深度学习

神经网络：

神经网络由多个神经元组成，每个神经元执行如下操作：

$$y = w \cdot x + b$$

深度学习常见参数

参数名	定义	定义说明
Epoch	训练轮数	整个训练数据集被模型完整训练多少遍。例如设置为 10，表示模型会“看”完所有数据 10 次。
Batch Size	批量大小	每次训练时输入模型的数据样本数量。例如 batch_size=32 表示每次用 32 条数据进行训练。
Learning Rate	学习率	控制模型参数更新步长的大小。学习率越大，更新越快；太大会导致不稳定，太小会收敛慢。
Optimizer	优化器	决定如何更新模型参数的算法，如 Adam、SGD 等。通常使用 Adam 或 AdamW。
Loss Function	损失函数	衡量模型预测结果与真实标签之间的差距，目标是通过训练不断减小这个差距。
Weight Decay	权重衰减	L2 正则化项，用于控制模型复杂度，防止过拟合。
Gradient Clipping	梯度裁剪	限制梯度的最大值，防止梯度爆炸，尤其在训练 RNN 或 Transformer 时常用。
Warmup Steps	预热步数	在训练开始阶段逐步增加学习率，使模型更稳定地进入训练状态。
Validation Set	验证集	从训练集中划分出来的一小部分数据，用于评估模型在训练过程中的表现。
Early Stopping	早停机制	当验证集上的性能不再提升时提前终止训练，避免浪费时间和过拟合。

梯度（Gradient）

在神经网络中，**梯度** 是损失函数（Loss）对模型参数（如权重 w 和偏置 b）的偏导数。

例如：

$$y = w \cdot x + b$$

再加一个损失函数，比如均方误差（MSE）：

$$L = \frac{1}{2}(y - y_{\text{true}})^2$$

我们要更新参数 w 和 b，就要计算梯度：

$$\frac{\partial L}{\partial w}, \quad \frac{\partial L}{\partial b}$$

链式法则：梯度是怎么算出来的？

我们以最简单的反向传播为例：

正向传播：

$$z = w \cdot x + b$$
$$y = \sigma(z) \quad \text{（比如 Sigmoid 激活）}$$
$$L = \text{loss}(y, y_{\text{true}})$$

反向传播：

我们要计算 $\frac{\partial L}{\partial w}$，使用链式法则：

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w}$$

其中：$\frac{\partial z}{\partial w} = x$

所以：

$$\frac{\partial L}{\partial w} = (\text{一些值}) \cdot x$$

结论：梯度 ∇w 与输入 x 成正比

也就是说，梯度的方向和大小都依赖于输入 x 的值。

激活函数

1.激活函数作用

神经网络是线性的，无法解决非线性的问题，加入激活函数就是给模型引入非线性能力；

不同的激活函数，特点和作用不同：

- `Sigmoid`和`tanh`的特点是将输出限制在`(0,1)`和`(-1,1)`之间，说明`Sigmoid`和`tanh`适合做概率值的处理，例如LSTM中的各种门；而`ReLU`就不行，因为`ReLU`无最大值限制，可能会出现很大值。
- `ReLU`适合用于深层网络的训练，而`Sigmoid`和`tanh`则不行，因为它们会出现梯度消失。

2.梯度爆炸和梯度消失

模型中的梯度爆炸和梯度消失问题：

1. 激活函数导致的梯度消失，像 `sigmoid`和 `tanh` 都会导致梯度消失；
2. 矩阵连乘也会导致梯度消失，这个原因导致的梯度消失无法通过更换激活函数来避免。**直观的说就是在反向传播时，梯度会连乘，当梯度都小于1.0时，就会出现梯度消失；当梯度都大于1.0时，就会出现梯度爆炸。**

如何解决梯度爆炸和梯度消失问题：

1. 上述第一个问题只需要使用像 ReLU 这种激活函数就可以解决；

2. 上述第二个问题没有能够完全解决的方法，目前有一些方法可以很大程度上进行缓解该问题，比如：对梯度做截断解决梯度爆炸问题、残差连接、normalize。由于使用了残差连接和 normalize 之后梯度消失和梯度爆炸已经极少出现了，所以目前可以认为该问题已经解决了。

