Exploring the Impact of Rectification: A Comparative Study of ReTanh to Modern Activation Functions

Alex Ho

STAT 613 - May 2021

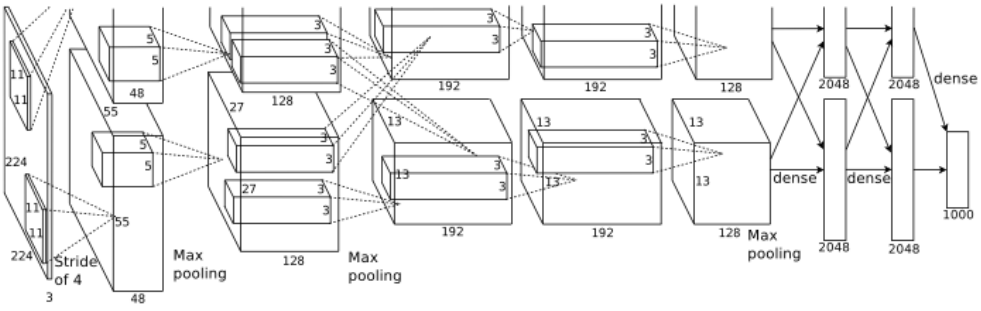

Figure 1: Illustration of AlexNet architecture from [8]

1 Literature Review

1.1 Convolutional Neural Networks

With the popularization of the backpropogation algorithm in [16], an era of machine learning using artificial neural networks emerged. Over time, multilayer perceptrons were improved upon and different varieties of artificial neural networks emerged. We have previously discussed Recurrent Neural Networks which are useful for modeling sequential data by using the output of one iteration as the input of the next iteration. Here we will discuss another specialized variety of neural network, the convolutional neural network. Convolutional networks perform conceptually in same way using the backpropogation algorithm and nonlinear activation functions, however instead of perceptrons, hidden layers consist of a kernel matrix which was used to convolve with the input by multiplication or other dot product. This architecture allowed the model to use local spatial structure from the inputs and intermediate hidden layers.

One of the first successful uses of convolutional networks was by [9] in 1989. Here, convolutional neural networks were used to recognize handwritten digits. This paper marked the start of computer vision related tasks using convolutional neural networks, however it did not gain significant ground until graphics processing units were shown to be useful in artificial neural networks [3]. The convolutional network used in this paper only had three hidden layers, and it wasn't until 2012 when "deep" convolutional networks were practically viable.

The ImageNet LSVRC-2010 contest was the first contest of its kind to have a large and high quality data set for computer vision tasks, containing 1.2 million high-resolution images coming

from 1000 different classes [1]. Learning this data set proved to be highly nontrivial for the first few years. Then in 2012, [8] made a breakthrough in convolutional network architecture. Achieving an error rate of 15.3%, the AlexNet architecture won the 2012 ImageNet Challenge by more than 10 percentage points compared to the runner up. It was the first of its kind to use rectified linear unit (ReLU) proposed in [12] as the activation function. It also boasted a massive 60 million trainable parameters from its five convolutional and three fully-connected layers. Training this model was only possible with GPU processing. This architecture is considered the start of the computer vision revolution which rapidly evolved in the following years. Today, convolutional networks can contain tens of millions of trainable parameters with over 100 convolutional layers [5]. An illustration of the architecture can be seen in Figure 1.

1.2 Activation Functions

From the lecture notes, we have seen that the first perceptrons used a simple step function to output 1 or -1 when the linear combination of inputs are greater than or less than a threshold value. However, when using the backpropogation algorithm for multilayer perceptron models, a differentiable function is required. In the recent decades, alternative activation functions have been developed to improve learning for deep neural network architectures.

1.2.1 Sigmoid Function

The Sigmoid activation function, also called the logistic function, was one of the first activation function used in artificial neural networks [16]. It was already known to be useful in logistic regression for binary classification, and takes the form $\sigma(x) = \frac{1}{1+e^{-x}}$. Its use has been limited after LeCun et al. showed in [10] that the sigmoid function is not ideal for learning in neural networks; its non-zero mean could lead to singular values in the Hessian. Moreover, it is prone to what is called "saturation" meaning that beyond certain values, learning is very slow due to small gradients. Given its shape, it faces saturation problems at both the positive and negative directions, and its use in modern machine learning is limited to output layers for binary classification.

