

POLYNOMIAL ACTIVATION FUNCTIONS

Anonymous authors

Paper under double-blind review

ABSTRACT

Activation is a nonlinearity function that plays a predominant role in the convergence and performance of deep neural networks. While Rectified Linear Unit (ReLU) is the most successful activation function, its derivatives have shown superior performance on benchmark datasets. In this work, we explore the polynomials as activation functions ($order \geq 2$) that can approximate continuous real valued function within a given interval. Leveraging this property, the main idea is to learn the nonlinearity, accepting that the ensuing function may not be monotonic. While having the ability to learn more suitable nonlinearity, we cannot ignore the fact that it is a challenge to achieve stable performance due to exploding gradients - which is prominent with the increase in order. To handle this issue, we introduce dynamic input scaling, output scaling, and lower learning rate for the polynomial weights. Moreover, lower learning rate will control the abrupt fluctuations of the polynomials between weight updates. In experiments on three public datasets, our proposed method matches the performance of prior activation functions, thus providing insight into a network's nonlinearity preference.

1 INTRODUCTION

Deep learning methods have achieved excellent results in visual understanding, visual recognition, speech, and natural language processing tasks (Krizhevsky et al. (2012), Lee et al. (2014), Goodfellow et al. (2014), Hochreiter & Schmidhuber (1997), Oord et al. (2016), Vaswani et al. (2017)). The convolutional neural networks (CNNs) first introduced in LeCun et al. (1999), is the foundation for numerous vision tasks. While recurrent neural networks, wavenet and the recent transformers with attention mechanism are the core algorithms used in speech and natural language processing. The commonality is the importance of deeper architectures that has both theoretical and empirical evidence (Serre et al. (2007), Simonyan & Zisserman (2015), Lee et al. (2014)).

One essential component for deep neural networks is the activation function that enables nonlinearity. While ReLUs are the most used nonlinearity, sigmoid and hyperbolic tangent are the traditional functions. Several derivatives of ReLU are presented in recent years that further improve the performance and minimize vanishing gradients issue (Maas et al. (2013), He et al. (2015a), Clevert et al. (2015), Ramachandran et al. (2019)). While most are fixed functions, the negative slope for Leaky ReLUs can be adjusted during the network design, and remains constant while training. Parametric ReLU adaptively changes the negative slope during training using a trainable parameter and demonstrate a significant boost in performance (He et al. (2015a)). A relatively new activation function, Swish, is derived by an automated search techniques (Ramachandran et al. (2019)). While the parameter β enables learning, the performance difference reported in the study between parametric and non-parametric versions is minimal. To this end, rather than using a fixed or heavily constrained nonlinearity, we believe that the nonlinearity learned by the deep networks can provide more insight on how they can be designed.

In this work, we focus on the use of polynomials as nonlinearity functions. We demonstrate the stability of polynomial of orders 2 to 9 by introducing scaling functions and initialization scheme that approximates well known activation functions. Experiments on three public datasets show that our method competes with state-of-the-art activation functions on a variety of deep architectures. Despite their imperfections, our method allows each layer to find their preferred nonlinearity during training. Finally, we show the learned nonlinearities that are both monotonic and non-monotonic.

2 RELATED WORK

In this section we review activation functions and their relative formulation. Sigmoid ($\sigma(x) = (1 + \exp(-x))^{-1}$), and the hyperbolic tangent ($\tanh(x) = (\exp(x) - \exp(-x)) / (\exp(x) + \exp(-x))$), are the usual nonlinearity functions for neural networks. However, the deeper networks suffer vanishing gradient issue (Bengio et al. (1994)) that have near zero gradients at the initial layers. Softplus ($f(x) = \log(1 + \exp(x))$) initially proposed in Dugas et al. (2000), is a smoother version of ReLU whose derivative is a sigmoid function.

Unlike sigmoid or tanh, ReLU ($ReLU(x) = \max(x, 0)$) can have gradient flow as long as the inputs are positive (Hahnloser et al. (2000), Nair & Hinton (2010), Glorot et al. (2011)). An extension of ReLU, called Leaky ReLUs (LReLU), allows a fraction of negative part to speed-up the learning process by avoiding the constant zero gradients when $x < 0$ (Maas et al. (2013)). LReLU are relatively popular for generative adversarial networks (Radford et al. (2015)).

$$LReLU(x) = \begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \quad \text{where } \alpha = 1.0$$

While the slope (α) is constant for LReLU, it is a learnable parameter for Parametric ReLU (PReLU), which has achieved better performance on image benchmark datasets (He et al. (2015a)). However, Exponential Linear Unit (ELU), another derivative of ReLU, has improved learning by shifting the mean towards zero and ensuring a noise-robust deactivation state (Clevert et al. (2015)).

