Data Mining Project - Swish Activation Function

Ted Watters

Abstract

This project seeks to expand on the work done by Google Brain on exploring activation functions in neural networks. [5] Specifically, look into which values of β generate a non-monotomic region and how that impacts training.

1. Introduction

There are various activation functions available in modern packages, including variations of the ReLU family and sigmoid family. Google Brain[5] reviewed a modified sigmoid function, nicknamed Swish, and found that it outperformed many other activation functions. Swish is defined as:

$$f(x) = \frac{x}{1 + e^{-\beta x}}$$

Graphically [5, Fig 4 and Fig 5]:

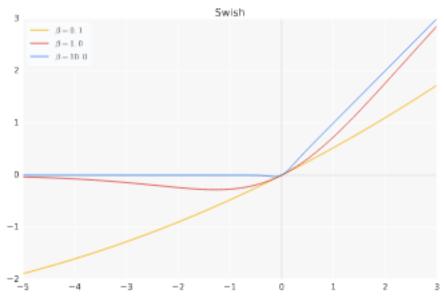


Figure 4: The Swish activation function.

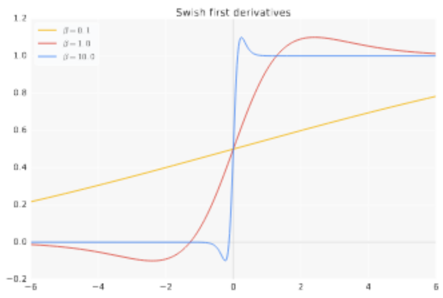


Figure 5: First derivatives of Swish.

As can be seen, depending on β, the activation function may be non-monotomic. This may not be ideal, as it is a ”convenient, but nonessential, property... If f is not monotonic and has multiple local maxima, additional and undesirable local extrema in the error surface may become introduced.” [2, pg. 307] The first derivative is:

$$f'(x) = \frac{1 + e^{-\beta x} + \beta e^{-\beta x} x}{(1 + e^{-\beta x})^2}$$

Email address: `ewatter4@jh.edu` (Ted Watters)

2. Regions where swish is decreasing

Examining further, $\{(1+e^{-\beta x})^2 > 0 : \forall \beta > 0, x \in \mathbb{R}\}$. So $f(x)$ is only decreasing when the numerator of the first derivative is less than 0:

$$\begin{aligned}
0 &> 1 + e^{-\beta x} + \beta e^{-\beta x} x \\
-1 &> e^{-\beta x} + \beta e^{-\beta x} x \\
-1 &> e^{-\beta x}(1+\beta x) \\
-e^{\beta x} &> 1 + \beta x
\end{aligned}$$

Let $u = -(1+\beta x)$ and $x = -\frac{u+1}{\beta}$, and using the Lambert function [8] [1]:

$$\begin{aligned}
1 &< \left(-(1 - \beta\frac{u+1}{\beta})\right) e^{\beta\frac{u+1}{\beta}} \\
1 &< e^{u+1}u \\
e^u u &> \frac{1}{e} \\
u &> W_0\left(\frac{1}{e}\right) \\
-\frac{u+1}{\beta} &> W_0\left(\frac{1}{e}\right) \\
x &< -\frac{1+W_0\left(\frac{1}{e}\right)}{\beta} \approx -\frac{1.278}{\beta}
\end{aligned}$$

Based on Figures 4 and 5 from the original paper [5], this may not be an intuitive answer. For instance, in Figure 4 $\beta = 0.1$ appears to be increasing over the entire interval. However, for $x \lessapprox -12.78$, the swish function is decreasing, but that is outside the limits of the figure. Similarly, for $\beta = 10$, it is not obvious from the figures whether swish is increasing as $x \to -\infty$. However, analytically we have shown it is decreasing for $x \lessapprox -0.128$.

Intuitively, it may make sense for to pick a value $\min \beta$ s.t. $f'(x) > 0$, $\forall x$. Minimum β is presumably optimal because it reduces linearity and maximizes the gradient. This hypothesis will be tested against the CIFAR-10 data set.

3. Tools

The authors of the paper [5] use various advanced neural networks for their analysis, including Mobile NASNet-A, ResNet-32, etc. For this project, I used a simpler CNN based on the TensorFlow tutorial [7]. The goal was to see if Swish outperformed ReLu in this context. Using existing code as a starting point [3]

[6] [4], I implemented a custom class in Python. Please note that this class allows for a trainable β, whereas the built in activation function in TensorFlow sets $\beta = 1$.