1.2.2 Tanh Function

The hyperbolic tangent function, shortened as the tanh function, is a smooth, zero-centered function with a range of -1 to 1. It became useful as an alternative to the sigmoid function because it had a mean of zero, aiding training performance with backpropogation [13]. However, given its similar shape to the sigmoid, it also suffers problems of saturation and vanishing gradients. Another property of the tanh function is that it has a maximum gradient value of 1 when the input value is 0. This property makes "dead neurons," neurons which contain an activation weight rarely used or trained due to a zero or near-zero gradient, because the tanh gradient approaches zero given values not near zero [13]. Other variations of the tanh were created, like the Hardtanh, which is a computationally cheaper piecewise linear function version of tanh with a range of -1 to 1.

1.2.3 ReLU Function

The rectified linear unit, or ReLU for short, is a simple function proposed in [12] which takes the form $ReLU(x) = \max(0, x)$. It solves the issue of vanishing gradients by allowing the gradient to simply be zero or 1. It is also very easy to compute, both in the forward pass and in backpropogation which may reduce training time. It has been the most popular activation function for hidden units of neural networks since [12] and is now considered the standard activation function [13]. Glorot and Bengio also note in [2] that ReLU activations have equal or greater performance than tanh despite hard non-linearity and an undefined gradient at zero, partly because of the weight sparsity it induces in the model. This sparisty, however, is equivalent to having dead neurons, so the leaky ReLU was proposed, taking the form of $\max(\alpha x, x)$ where α is some small non-negative value to allow for a non-zero gradient. [13] also notes that ReLU is prone to overfitting compared to the sigmoid as a result of its expressivity.

1.2.4 Swish Function

While the leaky ReLU fixed the problem of dead neurons, there was still an undefined gradient at zero, so [14] introduced the Swish function which takes the basic form $x\sigma(x)$ where $\sigma(x)$ is the sigmoid function. This function has similar characteristics to ReLU, however it was the first compound activation function proposed. The authors say that the main advantages of Swish are

that its smoothness and bounded from below characteristics solve the vanishing gradient problem, and provides informative gradients for very deep neural networks. The authors claim performance equal or greater than ReLU variants, and is one of the latest significant contributions to activation functions [13].

1.2.5 Adaptable Activation Functions

Beyond fixed activation function are adaptable activation functions which aim to have learnable parameters that exceed the performance of their fixed counterparts. An example of such functions is [4] and [14]. Goyal et al in [4] propose Self-Learnable Activation functions (SLAF) which uses a weighted sum of predetermined basis functions to achieve a claimed optimal activation function. They note that it can approximate most existing activation functions. A more simple adaptable activation function is Swish, which the authors also note can take the form of $x\sigma(\beta x)$ where β can be a learned parameter. While this is still an open field of study, it seems that the computational expense does not make up for the performance gains and the use of adaptable and learnable activation functions is limited to special cases.

2 Experiments and Results

2.1 Setup and Motivation

The experiments outlined in this section compare different activation functions for a simple standard convolutional network and a simple large convolutional network. The larger network will be used to evaluate how the activation functions vary for a deeper model where exploding or vanishing gradient flow may become a concern. The activation functions to be tested are ReLU, tanh, sigmoid, Swish, and an activation function which I call ReTanh. ReTanh takes inspiration from the ReLU activation function by rectifying the tanh function and takes the form of $ReTanh(x) = \max(0, tanh(x))$. To my knowledge, ReTanh or any other rectified tanh function has not been tested in any other experiments at the time of writing.

The motivation behind ReTanh is to compare to ReLU and evaluate the effect of rectifying. As discussed in the literature review, the tanh function saturates easily with a low upper bound, so we should expect there to be relatively lower performance compared to ReLU. However, there may be some improvements because rectifying solves half of the saturation problem found in tanh. These experiments will aim to show the effect that rectifying has on an activation function. A plot of ReTanh can be seen in 8.

We will be comparing the training, validation, and test accuracy as well as training loss between the functions. The accuracy will give a measurement for evidence of overfitting, and the training loss will show how fast and well the model is learning the data. The validation loss is not useful in this case because it is independent of the learning by gradient descent so it is not informative of how the activation functions differ.

