PolyLU: The Over-Powered Activation Function

Anonymous Authors

Anonymous Department
Anonymous University
Anonymous Place
{anonymous}@null.edu

Abstract

We present PolyLU, the polynomial linear unit, as an activation function with optimal performance characteristics, outperforming ReLU when applied to deep models. The celebrated ReLU activation function eliminates the vanishing gradient; our work takes this a step further, culminating in the field's first trainable activation function that is also resistant to adversarial attacks. Experiments demonstrate the potential of PolyLU and support a mathematical analysis of PolyLU's optimality.

1. Introduction

As machine learning applications continue to explode in variety and in nature, the humble activation function continues to power even the deepest neural networks. The last decade propelled ReLU into the spotlight, where it has remained as the foot soldier of choice in deep networks [4]. Despite dramatic advances in network architecture, security, and evaluation, the ReLU continues to enjoy immense popularity among researchers and engineers alike.

This paper proposes PolyLU, the polynomial linear unit, to succeed ReLU as the activation function of choice. While prior work (SELU [2], Swish [5]) unequivocally demonstrates a clear and unambigious improvement to ReLU, more work needs to be done in order to harness the total potential of the activation function. This thread of research culminates in another optimization problem, this time in the context of the activation function itself.

PolyLU is the first trainable activation function, comprised of high-powered polynomial curve fitting on ReLU. Experiments show a significant boost in performance on CIFAR-10 when applied to deep models. In addition, a theoretical analysis demonstrates that PolyLU is easily trained, and precisely powerful.

2. Background: Activation Function Engineering

When an artificial neuron receives inputs, it calculates a weighted sum, adds a bias, and then passes this result to its activation function. The activation function has the key role of calculating whether or not this particular neuron should be "activated" and passes the appropriate value onto the next layer of the neural net. Inspired by the biological threshold of firing within neurons inside the brain, activation functions control the exact outputs of each node within a neural network. They are often critical in determining the performance of a neural network architecture. As such, it is imperative to tailor the appropriate activation functions to specific machine learning needs. More complex neural networks require more finely-tuned and intelligently designed activation functions, and these needs have recently given rise to the field of activation function engineering.

Since the early 1990s, sigmoid has been one of the main activation functions used in modern machine learning [3]. However, it fell out of favor due to two main issues: (1) output values that are not centered at zero causing convergence issues, and (2) vanishing gradients. The tanh unit is similar to sigmoid; although it is scaled to be zero-centered, it still causes vanishing gradients.

ReLU, the rectified linear unit, was introduced as a simpler, faster, and higher performance activation function for neural networks in 2010 [4]. Compared to the continuous sigmoid and tanh functions, ReLU is defined as *f(x) = max(0, x)*, where *x* is a linear output (see Figure 1). ReLU has become the dominant activation function used in machine learning due to being much less computationally expensive than its predecessors and maintaining a constant gradient even at higher values, making it faster to converge.

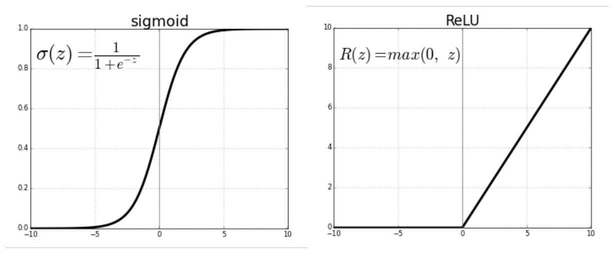

Figure 1. Sigmoid, one of the first widely used activation functions vs. ReLU, the current gold standard in activation function engineering [8].

Popular recent innovations in the field of activation function engineering include SELU and Swish. SELUs are scaled exponential linear units that act to induce self-normalization through their convergence properties [2]. Compared to explicit normalization techniques such as batch normalization, SELU will automatically converge towards zero mean and unit variance when propagated through many layers of a deep neural network, so that fully SELU networks will not have the vanishing or exploding gradient problems that ReLU architectures are often faced with. This property makes SELU better suited than ReLU for many deep neural net

DOI: http://dx.doi.org/10.1145/(to come)

architectures. SELU's implicit normalization also simplifies implementation - the performance of SELU neural networks is comparable to that of non-SELU architectures that have explicit batch, layer, and weight normalization.