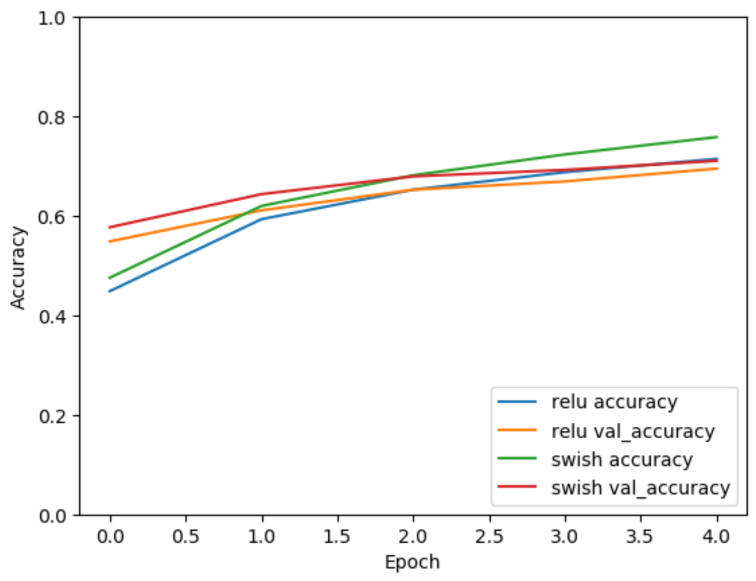
Because of the memory demands of this code, especially when running multiple iterations, I utilized an Amazon Web Services E2C instance (p2.xlarge) to run the final code. I utilized Python 3/ Tensor Flow 2, which was included in an Ubuntu Deep Learning AMI.

CIFAR-10 is a built in dataset to TensorFlow2. I utilized a different library to import SVHN.

My code is published at `https://github.com/tedwatters/swish-project/`

4. Swish Performance in CIFAR-10

Similar to the results in the paper, swish outperformed ReLu. This was true from the first to the last Epoch. Utilizing swish could either reduce the number of epochs needed to get to an acceptable answer, or improve in further accuracy



5. Range of Parameters in CIFAR-10

The authors of the paper [5] generated the following graphics, which imply that $\beta = 1$ is a good starting point (though training is recommended) and that the non-monotonic region is important for swish performance [5, Fig 4 and Fig 5].

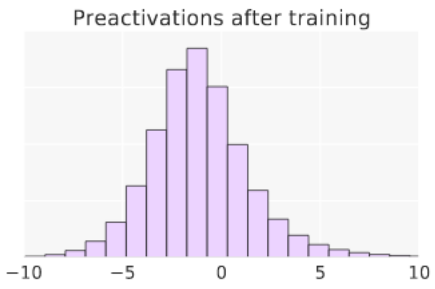


Figure 6: Preactivation distribution after training of Swish with $\beta = 1$ on ResNet-32.

