谷歌大脑提出新型激活函数Swish惹争议：可直接替换并优于ReLU？（附机器之心测试）

选自arXiv

机器之心编译

参与：路雪、蒋思源

> 近日，谷歌大脑团队提出了新型激活函数 Swish，团队实验表明使用 Swish 直接替换 ReLU 激活函数总体上可令 DNN 的测试准确度提升。此外，该激活函数的形式十分简单，且提供了平滑、非单调等特性从而提升了整个神经网络的性能。

在该论文中，谷歌大脑团队所提出了 Swish 激活函数：f(x) = x · sigmoid(x)，并通过基线实验表明其在绝大多数环境中可以替代当前比较流行的 ReLU 函数。不过在 Reddit 论坛上，该激活函数的性能与优点还是有些争议的，有的开发者发现该激活函数很多情况下可以比标准的 ReLU 获得更高的性能，而有些开发者则认为 Swish 激活函数并没有什么新意，我们应该关注于更加基础的研究。

Jean-Porte：我发现该论文要比 *ELU* 论文更加有意思。他们在激活函数空间上使用搜索技术，然后分析并执行一些试验。激活函数是十分重要的，自从 *ReLU* 函数流行以来，我们就很少在上面有比较大的进展，所以现在确实需要一批研究激活函数的论文以提升神经网络性能。

Turick：我们为什么不将该论文的创意结合早先提出的 *SELU*（缩放指数型线性单元），从而使 *Swish* 能实现自归一化而不需要使用使用批量归一化技术。如果我的推导没错的话，那么激活函数的形式就应该是 *1.67653251702 * x * sigmoid(x)*。

jbmlres：在《*Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning*》这篇论文中，所使用的激活函数难道不是类似的结构吗?

inkognit：该激活函数和 *Facebook* 提出的门控线性单元（*Gated Linear Unit*GLU）有点类似？/

通过观察 Reddit 上热烈的讨论，我们发现开发者对该激活函数的性能有很大的争议。有研究者使用 TensorFlow 或 Keras 实现了 Swish，并且常规情况下平均测试准确度大约有 0.5% 的提升，而有的开发者更是质疑该研究是不是灌了水，如上 jbmlres 所说的论文确实和该论文使用的激活函数十分相似。对此该论文的第一作者 Prajit Ramachandran 在 Reddit 上的回应如下：

大家好，我是该论文的第一作者，下面我会总结回应一些评论：

1. 正如上面所指出的，我们确实没有在文献综述中提及该篇提出了相同激活函数的论文。该误差在于我们没有彻底搜索相关的文献，我们真诚地致歉并很快会修改该论文。
2. 如论文中所述，我们搜索了许多形式的激活函数，x**CDF(x) 确实也在我们的搜索空间中，当我们发现它的性能并没有 x**sigmoid(x) 那么出色。
3. 我们也计划在使用推荐的初始化设置下实现 SELU 实验。
4. 激活函数的研究非常重要，因为它是深度学习的核心单元，即使激活函数只有少量的提升，但它也会因为大量的使用而获得极大的收益。ReLU 不仅在研究中十分常见，同时它在行业中也得到广泛的使用。因此替代 ReLU 对研究和产业都有实际的意义。
   我们希望该研究工作能为大家提供一组令人信服的实验结果，并鼓励使用 ReLU 的研究者和开发者至少可以尝试使用 Swish，如果大家的实验确实存在一些性能提升，那么我们就可以用它替换 ReLU。最重要的是，使用 Swish 替换 ReLU 是十分便捷的，我们不需要为该激活函数修改神经网络架构和初始化等。

对此，机器之心也尝试使用全连接神经网络测试该激活函数。我们测试的数据集是 MNIST，但因为一般情况下 3 层全连接网络就能获得 98.53% 的测试准确度，而这样的网络实在是太简单了，所以我们将其扩展到包含 10 个全连接隐藏层的神经网络。通过构建更复杂的网络，我们希望能增加优化和推断的困难，因此更能体现两个激活函数的区别。虽然增加模型复杂度很可能会产生过拟合现象，但我们只需要对比 ReLU 和 Swish 在复杂网络下的测试准确度就行了。

以下是扩展后的神经网络层级数与每一层的神经元数，通过这些参数，我们可以了解扩展后的神经网络架构：

```
INPUT_NODE = 784
LAYER1_NODE = 500
LAYER2_NODE = 500
LAYER3_NODE = 500
LAYER4_NODE = 500
LAYER5_NODE = 500
LAYER6_NODE = 500
LAYER7_NODE = 500
LAYER8_NODE = 300
LAYER9_NODE = 200
LAYER10_NODE = 100
OUTPUT_NODE = 10
```