$$ELU(x) = \begin{cases} \alpha(\exp(x) - 1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \quad \text{where } \alpha = 1.0$$

Gaussian Error Linear Unit, defined by $GeLU(x) = x\Phi(x)$, is a non-monotonic nonlinearity function (Hendrycks & Gimpel (2016)), where $\Phi(x)$ is the cumulative distribution function. Scaled Exponential Linear Unit (SELU) with self-normalizing property delivers robust training of deeper networks Klambauer et al. (2017).

$$SELU(x) = \lambda \begin{cases} \alpha \exp(x) - \alpha & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \quad \text{where } \lambda = 1.0507 \text{ and } \alpha = 1.6733$$

Unlike any of the above, Swish is a result of automated search technique Ramachandran et al. (2019). Using a combination of exhaustive and reinforcement learning based search techniques (Bello et al. (2017), Zoph & Le (2016)), the authors reported eight novel activation functions ($x * \sigma(\beta x)$, $\max(x, \sigma(x))$, $\cos(x) - x$, $\min(x, \sin(x))$, $(\tan^{-1}(x))^2 - x$, $\max(x, \tanh(x))$, $\text{sinc}(x) + x$, and $x * (\sinh^{-1}(x))^2$). Of which, $x * \sigma(\beta x)$ (Swish) and $\max(x, \sigma(x))$ matched or outperformed ReLU in CIFAR experiments, with the former showing better performance on ImageNet and machine translation tasks. The authors claim that the non-monotonic bump, controlled by β , is an important aspect of Swish. β can either be a constant or trainable parameter. To our knowledge this is the first work to search activation functions.

In contrast, we use polynomials with trainable coefficients to learn activation functions that have the ability to approximate a well-known or novel continuous nonlinearity. Furthermore, we demonstrate its stability and performance on three public benchmark datasets. We emphasize that our goal is to understand a network’s nonlinearity preference given the ability to learn.

3 POLYNOMIAL ACTIVATION

An n^{th} order polynomial with trainable weights $w_j, j \in 0, 1, \dots, n$ and its derivatives are defined as follows:

$$f_n(x) = \sum_{i=1}^n w_i * x^i = w_0 * x^0 + w_1 * x^1 + \dots + w_n * x^n$$

$$\frac{df_n(x)}{dx} = w_1 + w_2 * 2 * x + \dots + w_n * n * x^{n-1}$$

Given an input with zero mean and unit variance, a deeper network with multiple $f_n(x)$ suffer exploding gradients as $f_n(x)$ scales exponentially with increase in order (n) for $x > 1$ ($\lim_{x \rightarrow \infty} f_n(x)/x = \infty$). While this can be avoided by using $\text{sigmoid}(x)$, the approximations of $f_n(\text{sigmoid}(x))$ are limited for lower weights and saturates for higher weights. Moreover, the lower weights can suffer from vanishing gradients and the higher weights because of exploding gradients. To circumvent the exploding weights and gradients, we introduce dynamic input scaling, $g(\cdot)$. Consider $\mathbf{x}_i, i \in 1, 2, \dots, K$ is the output of a dense layer with K neurons, we define the dynamic input scaling $g(\cdot)$ as follows:

$$g(\mathbf{x}_i) = \frac{\sqrt{2} * \mathbf{x}_i}{\max_{1 \leq i \leq k} |\mathbf{x}_i|}$$

Essentially, we constrain $g(\cdot)$, such that $\max_{1 \leq i \leq k} |g(\mathbf{x}_i)| = \sqrt{2}$, regardless of the input dimensions. The choice of $\sqrt{2}$ allowed us to contain $f_n(g(x))$ for higher order's with larger w_n , especially in our mixed precision training. The primary advantage of the $g(\cdot)$ is that for any given order of the polynomial, we have $\max(f_n(g(\mathbf{x}_i))) \leq \|w_j\|_2 * \sqrt{2}^n$; this constraint allows us to normalize the output, resulting in $f_n(g(\mathbf{x}_i))/\sqrt{2}^n \leq \|w_j\|_2$. Observing the experiments in Section 4, we limit $\|w_j\|_2 \leq 3$ which allows us to explore prior activations as initializations. Essentially, when $\|w_j\|_2 > 3$, we renormalize the weights using $w_j * 3/\|w_j\|_2$. Therefore, we define n^{th} order polynomial activations as:

$$\begin{aligned} \text{PolynomialActivation}(\mathbf{x}_i) &= \frac{f_n(g(\mathbf{x}_i))}{\sqrt{2}^n} \\ &= \frac{w_0 * g(\mathbf{x}_i)^0 + w_1 * g(\mathbf{x}_i)^1 + \dots + w_n * g(\mathbf{x}_i)^n}{\sqrt{2}^n} \end{aligned}$$

Unlike the usual, polynomial activation is not for a scalar value and of course, not as simple as ReLU. The purpose of our proposal is to explore newer nonlinearities by allowing the network to choose an appropriate nonlinearity for each layer and eventually, design better activation functions on the gathered intuitions.

