SWISH: A SELF-GATED ACTIVATION FUNCTION

Prajit Ramachandran,*Barret Zoph, Quoc V. Le
Google Brain
{prajit,barretzoph,qvl}@google.com

ABSTRACT

The choice of activation functions in deep networks has a significant effect on the training dynamics and task performance. Currently, the most successful and widely-used activation function is the Rectified Linear Unit (ReLU). Although various alternatives to ReLU have been proposed, none have managed to replace it due to inconsistent gains. In this work, we propose a new activation function, named Swish, which is simply $f(x) = x \cdot \text{sigmoid}(x)$. Our experiments show that Swish tends to work better than ReLU on deeper models across a number of challenging datasets. For example, simply replacing ReLUs with Swish units improves top-1 classification accuracy on ImageNet by 0.9% for Mobile NASNet-A and 0.6% for Inception-ResNet-v2. The simplicity of Swish and its similarity to ReLU make it easy for practitioners to replace ReLUs with Swish units in any neural network.

1 INTRODUCTION

At the heart of every deep network lies a linear transformation followed by an activation function $f(\cdot)$. The activation function plays a major role in the success of training deep neural networks. Currently, the most successful and widely-used activation function is the Rectified Linear Unit (ReLU) (Hahnloser et al., 2000; Jarrett et al., 2009; Nair & Hinton, 2010), defined as $f(x) = \max(x, 0)$. The use of ReLUs was a breakthrough that enabled the fully supervised training of state-of-the-art deep networks (Krizhevsky et al., 2012). Deep networks with ReLUs are more easily optimized than networks with *sigmoid* or *tanh* units, because gradient is able to flow when the input to the ReLU function is positive. Thanks to its simplicity and effectiveness, ReLU has become the default activation function used across the deep learning community.

While numerous activation functions have been proposed to replace ReLU (Maas et al., 2013; He et al., 2015; Clevert et al., 2015; Klambauer et al., 2017), none have managed to gain the widespread adoption that ReLU enjoys. Many practitioners have favored the simplicity and reliability of ReLU because the performance improvements of the other activation functions tend to be inconsistent across different models and datasets.

In this work, we propose a new activation function called *Swish*. Swish is a smooth, non-monotonic function defined as $f(x) = x \cdot \text{sigmoid}(x)$. Our extensive experiments show that Swish consistently matches or outperforms ReLU on deep networks applied to a variety of challenging domains such as image classification and machine translation. On ImageNet, replacing ReLUs with Swish units improves top-1 classification accuracy by 0.9% on Mobile NASNet-A (Zoph et al., 2017) and 0.6% on Inception-ResNet-v2 (Szegedy et al., 2017). These accuracy gains are significant given that one year of architectural tuning and enlarging yielded 1.3% accuracy improvement going from Inception V3 (Szegedy et al., 2016) to Inception-ResNet-v2 (Szegedy et al., 2017).

2 SWISH

We propose a new activation function that we name *Swish*:

$$\boxed{f(x) = x \cdot \sigma(x)} \tag{1}$$

where $\sigma(x) = (1 + \exp(-x))^{-1}$ is the sigmoid function. See Figure 1 for the graph of Swish.

*Work done as a member of the Google Brain Residency program (g.co/brainresidency).

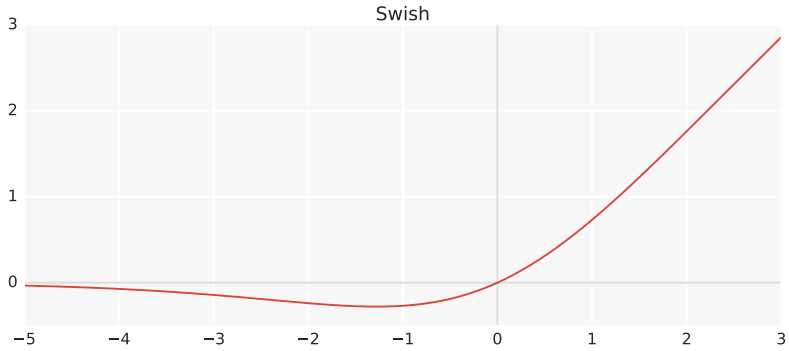


Figure 1: The Swish activation function.