以下是定义推断过程的代码，在这里我们首先使用的是 ReLU 激活函数：

```
layer1 = tf.nn.relu(tf.matmul(input_tensor, avg_class.avera 0])) + avg_class.average( 0])
ge(W[                                                      B[                          )
layer2 = tf.nn.relu(tf.matmul(layer1, avg_class.average(W 1])) + avg_class.average(B 1]))
[                                                          [
layer3 = tf.nn.relu(tf.matmul(layer2, avg_class.average(W 2])) + avg_class.average(B 2]))
[                                                          [
layer4 = tf.nn.relu(tf.matmul(layer3, avg_class.average(W 3])) + avg_class.average(B 3]))
[                                                          [
layer5 = tf.nn.relu(tf.matmul(layer4, avg_class.average(W 4])) + avg_class.average(B 4]))
[                                                          [
layer6 = tf.nn.relu(tf.matmul(layer5, avg_class.average(W 5])) + avg_class.average(B 5]))
[                                                          [
layer7 = tf.nn.relu(tf.matmul(layer6, avg_class.average(W 6])) + avg_class.average(B 6]))
[                                                          [
layer8 = tf.nn.relu(tf.matmul(layer7, avg_class.average(W 7])) + avg_class.average(B 7]))
[                                                          [
layer9 = tf.nn.relu(tf.matmul(layer8, avg_class.average(W 8])) + avg_class.average(B 8]))
[                                                          [
layer10 = tf.nn.relu(tf.matmul(layer9, avg_class.average(W9])) + avg_class.average(B 9])
[                                                          [                           )
return tf.matmul(layer10, avg_class.average(W[10])) + avg_class.average(B[10])
```

随后我们将定义推断过程所涉及的 ReLU 激活函数转换为 Swish 激活函数。因为 Swish 的表达式为 f(x) = x · sigmoid(x)，而 TensorFlow 自带 Sigmoid 函数，所以我们可以借助 tf.nn.sigmoid() 函数构建 Swish 激活函数：

```
ac_1=tf.matmul(input_tensor, avg_class.average(W[0])) + avg_class.average(B[0])
layer1 = ac_1*tf.nn.sigmoid(ac_1)
ac_2=tf.matmul(layer1, avg_class.average(W[1])) + avg_class.average(B[1])
layer2 = ac_2*tf.nn.sigmoid(ac_2)
ac_3=tf.matmul(layer2, avg_class.average(W[2])) + avg_class.average(B[2])
layer3 = ac_3*tf.nn.sigmoid(ac_3)
ac_4=tf.matmul(layer3, avg_class.average(W[3])) + avg_class.average(B[3])
layer4 = ac_4*tf.nn.sigmoid(ac_4)
ac_5=tf.matmul(layer4, avg_class.average(W[4])) + avg_class.average(B[4])
layer5 = ac_5*tf.nn.sigmoid(ac_5)
ac_6=tf.matmul(layer5, avg_class.average(W[5])) + avg_class.average(B[5])
layer6 = ac_6*tf.nn.sigmoid(ac_6)
ac_7=tf.matmul(layer6, avg_class.average(W[6])) + avg_class.average(B[6])
layer7 = ac_7*tf.nn.sigmoid(ac_7)
ac_8=tf.matmul(layer7, avg_class.average(W[7])) + avg_class.average(B[7])
layer8 = ac_8*tf.nn.sigmoid(ac_8)
ac_9=tf.matmul(layer8, avg_class.average(W[8])) + avg_class.average(B[8])
layer9 = ac_9*tf.nn.sigmoid(ac_9)
ac_10=tf.matmul(layer9, avg_class.average(W[9])) + avg_class.average(B[9])
layer10 = ac_10*tf.nn.sigmoid(ac_10)
return tf.matmul(layer10, avg_class.average(W[10])) + avg_class.average(B[10])
```

如上所示，我们先计算每一层的传播结果，如 ac_3 计算的是第二层神经网络的输出结果，即第三层神经网络的输入值。然后再计算当前层的激活值，如 layer3 使用

ac_3*tf.nn.sigmoid(ac_3) 构建激活函数并计算激活值。此外，我们构建的列表 W 包含了所有层级间的权重，列表 B 包含了每一层的偏置向量，而 avg_class.average() 是模型中定义的滑动平均操作。

