

Phish: A Novel Hyper-Optimizable Activation Function

Philip Naveen

December 18, 2021

Abstract

Deep-learning models estimate values using backpropagation. The activation function within hidden layers is a critical component to minimizing loss in deep neural-networks. Rectified Linear (ReLU) has been the dominant activation function for the past decade. Swish and Mish are newer activation functions that have shown to yield better results than ReLU given specific circumstances. Phish is a novel activation function proposed here. It is a composite function defined as $f(x) = x \text{TanH}(\text{GELU}(x))$, where no discontinuities are apparent in the differentiated graph on the domain observed. Four generalized networks were constructed using Phish, Swish, Sigmoid, and TanH. SoftMax was the output function. Using images from MNIST and CIFAR-10 databanks, these networks were trained to minimize sparse categorical crossentropy. A large scale cross-validation was simulated using stochastic Markov chains to account for the law of large numbers for the probability values. Statistical tests support the research hypothesis stating Phish could outperform other activation functions in classification. Future experiments would involve testing Phish in unsupervised learning algorithms and comparing it to more activation functions.

1 Introduction

Deep-learning algorithms are capable of solving complex problems. They use a series of synaptic weights and perceptrons to mimic the human thinking process. The success of training deep neural-networks (DNN) relies much on the activation function used in them. In each perceptron, two phases occur: a summation and transformation. In the summation, the inputs are multiplied with synaptic weights, which are initially generated at random, with a Hadamard product [12]. The transformation step consists of the summated vector being parsed through an activation function in addition to an optional bias [11]. Early architectures used TanH and Sigmoid extensively. However, the more complex DNNs required better activation functions.

The most commonly used activation function in DNNs is Rectified Linear (ReLU) [2]. It is a less probability inspired piecewise function with no discontinuities. It has a jump discontinuity when differentiated due to the sharp turn at the origin. Experiments demonstrated that ReLU increased the performance in DNNs, outperforming TanH [1] and Sigmoid [8]. However, ReLU has some faults. One of the biggest ones is the dying ReLU issue, but luckily leaky ReLU partially solved this issue via augmenting the negative domain of the function [3].

Swish and Mish are newer activation functions that have recently gained traction [9]. They are both composite and comprise at least one existing activation function. Unlike ReLU, these functions are non-linear, and their derivatives are void of discontinuities. They both perpetually increase and pass through the origin (0, 0). The new activation function created here would follow the parameters of Swish and Mish [6].

A new activation function was fabricated here. It is comedically named Phish. Phish is defined as $f(x) = x \text{TanH}(\text{GELU}(x))$. Phish is monotonic, unlike other activation functions where the slope is completely positive. On the interval from $[0, \infty]$, it is completely positive and passes through the origin (0, 0). Phish estimates update gradients for backpropagating DNN algorithms.

An experimental simulation to compare Phish to existing functions will be conducted. The levels of independent variable will be Phish, Swish, Sigmoid, and TanH. There was no control. The dependent variable was the minimization of sparse categorical crossentropy (SCC), which is one of the most common loss functions in classification. Several control variables will be held, such as the DNN layers, optimizer, output function, and learning rate.

2 Phish Activation Function

2.1 Backpropagation and Update Gradients

Activation functions are derived with the purpose of generating non-linearity to the inherently linear data transformed from the input layer of a neural-network. Backpropagation is the process where each synaptic weight in deep-learning algorithms are iteratively finetuned to complete a task using loss calculated between the expected and actual outcomes. Suppose there is a multilayer perceptron with weights, and biases adjusted through an arbitrary activation function $A(x)$. In this multilayer perceptron, as with most, the weights are defined the simple matrix