Like ReLU, Swish is unbounded above and bounded below. Unlike ReLU, Swish is smooth and non-monotonic. In fact, the non-monotonicity property of Swish distinguishes itself from most common activation functions. The derivative of Swish is

$$\begin{aligned} f'(x) &= \sigma(x) + x \cdot \sigma(x)(1 - \sigma(x)) \\ &= \sigma(x) + x \cdot \sigma(x) - x \cdot \sigma(x)^2 \\ &= x \cdot \sigma(x) + \sigma(x)(1 - x \cdot \sigma(x)) \\ &= f(x) + \sigma(x)(1 - f(x)) \end{aligned}$$

The first and second derivatives of Swish are shown in Figure 2. For inputs less than around 1.25, the derivative has a magnitude less than 1. The success of Swish implies that the gradient preserving property of ReLU (i.e., having a derivative of 1 when $x > 0$) may no longer be a distinct advantage in modern architectures. In fact, we show in the experimental section that we can train deeper Swish networks than ReLU networks when using BatchNorm (Ioffe & Szegedy, 2015).

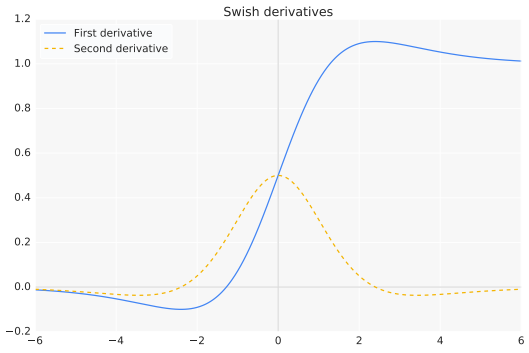


Figure 2: First and second derivatives of Swish.

An additional connection with ReLU can be seen if Swish is slightly reparameterized as follows:

$$f(x; \beta) = 2x \cdot \sigma(\beta x)$$

If $\beta = 0$, Swish becomes the linear function $f(x) = x$. As $\beta \to \infty$, the sigmoid approaches a 0-1 function, so Swish becomes like the ReLU function. This suggests that Swish can be loosely viewed as a smooth function, which nonlinearly interpolates between the linear function and the ReLU function. The degree of interpolation can be controlled by the model if β is set as a trainable parameter. We call this variant, without the factor of 2, *Swish-β*.

Our design of Swish was inspired by the use of the sigmoid function for gating in LSTM (Hochreiter & Schmidhuber, 1997) and highway networks (Srivastava et al., 2015). We simplified the gating mechanism by using the same value to gate itself, which we call *self-gating*. An advantage of self-gating is that it only requires a single scalar input, whereas normal gating requires multiple

scalar inputs. This property enables activation functions that use self-gating, such as Swish, to easily replace activation functions that take as input a single scalar (*pointwise* functions), such as the ReLU function, without needing to change the hidden size or the number of parameters.[1]

Practically, Swish can be implemented with a single line code change in most deep learning libraries, such as TensorFlow (Abadi et al., 2016) (e.g., `x * tf.sigmoid(x)` or `tf.nn.swish(x)` if using a version of TensorFlow released after the submission of this work). As a cautionary note, if BatchNorm (Ioffe & Szegedy, 2015) is used, the scale parameter should be set. Some high level libraries turn off the scale parameter by default due to the ReLU function being piecewise linear, but this setting is incorrect for Swish. For training Swish networks, we found that slightly lowering the learning rate used to train ReLU networks works well.

2.1 Properties of Swish

Our experiments show that Swish consistently outperforms or matches the ReLU function on a variety of deep models. While it is difficult to prove why one activation function outperforms another because of the many confounding factors that affect training, we believe that the properties of Swish being unbounded above, bounded below, non-monotonic, and smooth are all advantageous. In Figure 3, we plot other common activation functions, which will be useful to reference throughout this discussion.

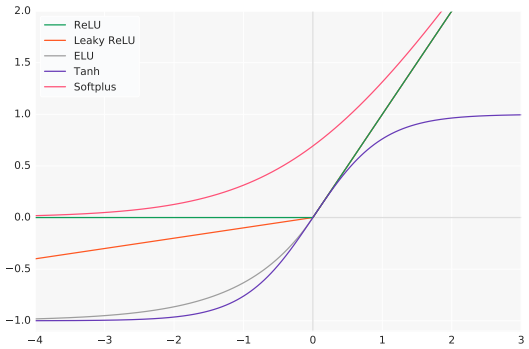


Figure 3: Common baseline activation functions. Best viewed in color.