首先我们使用的是标准的三层全连接网络（784；500；10），这种情况下测试准确度为 0.9829。而后我们将其扩展到 10 个隐藏层以测试 Swish 的效果。以下展示使用 ReLU 激活函数的全连接网络测试准确度：

```
After 0 training step(s), validation accuracy using average model is 0.0932
After 1000 training step(s), validation accuracy using average model is 0.9466
After 2000 training step(s), validation accuracy using average model is 0.962
After 3000 training step(s), validation accuracy using average model is 0.9662
After 4000 training step(s), validation accuracy using average model is 0.967
After 5000 training step(s), validation accuracy using average model is 0.9696
After 6000 training step(s), validation accuracy using average model is 0.9686
After 7000 training step(s), validation accuracy using average model is 0.9698
After 8000 training step(s), validation accuracy using average model is 0.97
After 9000 training step(s), validation accuracy using average model is 0.97
After 10000 training step(s), test accuracy using average model is 0.9666
```

如上所示，增加到 12 层后测试准确度有所降低，这可能是超参数没有达到最优，且整个模型更难以迭代更新权重。我们一共迭代更新了 10000 步，且每次更新所采用的批量大小为 100 个样本，这两个超参数的选择是根据标准三层全连接神经网络而设定的。此外，我们发现当层级扩展到 12 层时，原有的学习率很容易使模型发散，所以测试这两个激活函数所采用的学习率降低了 100 倍（learning_rate=0.008）。其它的超参数如正则化率、学习衰减率、移动平均衰减率都保持不变。以下是转化为 Swish 激活函数后的训练结果：

```
After 0 training step(s), validation accuracy using average model is 0.0902
After 1000 training step(s), validation accuracy using average model is 0.9516
After 2000 training step(s), validation accuracy using average model is 0.9614
After 3000 training step(s), validation accuracy using average model is 0.9668
After 4000 training step(s), validation accuracy using average model is 0.9688
After 5000 training step(s), validation accuracy using average model is 0.971
After 6000 training step(s), validation accuracy using average model is 0.9742
After 7000 training step(s), validation accuracy using average model is 0.9752
After 8000 training step(s), validation accuracy using average model is 0.9738
After 9000 training step(s), validation accuracy using average model is 0.975
After 10000 training step(s), test accuracy using average model is 0.9722
```

我们可以看到使用 Swish 激活函数的测试准确度比使用 ReLU 的高一点，但这个结果可能并不具有普遍意义。但至少我们可以说 Swish 在全连接神经网络中可以获得与 ReLU 相匹配的性能。原论文指出当全连接网络在 40 层以内时，Swish 只稍微优于 ReLU 激活函数，但当层级增加到 40 至 50 层时，使用 Swish 函数的测试准确度要远远高于使用 ReLU 的测试准确度。

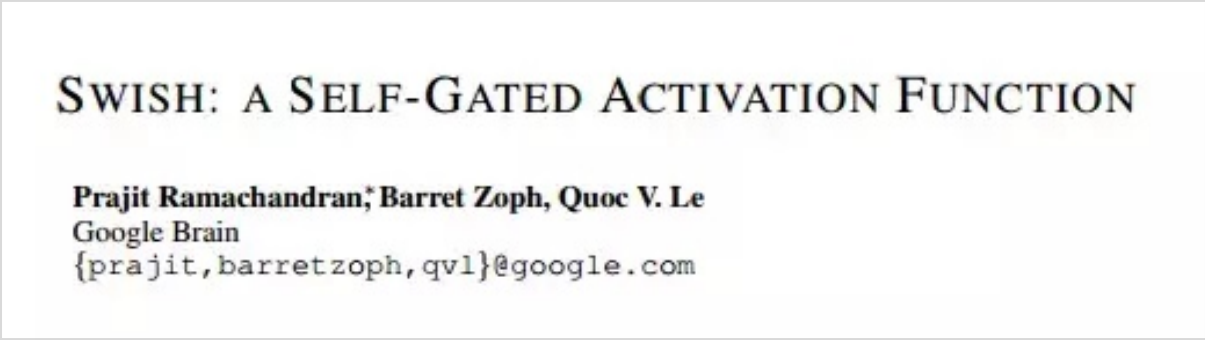
该全连接网络也很容易扩展到超越 40 层的情况，但这么深的层级对于计算力要求较高，所以读者也可以在机器之心 GitHub 下载该测试代码并进一步完成测评。

测试代码地址： https://github.com/jiqizhixin/ML-Tutorial-Experiment/blob/master/Experiments/swish_test.ipynb

添加全连接网络层级主要只需修改三个部分：第一是权重与偏置项，如定义新的 weights12 和 biases12，并将它们添加到列表 W 和 B 中；第二是定义 inference 函数中继续传播的过程，如 layer12 = tf.nn.relu(tf.matmul(layer11, avg_class.average(W[11])) + avg_class.average(B[11]))；最后是添加权重的正则项，如 regularazation=regularaztion + regularizer(W[11])。