Swish, or SiLU, is a sigmoid linear unit of the form $f(x) = x \cdot \sigma(\beta x)$ [5]. Swish was discovered at Google Brain using automated techniques in exhaustive and reinforcement learning-based search. Swish has been shown to far surpass both hand-designed functions and even ReLU, the current standard. For classification tasks on the ImageNet database, Swish outperforms ReLU in top-5 accuracy by an impressive 0.5% upon replacement in the Inception-ResNet-v2 architecture. It performs even better on mobile-sized nets, with a 1.4% increase on Mobile NASNet-A and 2.2% on MobileNet compared to ReLU.

While these functions were generated using architecture search, in this paper we propose using polynomial curve fitting to create a more mathematically precise model.

3. PolyLU: The over-powered activation function

We propose PolyLU (the Polynomial Linear Unit), a new general-purpose activation function designed to subsume all previous activation functions such as ReLU that are commonly used today. PolyLU is a drop-in replacement for any previous activation function, achieving provably better results while requiring little additional training time. The key insight behind PolyLU is to make the activation function trainable. By tuning the activation function to the particular network architecture and task, we can achieve better results. Formally, PolyLU is defined as follows:

$$\textsc{PolyLU}(x) = \sum_{i=0}^{n} w_i x^i$$

We first describe the theoretical properties of PolyLU. Then, we describe practical considerations in implementing PolyLU-based networks.

3.1 Theoretical properties

PolyLU subsumes all previous activation functions because PolyLU is provably optimal.

Theorem 3.1 (Optimality)**.** *PolyLU is the optimal activation function.*

Proof. Suppose $f(\cdot)$ is the optimal activation function, assumed to be continuous on a closed interval $[a, b]$, the range of inputs to the activation function in the neural network. By the Stone-Weierstrass theorem [9], this function can be uniformly approximated as closely as desired by a polynomial function. Because PolyLU models any polynomial function, PolyLU can be trained to be arbitrarily close to the optimal $f(\cdot)$ on the interval $[a, b]$. □

Even though PolyLU is proven to be optimal, one might question whether this optimal value can be efficiently found without using too much computing time or power. PolyLU would need to be comparable to ReLU in terms of computational simplicity in order to be considered a practical replacement.

Theorem 3.2 (Convergence)**.** *PolyLU can be trained to optimality.*

Proof. While PolyLU networks can be highly nonconvex, it has been shown that backpropagation [7] along with stochastic gradient descent [6] successfully optimizes these complicated, high-dimensional loss surfaces. □

A practical concern might be whether PolyLU can be efficiently computed.

Architecture	ReLU Accuracy	PolyLU
CNN 1	80%	**81%**
CNN 2	79%	**80.5%**
CNN 3	82.14%	**82.15%**

Table 1. A comparison of ReLU and PolyLU on a variety of CNN architectures for CIFAR-10. PolyLU surpasses ReLU in all experiments.

Theorem 3.3 (Efficiency)**.** *PolyLU can be computed efficiently.*

Proof. Using a naive approach, PolyLU can be computed in $O(n^2)$ time. The time complexity can be reduced to $O(n \log n)$ using the $O(\log n)$ approach for evaluating x^i. However, an even better approach is to use Horner's rule [10], which allows for computing PolyLU in $O(n)$ time, which is optimal. Applying PolyLU with degree n, where n is a constant, this translates to O(1) time, which makes PolyLU comparable to ReLU in terms of computational simplicity. □

3.2 Practical concerns

PolyLU can be a simple drop-in replacement for any other activation functions such as ReLU. Furthermore, PolyLU can be used as a replacement within a network that has already been trained. Because PolyLU is trainable and can approximate any other function, a network trained with another activation function $\sigma(\cdot)$ can be replaced with PolyLU initialized such that it approximates $\sigma(\cdot)$ on some chosen interval $[a, b]$ that includes the domain of the inputs to the activation function in the trained network. Finally, the network can be fine-tuned with PolyLU activation to reap the full benefits of using PolyLU with minimal additional computation.