Unboundedness is desirable because it avoids saturation, where training is slow due to near-zero gradients (Glorot & Bengio, 2010). Classical functions like *sigmoid* and *tanh* are bounded above and below, so the network must be carefully initialized to stay in the linear regime of these functions. The ReLU function was such a large improvement over *tanh* because it is unbounded above, which avoids saturation whenever $x > 0$. This property is so important that almost every recently-proposed successful activation function is unbounded above. Swish shares this property, with its positive side approaching the linear function as the input becomes more positive.

Furthermore, being bounded below may also be advantageous because of strong regularization effects. Functions that approach zero in the limit induce even larger regularization effects because large negative inputs are "forgotten". As shown in Figure 4, forgetting is especially important at the start of training when large negative preactivations are common. The property of approaching zero

[1]To further validate the quality of Swish, we used two search techniques (exhaustive and reinforcement learning) to find formulas of activation functions. We designed a search space of functions of the form $f(x) = f_1(f_2(x), f_3(x))$, where f_2 and f_3 are unary functions (e.g., sigmoid(x), $\sqrt{x}$, $\exp(-x^2)$, $\cos(x)$, αx, etc.) and f_1 is a binary function (e.g., $x \cdot y$, $x + y$, $\max(x, y)$, $\min(x, y)$, $\alpha x + (1 - \alpha)y$, etc.). The search was performed by replacing the ReLU function with the candidate activation function in a small ResNet-20, which was trained on CIFAR-10 for 10K steps. We used exhaustive search for small search spaces and reinforcement learning techniques (Zoph & Le, 2016; Bello et al., 2017; Zoph et al., 2017) for larger search spaces. We consistently found that Swish and its variants were the best performers, while the ReLU function and its variants were the second best performers. More complicated functions performed worse than simpler functions. Other functions that were discovered by the search that we found interesting are $f(x) = \cos(x) - x$ and $f(x) = \max(x, \tanh(x))$. However, these activation functions did not generalize as well as Swish in our experiments, so we did not pursue them further.

in the limit is satisfied by Softplus, ReLU, and Swish. Swish nonetheless differs from ReLU and Softplus because it produces negative outputs for small negative inputs due to its non-monotonicity. The non-monotonicity of Swish increases expressivity and improves gradient flow, which is important considering that many preactivations fall into this range (see Figure 4). This property may also provide some robustness to different initializations and learning rates.

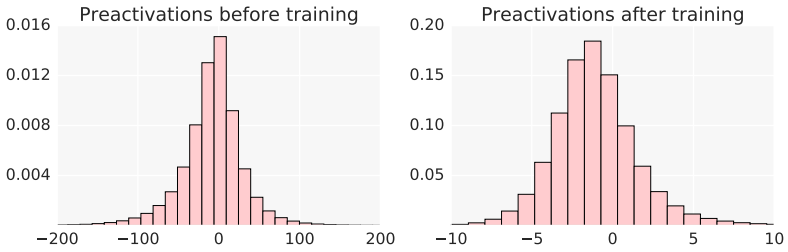


Figure 4: The distribution of preactivations to the final conv layer of a ResNet-32 with Swish.

Furthermore, smoothness plays a beneficial role in optimization and generalization. Figure 5 plots the *output* landscape of a random neural network with different activation functions. Specifically, we randomly initialize a 6-layered neural network, pass in as input the x and y coordinates of each point in a grid, and plot the scalar network output for each grid point. The output landscape of the ReLU network (which, in Figure 5, looks like an asterisk) has numerous sharp regions, potentially due to its non-smooth nature. In contrast, the Swish network output landscape is considerably smoother. The smoothness of the output landscape directly impacts the smoothness of the loss landscape. Intuitively, a smooth loss landscape is likely easier to optimize because it is more traversable, reducing sensitivity to initialization and learning rates.

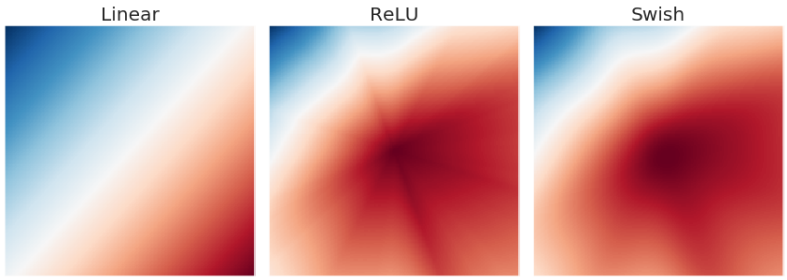


Figure 5: The output landscape of a random neural network, with different activation functions. The figure is generated by passing two scalars, the x and y coordinates of each position in a grid, into a randomly initialized neural network that outputs a single scalar. Linear refers to $f(x) = x$, i.e., no nonlinearity. The ReLU network output landscape has distinctive sharp regions of similar magnitudes whereas the the Swish network output landscape is more smooth. Best viewed in color.

