activation function, compare sigmoid, tanh, relu, leaky relu, elu, gelu, using examples and graph

以下是主流激活函数的对比，包含数学表达式、特性、代码示例和图示，帮助您直观理解它们的差异和适用场景：

1. 函数对比总览

函数	公式	输出范围	优点	缺点
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	(0, 1)	概率解释直观	梯度
Tanh	$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	(-1, 1)	零中心化	梯度
ReLU	$\mathrm{ReLU}(x) = \max(0, x)$	[0, +∞)	计算高效，缓解梯度消失	负区
Leaky ReLU	$\max(0.01x, x)$	(-∞, +∞)	缓解死亡神经元	需调
ELU	x if $x > 0$ else $\alpha(e^{x} - 1)$	(-α, +∞)	负区间平滑	计算
GELU	$x\Phi(x)$ （Φ为标准正态CDF）	(-∞, +∞)	Transformer常用，近似ReLU+随机性	计算

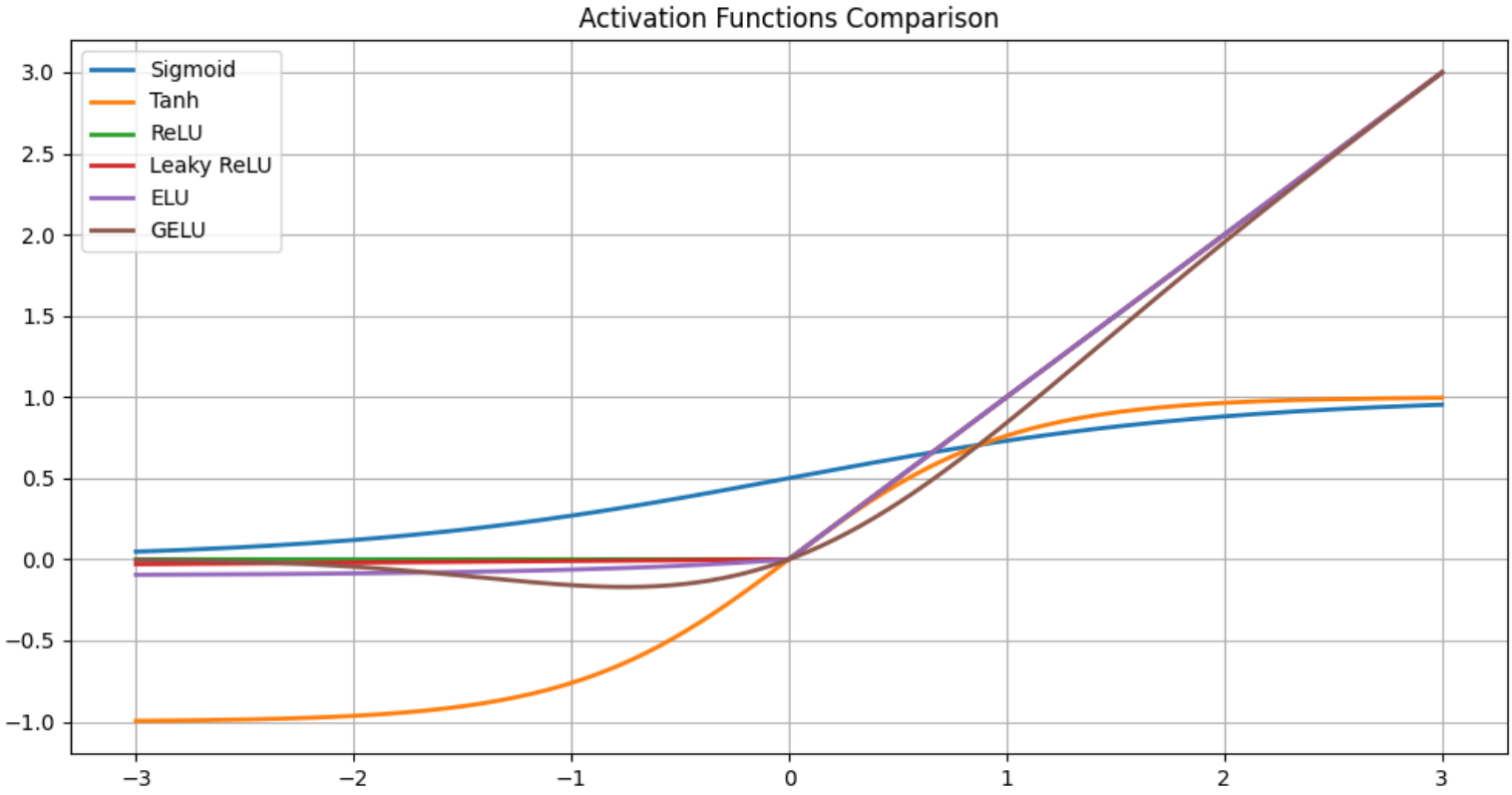
1. 函数对比总览

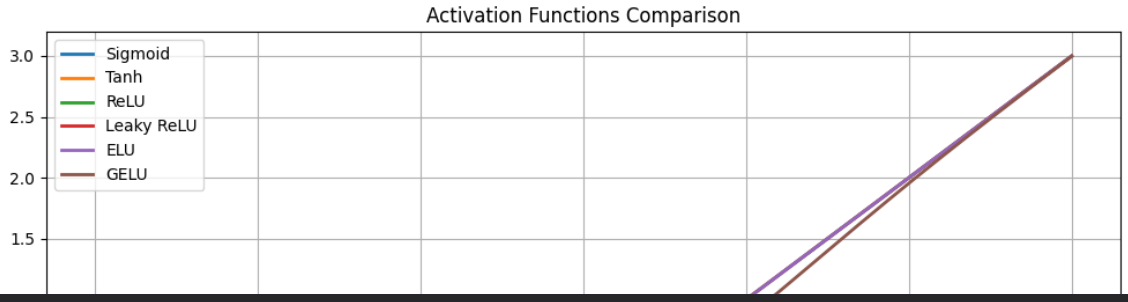
	输出范围	优点	缺点	死神经元风险
$\frac{1}{x}$	(0, 1)	概率解释直观	梯度消失严重	无
$\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	(-1, 1)	零中心化	梯度消失	无
$\max(0, x)$	[0, +∞)	计算高效，缓解梯度消失	负区间死亡	高
$x)$	(-∞, +∞)	缓解死亡神经元	需调斜率参数	低
se $\alpha(e^{x} - 1)$	(-α, +∞)	负区间平滑	计算复杂度略高	极低
标准正态CDF）	(-∞, +∞)	Transformer常用，近似ReLU+随机性	计算复杂	低

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-3, 3, 500)
functions = {