4. Experimental Design

To demonstrate the efficient training methodology described in Section 3.2, we evaluate PolyLU by starting with a pre-trained state-of-the-art classifier, replace the activations with PolyLU, and fine-tune the weights of the activation over several epochs of training with a small learning rate.

Our original classifier uses ReLU activations, so we initialize PolyLU by fitting PolyLU to match ReLU.

We evaluated different power levels (choices of the parameter n), and we achieved the best performance with a power level of $n = 9001$ [1], so we present only those results.

5. Results and Discussion

We show the results in Table 4. Note that PolyLU outperforms ReLU on three distinct CNN-based CIFAR classifiers. These experiments serve as validation for our theoretical results in Theorem 3. We use an experimental power of one on a training set the 10000 examples from the CIFAR-10 test set. For brevity, we show only the first 100 weights found by PolyLU:

$0.017149893401 * x^0 +$
$0.99 * x^1 +$
$0.00888631896714 * x^2 +$
$0.0137362291066 * x^3 +$
$0.0133124824318 * x^4 +$
$0.0198756410209 * x^5 +$
$0.00665728803177 * x^6 +$
$0.0109022705649 * x^7 +$
$0.00211887918294 * x^8 +$
$0.0123567407542 * x^9 +$

$0.00127715034082 * x^{10} +$
$0.0108572310948 * x^{11} +$
$0.00971274253379 * x^{12} +$
$0.0125689069994 * x^{13} +$
$0.0154566111239 * x^{14} +$
$0.00917150115876 * x^{15} +$
$0.00703062395675 * x^{16} +$
$0.00450949707661 * x^{17} +$
$0.00697652533347 * x^{18} +$
$0.00787468197186 * x^{19} +$
$0.0105139318401 * x^{20} +$
$0.0153303462748 * x^{21} +$
$0.0121670826804 * x^{22} +$
$0.0134887492809 * x^{23} +$
$0.014309604283 * x^{24} +$
$0.00618027251261 * x^{25} +$
$0.00166441532459 * x^{26} +$
$0.00178738625539 * x^{27} +$
$0.0134529528225 * x^{28} +$
$0.00819381179944 * x^{29} +$
$0.00873304426332 * x^{30} +$
$0.0190094556031 * x^{31} +$
$0.00129095455701 * x^{32} +$
$0.002341755654 * x^{33} +$
$0.00457339590339 * x^{34} +$
$0.0190407487358 * x^{35} +$
$0.00716893696706 * x^{36} +$
$0.00634182024273 * x^{37} +$
$0.00737034689557 * x^{38} +$
$0.00985595111914 * x^{39} +$
$0.0163211291087 * x^{40} +$
$0.000552493172454 * x^{41} +$
$5.28698472221237 * x^{42} +$
$0.0145356735677 * x^{43} +$
$0.0100844322799 * x^{44} +$
$0.0138138688979 * x^{45} +$
$0.00855620512433 * x^{46} +$
$0.0119446599231 * x^{47} +$
$0.016533640719 * x^{48} +$
$0.00677264108993 * x^{49} +$
$0.0161633208991 * x^{50} +$
$0.00419290655152 * x^{51} +$
$0.0185481394386 * x^{52} +$
$0.0198289242752 * x^{53} +$
$0.0151790190865 * x^{54} +$
$0.0127856411068 * x^{55} +$
$0.0166022890265 * x^{56} +$
$0.00190646326156 * x^{57} +$
$0.00632449772675 * x^{58} +$
$0.00806645149633 * x^{59} +$
$0.0197523113738 * x^{60} +$
$0.0131907782688 * x^{61} +$
$0.015286155723 * x^{62} +$
$0.00610818159537 * x^{63} +$
$0.00946131619365 * x^{64} +$
$0.0148055074097 * x^{65} +$
$0.0160995234252 * x^{66} +$
$0.016712190108 * x^{67} +$
$0.0135882313038 * x^{68} +$
$0.00288670763929 * x^{69} +$
$0.0136827136357 * x^{70} +$
$0.00925506284182 * x^{71} +$
$0.00416850801753 * x^{72} +$
$0.00219344177901 * x^{73} +$
$0.00821249496301 * x^{74} +$
$0.00717496068958 * x^{75} +$
$0.00978992638136 * x^{76} +$
$0.0105777596841 * x^{77} +$
$0.00878525838238 * x^{78} +$
$0.00145020565724 * x^{79} +$
$0.0174595742034 * x^{80} +$
$0.00692136907843 * x^{81} +$
$0.00656003476901 * x^{82} +$
$0.0102389741237 * x^{83} +$
$0.00890385889976 * x^{84} +$
$0.00356434449419 * x^{85} +$
$0.000609065174793 * x^{86} +$
$0.0187131891061 * x^{87} +$
$0.013509613309 * x^{88} +$
$0.0193990924049 * x^{89} +$
$0.0124852607295 * x^{90} +$
$0.0127394197891 * x^{91} +$
$0.0144571006024 * x^{92} +$
$0.00409722285667 * x^{93} +$
$0.0131037377593 * x^{94} +$
$0.000833472719764 * x^{95} +$
$0.0142001653643 * x^{96} +$
$0.00880959784533 * x^{97} +$
$0.00451920701253 * x^{98} +$
$0.00855274167317 * x^{99} + O(x^{100})$