3 Experiments

We first compare Swish to the ReLU activation function on a set of small scale experiments, and find that Swish consistently outperforms ReLU. We then benchmark Swish against ReLU and a number of recently proposed activation functions on challenging datasets, and find that Swish matches or exceeds the baselines on nearly all tasks.

The following sections will describe our experimental settings and results in greater detail. As a summary, Table 1 shows Swish in comparison to each baseline activation function we considered (which are defined in Section 3.2). The results in Table 1 are aggregated by comparing the performance Swish to the performance of different activation functions applied to a variety of models, such as Inception ResNet-v2 (Szegedy et al., 2017) and Transformer (Vaswani et al., 2017), across

multiple datasets, such as CIFAR, ImageNet, and English→German translation.[2] The improvement of Swish over other activation functions is statistically significant under a one-sided paired sign test.

Baselines	**ReLU**	**LReLU**	**PReLU**	**Softplus**	**ELU**	**SELU**
Swish $>$ Baseline	9	8	6	7	8	8
Swish $=$ Baseline	0	1	3	1	0	1
Swish $<$ Baseline	0	0	0	1	1	0

Table 1: The number of models on which Swish outperforms, is equivalent to, or underperforms each baseline activation function we compared against in our experiments.

3.1 Comparing Swish to ReLU

3.1.1 Swish improves the training of deep networks

We are interested in whether replacing ReLUs with Swish units can improve the training of deep networks. We train fully connected networks of varying depths on MNIST, with each layer having 1,024 neurons. We *do not* use residual connections (He et al., 2016a) because they enable the training of arbitrarily deep networks. We use He initialization (He et al., 2015) with BatchNorm (Ioffe & Szegedy, 2015) to lessen the dependence on initialization. The network is optimized using SGD with momentum on a batch size of 128, and for fair comparison, we try the same number of learning rates for each activation function. Figure 6 plots the median result of 3 runs.

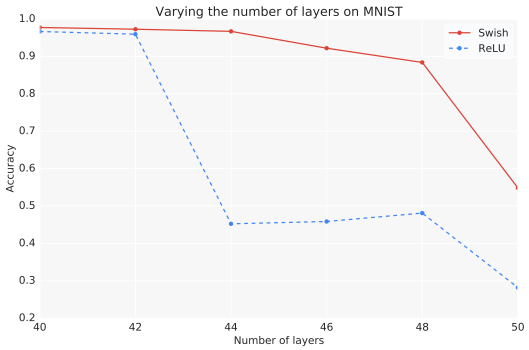


Figure 6: The test accuracy when changing the number of layers in fully connected networks with different activation functions on MNIST. The results plotted are the medians of 3 runs.

In our experiments, both activation functions achieve similar performances up to 40 layers. However, as shown in the figure, Swish outperforms ReLU by a large margin in the range between 40 and 50 layers when optimization becomes difficult. In very deep networks, Swish achieves higher test accuracies than ReLU. This result challenges the conventional wisdom that ReLU performs well because it does not squish gradients; Swish is able to train deeper networks despite having gradient squishing properties.

3.1.2 Swish robustly outperforms ReLU with different batch sizes

We are interested in understanding the sensitivity of Swish and ReLU to hyperparameters, especially batch size. We ran preactivation ResNet-32 models with different batch sizes on CIFAR-10 (Krizhevsky & Hinton, 2009), and plot the median result of 5 runs in Figure 7.

As expected, the performance of both activation functions decreases as the batch size increases, potentially due to sharp minima (Keskar et al., 2017). However, Swish outperforms ReLU on every

[2]To avoid skewing the comparison, each model type is compared just once. A model with multiple results is represented by the average of its results. Specifically, the models with averaged results are (a) ResNet-164, Wide ResNet 28-10, and DenseNet 100-12 across the CIFAR-10 and CIFAR-100 results, (b) Mobile NASNet-A and Inception-ResNet-v2 across the 3 runs, and (c) WMT Transformer model across the 4 newstest results.

