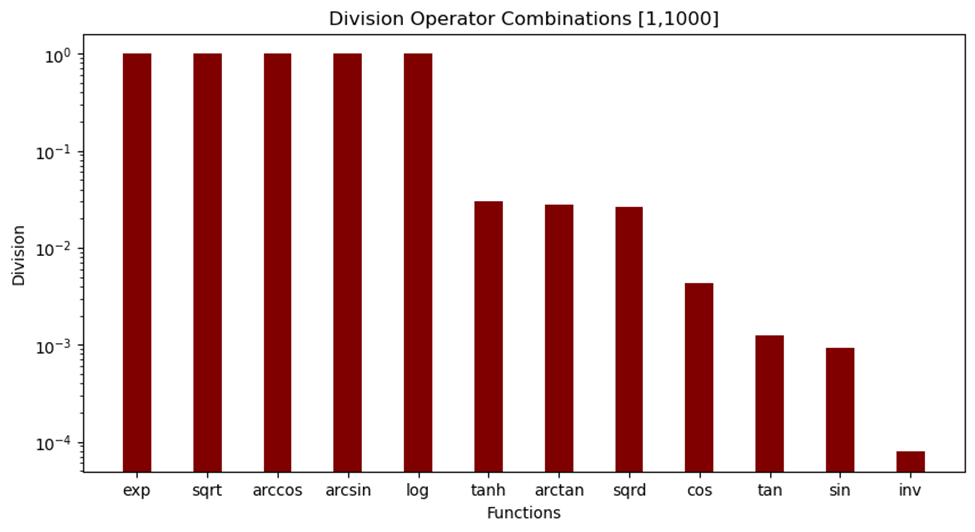


Figure 5.1 represents the sharpness of the unary operators combined with the subtraction operator. As seen in figure 4, the same trend occurs for all graphs within the range of [1,1000].

**Figure 5.2**



**Figure 5.3**



**Figures 6**

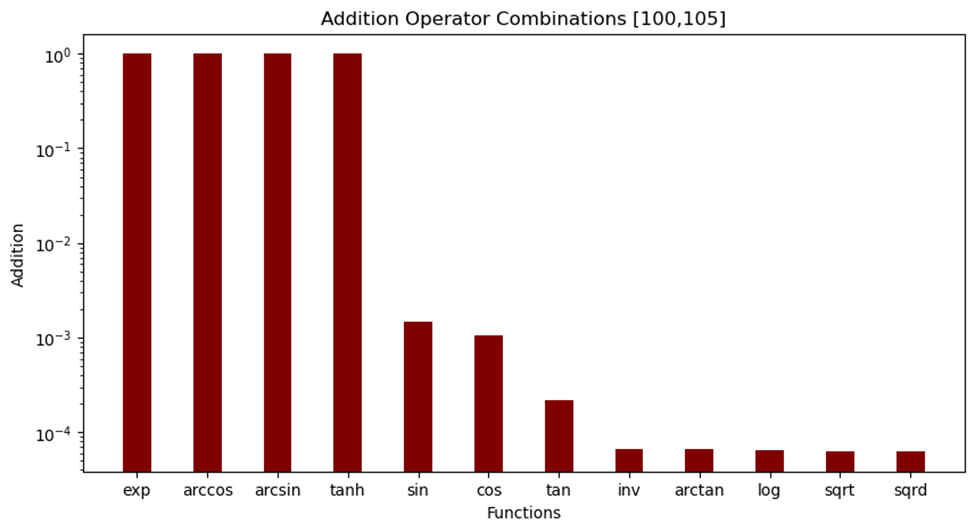
****

Figure 6 shows the addition operator combinations with the unary operators within the range [100,105]. For this range, the exp and tanh operators are often broken, as opposed to the sqrt and log operators for the [-1,1] range. Since this range is not close to 0, the operators that usually break like log and sqrt are on the low sharpness level and stable.