2.2 Data

The data set which will be used to compare the different models will be the CIFAR-10 data set [7] which is a well-known and standard baseline. The CIFAR-10 data set consists of 10 image classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck) where each class has exactly 5000 examples in the training set and 1000 examples in the test set. Each image is 32 by 32 pixels and in red, green, blue channels giving each image dimensions of $32 \times 32 \times 3$. We will also use

CIFAR-10.1 [15] which was developed as an additional testing set with similar distributions to the original CIFAR-10 set. CIFAR-10.1 consists of roughly 2000 additional test images developed years after the original data set was released, and were selected as a subset of the TinyImages data set [17]. It was designed to minimize the shift in internal distributions relative to the original data set and showed to be useful in evaluating models for implicit and direct overfitting.

This data set is useful for our use case because it is sufficiently complex to be nontrivial. Training on other datasets like MNIST [11] can easily reach less than 10 percent error rate with a fully connected neural network with a single hidden layer within five epochs. Moreover, the dimensions of the images are small enough such that several experiments and trials can be run even with limited computational resources.

2.3 Implementation

The implementation of the models follows a simple convolutional network architecture. We use standard basic image pre-processing on the training set: the images are padded with 6 pixels on each side and randomly horizontally flipped, and then randomly cropped back to the original $32 \times 32 \times 3$ dimensions. The brightness is then adjusted randomly with a maximum delta of 0.5 so that the image brightness is in the range of 0.5 to 1.5 where 1 is the original brightness. The standard architecture starts with a 16-filter convolutional layer, followed by two each of 32-filter, 64-filter, and 128-filter convolutional layers with a 2D average pooling layer between layers where filter size increases. The larger model also starts with a single 16-filter convolutional layer, along with three each of 32-filter, 64-filter, 128-filter, and 256-filter convolutional layers with a 2D average pooling layer between layers where filter size increases. The model ends with a global average pooling layer connected to the 10-way fully-connected output layer with softmax. In all, there are a total of eight trainable layers with 292,490 trainable parameters for the standard model and 14 trainable layers with just under 2 million trainable parameters for the larger model.

The default TensorFlow hyperparameters were used. Using the categorical cross entropy as the loss function, we minimized it using the Adam optimizer [6] with the default TensorFlow hyperparameters (learning rate $= 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1 \times 10^{-7}$). Each model was trained using a batch size of 64 over 50 or 100 epochs. Models were created using Google's TensorFlow version 2.3 and trained on Google Colab. The link to the Colab Notebook can be found

here.

2.4 Results

A summary of the results can be found in Tables 1, 2, 3 and figures 2 and 3 with additional figures in the Appendix.

Function	Time (ms/step)	Min Training Loss
Sigmoid	11	0.9364
Tanh	11	0.6373
ReLU	13.5	0.1595
ReTanh	13	0.2976
Swish	13	0.1618

Table 1: Summary of time and loss data for each activation function over 50 epochs for standard sized model.

Function	Training Accuracy	Validation Accuracy	Test Accuracy
ReLU	0.963	0.8481	0.7313
ReTanh	0.9217	0.8418	0.7279
Swish	0.9616	0.856	0.7447
Tanh	0.8076	0.7914	0.6541

Table 2: Summary of accuracy for each activation function over 100 epochs with standard sized model.

Function	Training Accuracy	Validation Accuracy	Test Accuracy
ReLU	0.9390	0.8464	0.7303
ReTanh	0.8575	0.8018	0.6660
Swish	0.9346	0.8289	0.7195
Tanh	0.6687	0.6935	0.5572

Table 3: Summary of accuracy for each activation function over 100 epochs with large sized model.

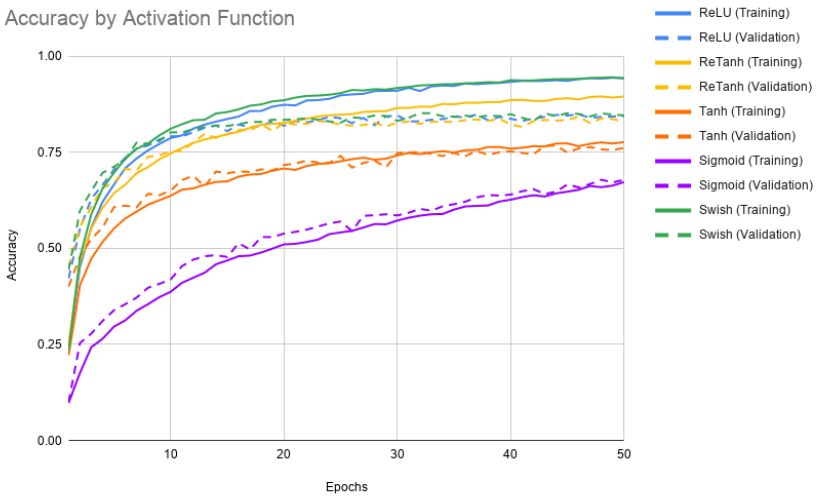

Figure 2: Plot of accuracy over 50 epochs for each activation function for the standard sized model.