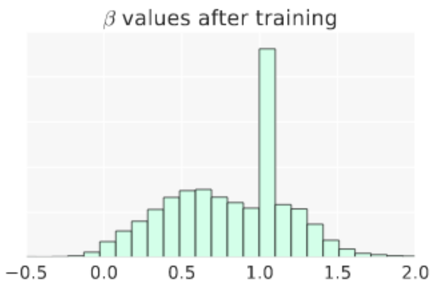
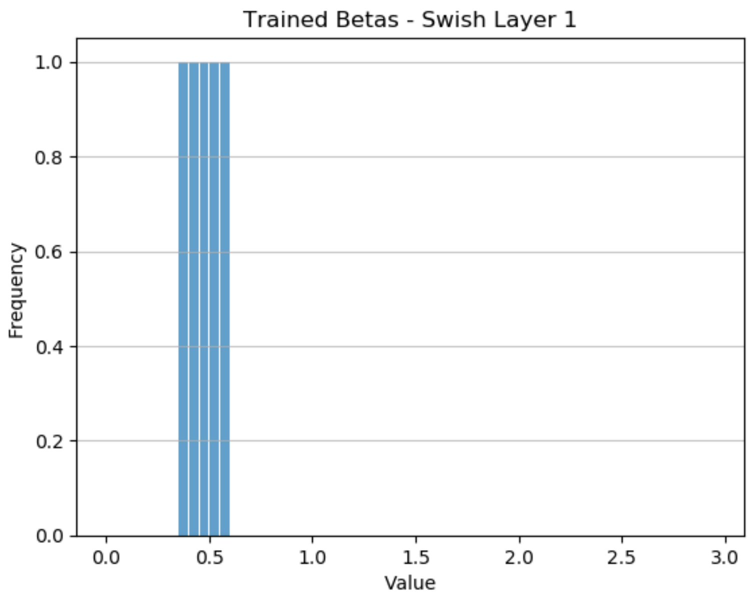
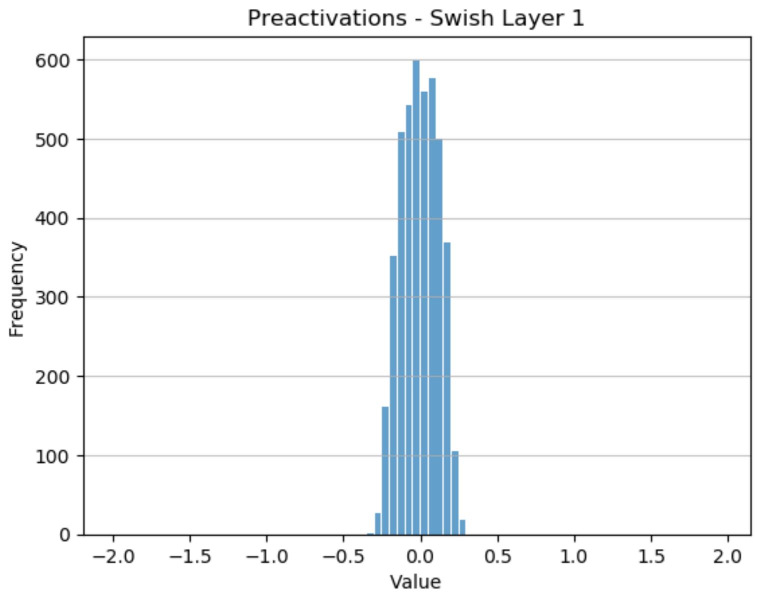
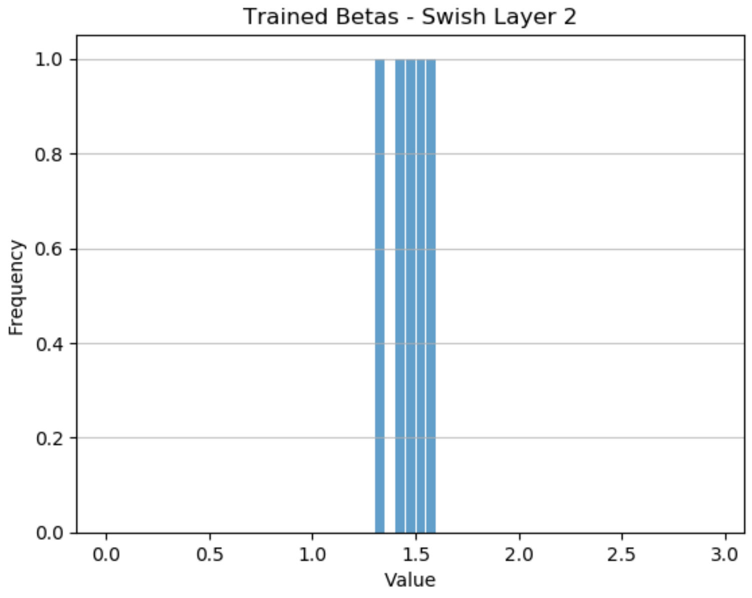


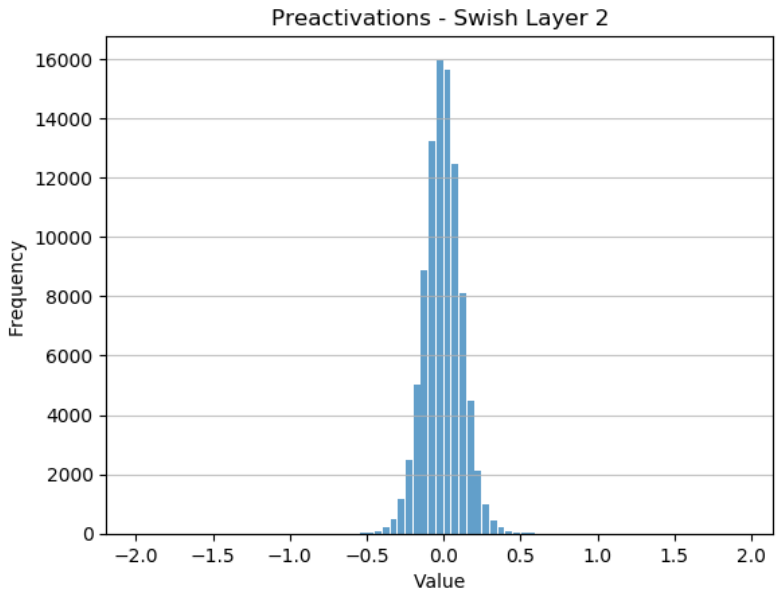
Figure 7: Distribution of trained β values of Swish on Mobile NASNet-A.

