0.1 Position-Wise Feed-Forward Network

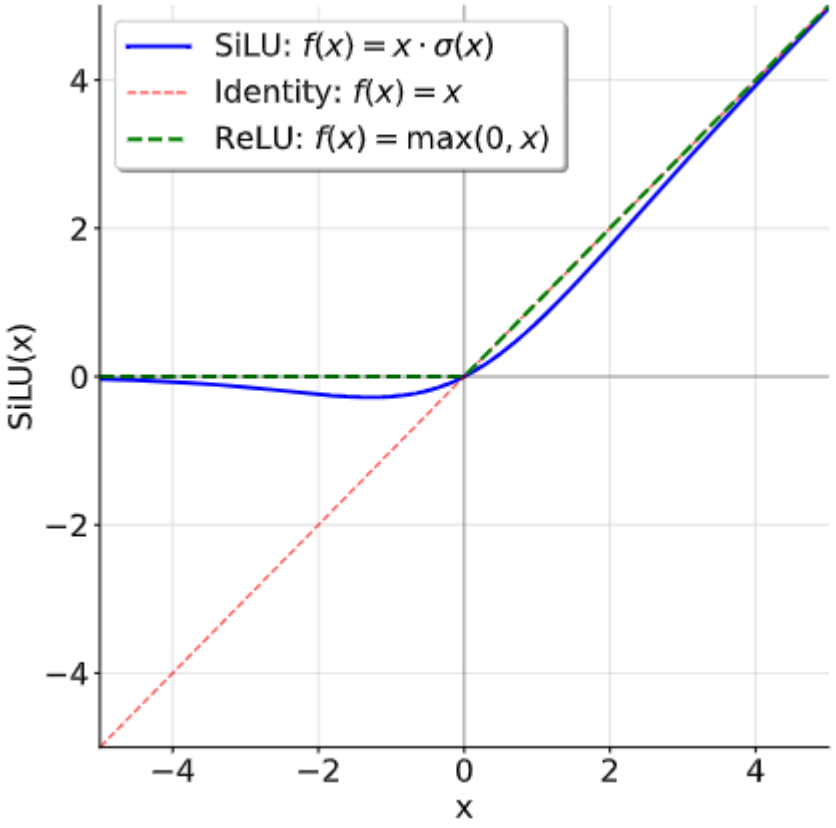
0.1.1 激活函数

在原始 Transformer 中，FFN 使用的是 ReLU 激活函数：$\text{ReLU}(x) = \max(0, x)$ 。

- 它的特点是简单、计算快，但它在 $x < 0$ 时导数完全为 0，且在 0 点不平滑。

现代大模型普遍换用了 **SiLU (Sigmoid Linear Unit)**，也叫 Swish 。

- **公式**：$\text{SiLU}(x) = x \cdot \sigma(x) = \frac{x}{1+e^{-x}}$ 。
- **特点**：
  1. **平滑（Smooth）**：它处处可导，没有 ReLU 在 0 点的那个折角。
  2. **非单调（Non-monotonic）**：在 x 略小于 0 时，它会有一个小小的下凹（允许少量的负值保留），而不是像 ReLU 那样一刀切成 0。



- 可以把它看作是一个“软化”版的 ReLU，它保留了更多微弱的负信号，有助于梯度的精细调整。

0.1.2 Gated Linear Unit (GLU，门控线性单元)

GLU 的原始定义如下:

$$GLU(x) = \sigma(W_1 x) \odot (W_2 x)$$

- $W_2 x$（**信息路径 / Information Path**）：
  - 这是一个纯线性的变换。它代表了原始的信息或特征值。
  - 可以把它看作是数据的“实际内容信息”。
- $\sigma(W_1 x)$（**门控路径 / Gate Path**）：

  - σ 是 Sigmoid 函数，将输出压缩到 $(0,1)$ 区间。
  - W_1x 计算的是一个控制信号。
  - 这代表了通过率（概率）。0 代表完全关闭，1 代表完全通过。
- ⊙（**逐元素乘法 / Hadamard Product**）：
  - 这是核心交互。信息乘以通过率。

0.1.3 从 FFN 到 SwiGLU

以前的 FFN 走地是升维 -> 激活 -> 降维这条路，现在引入了 Gated Linear Unit 机制：

$$\text{FFN}(x) = \text{SwiGLU}(x, W_1, W_2, W_3) = W_2(\text{SiLU}(W_1x) \odot (W_3x))$$

这里有 3 个权重矩阵，而传统的 FFN 只有 2 个，分别是：

1. W_1 **(Gate Projection)**：
   - 输入 x 乘以 W_1，然后通过 SiLU 激活。
   - 这部分充当了“门控”的角色（虽然 SiLU 不是严格的 0-1，但起到了非线性筛选的作用）。
2. W_3 **(Up Projection)**：
   - 输入 x 乘以 W_3，不经过激活函数。
   - 这部分保留了原始的线性特征信息。
3. ⊙ **(Element-wise Product)**：
   - 将上述两部分逐元素相乘。这里就是“门控”发生的地方：W_1 路径筛选并控制 W_3 路径的信息保留量。
4. W_2 **(Down Projection)**：
   - 将融合后的结果映射回原来的维度。

```python
class SwiGLU(nn.Module):

    def __init__(self, d_model: int, d_ff: int):
        super().__init__()
        self.w1 = Linear(d_model, d_ff)
        self.w2 = Linear(d_ff, d_model)
        self.w3 = Linear(d_model, d_ff)

    def forward(self, x):
        return self.w2(silu(self.w1(x)) * self.w3(x))

def silu(x: torch.Tensor):
    return x * torch.sigmoid(x)
```

问题：为什么 FFN 中要先升维再降维？

“先升维再降维”本质上是一个“解压 -> 处理 -> 压缩”的过程。

- 在低维空间（d_{model}），信息是高度压缩和抽象的，便于传输（节省显存和通信带宽）。
- 在高维空间（d_{ff}），信息被解压，便于神经网络利用非线性函数进行精细的特征提取和知识检索。

就像做统计分析：原始数据表（低维）可能看不出规律，把它通过特征工程扩展成巨大的交互项矩阵（升维），用 Lasso 回归筛选出重要变量（激活/门控），最后输出一个简洁的模型系数（降维）