$$w^{[l]} = \begin{bmatrix} w_{1,1}^{[l]} & w_{1,2}^{[l]} & w_{1,3}^{[l]} & \dots & w_{1,n}^{[l]} \\ w_{2,1}^{[l]} & w_{2,2}^{[l]} & w_{2,3}^{[l]} & \dots & w_{2,n}^{[l]} \\ w_{3,1}^{[l]} & w_{3,2}^{[l]} & w_{3,3}^{[l]} & \dots & w_{3,n}^{[l]} \\ \dots & \dots & \dots & \dots & \dots \\ w_{n^{[l]},1}^{[l]} & w_{n^{[l]},2}^{[l]} & w_{n^{[l]},3}^{[l]} & \dots & w_{n^{[l]},n^{[l-1]}}^{[l]} \end{bmatrix} \in \mathbb{R}^{n^{[l]} \times n^{[l-1]}} \quad (1)$$

to represent the baseline values, which are usually randomly generated within a $[-1, 1]$ interval. The less complex bias vectors can be represented by a one dimensional version of the matrix seen above. In addition, the weighted sum (aka. the values parsed through the activation function) is

$$z = \sum_{i=1}^n x_i w_i + b \quad (2)$$

The weighted input can be obtained and parsed through $A(x)$ for the intermediate column vector

$$z^{[l]} = \begin{pmatrix} z_1^{[l]} \\ z_2^{[l]} \\ z_3^{[l]} \\ \cdot \\ \cdot \\ \cdot \\ z_n^{[l]} \end{pmatrix} \in \mathbb{R}^{n^{[l]}} \quad (3)$$

where there exists elements until the n th degree. To calculate the update gradient, the rate of change in loss L must be determined. Theoretically, though impractical, this can be determined via calculating the slope between two datapoints with an infinitesimal distance. The standard error can be approximated via finding the instantaneous rate of change in loss (eg. determining a partial derivative in respect to z). When

$$\frac{\Delta L}{\Delta z_k^{[l]}} \approx \frac{\partial L}{\partial z_k^{[l]}} \quad (4)$$

the calculated error can be propagated to every weight in the neural-network. Using the weighted input, loss derivative, and activation function derivative, the update gradient can be calculated using basic algebra such that

$$\frac{\Delta L}{\Delta z_k^{[l]}} = \frac{\partial L}{\partial z^{[l-1]}} = \frac{\partial L}{\partial z^{[l]}} A^{[l-1]'}(z^{[l-1]}) \quad (5)$$

across many iterations. Due to space constraints, optimization and further analysis of partial derivatives has been omitted. As can be seen, the activation function and its derivative are critical in the training of deep neural-networks (DNNs) in supervised classification, or in unsupervised classification (eg. discriminators in generative adversarial networks). Substituting various activation functions can vastly alter the minimization of loss.

2.2 Derivation and Implementation

Much like Mish, Phish is a composite function. It comprises two existing activation functions, those being TanH and GELU¹, is defined as

$$\text{GELU}(x) = \frac{x}{2} \left[1 + \frac{2}{\sqrt{\pi}} \int_0^{\frac{x}{\sqrt{2}}} e^{-t^2} dt \right] \quad (6)$$

to approximate ReLU

$$\text{ReLU}(x) = x^+ = \max(0, x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases} \quad (7)$$

such that no discontinuities occur on the differentiated graph. ReLU is perhaps the most used activation function in DNNs. It has shown to be effective in large-scale classification problems, often used in image classification.

The outer activation function TanH², is defined by

$$\text{TanH}(x) = \frac{e^x - e^{-x}}{e^{-x} + e^x} \quad (8)$$

Since Phish is expressed in terms of other equations and variables, the true form of the equation can be determined. Therefore, through substituting variables and rearranging the terms, the Phish equation in the most pure form can be defined as

$$\text{Phish}(x) = x(\text{TanH} \circ \text{GELU}) = x \frac{e^{\frac{x}{2} \left[1 + \frac{2}{\sqrt{\pi}} \int_0^{\frac{x}{\sqrt{2}}} e^{-t^2} dt \right]} - e^{-\frac{x}{2} \left[1 + \frac{2}{\sqrt{\pi}} \int_0^{\frac{x}{\sqrt{2}}} e^{-t^2} dt \right]}}{e^{-\frac{x}{2} \left[1 + \frac{2}{\sqrt{\pi}} \int_0^{\frac{x}{\sqrt{2}}} e^{-t^2} dt \right]} + e^{\frac{x}{2} \left[1 + \frac{2}{\sqrt{\pi}} \int_0^{\frac{x}{\sqrt{2}}} e^{-t^2} dt \right]}} \quad (9)$$