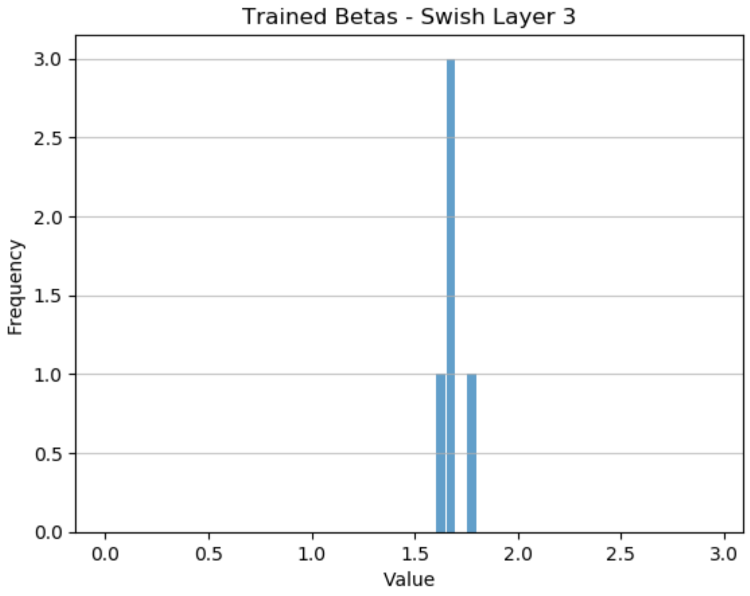
I conducted a similar analysis, using the tools listed above. The authors were not clear about the definition of the pre-activations or where swish was used. I utilized swish in 4 different layers, and compartmentalized results accordingly.