2.5 Discussion

2.5.1 Standard Model

We can see from the plots of the standard sized model over 50 epochs that the sigmoid and tanh functions performed noticeably worse than the other activation functions in terms of accuracy as well as training loss. However it is also interesting to note that they did perform better in terms of speed and overfitting, as both were faster and did not diverge greatly between the training and validation accuracy.

Comparing the higher performance activation functions ReLU and Swish, we can see that they performed nearly identically in terms of training and validation accuracy, even over 100 epochs, with the standard sized model. The Swish function achieved very slightly higher accuracy and slightly lower loss, however it may be able to be attributed to the random initialization of the weights. They also showed signs of significant overfitting through diverging training and validation accuracy, however the test accuracy was still higher than the tanh test accuracy over 100 epochs.

The ReTanh function performed surprisingly well in the standard model. It achieved very similar performance to the Swish and ReLU functions with regard to accuracy over 100 epochs, however the training loss was greater, almost double that of Swish and ReLU over 100 epochs. The ReTanh function also indicated lower levels of overfitting the training data because the difference

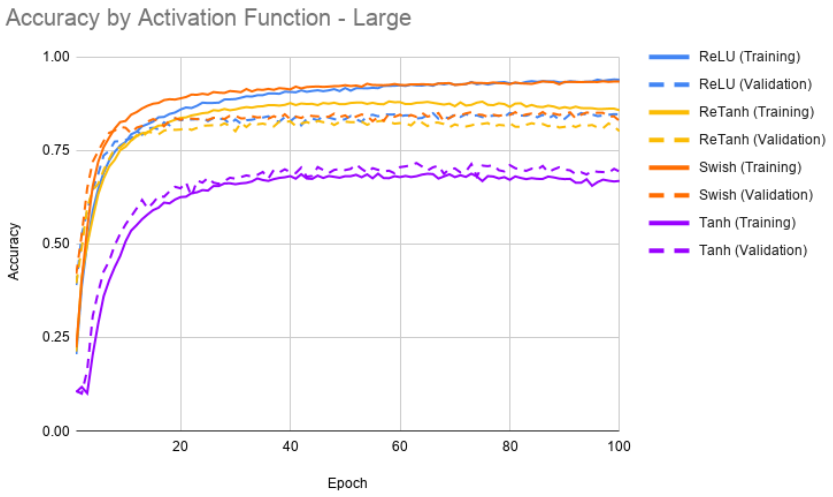

Figure 3: Accuracy by activation function for the large convolutional neural network model over 100 epochs.

between the validation and training accuracy was noticeably lower than the same difference in the ReLU and Swish functions over 100 epochs. The execution time was also comparable to ReLU and Swish at about 13 milliseconds per step. Overall for the standard sized model, there is no clear advantage to using ReLU or Swish over the ReTanh function.

2.5.2 Large Model

With the larger model, the differences between the activation functions started to emerge. As a baseline, tanh was still tested to compare to ReTanh even though poor performance is expected. Tanh performed as expected, significantly worse than ReLU, ReTanh, and Swish. After 100 epochs, the training and validation accuracy for the ReLU and Swish functions were nearly identical, though, Swish saw the quickest increase in accuracy compared to the other models, followed closely by ReLU. ReTanh saw performance somewhere between the top performers and its parent tanh, however it still continued to see lower overfitting compared to ReLU and Swish. The validation accuracy for ReTanh was only slightly lower than that of ReLU and Swish, however the training accuracy was noticably lower than the two.

2.6 Conclusion and Future Work

Rectifying the hyperbolic tangent function yielded unexpected results. With smaller models where gradient flow and vanishing gradients are less of a concern, it appears even a limited range ReTanh performs comparably and even better than the state of the art ReLU and Swish functions with regard to accuracy and overfitting. However for larger models, differences emerge and the more expressive ReLU and Swish functions yield greater accuracy and training speed.

When looking at the impact of a simple rectifier, it is clear that it improves upon older activation functions. The improvement gives evidence that even with a very limited expressiveness, solving part of the vanishing gradient problem greatly improves performance, both in large and small models. These results bring to question how much activation functions have truly been improved upon if such a simple alteration to a sub-optimal function increases performance to almost the same level.