**Figure 6.1**

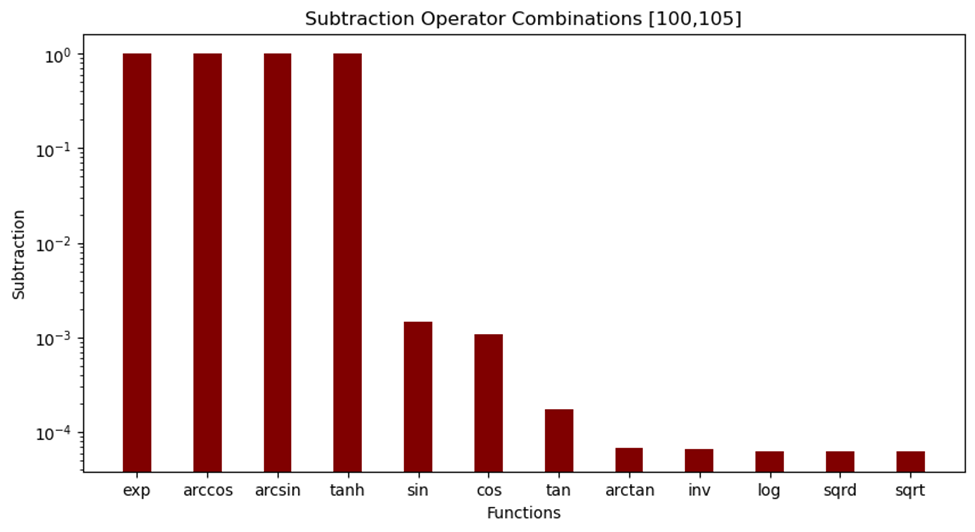
****

Figure 6.1 displays the subtraction operator combinations for the unary operators. The first four operator combinations are broken probably because when paired with subtraction, the difference between the numbers are either too small or too big to make precise results.

**Figure 6.2**

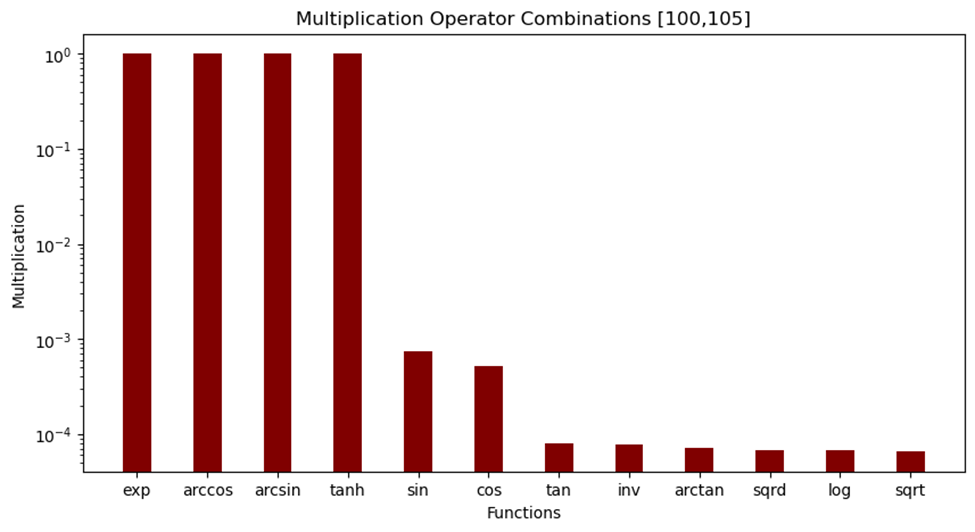
****

Figure 6.2 shows the multiplication operator combinations with the unary operators. Like the subtraction and addition operators, the first four operators show as broken and sqrt and log are both stable with low sharpness levels.

**Figure 6.3**

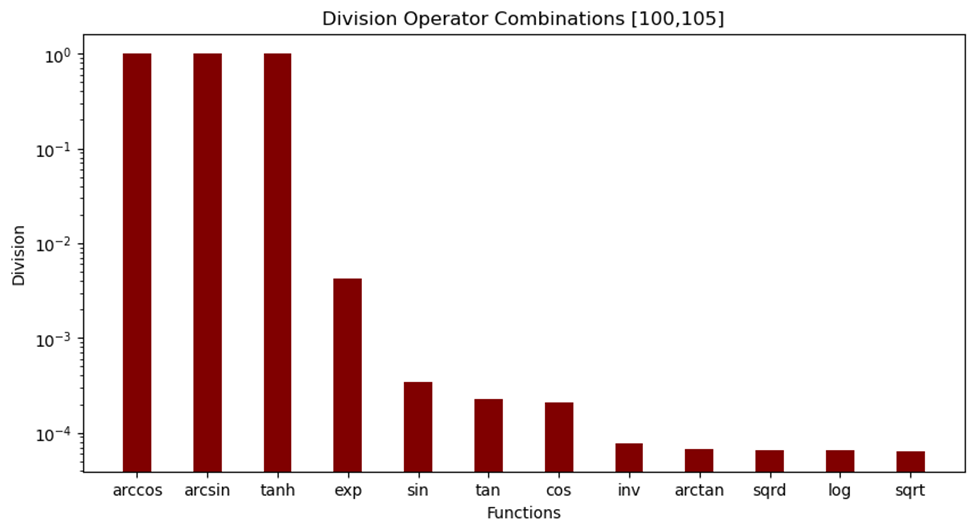
****

Figure 6.3 represents the division operator combinations with unary operators. The division operator seems to be stable with its combinations. This is the only case in which the exp operator doesn’t break, indicating that the division operator doesn’t produce extremely large or small results with the exp operator.