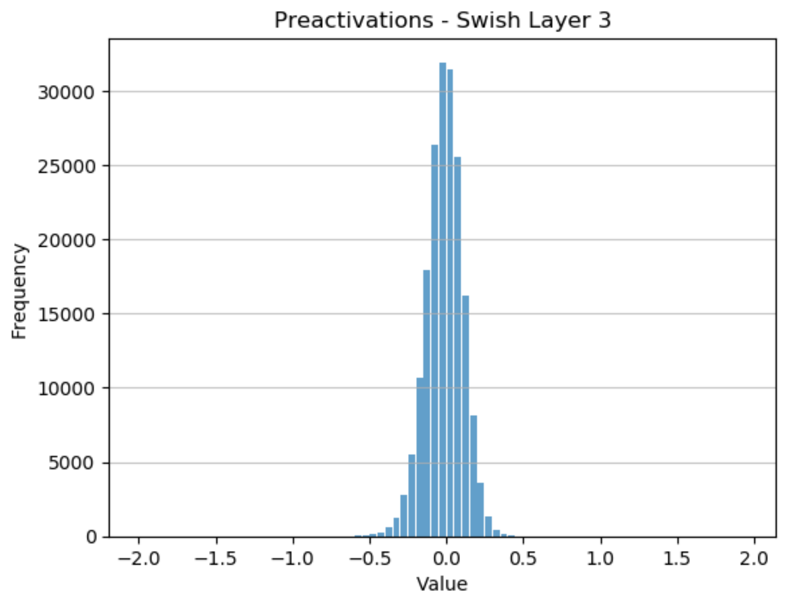


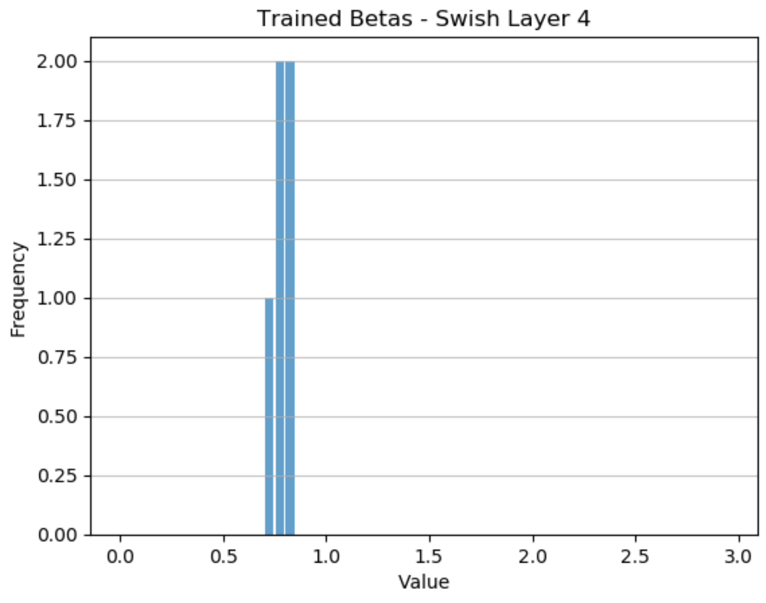

Preactivations - Swish Layer 1
Frequency
Value

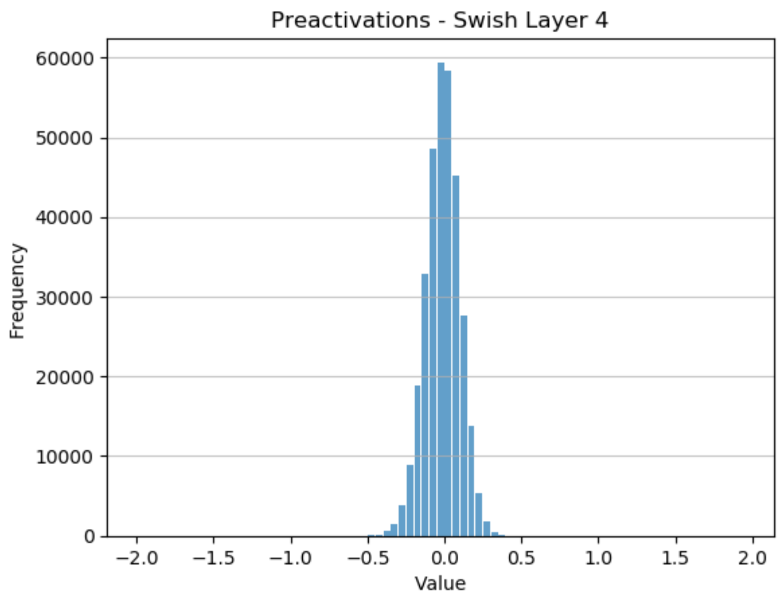

Trained Betas - Swish Layer 2
Frequency
Value


Preactivations - Swish Layer 2
Frequency
Value
16000
14000
12000
10000
8000
6000
4000
2000
0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


Trained Betas - Swish Layer 3
Frequency
Value
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0


Preactivations - Swish Layer 3
Frequency
Value


Trained Betas - Swish Layer 4
Frequency
Value



The big take aways from these figures are that:

- Optimal β depends on the layer. The initial Cov2D and single Dense layer were $\beta < 1$, where as the remaining Cov2D layers were $\beta > 1$
- The distribution of pre-activations was much different than the papers. One cause of this may have been that I normalized the pixel values from 1-255 down to the range [0,1]. There may also be different definitions of pre-activation that the authors used.
- The standard deviations for the preactivations in the first and last layer were lower than the middle two, which correlates with the differences in trained β.

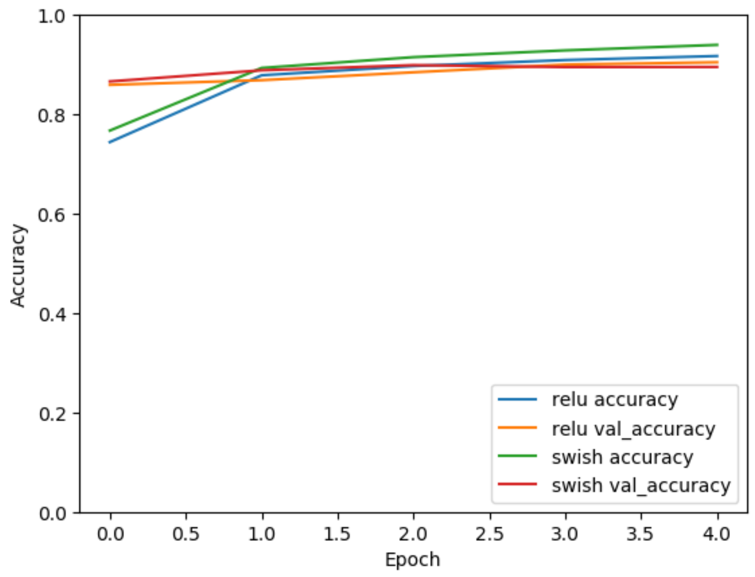
Comparing this with the hypothesis that β trains to stay out of the non-monotonic portion of swish, I found that:

- Virtually no pre-activations were in the region where swish was decreasing, thus swish was trained to be monotonic.
- Since our hypothesis stated that minimum β would be best, we would have expected that as standard deviation of the pre-activations was reduced, β would have been reduced. However, the opposite occured.

6. Swish Performance in SVHN

SVHN is a larger data set, and there is presumably less variance between characters in house numbers than there is in the various objects in CIFAR-10.