除了全连接网络以外，原论文还详细对比了各种深度网络如 Inception-v4、MobileNet、Mobile NASNet-A 等，原论文表示新的激活函数总体上要优于传统的 ReLU 函数。下面我们简单介绍了谷歌大脑提出的新论文——Swish：一种自门控激活函数。

论文链接： https://arxiv.org/abs/1710.05941



SWISH: A SELF-GATED ACTIVATION FUNCTION

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

摘要：深度网络中激活函数的选择对训练过程和任务性能有很大影响。目前，最成功、使用最广泛的激活函数是修正线性单元（Rectified Linear Unit，ReLU）。尽管出现了很多修正 ReLU 的激活函数，但是无一可以真正替代它。本论文中，我们提出了一种新型激活函数 Swish：f(x) = x · sigmoid(x)。我们在多个难度较高的数据集上进行实验，证明 Swish 在深层模型上的效果优于 ReLU。例如，仅仅使用 Swish 单元替换 ReLU 就能把 Mobile NASNetA 在 ImageNet 上的 top-1 分类准确率提高 0.9%，Inception-ResNet-v 的分类准确率提高 0.6%。Swish 的简洁性及其与 ReLU 的相似性使从业者可以在神经网络中使用 Swish 单元替换 ReLU。

Neural architecture search with reinforcement learning

2 Swish

我们提出一个新的激活函数 Swish：

$$f(x) = x \cdot \sigma(x)$$

其中σ(x) = 1/(1 + exp(−x)) 是 Sigmoid 函数。图 1 展示的是 Swish 函数的图像：

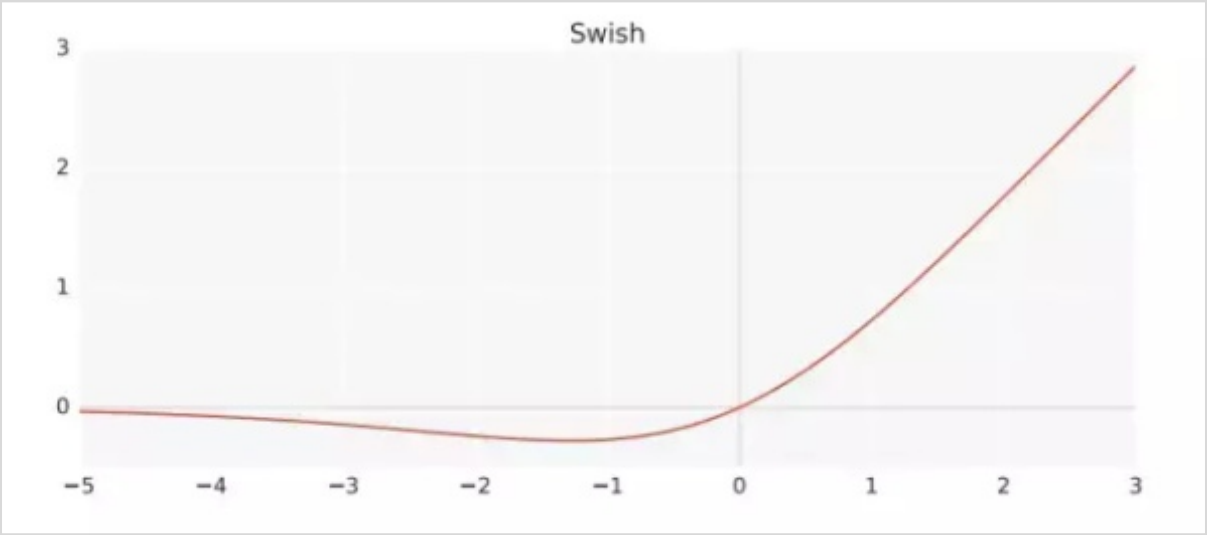


图 *1*： *Swish* 激活函数

和 ReLU 一样，Swish 无上界有下界。与 ReLU 不同的是，Swish 是平滑且非单调的函数。事实上，Swish 的非单调特性把它与大多数常见的激活函数区别开来。Swish 的导数是

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

Swish 的一阶导和二阶导如图 2 所示。输入低于 1.25 时，导数小于 1。Swish 的成功说明 ReLU 的梯度不变性（即 x > 0 时导数为 1）在现代架构中或许不再是独有的优势。事实上，实验证明在使用批量归一化（Ioffe & Szegedy, 2015）的情况下，我们能够训练出比 ReLU 网络更深层的 Swish 网络。