Using the backpropagation equation derived in the introduction, the activation function A(x) can be substituted with any activation function to simulate the calculation of update gradients. Such gradients for Phish require its derivative. Update gradient calculation can be formulated via substituting the Phish derivative.

$$\therefore \frac{\Delta L}{\Delta z_k^{[l]}} = \frac{\partial L}{\partial z^{[l-1]}} = \frac{\partial L}{\partial z^{[l]}} \text{Phish}^{[l-1]'}(z^{[l-1]}) \quad (10)$$

Based on the assumption that

$$\text{erf}(z) = \frac{2}{\pi} \int_0^z e^{-t^2} dt \quad (11)$$

where z is any complex number, the derivative can be calculated by substituting integrals, rearranging the terms, and applying the chain rule onto all sides.

$$\therefore \frac{\Delta L}{\Delta z_k^{[l]}} = \frac{\partial L}{\partial z^{[l-1]}} \times \frac{d}{dz^{[l-1]}} \left[\frac{z^{[l-1]} e^{z^{[l-1]} \left(\text{erf}\left(\frac{z^{[l-1]}}{\sqrt{2}}\right) + 1\right) - 1}}{e^{z^{[l-1]} \left(\text{erf}\left(\frac{z^{[l-1]}}{\sqrt{2}}\right) + 1\right) + 1}} \right] \quad (12)$$

3 Evaluation

A simulation was conducted to compare activation functions. The levels of independent variable were Phish, Swish, Sigmoid, and TanH. Phish was the control. The minimization of loss was studied using DNNs.

¹GELU is an approximation of the ReLU activation function defined as The main implementation of such a function is to avoid the large jump discontinuity apparent in ReLU, which occurs at the origin (0, 0) on the Cartesian coordinate system. The non-linear function seems to outperform ReLU and ELU in certain tasks in language processing and classification.

²TanH is the analogue hyperbolic tangent function often used throughout trigonometry. Similar in concept to Sigmoid, it has two horizontal asymptotes. However, these exist at $y=\pm 1$, which indicates that the domain is half negative. Therefore, TanH_+ exists only rightward of the origin (0, 0), which it crosses.

One Intel i7 computer was obtained. Python3 was installed onto the machine with machine-learning and linear algebra dependencies. For the procedure, 170,000 training and 50,000 testing images were gathered. The images were preprocessing via normalization and cropping. The pre-processing was limited to generalize the training process.

A generic neural-network was fabricated for testing. It comprised an input layer, four hidden layers, and an output layer. The output layer was always used SoftMax. The models were compiled with the Adam optimizer. Binary classification crossentropy loss

$$J(w) = \frac{-1}{n} \sum_{j=1}^n [y_j \log(\hat{y}_j) + (1 - y_j) \log(1 - \hat{y}_j)] \quad (13)$$

was substituted with SCC.

$$\therefore J(w) = - \sum_{j=1}^n y_j \log(\hat{y}_j) \quad (14)$$

where w represents the arbitrary parameters of a given network with the y values representing the predicted and true labels. This was done so the network would assume correct classifications can only be a single prediction.

SoftMax was used for the output layer. It is a deep-learning probability distribution function used in multi-class identification problems. It is