4 WEIGHT INITIALIZATION FOR POLYNOMIAL ACTIVATIONS

Initialization of polynomial weights play a significant role in the stability and convergence of networks. We start by examining this issue with random initializations. In this scenario, we start with Network-1 defined in Section 5.1 and polynomial activation with $n = 2$. We train all the networks using stochastic gradient descent (SGD) with a learning rate of 0.1 and a batch size of 256. To speed-up this experiment, we use mixed precision training. By varying $w_i \in \{-1, -0.95, 0.90, \dots, 1\}$, we train each possible combinations ($41^3 = 68921$) of initialization for an epoch. We use same initialization for all the three activations. The weights of both convolution and linear layers are initialized using He et al. (2015a) for each run. Given the simplicity of MNIST hand written digits dataset, we consider any network with test error of $\leq 10\%$ as fast converging initialization. In comparison, all the networks reported in Table2 achieved a test error of $\leq 5\%$ after the first epoch. One interesting observation with this experiment is that the networks initialized with $f_n(x) \approx x$ and $f_n(x) \approx -x$ have always achieved a test error of $\leq 5\%$. We extended our experiment to $n = 3$ with $w_i \in \{-1, -0.8, -0.6, \dots, 1\}$ and observed the same. Our second observation is that the weights closer to 0 never converged.

To this end, rather than initializing the polynomial weights with a normal or uniform distribution, we instead used weights that can approximate an activation function, $F(x)$. The advantage is that we can start with some of the most successful activation functions, and gradually allow the weights to choose an appropriate nonlinearity. It is important to understand that the local minima for $f_n(x) \approx F(x)$ is not guaranteed since x does not satisfy boundary conditions, regardless of it being continuous (Weierstrass theorem). While lower order polynomial suffer more, failure often leads to $f_n(x) \approx x$, which is stable.

Along with $f_n(x) = x$ ($w_1 = 1$), we investigate ReLU, Swish ($\beta = 1$) and TanH approximations as initializations that are derived by minimizing the sum of squared residuals of $f_n(x) - ReLU(x)$, $f_n(x) - Swish(x)$, and $f_n(x) - tanh(x)$ using the least squares within the bounds $-5 \leq x \leq 5$. Essentially, for an n^{th} order polynomial, we initialize the w_j such that the $f_n(x) \approx ReLU(x)$, $f_n(x) \approx Swish(x)$, or $f_n(x) \approx tanh(x)$. Figure 1 shows the approximations of ReLU, Swish and TanH using polynomials of orders 2 to 9, and their respective initializations used in this experiment are in Table 1. On observation, Swish approximations are relatively more accurate when compared to ReLU and TanH.

Using Network-1 and Network-2 defined in Section 5.1, we evaluate the stability and performance of each of the four initializations for order’s 2 to 9 under the following optimization setting: **1)** SGD with a learning rate of 0.1 **2)** SGD with a learning rate of 0.05 **3)** SGD with a learning rate of 0.01 **4)** Adam (Kingma & Ba (2014)) with a learning rate of 0.001. By varying the batch size from 2^3 to 2^{10} , we trained Network-1 and Network-2 in each setting for 10 epochs to minimize cross entropy loss on MNIST data. In total, we train 512 configurations (2 different networks * 8 different orders * 4 different optimization settings * 8 different batch sizes) per initialization, and compute the test error at the end of 10^{th} epoch.

We observed that $\sim 22\%$ of our networks with order ≥ 6 failed to converge. On monitoring the gradients, we reduced the learning rate (lr) for the polynomial weights to $lr * 2^{-0.5*order}$ ¹, resulting in the convergence of all the 512 configurations. The issue is that the polynomials fluctuate heavily between the updates due to high backpropogated gradients.

The networks whose activations are initialized with $f_n(x) \approx Swish(x)$ outperformed in 167 experiments, followed by $f_n(x) \approx ReLU(x)$ in 155, $f_n(x) = x$ in 148 and $f_n(x) \approx tanh(x)$ in 42. The results suggest that the Swish approximations as initializations are marginally better when compared to the rest. It is important to note that the activation after training does not resemble Swish or any other approximations that are used during the initialization. Instead, allows the network to converge faster.