6. Conclusion

We contribute PolyLU, a general purpose high-dimensional polynomial fitted activation function. Through a theoretical analysis as well as experiments on CIFAR-10, we show that PolyLU is strictly better than ReLU and other proposed functions in the activation function engineering domain when applied to deep models. Notably, PolyLU outperforms ReLU at top 1 classification tasks by a non-trivial 1.0% upon replacement in a standard CNN architecture.

Not only does PolyLU provide a boost in performance, PolyLU neural networks are also more resistant to attacks with adversarial examples due to the increased computational power required for an adversary to differentiate through the activations. We are relu optimistic that, with a power level over 9000, PolyLU's improvement in performance will overpower the current standards in activation function engineering in extrapolating to other challenging data sets and architectures as well.

References

[1] Dragonball Wiki. It's over 9000! URL http://dragonball.wikia.com/wiki/It%27s_Over_9000!

[2] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-normalizing neural networks. *CoRR*, abs/1706.02515, 2017. URL http://arxiv.org/abs/1706.02515.

[3] H. N. Mhaskar and C. A. Micchelli. How to choose an activation function. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, pages 319–326, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc. URL http://dl.acm.org/citation.cfm?id=2987189.2987230.

[4] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress. ISBN 978-1-60558-907-7. URL http://dl.acm.org/citation.cfm?id=3104322.3104425.

[5] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. *CoRR*, abs/1710.05941, 2017. URL http://arxiv.org/abs/1710.05941.

[6] H. Robbins and S. Monro. A stochastic approximation method. *Ann. Math. Statist.*, 22(3):400–407, 09 1951. doi: 10.1214/aoms/1177729586. URL https://doi.org/10.1214/aoms/1177729586.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[8] S. Sharma. Activation functions: Neural networks – towards data science, Sep 2017. URL https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[9] M. H. Stone. The generalized weierstrass approximation theorem. *Mathematics magazine*, 21:167–183; 237–254, 1948.

[10] W. Van Assche. P. borwein and t. erdélyi;polynomials and polynomial inequalities. *J. Approx. Theory*, 86(3):361–363, Sept. 1996. ISSN 0021-9045. doi: 10.1006/jath.1996.0078. URL http://dx.doi.org/10.1006/jath.1996.0078.