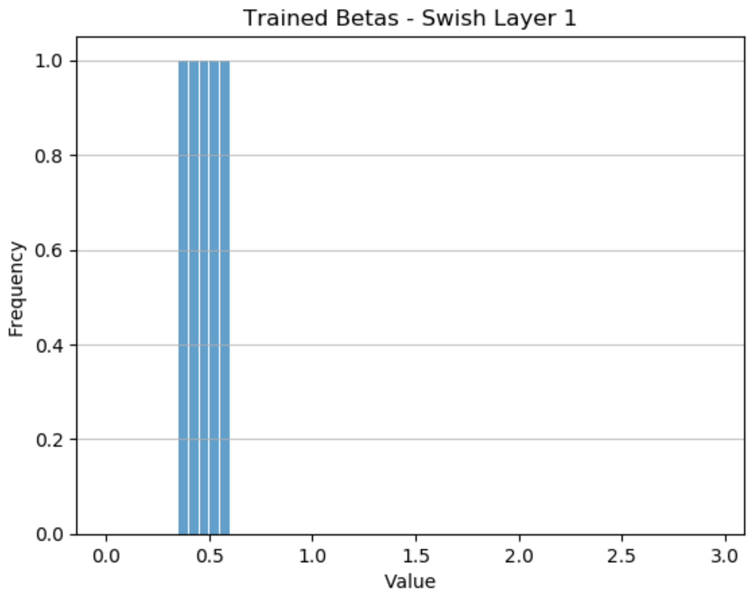
Presumably, our CNN would perform better with this data set. Looking at how effective swish was here:

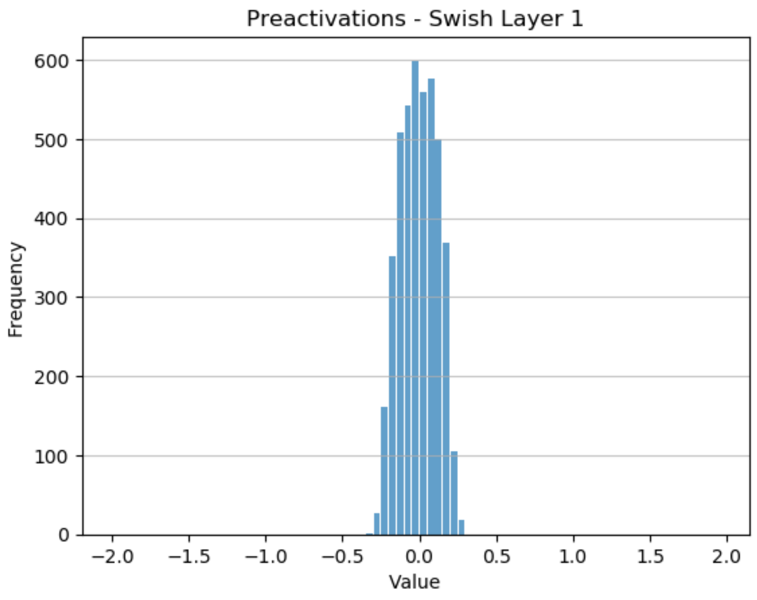


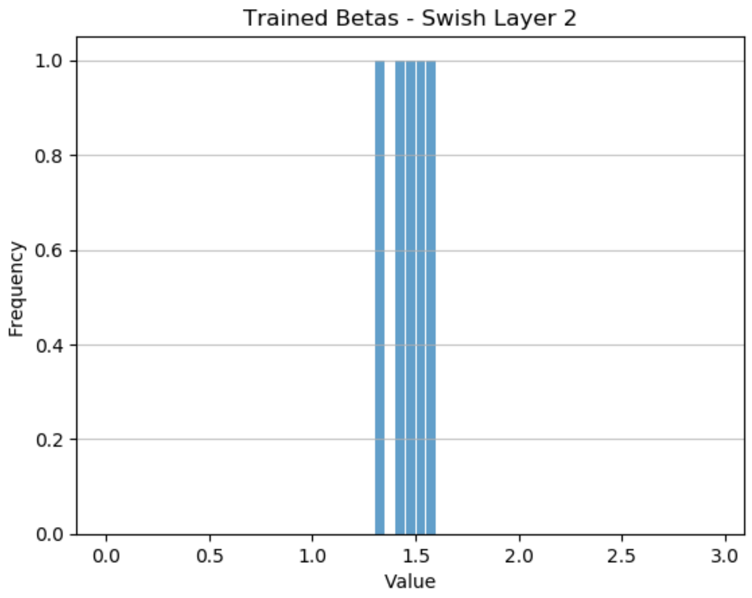
As shown, swish outperformed ReLu at all epochs. Additionally, overall performance was over 90