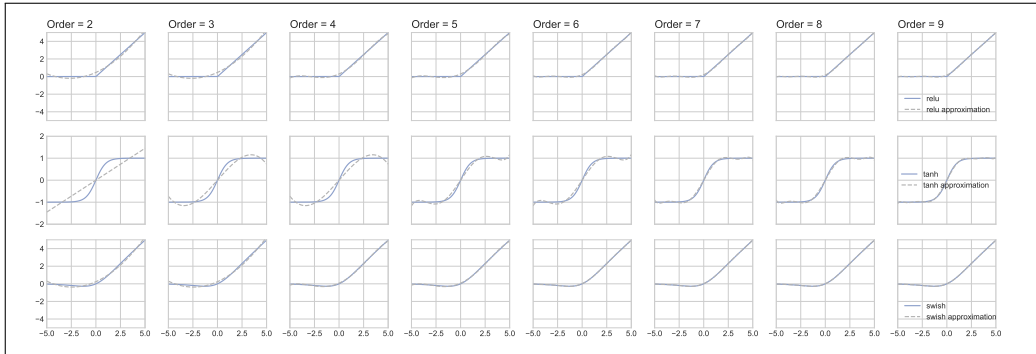


Figure 1: ReLU, Tanh, and Swish approximations derived for initializing polynomial activations.

5 BENCHMARK RESULTS

We evaluate the polynomial activations on three public datasets and compare our work with seven state-of-art activation functions reported in the literature. On CIFAR, we present the accuracies of the networks with polynomial activations presented in Ramachandran et al. (2019).

Here on, we initialize polynomial activations with $f_n(x) \approx Swish(x)$ weights. We set $\alpha = 0.01$ for LReLU, $\alpha = 1.0$ for ELUs, $\lambda = 1.0507$ and $\alpha = 1.6733$ for SELU, and a non-trainable $\beta = 1$ for Swish (Swish-1). We adopt the initialization proposed in He et al. (2015a) for both convolution and linear layers, and implement all the models in PyTorch.

¹Gradient clipping was not effective in our experiments.

Table 1: Derived coefficients used to initialize the weights of polynomial activations by approximating ReLU, Swish and TanH