> **Important**
>
> # 激活函数对比表
>
> | 激活函数 | 设计思路 | 反向传播优势 |
> | --- | --- | --- |
> | **Sigmoid** | 将输出压缩到 (0,1)，用于二分类、门控机制等场景。模拟神经元“是否激活”的概率输出。 | 平滑可导，便于梯度计算；适合输出为概率的场景，如二分类输出层。 |
> | **Tanh** | 在 Sigmoid 基础上改进，输出范围调整为 (-1, 1)，使输出更接近 zero-centered。 | 输出均值接近 0，缓解 ReLU 中存在的方向受限问题，更适合循环网络（RNN）等结构。 |
> | **ReLU** | 简化激活方式，只保留正区间信息，负区间直接置零。旨在加快训练速度并避免饱和。 | 正区间的导数恒为 1，极大缓解梯度消失问题；计算效率高，是现代深度学习最常用激活函数之一。 |
> | **Leaky ReLU** | 改进 ReLU，在负区间引入一个很小的斜率（如 0.01），解决 Dead ReLU 问题。 | 所有输入都有非零梯度，缓解神经元“死亡”现象，适合深层网络训练。 |
> | **PReLU** | 进一步改进 Leaky ReLU，将负区间的斜率设为可学习参数，提升模型自适应能力。 | 负区间的斜率在训练中动态调整，提高表达能力和泛化性能。 |
> | **RReLU** | 在训练时对负区间的斜率进行随机采样(高斯分布)，推理时取期望值，增强鲁棒性。 | 引入噪声扰动，防止过拟合，适用于数据增强场景下的激活建模。 |
> | **ELU** | 在负区间引入指数形式的非线性变换，使得输出整体分布更接近 zero-centered。 | 输出均值接近 0，加速收敛；缓解 Dead ReLU 问题；负区间的平滑变化有助于梯度稳定传播。 |
> | **GELU** | 基于标准正态分布设计，输入以一定概率作为输出保留或抑制，融合了概率建模思想。 | 函数处处连续光滑，结合概率机制提升表达能力，表现优于 ReLU，广泛用于 Transformer 架构中。 |
> | **Swish** | 由 Google 提出，形式为 $x \cdot \sigma(x)$，是非单调且无上界的激活函数。 | 非单调性使其具有更强的表达能力，适合深层网络；无上界避免饱和，梯度传播更稳定。 |
> | **GLU（Gated Linear Unit）** | 通过门控机制控制信息流动，形式为$x \otimes \sigma(g(x))$，其中 g(x) 是另一分支输出。 | 门控机制带来更强的信息筛选能力，能提升模型表达力；广泛应用于 LLaMA、PaLM 等大模型中的 FFN 层。 |

为什么当输入值很小的时候激活函数的输出值都趋近于0?

核心原因解析

1. **【防止信息过载】——抑制无用/不重要信号**
   神经网络每一层都在提取特征；如果所有输入都保留（不管大小），网络会记住很多噪声或无关信息；
   通过让小负值输出接近 0，相当于实现了 一种软性的"注意力筛选"机制 ； 这有助于模型专注于重要的输入特征，忽略干扰信息。
2. **【缓解梯度爆炸/消失】——使反向传播更稳定**
   在反向传播中，梯度是链式相乘的；如果激活函数在负区间的导数也很大，会导致梯度不断放大（爆炸）；
   而如果输出和导数都很小，就能避免这种不稳定现象。
   比如：

   1.ReLU 的导数在负区间为 0 → 完全阻断梯度；

   2.Swish 和 GELU 的导数逐渐变小 → 平滑衰减梯度
3. **【提高稀疏性】——模仿生物神经元特性**
   生物神经元并不是对每个刺激都响应；只有当刺激足够强时才会"激活"；所以人工神经网络引入"负输入→输出趋近于0"的机制，是为了模拟这种稀疏激活行为； 这样可以让模型学习到更有意义的表示。
   举例：如果某个神经元对当前输入几乎没反应（输出为 0），说明它可能只对某些特定模式敏感；这种稀疏性提升了模型的表达能力和泛化性能。
4. **【零中心化】——让数据分布更合理**
   如果激活函数的输出均值远离 0（如 Sigmoid 输出总大于 0），会导致下一层输入数据偏移；导致参数更新方向受限（梯度震荡）；
   所以**现代激活函数（如 ELU、Swish、GELU）都在努力让输出接近 zero-centered；让神经网络的学习过程更高效、收敛更快。**
5. **【门控机制】——控制信息流动（如 GLU）**
   在像 GLU、SwiGLU 这类激活函数中，负值输出代表"关闭"；当某个输入特别小（如 -10），sigmoid(g(x)) 接近 0，意味着"这个输入不重要"，于是被抑制；这是一种显式的"门控机制"，可以实现信息流的精细控制。