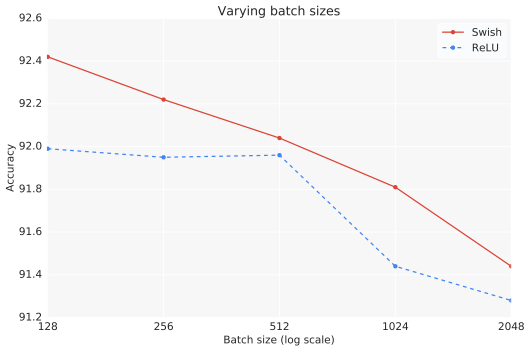


Figure 7: Test set accuracy when changing the batch size of ResNet-32 with different activation functions on CIFAR-10.

batch size, suggesting that the performance difference between the two activation functions remains even when varying the batch size.

3.2 Benchmarking Swish against other common activation functions

We compare Swish against several additional baseline activation functions on a variety of models and datasets. Since many activation functions have been proposed, we choose the most common activation functions to compare against, and follow the guidelines laid out in each work:

- Leaky ReLU (*LReLU*) (Maas et al., 2013):

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases}$$

  where $\alpha = 0.01$. LReLU enables a small amount of information to flow when $x < 0$.
- Parametric ReLU (*PReLU*) (He et al., 2015): The same form as LReLU but α is a learnable parameter. Each channel has a shared α which is initialized to 0.25.
- Softplus (Nair & Hinton, 2010): $f(x) = \log(1 + \exp(x))$. Softplus is a smooth function with properties similar to Swish, but is strictly positive and monotonic. It can be viewed as a smooth version of ReLU.
- Exponential Linear Unit (*ELU*) (Clevert et al., 2015):

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

  where $\alpha = 1.0$
- Scaled Exponential Linear Unit (*SELU*) (Klambauer et al., 2017):

$$f(x) = \lambda \begin{cases} x & \text{if } x \geq 0 \\ \alpha(\exp(x) - 1) & \text{if } x < 0 \end{cases}$$

  with $\alpha \approx 1.6733$ and $\lambda \approx 1.0507$.

These baseline activation functions are also shown in Figure 3. In the following sections, to make the results easy to read, we boldface all results that are close to the best result on each model. Note that our results may not be directly comparable to the results in the corresponding works due to differences in our training setup.

3.2.1 CIFAR

We first compare Swish to all the baseline activation functions on the CIFAR-10 and CIFAR-100 datasets (Krizhevsky & Hinton, 2009). We use the preactivation ResNet-164 (He et al., 2016b),

Wide ResNet 28-10 (WRN) (Zagoruyko & Komodakis, 2016), and DenseNet 100-12 (Huang et al., 2017) models. We implement the 3 models in TensorFlow and replace the ReLU function with a different activation function. We use the same hyperparameters described in each work, such as optimizing using SGD with momentum, and follow previous works by reporting the median of 5 different runs.

Model	ResNet	WRN	DenseNet
LReLU	94.2	**95.6**	**94.7**
PReLU	94.1	95.1	94.5
Softplus	**94.6**	94.9	**94.7**
ELU	94.1	94.1	94.4
SELU	93.0	93.2	93.9
ReLU	93.8	95.3	**94.8**
Swish	**94.7**	**95.5**	**94.8**

Table 2: CIFAR-10 accuracy.

Model	ResNet	WRN	DenseNet
LReLU	74.2	78.0	83.3
PReLU	74.5	77.3	81.5
Softplus	**76.0**	**78.4**	**83.7**
ELU	75.0	76.0	80.6
SELU	73.2	74.3	80.8
ReLU	74.2	77.8	**83.7**
Swish	75.1	**78.5**	**83.8**

Table 3: CIFAR-100 accuracy.

The results in Tables 2 and 3 show how Swish consistently matches or outperforms ReLU on every model for both CIFAR-10 and CIFAR-100. Swish also matches or exceeds the best baseline performance on almost every model. Importantly, the "best baseline" changes between different models, which demonstrates the stability of Swish to match these varying baselines. Softplus, which is smooth and approaches zero on one side, similar to Swish, also has strong performance, providing additional evidence their shared properties are important for activation function performance.

3.2.2 IMAGENET

Next, we benchmark Swish against the baseline activation functions on the ImageNet 2012 classification dataset (Russakovsky et al., 2015). ImageNet is widely considered one of most important image classification datasets, consisting of a 1,000 classes and 1.28 million training images. We evaluate on the validation dataset, which has 50,000 images.