Activation	Approximation
ReLU	$f_2(x) = 0.47 + 0.50 * x + 0.09 * x^2$
Swish	$f_2(x) = 0.24 + 0.50 * x + 0.10 * x^2$
TanH	$f_2(x) = 2.5e^{-12} + 0.29 * x - 4.0e^{-12} * x^2$
ReLU	$f_3(x) = 0.47 + 0.50 * x + 0.09 * x^2 - 1.7e^{-10} * x^3$
Swish	$f_3(x) = 0.24 + 0.50 * x + 0.10 * x^2 - 1.2e^{-10} * x^3$
TanH	$f_3(x) = 1.0e^{-9} + 0.51 * x - 2.2e^{-10} * x^2 - 0.01 * x^3$
ReLU	$f_4(x) = 0.29 + 0.50 * x + 0.16 * x^2 + 1.8e^{-10} * x^3 - 3.3e^{-3} * x^4$
Swish	$f_4(x) = 0.07 + 0.50 * x + 0.17 * x^2 + 1.8e^{-10} * x^3 - 3.2e^{-3} * x^4$
TanH	$f_4(x) = -1.1e^{-8} + 0.51 * x + 4.5e^{-9} * x^2 - 0.01 * x^3 - 1.5e^{-10} * x^4$
ReLU	$f_5(x) = 0.29 + 0.50 * x + 0.16 * x^2 - 1.6e^{-8} * x^3 - 3.3e^{-3} * x^4$ $+ 7.5e^{-10} * x^5$
Swish	$f_5(x) = 0.07 + 0.50 * x + 0.17 * x^2 - 1.5e^{-8} * x^3 - 3.2e^{-3} * x^4$ $+ 6.9e^{-10} * x^5$
TanH	$f_5(x) = 3.0e^{-8} + 0.67 * x - 2.4e^{-8} * x^2 - 0.04 * x^3 + 1.7e^{-9} * x^4$ $+ 1.1e^{-3} * x^5$
ReLU	$f_6(x) = 0.21 + 0.50 * x + 0.23 * x^2 + 7.6e^{-8} * x^3 - 1.1e^{-2} * x^4$ $- 3.5e^{-9} * x^5 + 2.3e^{-4} * x^6$
Swish	$f_6(x) = 0.02 + 0.50 * x + 0.21 * x^2 - 1.0e^{-7} * x^3 - 8.1e^{-3} * x^4$ $+ 3.7e^{-9} * x^5 + 1.4e^{-4} * x^6$
TanH	$f_6(x) = -3.7e^{-8} + 0.67 * x + 2.6e^{-8} * x^2 - 0.04 * x^3 - 4.9e^{-9} * x^4$ $+ 1.1e^{-3} * x^5 + 3.4e^{-10} * x^6$
ReLU	$f_7(x) = 0.21 + 0.50 * x + 0.23 * x^2 + 1.4e^{-6} * x^3 - 1.1e^{-2} * x^4$ $- 1.2e^{-7} * x^5 + 2.3e^{-4} * x^6 + 2.7e^{-9} * x^7$
Swish	$f_7(x) = 0.02 + 0.50 * x + 0.21 * x^2 + 4.8e^{-8} * x^3 - 8.1e^{-3} * x^4$ $- 6.5e^{-9} * x^5 + 1.4e^{-4} * x^6 + 1.9e^{-10} * x^7$
TanH	$f_7(x) = -2.2e^{-6} + 0.79 * x - 3.4e^{-7} * x^2 - 0.09 * x^3 - 1.2e^{-10} * x^4$ $+ 4.8e^{-3} * x^5 + 8.5e^{-10} * x^6 - 9.1e^{-5} * x^7$
ReLU	$f_8(x) = 0.17 + 0.50 * x + 0.29 * x^2 - 1.4e^{-7} * x^3 - 2.6e^{-2} * x^4$ $+ 9.7e^{-9} * x^5 + 1.2e^{-3} * x^6 - 2.0e^{-10} * x^7 - 2.1e^{-5} * x^8$
Swish	$f_8(x) = 6.5e^{-3} + 0.50 * x + 0.23 * x^2 + 2.2e^{-8} * x^3 - 0.01 * x^4$ $- 1.7e^{-9} * x^5 + 4.7e^{-4} * x^6 + 4.0e^{-11} * x^7 - 7.0e^{-6} * x^8$
TanH	$f_8(x) = 1.3e^{-6} + 0.79 * x - 8.0e^{-7} * x^2 - 0.09 * x^3 + 1.3e^{-7} * x^4$ $+ 4.8e^{-3} * x^5 - 8.2e^{-9} * x^6 - 9.1e^{-5} * x^7 + 1.6e^{-10} * x^8$
ReLU	$f_9(x) = 0.17 + 0.50 * x + 0.29 * x^2 + 9.0e^{-8} * x^3 - 2.6e^{-2} * x^4$ $- 1.2e^{-8} * x^5 + 1.2e^{-3} * x^6 + 6.5e^{-10} * x^7 - 2.1e^{-5} * x^8 - 1.1e^{-11} * x^9$
Swish	$f_9(x) = 6.5e^{-3} + 0.50 * x + 0.23 * x^2 - 1.3e^{-8} * x^3 - 0.01 * x^4$ $+ 1.8e^{-9} * x^5 + 4.7e^{-4} * x^6 - 9.5e^{-11} * x^7 - 7.0e^{-6} * x^8 + 1.7e^{-12} * x^9$
TanH	$f_9(x) = -5.7e^{-8} + 0.87 * x + 1.0e^{-7} * x^2 - 0.13 * x^3 - 2.3e^{-8} * x^4$ $+ 1.2e^{-2} * x^5 + 1.6e^{-9} * x^6 - 5.0e^{-4} * x^7 - 3.3e^{-11} * x^8 + 7.8e^{-6} * x^9$

5.1 MNIST

The MNIST dataset consists greyscale handwritten digits (0-9) of size 28 x 28 pixel, with 60,000 training and 10,000 test samples, no augmentation was used for this experiment. We consider four different networks to understand the behaviour of the proposed activation for orders 2 to 9 on network depth. In each network, there are k convolutional layers (ends with an activation function), followed by a dropout ($p = 0.2$), a linear with 64 output neurons, an activation, and a softmax layer. The number of convolutional layers, k , in Network-1, Network-2, Network-3 and Network-4 is two, four, six and eight, respectively. The choice of networks is driven by our curiosity to investigate the

stability of the proposed activation function. The filter size in the first convolutional layer is 7x7, and then the rest are 3x3 filters. The number of output channels in the first $k/2$ convolutional layers is 64, and then the rest are 128. The subsampling is performed by convolutions with a stride of 2 at $k/2$ and k layer. We train each configuration for 60 epochs using stochastic gradient descent with a momentum of 0.9. The initial learning rate is set to 0.01 and reduced to 0.001 at epoch 40². Realizing the effect of learning rate on polynomial activations from the weight initialization experiments, we use a lower learning rate only for the polynomial weights, $lr * 2^{-0.5*order}$.