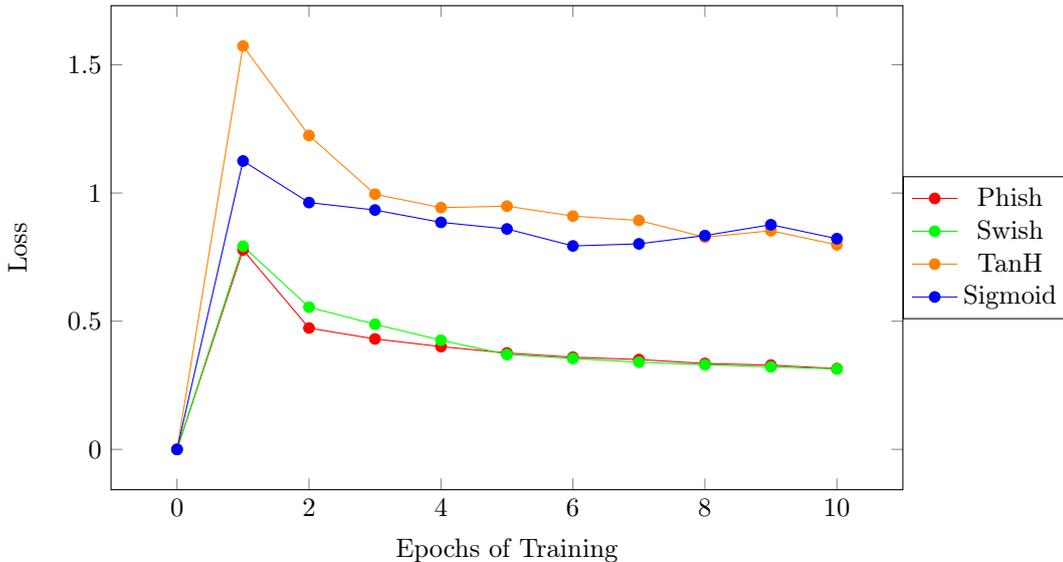
$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \quad (15)$$

where the input and output functions of the network calculate for the input vector. During testing, $K=10$ was constant because each of the databanks used had ten possible labels.

The levels of independent variable were tested. This was done twenty-five separate times for each activation function. The minimization of SCC was recorded.

This project was conducted on a laptop without graphics processing units or cloud servers. Therefore, a large scale cross-validation was not reasonable. A memoryless stochastic model was more favorable for such a purpose. Thus a Markov chain was developed to simulate the process, which can be seen in the appendix.

Graph 1: The Effect of Activation Functions on Minimizing Sparse Categorical Crossentropy



The effect of activation functions on minimizing the loss in classification for DNNs was tested. Various datasets were used to simulate classification backpropagation. Phish (red), Swish (green), Sigmoid (blue), and TanH (orange) can be seen in graph 1.

This particular graph shows the trend when training on MNIST fashion. The graph was the average loss across epochs calculated from twenty-five trials. Across the various epochs, it can be

seen that Phish and Swish had a similar minimization of SCC. TanH and Sigmoid had significantly lower reduction of loss compared to Swish and Phish. From the data collected it could be inferred that

Similar patterns were apparent when the networks trained on MNIST numbers and CIFAR-10 image databanks. Phish consistently outperformed TanH and Sigmoid. It was either on-par or slightly superior to Swish. The results of the experiment show that Phish is a promising alternative activation function.

Table 1: Statistical Analysis at 0.05 Significance Level under 48 Degrees of Freedom

T-Test	Calculated Value	Table Value	Result
Phish vs. TanH	56.331	2.011	Significant
Phish vs. Sigmoid	35.088	2.011	Significant
Phish vs. Swish	1.996	2.011	Not Significant
Swish vs. TanH	55.846	2.011	Significant
Swish vs. Sigmoid	33.785	2.011	Significant
Sigmoid vs. TanH	17.208	2.011	Significant

This data table shows the compared levels of independent variable. Six independent parametric t-tests³ were calculated to determine the significance of the data collected. The value of significance was at 0.05, and was granted 48 degrees of freedom. A table value of 2.011 was used. A null hypothesis was generated. It stated that there would be no difference between any of the tested activation functions when given the task of minimizing sparse categorical crossentropy.

Five of the six comparisons were significant. The Phish vs. Swish test was not significant, which showed that through the testing, both activation functions delivered similar results by the tenth epoch. This logic can be seen in the similarities in calculated values between Phish, Swish, and the other functions. Phish vs. TanH delivered the greatest difference in performance, with Phish on average having the lowest loss and TanH having the highest. The variance of the Swish and Sigmoid datasets were also noticeably higher than the other two functions.