We compare all the activation functions on a variety of architectures designed for ImageNet: Inception-ResNet-v2, Inception-v4, Inception-v3 (Szegedy et al., 2017), MobileNet (Howard et al., 2017), and Mobile NASNet-A (Zoph et al., 2017). All these architectures were designed with ReLUs. We again replace the ReLU activation function with different activation functions and train for a fixed number of steps, determined by the convergence of the ReLU baseline. For each activation function, we try 3 different learning rates with RMSProp (Tieleman & Hinton, 2012) and pick the best.[3] All networks are initialized with He initialization (He et al., 2015). To verify that the performance differences are reproducible, we run the Inception-ResNet-v2 and Mobile NASNet-A experiments 3 times with the best learning rate from the first experiment. We plot the learning curves for both models in Figure 8.

Model	Top-1 Acc. (%)			Top-5 Acc. (%)		
LReLU	73.8	73.9	74.2	91.6	91.9	91.9
PReLU	74.6	74.7	74.7	**92.4**	**92.3**	**92.3**
Softplus	74.0	74.2	74.2	91.6	91.8	91.9
ELU	74.1	74.2	74.2	91.8	91.8	91.8
SELU	73.6	73.7	73.7	91.6	91.7	91.7
ReLU	73.5	73.6	73.8	91.4	91.5	91.6
Swish	74.6	74.7	74.7	92.1	92.0	92.0
Swish-β	**74.9**	**74.9**	**75.2**	**92.3**	**92.4**	**92.4**

Table 4: Mobile NASNet-A on ImageNet, with 3 different runs ordered by top-1 accuracy.

[3]For some of the models with ELU, SELU, and PReLU, we train with an additional 3 learning rates (so a total of 6 learning rates) because the original 3 learning rates did not converge.

Model	Top-1 Acc. (%)			Top-5 Acc. (%)		
LReLU	79.5	79.5	79.6	94.7	94.7	94.7
PReLU	79.7	79.8	80.1	94.8	94.9	94.9
Softplus	80.1	**80.2**	**80.4**	**95.2**	**95.2**	**95.3**
ELU	75.8	79.9	80.0	92.6	95.0	95.1
SELU	78.9	79.2	79.2	94.4	94.4	94.5
ReLU	79.5	79.6	79.8	94.8	94.8	94.8
Swish	**80.2**	**80.3**	**80.4**	**95.1**	**95.2**	**95.2**

Table 5: Inception-ResNet-v2 on ImageNet with 3 different runs. Note that the ELU sometimes has instabilities at the start of training, which accounts for the first result.

Model	Top-1 Acc. (%)	Top-5 Acc. (%)
LReLU	72.5	91.0
PReLU	**74.2**	**91.9**
Softplus	73.6	91.6
ELU	73.9	91.3
SELU	73.2	91.0
ReLU	72.0	90.8
Swish	**74.2**	91.6
Swish-β	**74.2**	**91.7**

Table 6: MobileNet on ImageNet.

Model	Top-1 Acc. (%)	Top-5 Acc. (%)
LReLU	78.4	94.1
PReLU	77.7	93.5
Softplus	**78.7**	**94.4**
ELU	77.9	93.7
SELU	76.7	92.8
ReLU	78.4	**94.2**
Swish	**78.7**	**94.2**

Table 7: Inception-v3 on ImageNet.

Model	Top-1 Acc. (%)	Top-5 Acc. (%)
LReLU	79.3	**94.7**
PReLU	79.3	94.4
Softplus	**79.6**	**94.8**
ELU	**79.5**	94.5
SELU	77.8	93.7
ReLU	79.2	94.6
Swish	79.3	**94.7**

Table 8: Inception-v4 on ImageNet.

The results in Tables 4-8 show strong performance for Swish. On Inception-Resnet-v2, Swish outperforms ReLU by a nontrivial 0.6%. Swish performs especially well on mobile sized models, with a 0.9% boost on Mobile NASNet-A and a 2.2% boost on MobileNet over ReLU.