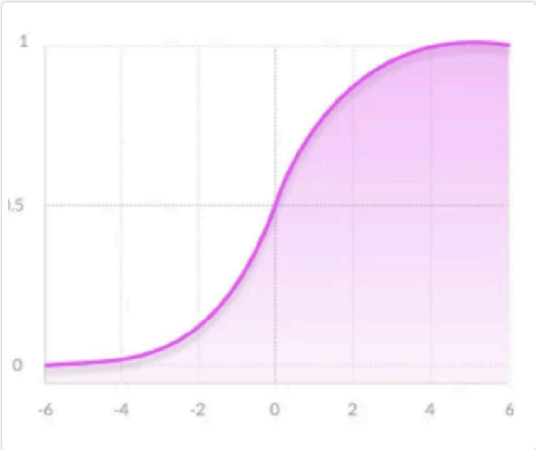
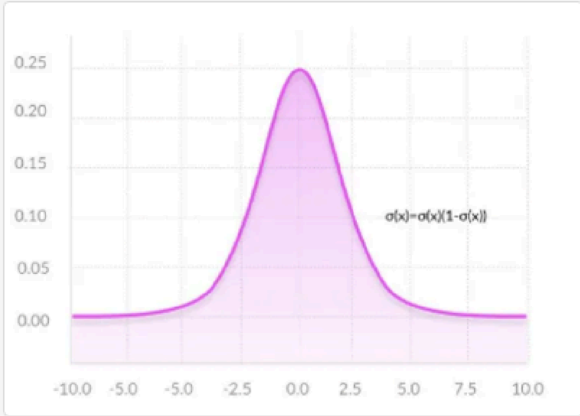
3.Sigmoid

Sigmoid 函数公式：

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

导数公式：

$$\sigma'(z) = \sigma(z)(1-\sigma(z))$$

图1：sigmoid函数图像	图2：sigmoid导数图像
	

优点：

- 平滑，易于求导；
- 取值范围是 $(0, 1)$，可直接用于求概率值的问题或者分类问题；
- 比如 LSTM 中的门、二分类或多标签分类问题。

缺点：

- 计算量大，包含幂运算和除法运算；
- Sigmoid 的导数取值范围是 $[0, 0.25]$，最大值小于 1，在反向传播中容易出现**梯度消失**；
- 输出不是 zero-centered，会导致当前层接收到上一层非零均值的信号作为输入，随着网络加深，会改变数据的原始分布。

> ⓘ **Note**
>
> **1. 输入非零中心化（Non-zero-centered Input）**
>
> - 假设某一层神经元的输出都是正数（例如使用 Sigmoid 激活函数）；
> - 那么下一层神经元接收到的输入数据 x 也都是正数；
> - 这导致该层的输入数据分布偏移，不再是围绕 0 分布（non-zero-centered）。
>
> ---
>
> **2. 梯度更新方向受限**
>
> 考虑一个简单的线性变换：
>
> $$y = w \cdot x + b$$
>
> 在反向传播中，权重 w 的梯度为：
>
> $$\nabla w = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial w} = \frac{\partial L}{\partial y} \cdot x$$
>
> 如果所有 $x > 0$，那么：
>
> - 权重 w 的梯度要么全为正，要么全为负；
> - 参数更新只能朝一个方向进行（全部增加或全部减少）；
> - 更新路径变得“曲折”，不能高效地收敛。
>
> ---

> **3. 影响训练效率**
>
> 这种现象被称为 **梯度更新的震荡（oscillation）**：
>
> - 导致模型收敛速度变慢；
> - 容易陷入局部最优；
> - 不利于深层网络的学习稳定性

4. Tanh

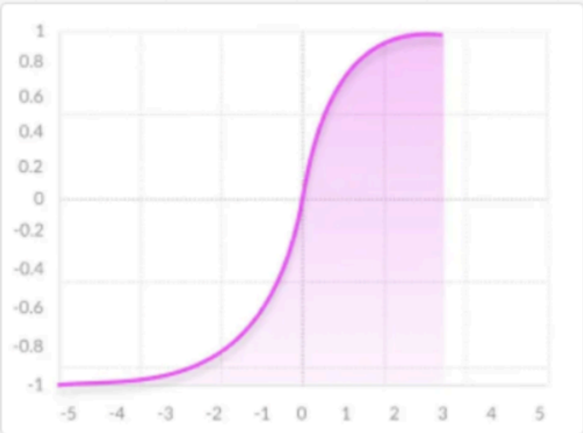
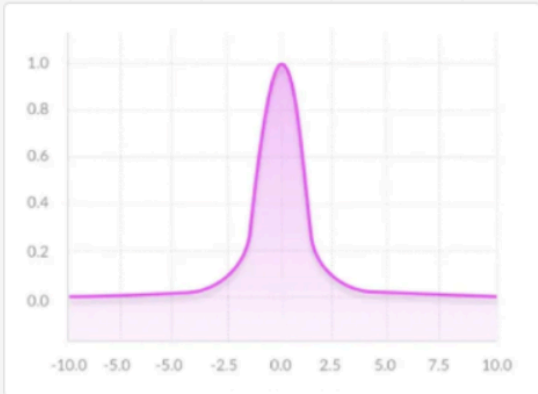
Tanh 函数公式：