4 Discussion

4.1 Procedural Flaws

There were many sources for error in the experimentation done to determine the properties of Phish. The first was that Phish was only compared to three other activation functions. Another flaw was that only one architecture was tested for classification, where many could have been tested. Other combinations of optimizers, metrics, losses, and layers may result in different findings.

In addition, a true simulation of the loss was never conducted. Only stochastic replications of limited simulations were analyzed. This was due to the limitations of devices in this research, as only a single laptop was available. While Markov chains can simulate probability, they cannot predict the change in probability.

To remedy these errors in the future, various types of classification algorithms could be tested using the activation functions. More functions could be compared as well. Lastly, better computers and cloud servers could be used to conduct the advanced simulations required to test Phish, that would otherwise impractical on a laptop.

4.2 Future Applications

Future applications of the activation function proposed in this research may vary. The first application would be further testing on types of datasets. MNIST and CIFAR-10 were used in this research. MNIST is a relatively simple dataset that most deep-learning models could solve [13]. CIFAR-10 consists of RGB images, which requires better models to solve [7]. Still, testing Phish on MNIST and CIFAR-10 only would limit knowledge on its properties. ImageNet is a

³T-tests were a favorable method to testing for significance in this dataset. Traditionally, a one way ANOVA test could imply significance. However, that is suitable for identifying differences and patterns holistically. This experiment sought to determine whether Phish is more or less effective at loss minimization. Therefore, comparing pairs of activation functions using independent t-tests was more ideal.

public databank consisting of RGB images with an average resolution of 469×387 pixels [10]. It is organized according to the WordNet hierarchy, and is often used when to test pretrained convolutional neural-networks.

Specialized layers in the networks used for testing were omitted throughout evaluation. Further testing could determine the effect of Phish on such models. Specific examples would include recurrent neural-networks. These networks were engineered to solve the vanishing gradient problem [14]. Gated recovery unit and long-short term memory algorithms are extensions of recurrent neural-networks [5]. When testing time series data, Phish could be implemented in these algorithms via substituting Sigmoid layers.

Another example of a future study would be utilizing Phish in generative adversarial networks. These algorithms comprise of two models, often multilayer perceptrons, engaging in a minimax game. The first model is the generator, which captures the distribution of a given dataset. The second one is the discriminator, which differentiates samples from the dataset and ones generated by the generator. Ideally, the loss of the discriminator would be maximized with the accuracy yielding $\frac{1}{2}$ everywhere. Testing Phish in a model with the purpose of maximizing loss would be an interesting future study [4].

5 Conclusion

Phish is a novel non-monotonic activation function. It delivered higher performance in MNIST and CIFAR-10 image classification than Sigmoid and TanH. It rivals Swish in loss minimization. The function perpetually increases without an upper bound. Its derivative is always positive. Phish evaluates calculations that increase the speed of loss minimization. Unlike ReLU, Phish is fully differentiable. Future studies could involve training generative adversarial networks with Phish and examining the performance. This project was conducted under adult approval with antivirus software.

References

- [1] Kamel Abdelouahab, Maxime Pelcat, and François Berry. Why tanh is a hardware friendly activation function for cnns. *Proceedings of the 11th International Conference on Distributed Smart Cameras*, 2017.
- [2] Abien Fred Agarap. Deep learning using rectified linear units (relu). *ArXiv*, abs/1803.08375, 2018.
- [3] Arun Kumar Dubey and Vanita Jain. Comparative study of convolution neural network’s relu and leaky-relu activation functions. *Lecture Notes in Electrical Engineering*, 2019.
- [4] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014.
- [5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.
- [6] Diganta Misra. Mish: A self regularized non-monotonic activation function. In *BMVC*, 2020.
- [7] Suyesh Pandit and Sushil Kumar. Improvement in convolutional neural network for cifar-10 dataset image classification. *International Journal of Computer Applications*, 176:25–29, 2020.
- [8] Heny Pratiwi, Agus Perdana Windarto, Susliansyah Susliansyah, Ririn Restu Aria, Susi Susilowati, Luci Kanti Rahayu, Yuni Fitriani, Agustiena Merdekawati, and Indra Riyana Rahadjeng. Sigmoid activation function in selecting the best model of artificial neural networks. 2020.
- [9] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Swish: a self-gated activation function. *arXiv: Neural and Evolutionary Computing*, 2017.