Table2 shows the test error (median of 5 different runs) measured at the end of the final epoch. Polynomial activations with order 2 and 3 either match or outperform other activation functions. While our method performs well overall, we observed that the order greater than three is unnecessary. It does not improve accuracy, and increases both complexity and computation cost. Figure 2 shows the learned polynomial activation functions for network’s 1, 2 and 3 across two different runs. While lower order nonlinearities are similar to parabola, higher orders tend to avoid most information. We believe the reason to avoid information by the higher orders comes from the simplicity of the data. Overall, polynomial activations and PReLUs have performed better.

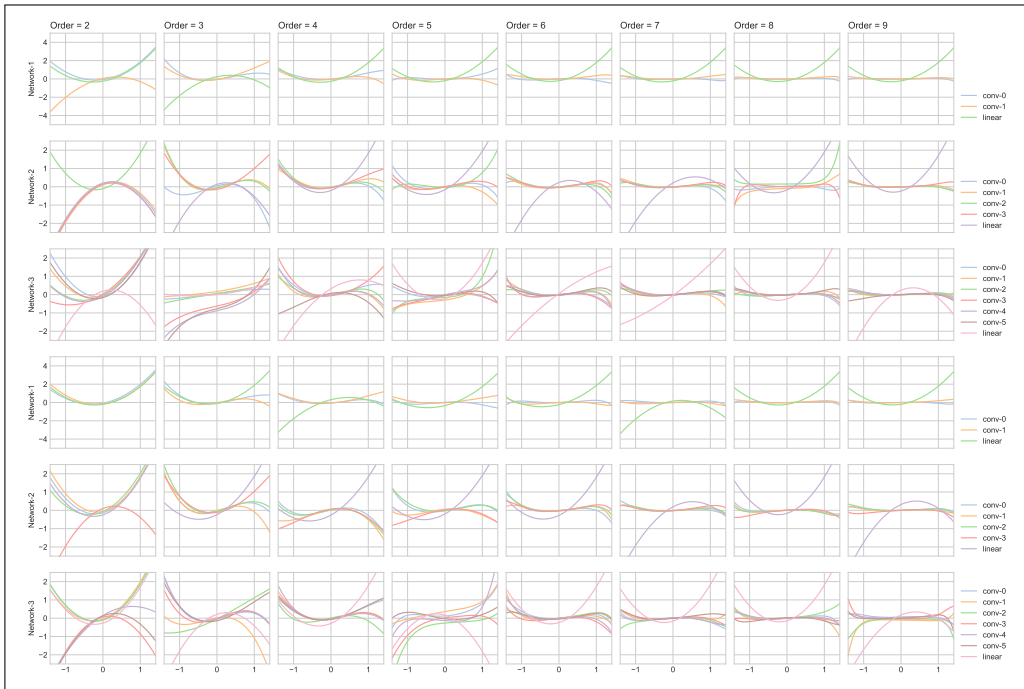


Figure 2: Learned polynomial activations across two different runs by Network-1 (1st and 4th row), Network-2 (2nd and 5th row), and Network-3 (3rd and 6th row) on MNIST dataset. Best viewed in color.

5.2 CIFAR

The CIFAR-10 and CIFAR-100 datasets consists of color images from 10 and 100 labels, respectively (Krizhevsky (2009)). There are 50,000 training and 10,000 test samples with a resolution of 32 x 32 pixels. The training data is augmented with random crop and random horizontal flip (Lee et al. (2014)). We extend the experiments reported on CIFAR in Ramachandran et al. (2019) with polynomial activations using the ResNet-164 (R164) (He et al. (2015b)), Wide ResNet 28-10 (WRN) (Zagoruyko & Komodakis (2016)), and DenseNet 100-12 (Dense) (Huang et al. (2017)) models. We replicate the architecture (with two changes) and training setting from the original papers, and switch the ReLU with polynomial activations. The two changes are 1. lower the learning rate only for the polynomial weights, 2. disable the polynomial activation right before the average

²Initial learning rate for ELU, PReLU and SELU is dropped to 0.005 due to convergence issues.