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = \frac{2}{1 + e^{-2z}} - 1$$

可以看出，Tanh 是由 Sigmoid 经过平移和拉伸得到的。**Tanh 的取值范围是：**$(-1, 1)$

导数公式：

$$\tanh'(x) = 1 - (\tanh(x))^2$$

图3：tanh函数图像	图4：tanh导数图像
	

优点：

- Tanh 是对 Sigmoid 的一种改进，解决了输出均值不为 0 的问题；
- Tanh 的导数取值范围是 $(0, 1)$，相比 Sigmoid 在反向传播中梯度消失问题稍有缓解；

缺点：

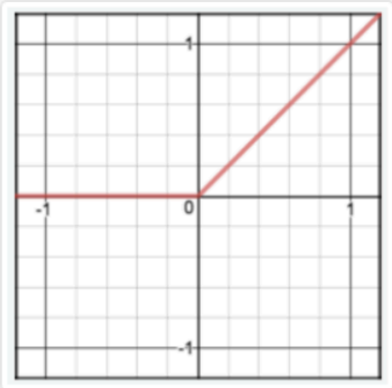
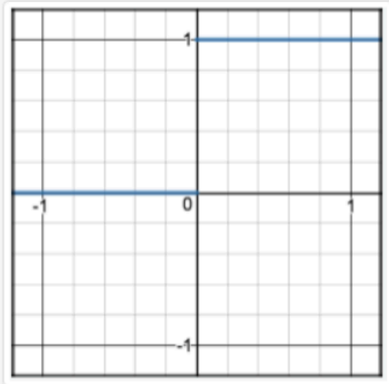
- 依然存在梯度消失问题；
- 同样含有幂运算，计算量较大。

5. ReLU 系列

5.1 ReLU

ReLU 全称为 **Rectified Linear Unit**，即修正线性单元函数。该函数公式如下：

$$\mathrm{ReLU}(z) = \begin{cases} z, & z > 0 \\ 0, & z \leq 0 \end{cases}$$

图5：ReLU函数	图6：导数
$\mathrm{ReLU}(z)=\begin{cases}0 & \text{if } z\leqslant 0\\ z & \text{if } z>0\end{cases}$	$\mathrm{ReLU}'(z)=\begin{cases}0 & \text{if } z\leqslant 0\\ 1 & \text{if } z>0\end{cases}$
	

优缺点：

- 相比于 `sigmoid`、`tanh` 这两个激活函数，`ReLU` 激活函数的优缺点如下：
    - 当 `z>0` 时，ReLU 激活函数的导数恒为常数1，这就避免了 sigmoid 和 tanh 会在神经网络层数比较深的时候出现的梯度消失的问题；
    - 计算复杂度低，不再含有幂运算，只需要一个阈值就能够得到其导数；
    - 经过实际实验发现，**使用 ReLU 作为激活函数，模型收敛的速度比 sigmoid 和 tanh 快**；
    - 当z<0时，ReLU 激活函数的导数恒为常数0，这既带来了一些有利的方面，也导致了一些坏的方面，分别进行描述。
        - 有利的方面：在深度学习中，目标是从大量数据中学习到关键特征，也就是把密集矩阵转化为稀疏矩阵，保留数据的关键信息，去除噪音，这样的模型就有了鲁棒性。ReLU 激活函数中将 `z<0` 的部分置为0，就是产生稀疏矩阵的过程。
        - 坏的方面：将 `z<0` 的部分梯度直接置为0会导致 Dead ReLU Problem(神经元坏死现象)。**可能会导致部分神经元不再对输入数据做响应，无论输入什么数据，该部分神经元的参数都不会被更新。**（这个问题是一个非常严重的问题，后续不少工作都是在解决这个问题）
    - ReLU 有可能会导致梯度爆炸问题，解决方法是梯度截断；
    - ReLU 的输出不是 0 均值的，这个和 sigmoid 类似。（后续的优化工作 ELU 在该问题上解决的比较好，ELU 的输出是近似为0的）

5.2 Leaky ReLU

为了解决 ReLU 的 Dead ReLU 问题，提出了 **Leaky ReLU（渗漏整流线性单元）**，是 ReLU 的一个变体。

在 $z>0$ 的部分与 ReLU 相同；
在 $z\leq 0$ 的部分，采用一个非常小的斜率（如 0.01）。

公式：

$$\mathrm{LeakyReLU}(z)=\begin{cases}0.01z & \text{if } z\leq 0\\ z & \text{if } z>0\end{cases}$$