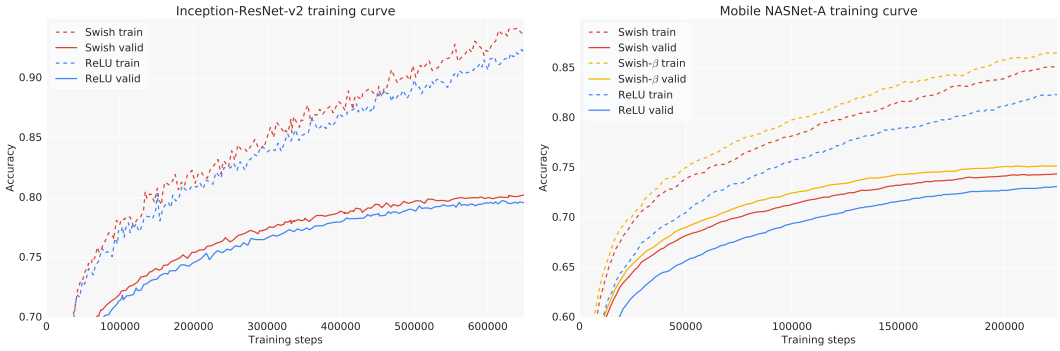
Swish also matches or exceeds the best performing baseline on most models, where again, the best performing baseline differs depending on the model. For example, Swish performs just as well as PReLU on the mobile sized models despite PReLU having the advantage of additional trainable parameters on a smaller model. To account for this advantage, we try the Swish-β variant with a trainable β parameter per channel. Swish-β achieves an additional gain of 0.5% top-1 accuracy on the Mobile NASNet-A model, pushing the difference between Swish and ReLU to a sizeable 1.4%. Softplus achieves accuracies comparable to Swish on the larger models, but performs worse on both mobile sized models. For Inception-v4, the gains from switching between activation functions is more limited, and Swish slightly underperforms Softplus and ELU. In general, the results suggest that switching to Swish improves performance with little additional tuning.

3.2.3 Machine Translation

We additionally benchmark Swish on the domain of machine translation. We train machine translation models on the standard WMT 2014 English→German dataset, which has 4.5 million training sentences, and evaluate on 4 different newstest sets using the standard BLEU metric. We use the attention based Transformer (Vaswani et al., 2017) model, which utilizes ReLUs in a 2-layered feed-forward network between each attention layer. We train a 12 layer "Base Transformer" model with 2 different learning rates[4] for 300K steps, but otherwise use the same hyperparameters as in the original work, such as using Adam (Kingma & Ba, 2015) to optimize.

Table 9 shows that Swish outperforms or matches the other baselines on machine translation. Swish does especially well on newstest2016, exceeding the next best performing baseline by 0.6 BLEU points. The worst performing baseline function is Softplus, demonstrating inconsistency in performance across differing domains. In contrast, Swish consistently performs well across multiple domains.

[4]We tried an additional learning rate for Softplus, but found it did not work well across all learning rates.



(a) Inception-ResNet-v2.

(b) Mobile NASNet-A.

Figure 8: Training curves on ImageNet. Best viewed in color.

Model	newstest2013	newstest2014	newstest2015	newstest2016
LReLU	**26.2**	**27.9**	**29.8**	33.4
PReLU	**26.3**	**27.7**	**29.7**	33.1
Softplus	23.4	23.6	25.8	29.2
ELU	24.6	25.1	27.7	32.5
SELU	23.7	23.5	25.9	30.5
ReLU	**26.1**	**27.8**	**29.8**	33.3
Swish	**26.2**	**28.0**	**30.1**	**34.0**

Table 9: BLEU score of a 12 layer Transformer on WMT English→German.

4 Related Work

Pointwise functions transform one scalar to another scalar. While this work focuses on pointwise nonlinear activation functions, there are many types of activation functions used in deep networks. *Many-to-one* functions, like max pooling, maxout (Goodfellow et al., 2013), and gating (Hochreiter & Schmidhuber, 1997; Srivastava et al., 2015; van den Oord et al., 2016; Dauphin et al., 2016), derive their power from combining multiple sources in a nonlinear way. *One-to-many* functions, like Concatenated ReLU (Shang et al., 2016), improve performance by extracting more nonlinear signal from a single source. Finally, *many-to-many* functions, such as BatchNorm (Ioffe & Szegedy, 2015) and LayerNorm (Ba et al., 2016), induce powerful nonlinear relationships between their inputs.

While much work has focused on proposing new activation functions (Maas et al., 2013; Agostinelli et al., 2014; He et al., 2015; Clevert et al., 2015; Klambauer et al., 2017; Qiu & Cai, 2017), few studies, such as Xu et al. (2015), have systematically compared different activation functions. To the best of our knowledge, this is the first study to compare pointwise activation functions across multiple challenging datasets.