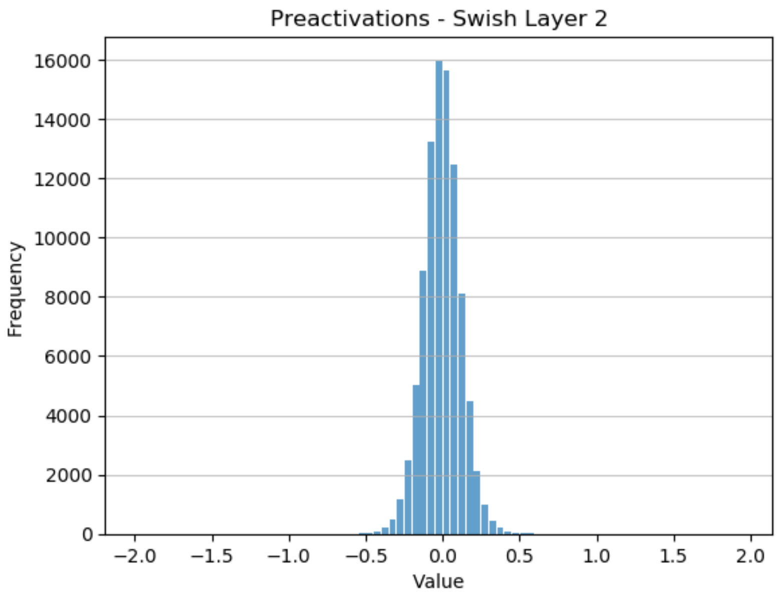
7. Range of Parameters in SVHN

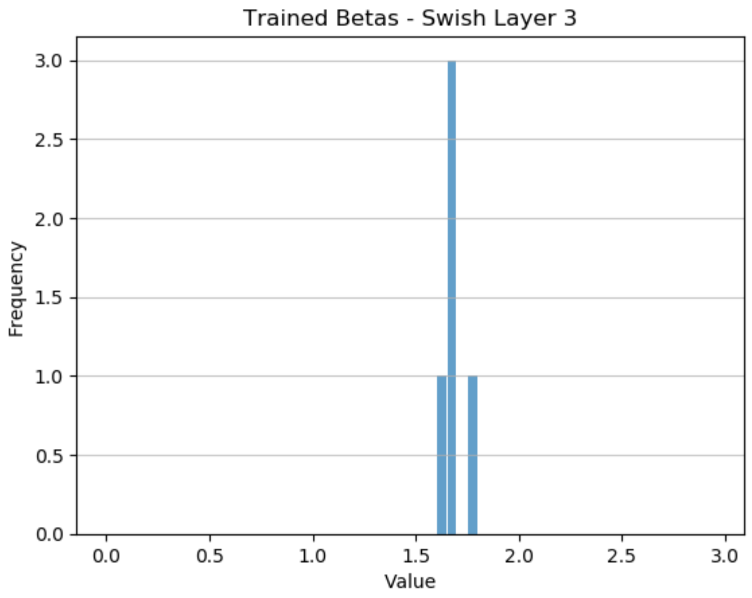
Since the neural network is the same, the data is the same type, and the incoming data has been normalized, we would expect to see similar trends in SVHN that we saw in CIFAR-10.

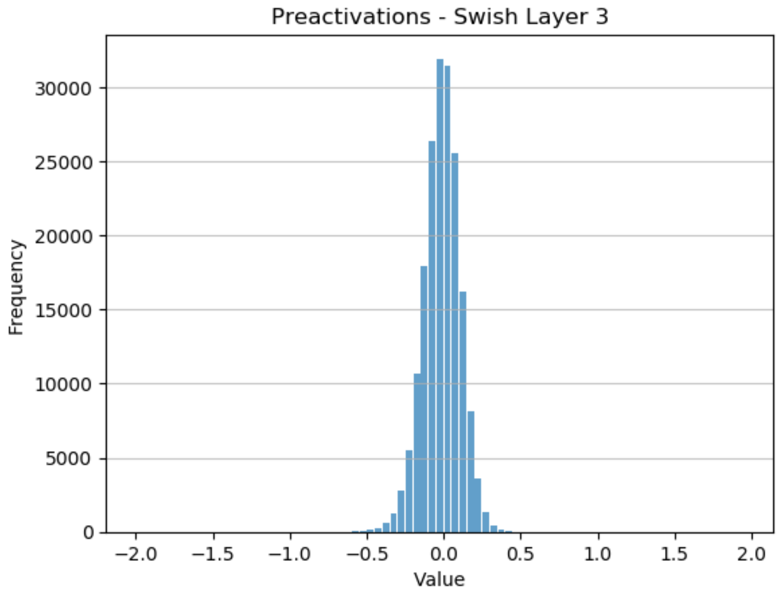

Trained Betas - Swish Layer 1
Frequency
Value
0.0
0.2
0.4
0.6
0.8
1.0
0.5
1.5
2.0
2.5
3.0