特点：

能在一定程度上缓解 Dead ReLU 问题；效果不稳定，实际应用较少。

5.3 PReLU、RReLU

PReLU（Parametric ReLU）：

PReLU 是 Leaky ReLU 的改进版本，其负区间的斜率 α 是可学习参数。

公式：

$$\mathrm{PReLU}(z) = \begin{cases} \alpha \cdot z & \text{if } z \leq 0 \\ z & \text{if } z > 0 \end{cases}$$

其中 α 是通过反向传播学习得到的。

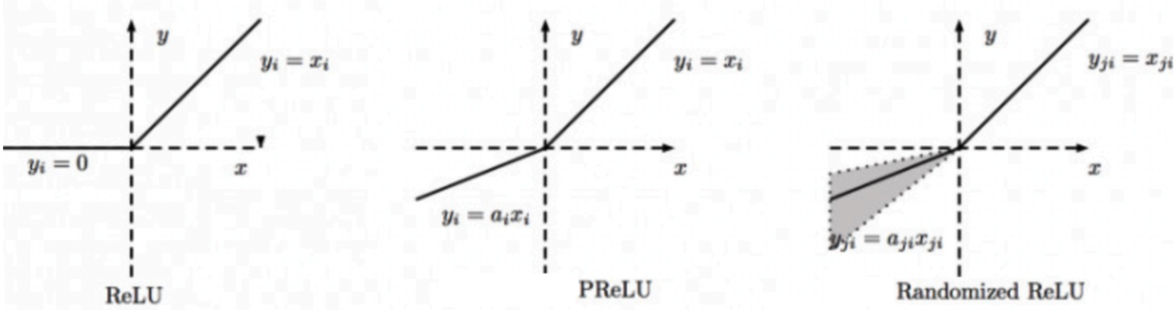
RReLU（Randomized ReLU）：

RReLU 在训练过程中对负区间的斜率进行随机采样，在推理时取期望值。

公式：

$$\mathrm{RReLU}(z) = \begin{cases} \alpha \cdot z & \text{if } z \leq 0 \\ z & \text{if } z > 0 \end{cases}$$