Table 2: Comparison of activation functions on MNIST

Method	Network-1	Network-2	Network-3	Network-4
ReLU	0.77	0.66	0.51	0.50
LReLU	0.74	0.57	0.54	0.59
PReLU	0.71	0.55	0.45	0.51
ELU	0.86	0.54	0.58	0.68
GELU	0.79	0.63	0.55	0.67
SELU	0.76	0.69	0.63	0.68
Swish-1	0.75	0.63	0.66	0.52
Ours ($n=2$)	0.64	0.48	0.51	0.55
Ours ($n=3$)	0.61	0.50	0.54	0.46
Ours ($n=4$)	0.69	0.62	0.69	0.66
Ours ($n=5$)	0.71	0.58	0.59	0.49
Ours ($n=6$)	0.67	0.65	0.58	0.50
Ours ($n=7$)	0.83	0.59	0.62	0.71
Ours ($n=8$)	0.81	0.69	0.68	0.54
Ours ($n=9$)	0.75	0.71	1.03	0.58

Table 3: Comparison of various activation functions performance on CIFAR using ResNet-164 (R164), Wide ResNet 28-10 (WRN), and DenseNet 100-12 (Dense)

Method	CIFAR10			CIFAR100		
	R164	WRN	Dense	R164	WRN	Dense
Softplus	94.6*	94.9*	94.7*	76.0*	78.4*	83.7*
ReLU	93.8*	95.3*	94.8*	74.2*	77.8*	83.7*
LReLU	94.2*	95.6*	94.7*	74.2*	78.0*	83.3*
PReLU	94.1*	95.1*	94.5*	74.5*	77.3*	81.5*
ELU	94.1*	94.1*	94.4*	75.0*	76.0*	80.6*
GELU	94.3*	95.5*	94.8*	74.7*	78.0*	83.8*
SELU	93.0*	93.2*	93.9*	73.2*	74.3*	80.8*
Swish-1	94.7*	95.5*	94.8*	75.1*	78.5*	83.8*
Swish	94.5*	95.5*	94.8*	75.1*	78.0*	83.9*
Ours ($n = 2$)	94.0	94.7	93.7	75.1	78.6	79.5
Ours ($n = 3$)	93.6	95.6	94.1	74.6	77.8	78.9

*Reported in Ramachandran et al. (2019)

pool as the gradients are usually high during backpropogation. We train using SGD with an initial learning rate of 0.1, a momentum of 0.9 and a weight decay of 0.0005.

Table3 shows the test accuracy (median of 5 different runs) measured at the end of final epoch for both CIFAR-10 and CIFAR-100. The results are comparable and the best performance occurs with Wide ResNet which is shallow when compared to DenseNet and ResNet with relatively fewer parameters. In this case, most activations are non-monotonic signifying its importance stated in Ramachandran et al. (2019). We also observe that the information allowed through initial layers is lower when compared to the deeper layers. One reason is that the residual connections is common in all the networks. Our numbers on CIFAR100 for DenseNet is lower when compared to Ramachandran et al. (2019), however, we are comparable to the original implementation that reported an accuracy of 79.80 using ReLUs (Huang et al. (2017)).