**Discussion**

In analyzing the behavior of the sharpness of various mathematical operators under different data ranges, several patterns emerge. The square root and logarithmic operators consistently display broken behavior across both the [-1,1] and [1,1000] ranges, indicating that they should be avoided if presented with data scenarios similar to these. The inverse operator shows variability, spreading across instances of being broken, increasing sharpness, and decreasing sharpness. Being broken means that the operator returns either an imaginary number or not a real value (nan). Operators paired as the second layer with square root and logarithmic operations frequently become broken (indicated in yellow), displaying a systemic issue with these combinations due to negative numbers. This demonstrates that these combinations should be avoided to achieve a more stable model.

Exponentiation seems particularly vulnerable among paired operators remaining broken for all combinations like the square root and logarithmic operators. In contrast, cosine maintains stability as it adjusts in sharpness when paired with other operators, avoiding breaks. Sin behaves similarly but with slightly less stability due to occasional breaks when interacting with square root and logarithmic functions.

Tan, along with the squared operator, appear less stable overall, consistently decreasing in sharpness across most paired operators. Arctan and tanh show relative stability, breaking only when paired with square root and logarithm functions but otherwise maintaining consistent sharpness levels across different operators. This analysis reveals complex dynamics in how mathematical operations interact and perform under varying computational conditions.

**Conclusion**

The investigation into the characteristics of the sharpness metric and its application across various mathematical operators and data ranges shows the importance of sharpness in enhancing model stability and generalization. The findings reveal that operators like square root and logarithmic functions exhibit unstable behaviors in specific data ranges, specifically near 0, suggesting their avoidance in scenarios where stability is paramount. On the other hand, operators such as cosine and arctan demonstrate greater robustness, maintaining more consistent sharpness across different conditions. The study's exploration of perturbation effects further illuminates how varying the amount and percentage of perturbations influences the stability and accuracy of model predictions. By analyzing both single-layer and double-layer interactions, the research provides valuable insights into operator compatibility and the importance of choosing the right combinations to minimize sharpness and avoid potential errors. Overall, this investigation not only reinforces the utility of the sharpness metric in genetic programming but also offers an understanding of how different operators behave under diverse conditions.

**References**

[1] [StackGP/StackGP.py at main · hoolagans/StackGP · GitHub](https://github.com/hoolagans/StackGP/blob/main/StackGP.py)

[2] Bakurov, I. Haut, N. Banzhaf, W. (2024). Sharpness-Aware Minimization in Genetic Programming. *ArXiv.org*. <https://doi.org/10.48550/arXiv.2405.10267>

[3] Vladislavleva, E.J., Smits, G.F., den Hertog, D.: Order of nonlinearity as a complexity measure

for models generated by symbolic regression via pareto genetic programming. IEEE Transac-

tions on Evolutionary Computation 13(2), 333–349 (2009). DOI 10.1109/TEVC.2008.926486

[4] Vanneschi, L., Castelli, M., Silva, S.: Measuring bloat, overfitting and functional complex-

ity in genetic programming. In: Proceedings of the 12th annual conference on Genetic and

evolutionary computation, pp. 877–884 (2010)

[5] Goncalves, I., Silva, S., Melo, J.B., Carreiras, J.M.B.: Random sampling technique for over-

fitting control in genetic programming. In: A. Moraglio, S. Silva, K. Krawiec, P. Machado,

C. Cotta (eds.) Genetic Programming, pp. 218–229. Springer Berlin Heidelberg, Berlin, Hei-

delberg (2012)

[6] Da Costa, L., Landry, J.A., Levasseur, Y.: Treating noisy data sets with relaxed genetic

programming. In: Artificial Evolution: 8th International Conference, Evolution Artificielle,

EA 2007, Tours, France, October 29-31, 2007, Revised Selected Papers, p. 1–12. Springer-

Verlag, Berlin, Heidelberg (2023). DOI 10.1007/978-3-540-79305-2 1. URL https:

//doi.org/10.1007/978-3-540-79305-2\_1

[7] Vanneschi, L., Gustafson, S.: Using crossover based similarity measure to improve genetic

programming generalization ability. In: Proceedings of the 11th Annual conference on Genetic

and evolutionary computation, pp. 1139–1146 (2009)