其中 α 是从高斯分布中随机生成的值。



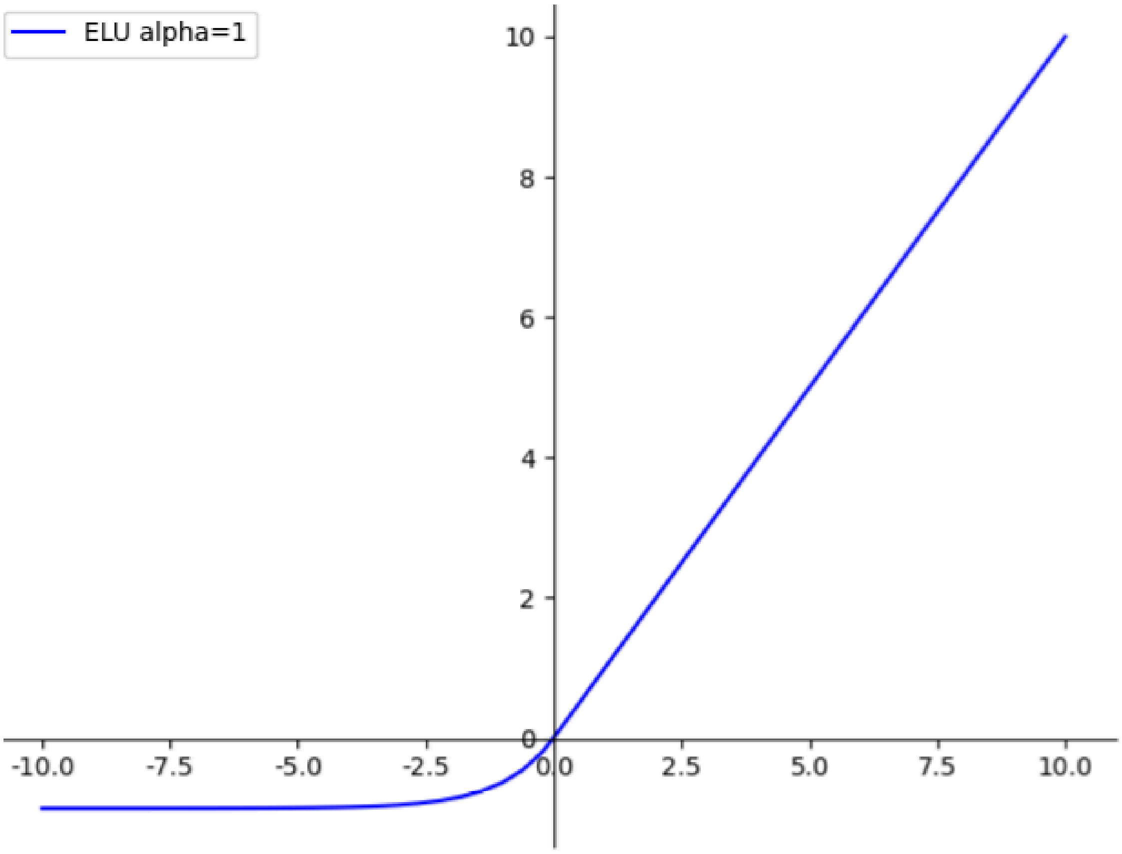
5.4 ELU（Exponential Linear Unit）

ELU 是为了解决 ReLU 的问题而提出的激活函数。与 ReLU 不同的是，ELU 在负区间具有非零输出，使得输出均值接近于 0。

公式：

$$\mathrm{ELU}(x) = \begin{cases} x & x > 0 \\ \alpha(e^x - 1) & x \leq 0 \end{cases}$$

其中 α 是可学习参数。



特点：

其中 α不是固定的，是通过反向传播学习出来的。ELU的一个小问题是需要exp计算，运算量会更大一些。

- 融合了sigmoid和ReLU，左侧具有软饱和性，右侧无饱和性。
- 右侧线性部分使得ELU能够缓解梯度消失，而左侧软饱能够让ELU对输入变化或噪声更鲁棒。
- ELU的输出均值接近于零，所以收敛速度更快。

---

6. GeLU（Gaussian Error Linear Unit）

GeLU 出自 2016 年论文《Gaussian Error Linear Units (GELUs)》，是一种基于概率思想设计的激活函数。

6.1 介绍

> **Tip**
>
> Relu函数中包含两种映射：一个是恒等映射（identity mapping），当输入值大于零时就是恒等映射；一个是置零映射（zero mapping），当输入值小于等于零时就是置零映射。
>
> 参考 ReLU 激活函数，设计另外一个包含恒等映射和置零映射的激活函数，并且参考 ReLU 函数来看，新激活函数应该有如下性质：
>
> 1. 在输入 `x` 满足某些条件时，为恒等映射；
> 2. 在输入 `x` 满足另外一些条件时，为置零映射；
> 3. 在输入 `x` 是一个较大的正值时，更希望为恒等映射；在输入 `x` 为一个较小的负值时，更希望是一个置零映射；

以上就是想要新设计的激活函数的性质。GeLU 的核心思想是根据输入值的概率分布决定是否保留或抑制该值。

公式：

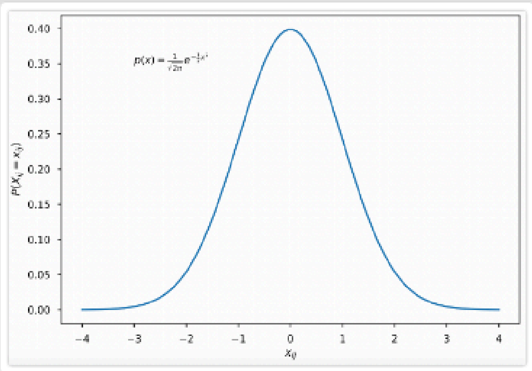
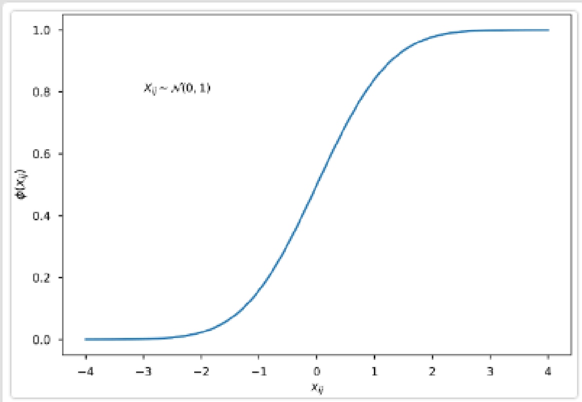
$$\mathrm{GELU}(x) = x \cdot \Phi(x)$$

其中 $\Phi(x)$ 是标准正态分布的累积分布函数。

概率解释：

下面的图7和图8是标准正态分布的概率密度函数和累积分布函数的图像。接下来根据下图8中的累积分布函数设计一个新的函数。

符号定义：输入值用 x表示，$\phi(\cdot)$表示下图8中的正态分布的累积分布函数，$f(\cdot)$表示新设计的函数。

图7：标准正态分布概率密度函数	图8：标准正态分布累积分布函数
$p(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$	$\phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}t^2} dt$
	