Future work could include comparing the "leaky" versions of the ReLU and ReTanh as well as with deeper models with additional methods, like dropout and batch normalization. Testing the effects of rectifying other functions could also be another area of work, although a vertically shifted and rectified sigmoid function would expect to have similar performance to ReTanh because they share the same shape. A more abstract future direction would be to pay attention to activation functions more in neural network research, as they are the aspect which defines how the nonlinearity and gradient of neural networks behave. Balancing expressiveness by finding a computationally efficient activation function which can mitigate overfitting would remove the need for heuristics to avoid overfitting like dropout.

Though activation functions are not a popular area of modern research, this experiment shows that investing more effort into this field may yield high rewards, like improvements in overfitting and training time among others.

References

[1] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[2] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[4] Mohit Goyal, Rajan Goyal, and Brejesh Lall. Learning activation functions: A new paradigm for understanding neural networks. *arXiv preprint arXiv:1906.09529*, 2019.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[7] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[9] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[10] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[11] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[12] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

[13] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[14] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[15] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do cifar-10 classifiers generalize to cifar-10? 2018. https://arxiv.org/abs/1806.00451.

[16] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[17] Antonio Torralba, Rob Fergus, and William T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, 2008.

3 Appendix

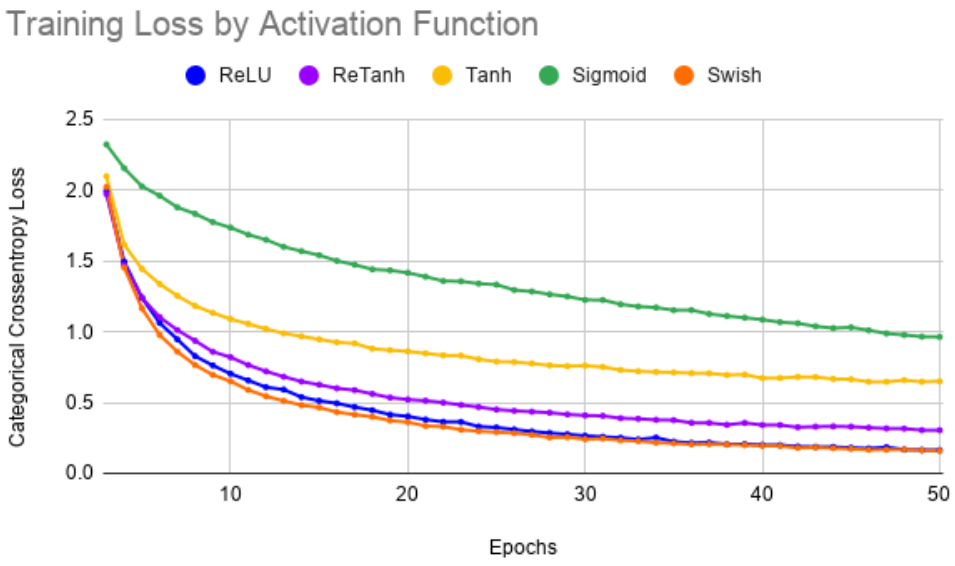

Figure 4

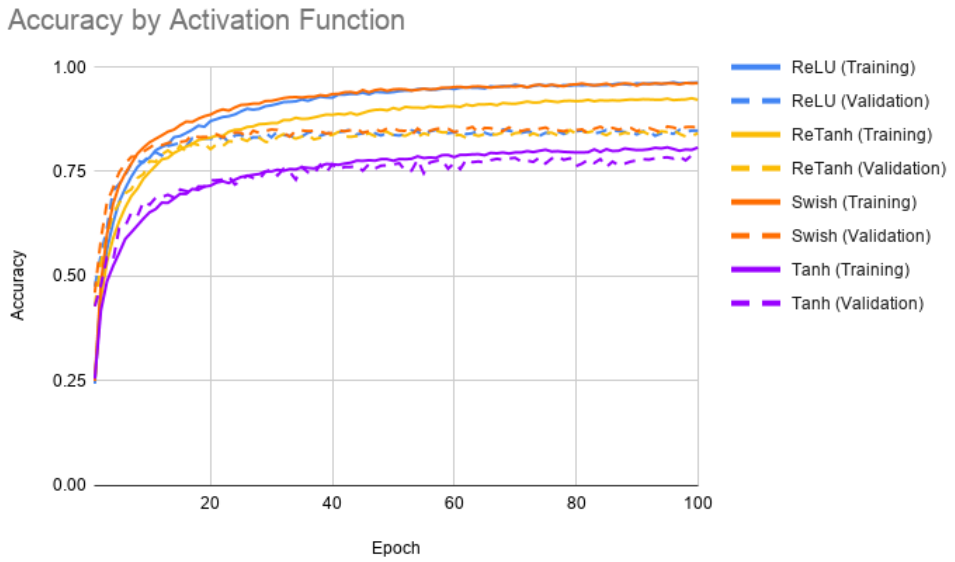

Figure 5

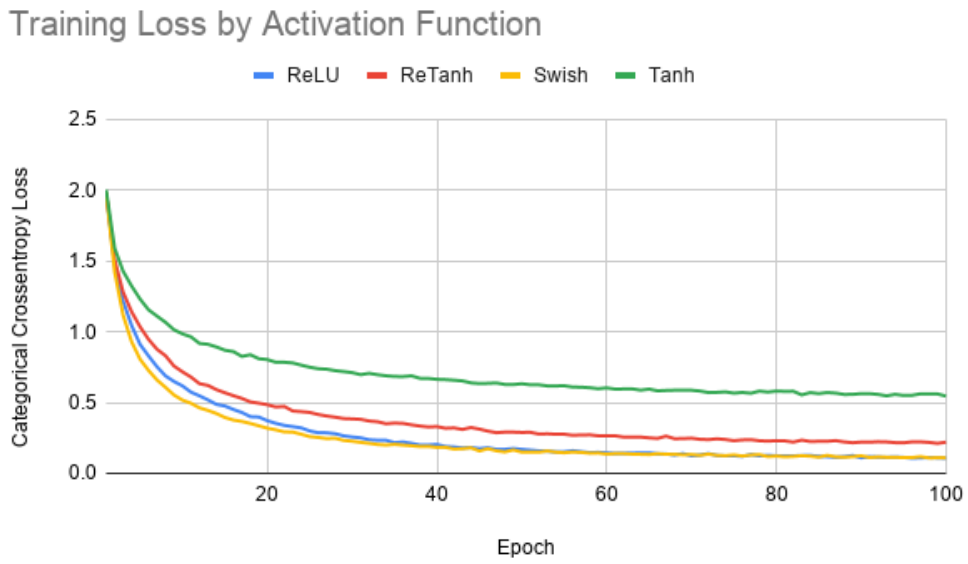

Figure 6

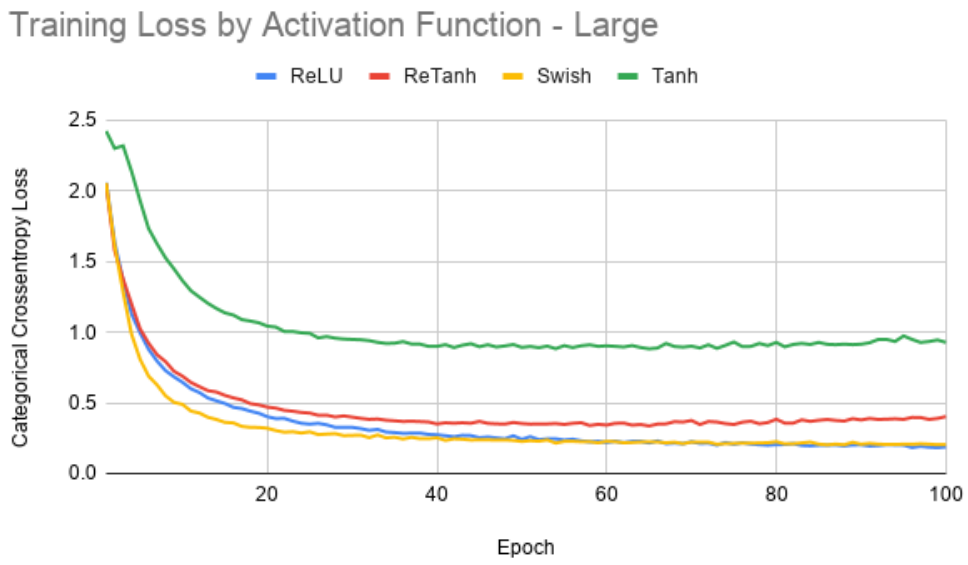

Figure 7

Figure 8: Plot of the ReTanh function.