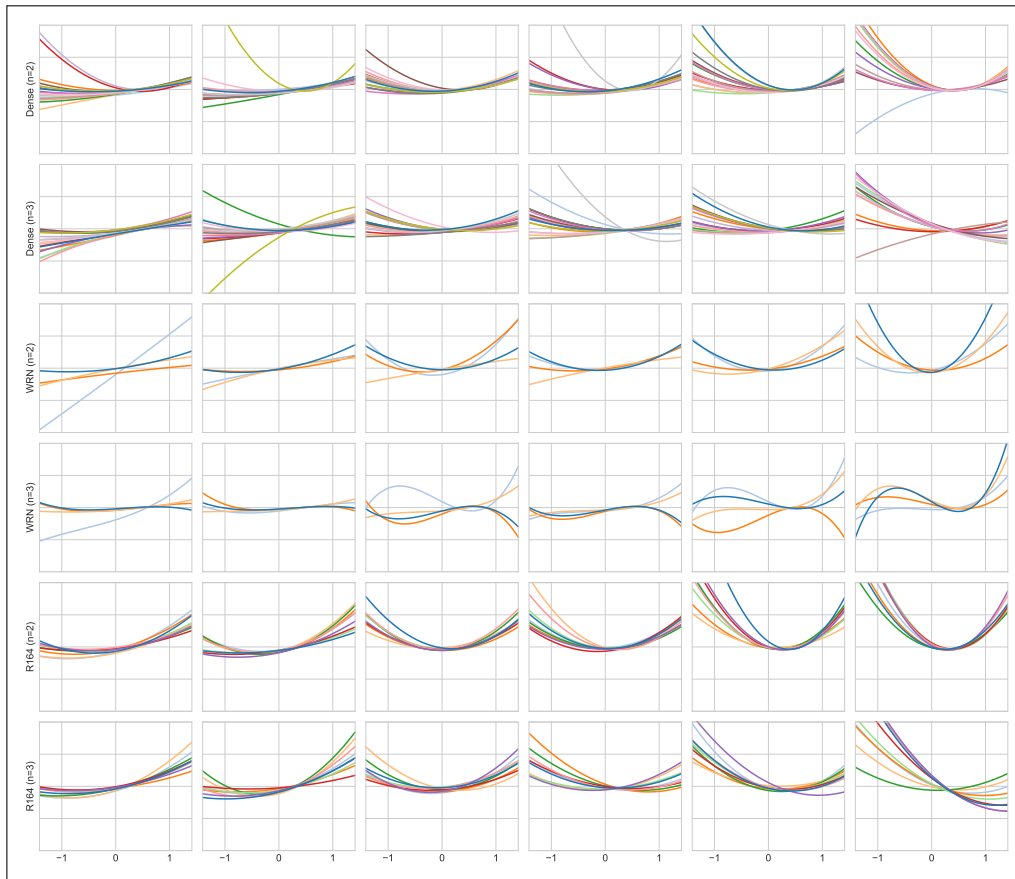


Figure 3: Learned polynomial activations on CIFAR10 data. Each row represents activations for a given network. All the activations in a network are split into 6 columns for better visibility. For R164, we only plot the first activation of each residual block for better visibility. Best viewed in color.

6 CONCLUSIONS

We proposed a polynomial activation function that learns the nonlinearity using trainable coefficients. Our contribution is stabilizing the networks with polynomial activation as a nonlinearity by introducing scaling, initialization technique and applying a lower learning rate for the polynomial weights, which provides more insight about the nonlinearity preferred by networks. The resulting nonlinearities are both monotonic and non-monotonic in nature. In our MNIST experiments, we showed the stability of our method with orders 2 to 9 and achieved superior performance when compared to ReLUs, LReLUs, PReLUs, ELUs, GELUs, SELUs and Swish. In our CIFAR experiments, the performance by replacing ReLUs with polynomial activations using DenseNet, Residual Networks and Wide Residual Networks is on par with eight state-of-the-art activation functions.

While the increase of parameters is negligible, our method is computationally expensive. We believe that by designing networks with simpler activations like ReLU for the initial layers, followed by layers with polynomial activations can further improve accuracies.

REFERENCES

Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V. Le. Neural optimizer search with reinforcement learning. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp.

- 459–468, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL <http://proceedings.mlr.press/v70/bello17a.html>.
- Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning longterm dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5, 1994.
- Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2015.
- Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. Incorporating second-order functional knowledge for better option pricing. In *Proceedings of the 13th International Conference on Neural Information Processing Systems, NIPS’00*, pp. 451–457, Cambridge, MA, USA, 2000. MIT Press. URL <http://dl.acm.org/citation.cfm?id=3008751.3008817>.
- Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. *Proceedings of Machine Learning Research*, 2011.
- Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2, NIPS’14*, pp. 2672–2680, Cambridge, MA, USA, 2014. MIT Press. URL <http://dl.acm.org/citation.cfm?id=2969033.2969125>.
- Richard H.R. Hahnloser, Rahul Sarpeshkar, Misha A. Mahowald, Rodney J. Douglas, and Hyunjune Sebastian Seung. Digital selection and analogue amplification coexist in a cortex- inspired silicon circuit. *Nature*, 405, 2000.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing humanlevel performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, 2015a.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015b.
- Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2016.
- Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL <http://dx.doi.org/10.1162/neco.1997.9.8.1735>.
- Gao Huang, Zhuang Liu, Kilian Q. Weinberger, and Laurens van der Maaten. Densely connected convolutional. In *Conference on Computer Vision and Pattern Recognition*, 2017.
- Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.
- Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. 2017.
- Alex Krizhevsky. Learning multiple layers of features from tiny images, 2009.
- Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, NIPS’12*, pp. 1097–1105, USA, 2012. Curran Associates Inc. URL <http://dl.acm.org/citation.cfm?id=2999134.2999257>.
- Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*, pp. 319–, London, UK, UK, 1999. Springer-Verlag. ISBN 3-540-66722-9. URL <http://dl.acm.org/citation.cfm?id=646469.691875>.
- Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets, 2014.

- Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning*, 2013.
- Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning, ICML'10*, pp. 807–814, USA, 2010. Omnipress. ISBN 978-1-60558-907-7. URL <http://dl.acm.org/citation.cfm?id=3104322.3104425>.
- Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. 2016. URL <http://arxiv.org/abs/1609.03499>. cite arxiv:1609.03499.
- Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.
- Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. *ICLR*, 2019.
- Thomas Serre, Gabriel Kreiman, Minjoon Kouh, Charles Cadieu, Ulf Knoblich, and Tomaso Poggio. A quantitative theory of immediate visual recognition. *PROG BRAIN RES*, pp. 33–56, 2007.
- Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.
- Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference*, 2016.
- Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *ArXiv*, abs/1611.01578, 2016.