- [10] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115:211–252, 2015.
- [11] Siddharth Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. 2020.
- [12] Vinita Silaparasetty. Perceptrons. *Deep Learning Projects Using TensorFlow 2*, 2020.
- [13] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *ArXiv*, abs/1708.07747, 2017.
- [14] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *ArXiv*, abs/1409.2329, 2014.

6 Appendix

6.1 Markov Chains

A Markov chain is a stochastic graph model based in probability. They are favorable for simulating large networks of events because they are memoryless. Each chain yields a stochastic transition matrix (STM).

The Markov model on Ω results in the stochastic process $(X_0, X_1, X_2, \dots, X_t)$ in which the transition state between x and y complies with the properties

$$X_t \in \Omega, \forall t \tag{16}$$

and

$$\mathbb{P}[X_{t+1} = y | X_t = x, X_{t-1} = x_{t-1}, \dots, X_0 = x_0] = \mathbb{P}[X_{t+1} = y | X_t = x] =: P(x, y) \tag{17}$$

In addition, the STM exists with non-negativity

$$\forall x, y \in \Omega, P(x, y) \geq 0 \tag{18}$$

and stochasticity

$$\sum_{y \in \Omega} P(x, y) = 1, \forall x \in \Omega \tag{19}$$

where each row converges to 1. This Markov model is continuous, with no termination node with 100% probability of returning to itself on the graph, and a 0% chance of transferring to any other stage. In addition, there is technically an appropriate start node in this graph. However, since this Markov model will be ran for extended periods of time, the law of large numbers states that the probability of the event occurring will be affected minimally from the first event, especially since there are only two possible stages in this model. Therefore, the chance of starting at either stage was 50% always.

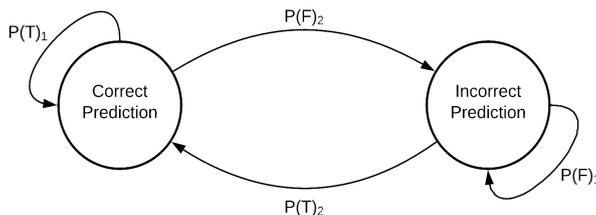


Figure 1: Markov Chain Model used for Simulating Cross-Validations for Deep Neural-Networks

The Markov chain utilized here is a two stage graph with four global locations and two local ones for each stage. An minimalistic representation of the adjacency matrix could be fabricated accordingly with 4×4 dimensions. The probability values were guaranteed using the STM

$$\omega_{ij} = \Omega = \begin{bmatrix} P_1(T) & P_2(T) \\ P_2(F) & P_1(F) \end{bmatrix} \quad (20)$$

The Markov simulation was conducted where the $P_1(T)$, $P_1(F)$, $P_2(T)$, and $P_2(F)$ were retrieved from a cross-validation. For each activation function, a DNN was trained across ten epochs. The prediction ratios were implanted into four graphs. Each graph was simulated for 10,000 iterations twenty-five times to follow the ideal experimental design.

6.2 Deep Neural-Networks

A generalized model building framework is ideal when testing narrow components of deep-learning models such as activation functions. This is to minimize the lurking/confounding variables that may be introduced via convolutional, pooling, and recurrent layers. Forget and memory gates were also omitted from the model for the same reasons.

The testing model comprised an initial flattening layer, six hidden layers, and one output layer. The flattening layer manipulated the image data into a one-dimensional array for the next layer. The six hidden layers used one of the four activation functions tested and contained between 32-128 layers each. The output layer was always ten neurons, because MNIST, and CIFAR-10 both have ten classes. It used SoftMax instead of a Sigmoid, as probability of classification was distributed between more than two classes. The models trained using sparse categorical crossentropy loss and the Adam optimizer, which combines aspects of the previously engineered AdaGrad and RMSProp methods.