设计的新函数：给定输入值x，函数 $f(x)$的输出值以 $\phi(x)$的概率采用恒等映射，以 1-$\phi(x)$的概率采用置零映射。也就是下述公式:

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

- 当输入 x 是一个较大的正值时，从图8中可以看出$\phi(x)$的函数图像逐渐趋近于1，由于函数 $f(x)$的输出值以 $\phi(x)$的概率采用恒等映射，所以有接近于1的概率采用恒等映射；
- 当输入x 是一个较小的负值时，$\phi(x)$趋近于0，由于函数$f(x)$以 1-$\phi(x)$的概率采用置零映射，所以有接近于1的概率采用置零映射；

如果改用图7中概率密度函数（PDF）来表示，则函数形式如下：

$$f(x) = x \cdot p(X < x) + 0 \cdot (1 - p(X < x)) = x \cdot p(X < x)$$

其中：

- x 表示实际的输入值；
- X 表示服从标准正态分布的随机变量。

最终得到了 GELU（Gaussian Error Linear Unit）的常见形式：

$$\mathrm{GELU}(x) = x \cdot p(X < x) = x \cdot \Phi(x)$$

其中：$\Phi(x)$ 是标准正态分布的累积分布函数。

6.2 函数及导数

GeLU 公式为：

$$GELU = x \cdot \Phi(x)$$

使用该函数作为激活函数时，需要求解其导数。对其求导可得：

$$\frac{d}{dx}GELU = \Phi(x) + x\frac{d}{dx}\Phi(x) = \Phi(x) + x \cdot p(X = x)$$

其中 X 是随机变量，$p(X = x)$ 是图7中的标准正态分布概率密度函数中，随机变量取值为 x 时的值。

GELU 函数及其导数的图像如下所示:可以看出其函数图像和 ReLU 非常相似，其导数图像也和 ReLU 的导数图像非常相似，不过该图像是连续的。

图9：GELU函数

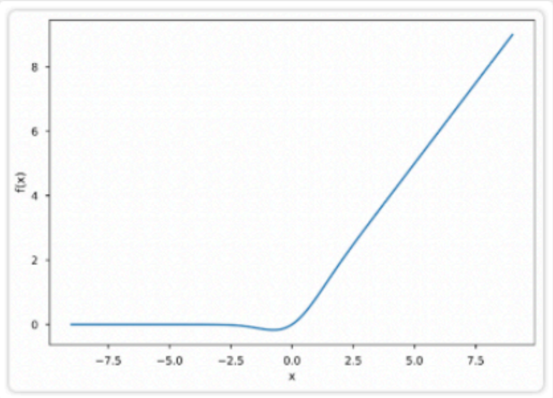
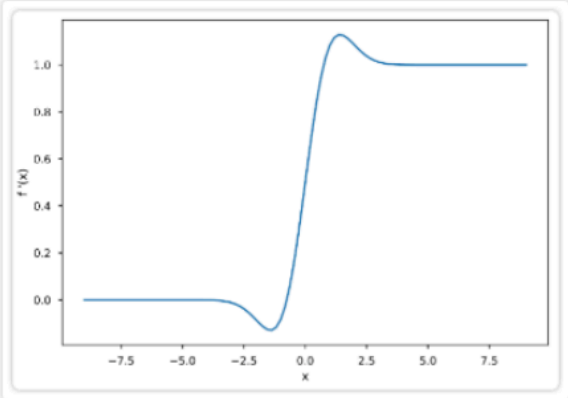


图10：GELU导数



GELU 激活函数的优缺点：

1.从其函数图像可以看出，在负值区域，不再全为0，这解决了 Dead ReLU 问题；
2.GELU 函数是处处连续、光滑可导的；

6.3 精确计算

对于 GeLU 的加速计算有两种方法。

第一种方法是精确求解。有一个函数为 Gauss Error function (gef)，由于使用率非常高所以在常见的库（比如TensorFlow、PyTorch）中都有针对该函数的优化，该函数的公式如下。

$$\text{erf}(y) = \frac{2}{\sqrt{\pi}}\int_0^y e^{-t^2}dt$$

所以如果能够先求解出 $\text{erf}(\cdot)$，再由该函数求解出 $\Phi(x)$，那么可以加快计算。下面省略具体的推导过程，直接给出计算公式：

$$\Phi(x) = \frac{1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)}{2}$$

另一种方法是不精确求解，而是求解其近似值。为了加速计算，还可以使用近似计算的方式。GELU 的近似公式如下所示：

$$GELU = 0.5 * x\left(1 + \tanh\left[\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right]\right)$$