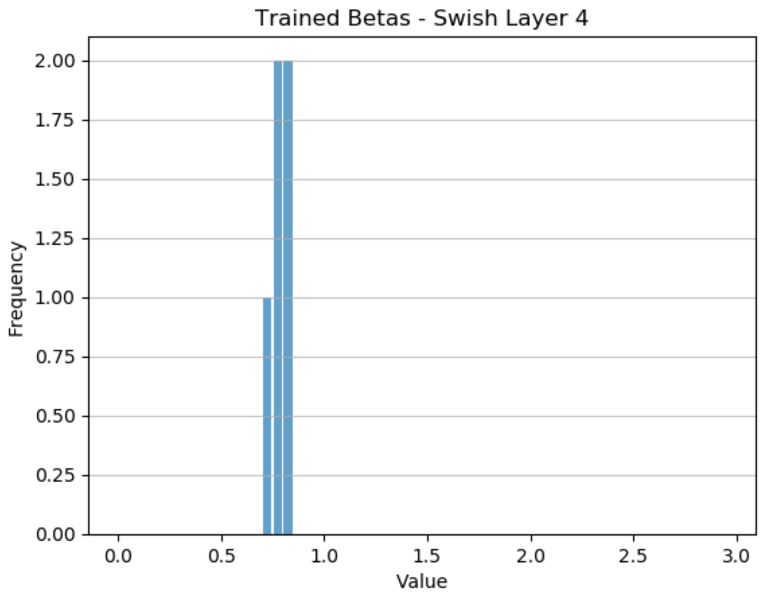

Preactivations - Swish Layer 1
Frequency
Value
0
100
200
300
400
500
600
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

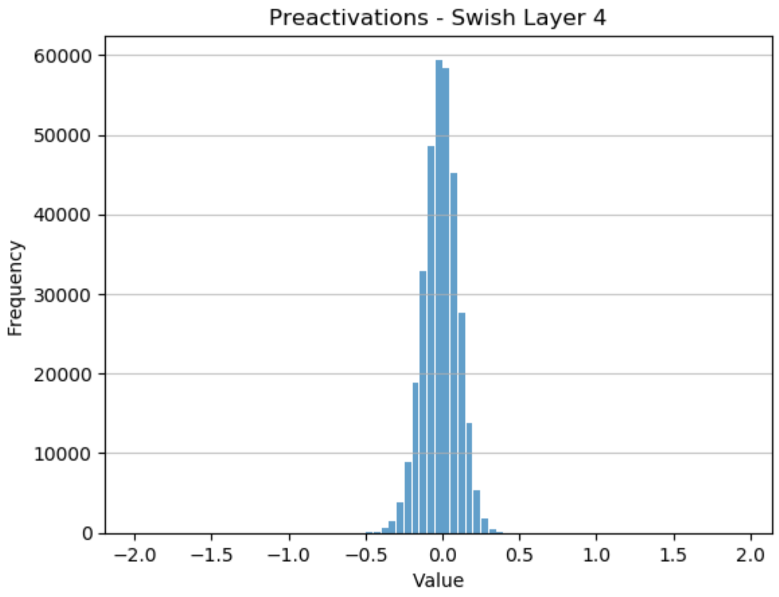

Trained Betas - Swish Layer 2
Frequency
Value
0.0
0.2
0.4
0.6
0.8
1.0
0.5
1.0
1.5
2.0
2.5
3.0


Preactivations - Swish Layer 2
Frequency
Value
0
2000
4000
6000
8000
10000
12000
14000
16000
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


Trained Betas - Swish Layer 3
Frequency
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Value


Preactivations - Swish Layer 3
Frequency
30000
25000
20000
15000
10000
5000
0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
Value





Indeed, the same trends appear as in CIFAR-10 when looking at β and pre-activations.

8. Conclusions

- Swish outperforms ReLu in simple neural networks in both the CIFAR-10 and SVHN data sets
- Swish does not need pre-activations in the non-monotonic regions to outperform ReLu
- The performance of swish is likely very dependent on the neural network architecture and data set. Other than knowing that swish should be attempted during optimization, there are few other ground rules
- If swish is going to be tested, recommend building your own class, because Tensor Flow limits swish to $\beta = 1$ and does not allow it to be a trained parameter.

9. Future Work

- Utilize more advance neural network architectures to see how they affect optimal β
- Do not normalize image data to be in the range [0,1], and see how those pre-activations affect optimal β
- Review addtional definitions of pre-activations to ensure consistent usage.
- Attempt to use swish on non-image data sets, such as language processing.
- Improve source code and graphics
- Conduct more data runs to see if there is a definable correlation between optimal β and standard deviation of the pre-activations.

References

[1] Avraham Adler. *lamW: Lambert-W Function*, 2015. R package version 1.3.3.

[2] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley, New York, 2 edition, 2001.

[3] Neoanarika Ang Ming Liang. Searching-for-activation-functions, 2018.

[4] Nicholas Ollis. Implementing swish activation function in keras, 2020. Online; accessed 12-November-2020.

[5] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. *CoRR*, abs/1710.05941, 2017.

[6] Simon Larsson. How to create custom activation functions in keras / tensorflow?, 2020. [Online; accessed 12-November-2020].

[7] TensorFlow. Convolutional neural network (cnn), 2020. [Online; accessed 17-November-2020].

[8] Wikipedia contributors. Lambert w function — Wikipedia, the free encyclopedia, 2020. [Online; accessed 12-November-2020].