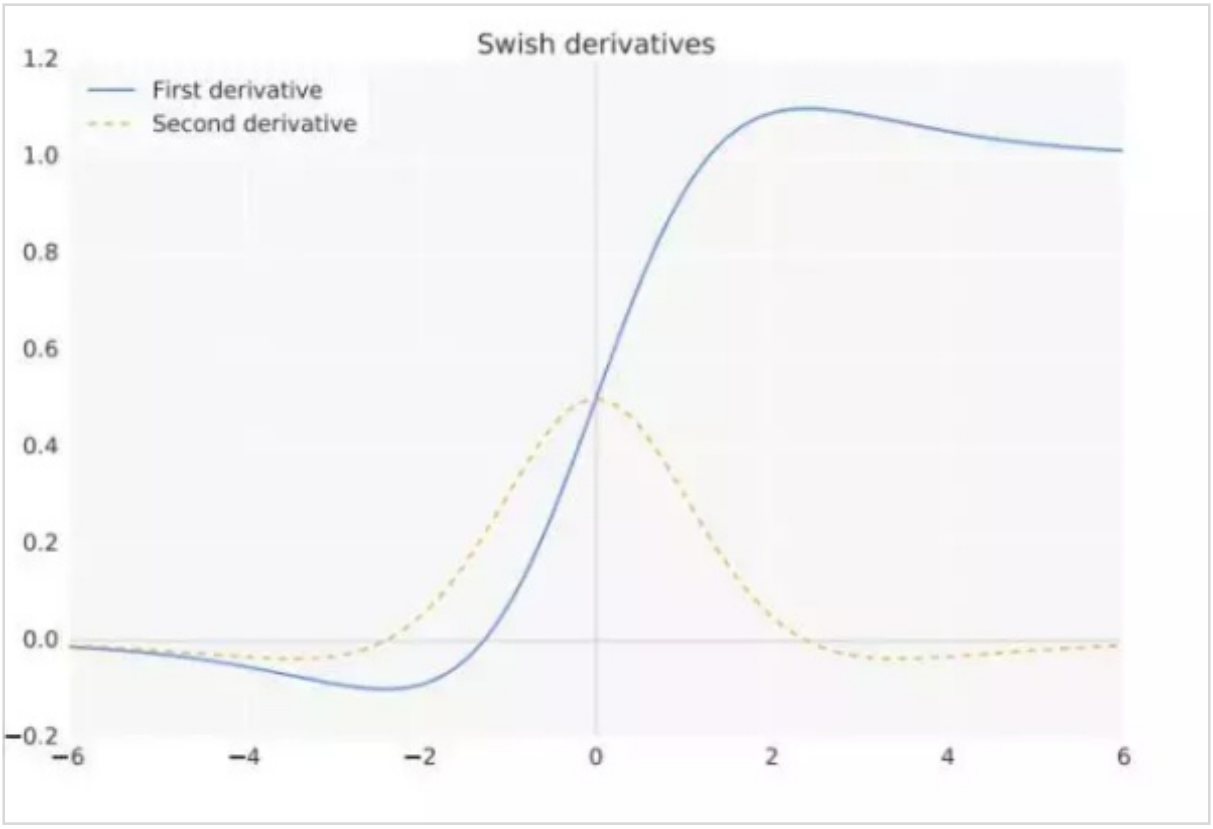


图 2：*Swish* 的一阶导数与二阶导数。

如果 Swish 按照下列方式进行重新参数化，则我们可以看到它与 ReLU 函数的联系。

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

如果 β = 0，则 Swish 变成线性函数 f(x) = x。当 β → ∞，Sigmoid 更接近 0-1 函数（指示函数），则 Swish 更像 ReLU 函数。这说明 Swish 可以宽泛地被视为平滑的函数，非线性地内插在线性函数和 ReLU 函数之间。当β被设置为可训练参数时，内插程度可以由模型控制。我们称该变量（不包括 x 前面的系数 2）为 Swish-β。

Swish 的设计受到 LSTM 和 highway network 中使用 sigmoid 函数进行门控的启发。我们使用同样的值进行门控来简化门控机制，称为自门控（self-gating）。自门控的优势是它仅需要一个简单的标量输入，而正常的门控需要多个标量输入。该特性令使用自门控的激活函数如 Swish 能够轻松替换以单个标量作为输入的激活函数（如 ReLU），无需改变参数的隐藏容量或数量。

事实上，在 TensorFlow 等大多数深度学习库中只需更改一行代码即可实现 Swish 函数（Abadi et al., 2016）。需要注意的是，如果使用 BatchNorm（Ioffe & Szegedy, 2015），则应设置缩放参数（scale parameter）。由于 ReLU 函数是分段线性函数，一些高级别的库默认关闭缩放参数，但是该设置不适用于 Swish。在训练 Swish 网络时，我们发现稍微降低用于