Our study shows that Swish consistently outperforms ReLU on deep models. The strong performance of Swish challenges conventional wisdom about ReLU. Hypotheses about the importance of the gradient preserving property of ReLU seem unnecessary when residual connections (He et al., 2016a) enable the optimization of very deep networks. A similar insight can be found in the fully attentional Transformer (Vaswani et al., 2017), where the intricately constructed LSTM cell (Hochreiter & Schmidhuber, 1997) is no longer necessary when constant-length attentional connections are used. Architectural improvements lessen the need for individual components to preserve gradients.

However, even though ReLU empirically underperforms Swish, ReLU benefits from a comparatively simpler theoretical analysis due to its piecewise linear form. For example, He et al. (2015) derived a simple extension of the linear initialization of Glorot & Bengio (2010) that is used widely for ReLU networks, Dinh et al. (2017) used the linearity of ReLU to form an equivalent sharp reparameterization of networks, and Raghu et al. (2016) measured expressivity by counting the number

of piecewise linear regions learned by a ReLU network. It is unclear whether the smooth, non-monotonic Swish will be amenable to theoretical analysis.

Despite the differences in functional form, both ReLU and Swish have biological connections (Glorot et al., 2011). Swish is reminiscent of certain activation functions observed in models of vertebrate retinal neurons. For example, Baccus & Meister (2002) modeled the output of salamander ganglion cells with a linear unit followed by a pointwise activation function ("LN model") and found that the latter attains a non-monotonic "U" shape. While this model treats more than one nonlinear layer of neurons as a single linear unit, Real et al. (2017) suggest that some retinal bipolar cells exhibit non-monotonic activation functions that are also similar to Swish.

5 Conclusion

Swish is a novel activation function with the form $f(x) = x \cdot \text{sigmoid}(x)$. Swish has the properties of one-sided boundedness at zero, smoothness, and non-monotonicity, which may play a role in the observed efficacy of Swish and similar activation functions. Our experiments used models and hyperparameters that were designed for ReLU and just replaced the ReLU activation function with Swish; even this simple, suboptimal procedure resulted in Swish consistently outperforming ReLU and other activation functions. We expect additional gains to be made when these models and hyperparameters are specifically designed with Swish in mind. The simplicity of Swish and its similarity to ReLU means that replacing ReLUs in any network is just a simple one line code change.

Acknowledgements

We thank Esteban Real, Geoffrey Hinton, Irwan Bello, Jon Shlens, Kathryn Rough, Mohammad Norouzi, Navdeep Jaitly, Niki Parmar, Sam Smith, Simon Kornblith, Vijay Vasudevan, and the Google Brain team for help with this project.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *USENIX Symposium on Operating Systems Design and Implementation*, volume 16, pp. 265–283, 2016.

Forest Agostinelli, Matthew Hoffman, Peter Sadowski, and Pierre Baldi. Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*, 2014.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. In *Advances in Neural Information Processing Systems*, 2016.

Stephen A Baccus and Markus Meister. Fast and slow contrast adaptation in retinal circuitry. *Neuron*, 36(5): 909–919, 2002.

Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V Le. Neural optimizer search with reinforcement learning. In *International Conference on Machine Learning*, pp. 459–468, 2017.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. *arXiv preprint arXiv:1612.08083*, 2016.

Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. In *International Conference on Machine Learning*, 2017.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256, 2010.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 315–323, 2011.

Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In *International Conference on Machine Learning*, 2013.

Richard HR Hahnloser, Rahul Sarpeshkar, Misha A Mahowald, Rodney J Douglas, and H Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789): 947, 2000.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pp. 630–645. Springer, 2016b.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. In *Conference on Computer Vision and Pattern Recognition*, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456, 2015.

Kevin Jarrett, Koray Kavukcuoglu, Yann LeCun, et al. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, 2009.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations*, 2017.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *arXiv preprint arXiv:1706.02515*, 2017.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Technical report, University of Toronto, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning*, volume 30, 2013.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning*, 2010.

Suo Qiu and Bolun Cai. Flexible rectified linear units for improving convolutional neural networks. *arXiv preprint arXiv:1706.08098*, 2017.

Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. On the expressive power of deep neural networks. In *International Conference on Machine Learning*, 2016.

Esteban Real, Hiroki Asari, Tim Gollisch, and Markus Meister. Neural circuit inference from function to structure. *Current Biology*, 27(2):189–198, 2017.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. Understanding and improving convolutional neural networks via concatenated rectified linear units. In *International Conference on Machine Learning*, pp. 2217–2225, 2016.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, pp. 4278–4284, 2017.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems*, pp. 4790–4798, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

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. In *International Conference on Learning Representations*, 2016.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2017.