7. Swish

出自 2017 年的论文《Searching for Activation Functions》

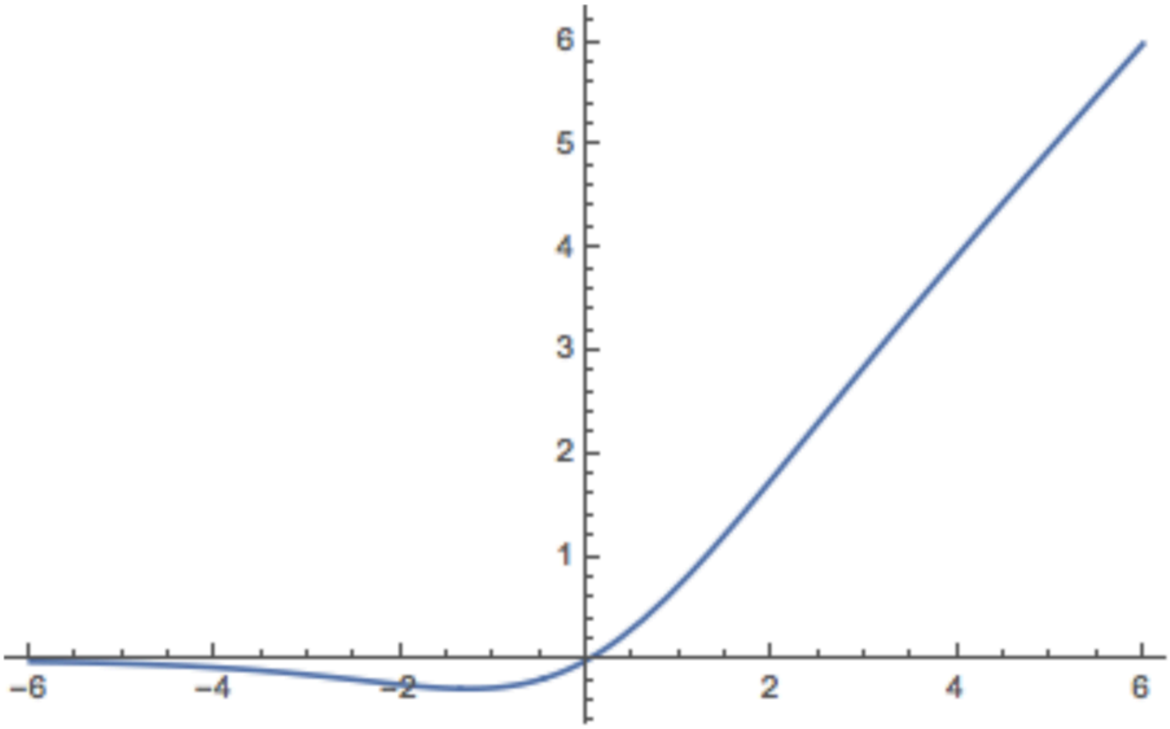
该激活函数的公式为：

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

Swish 导数：

$$f'(x) = \sigma(x) + x \cdot \sigma(x) \cdot (1 - \sigma(x)) = x \cdot \sigma(x) + \sigma(x)(1 - x \cdot \sigma(x)) = f(x) + \sigma(x) \cdot (1 - f(x))$$

该激活函数的图像为：



Swish 特点：

- 和 ReLU 一样，没有上边界，因此不会出现梯度饱和现象；
- 有下边界，可以产生更强的正则化效果（x 左半轴慢慢趋近于 0）；
- 非单调；
- 处处连续且可导，更容易训练；

> ⓘ **Note**
>
> 关于正则化效果：x 轴越靠近左半轴，纵坐标的值越小，甚至接近于 0，如果 x 值是 -10，那么经过激活之后的值接近于 0，就可以一定程度上过滤掉一部分信息，起到正则化的效果。

8. GLU

PaLM 和 LLaMA 中都使用 SwiGLU 替换了 FFN。

出自 2017 年的论文《Language Modeling with Gated Convolutional Networks》

GLU 全称为 Gated Linear Unit，即门控线性单元函数。

参考 ReLU 激活函数，激活函数 GLU 的公式为如下形式：

$$\mathrm{GLU}(x) = x \otimes \sigma(g(x))$$

这里有一个新符号 $g(x)$ 表示的是向量 x 经过一层 MLP 或者卷积,$\otimes$ 表示两个向量逐元素相乘,σ 表示 Sigmoid 函数。

> **Note**
>
> 当 $\sigma(g(x))$ 趋近于 0 时，表示对 x 进行阻断;当 $\sigma(g(x))$ 趋近于 1 时，表示允许 x 通过以此实现门控激活函数的效果