训练 ReLU 网络的学习率效果很好。

2.1 Swish 的特性

我们的实验证明 Swish 在多个深度模型上的性能持续优于或与 ReLU 函数相匹配。由于训练会受多种因素的影响，我们很难证明为什么一个激活函数会优于另一个。但是我们认为 Swish 无上界有下界、非单调且平滑的特性都是优势。我们在图 3 中绘出了其他常见激活函数，这有利于讨论的进行。

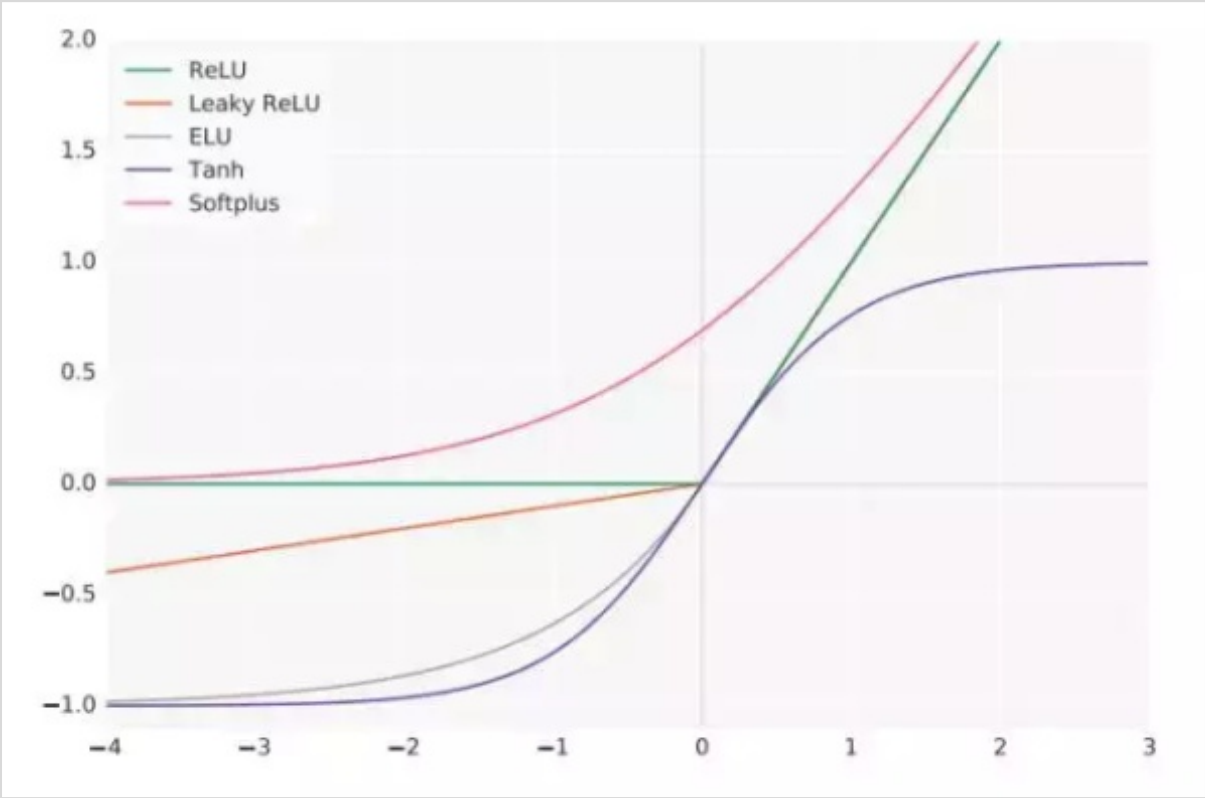


图 3：常见的基线激活函数。

3 实验

本论文在不同的任务、数据集上使用不同的模型来测试主流的激活函数性能。如下表 1 展示了 Swish 与基线激活函数之间的对比，该表总结了 Swish 激活函数和各种主流激活函数在基线性能上的差别，这些对比结果是通过不同模型（Inception 、MobileNet 等）在不同数据集（MNIST、CIFAR 或 ImageNet 等）上使用不同激活函数而实现的。

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

表 *1*：该实验中，*Swish* 要优于其他激活函数的模型数量。其中使用 *Swish* 的模型性能普遍要比使用其它激活函数的模型性能高。

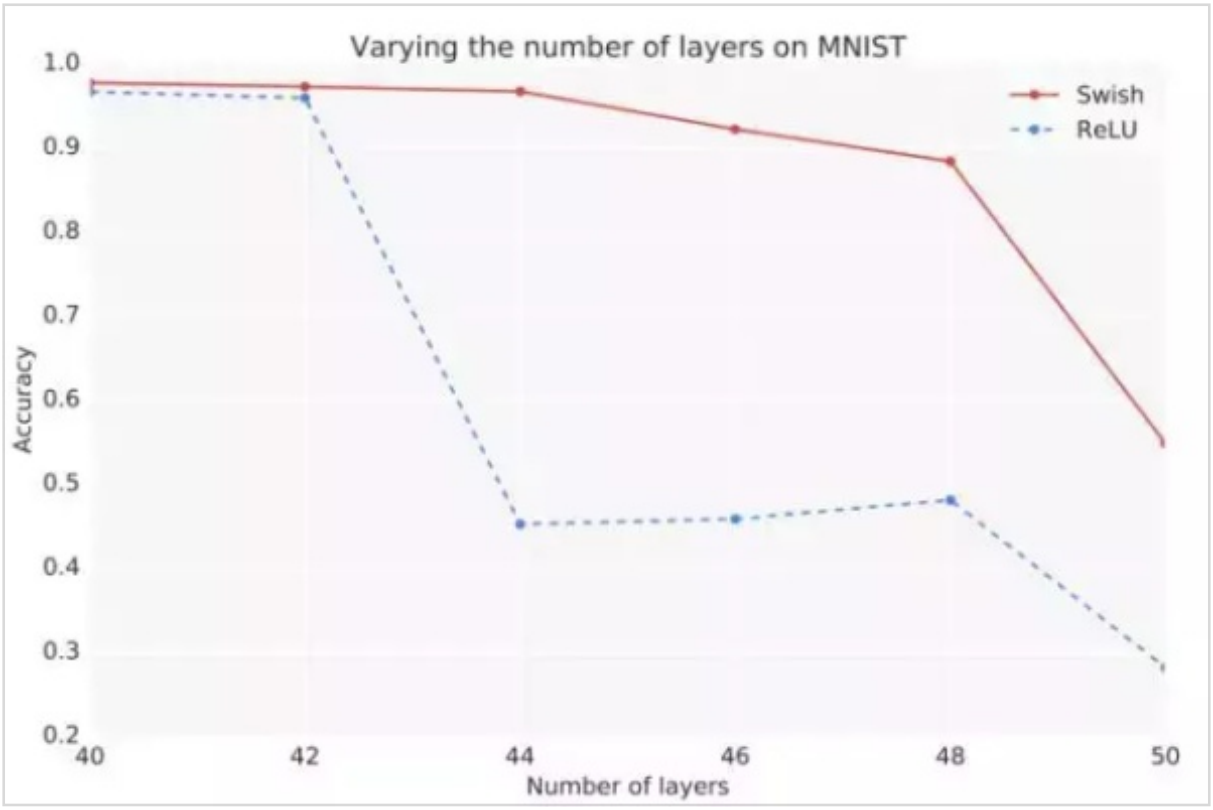


图 *6*：改变带有不同激活函数的全连接层级数时，其在 *MNIST* 数据集上的性能变化。以上取三次运行的中位数。

在我们的实验中，如果全连接网络的层级在 40 层以内，那么不同激活函数所表现出的性能没有显著性区别。而从 40 层增加到 50 层中，Swish 要比 ReLU 表现得更加优秀，因为随着层级的增加，优化将变得更加困难。在非常深的网络中，Swish 相对于 ReLU 能实现更高的测试准确度。虽然传统来说 ReLU 因为其梯度不会缩减（梯度消失问题）而表现出高效的性能，但现在 Swish 尽管有梯度缩减（梯度不为常量）现象，但其更适合训练深层神经网络。

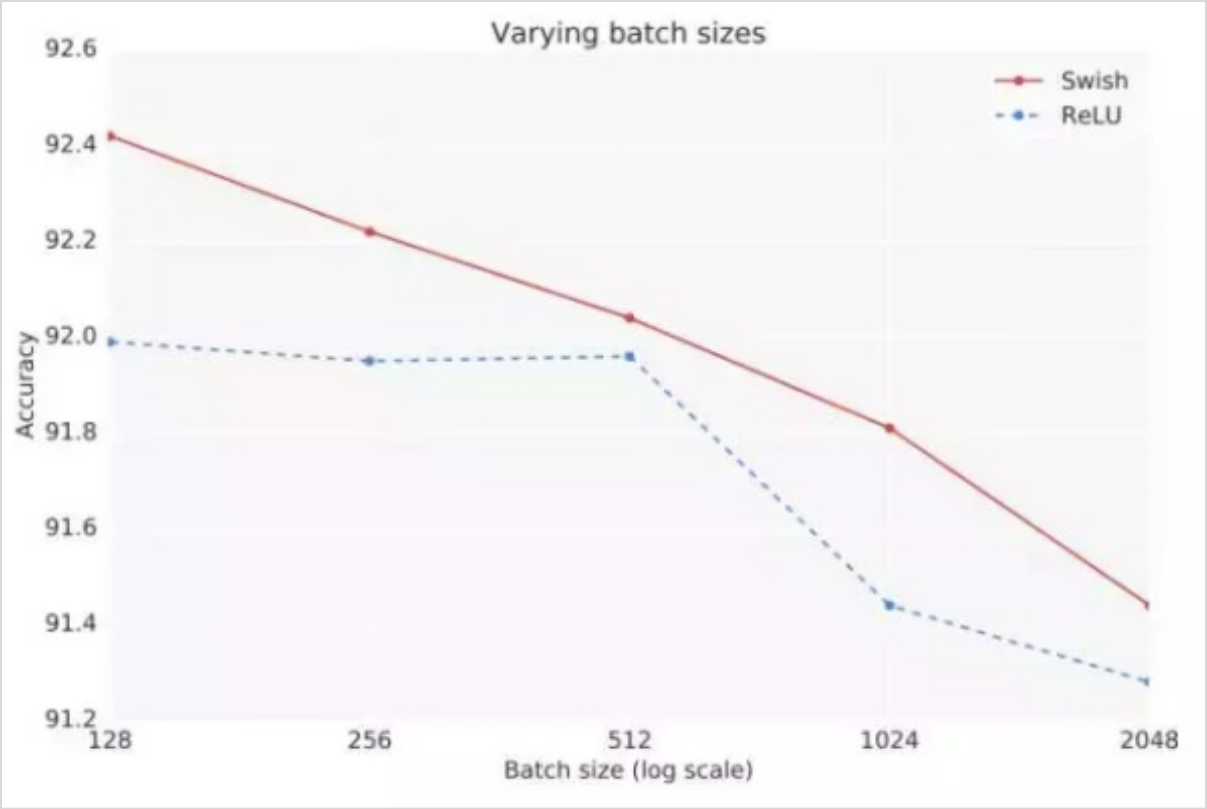


图 7：改变使用不同激活函数的 *ResNet-32* 模型批量大小时，其在 *CIFAR-10* 数据集上的测试准确度。

Swish 在每一个批量大小的性能都要比 ReLU 激活函数好，这意味着两种激活函数的性能对比并不随着批量大小的变化而变化。

此外，我们还重点比较了主流激活函数在 ImageNet 多个架构上的性能，这些架构包括：Inception-ResNet-v2、Inception-v4、Inception-v3（Szegedy et al., 2017）、MobileNet（Howard et al., 2017）和 Mobile NASNet-A（Zoph et al., 2017）。所有架构都是为 ReLU 而设计的，但我们可以用不同的激活函数替换 ReLU，然后给定所有模型相同的迭代次数并进行训练，训练步数由 ReLU 基线的收敛来决定。

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**

表 4：*ImageNet* 上的 *Mobile NASNet-A*，针对 *top-1* 准确率进行了 *3* 次不同的运行。

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**

表 5：*ImageNet* 上的 *Inception-ResNet-v2* 的 *3* 次不同运行结果。注意：*ELU* 有时在训练开始时存在不稳定的问题，这是其在第一次训练取得较低结果的原因。

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**

表 6：*MobileNet* 使用不同激活函数在 *ImageNet* 上的训练结果。

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**

表 8：*Inception-v4* 使用不同激活函数在 *ImageNet* 上的训练结果。

表 4-8 的结果表明 Swish 的强大性能。Swish 在移动端模型上的性能尤其地好，在 Mobile NASNet-A 上的性能比 ReLU 高 0.9%，在 MobileNet 上的性能比 ReLU 高 2.2%。此外，Swish 在大部分模型上的性能都匹配或优于最佳性能的基线（最佳性能基线随着模型的改变而改变）。

5 结语

Swish 是一种新型激活函数，公式为： f(x) = x · sigmoid(x)。Swish 具备无上界有下界、平滑、非单调的特性，这些都在 Swish 和类似激活函数的性能中发挥有利影响。我们在实验中使用了专为 ReLU 设计的模型和超参数，然后用 Swish 替换掉 ReLU 激活函数；仅仅是如此简单、非最优的迭代步数仍使得 Swish 持续优于 ReLU 和其他激活函数。我们期待当模型和超参数都专为 Swish 设计的时候，Swish 还能取得进一步的提升。Swish 的简洁性及其与 ReLU 的相似性意味着在任何网络中替代 ReLU 都只是改变一行代码这么简单的事。

/原文链接：https://www.reddit.com/r/MachineLearning/comments/773epu/r_swish_a_selfgated_activation_function_google//

谷歌大脑提出新型激活函数Swish惹争议：可直接替换并优于ReLU？（附机器之心测试）