    "Sigmoid": lambda x: 1/(1+np.exp(-x)),
    "Tanh": np.tanh,
    "ReLU": lambda x: np.maximum(0, x),
    "Leaky ReLU": lambda x: np.maximum(0.01*x, x),
    "ELU": lambda x: np.where(x>0, x, 0.1*(np.exp(x)-1)),
    "GELU": lambda x: 0.5*x*(1 + np.tanh(np.sqrt(2/np.pi)*(x +
        0.044715*x**3)))
}

plt.figure(figsize=(12, 6))
for name, func in functions.items():
    plt.plot(x, func(x), label=name, lw=2)
plt.title("Activation Functions Comparison")
plt.legend()
plt.grid()
plt.show()
```


Activation Functions Comparison
Sigmoid
Tanh
ReLU
Leaky ReLU
ELU
GELU
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−3
−2
−1
0
1
2
3



3. 具体分析

(1) Sigmoid

- **用例**：二分类输出层（概率输出）

```python
model.add(Dense(1, activation='sigmoid'))  # 二分类任务
```

- **问题**：梯度在 $|x| > 3$ 时接近0，导致深层网络难以训练。

(2) Tanh

- **用例**：RNN隐藏层（LSTM/GRU）

```python
model.add(LSTM(64, activation='tanh'))  # 传统RNN结构
```

- **优势**：输出以0为中心，缓解梯度偏移问题。

(3) ReLU

- **用例**：CNN和全连接层

```python
model.add(Conv2D(32, kernel_size=3, activation='relu'))
```

- **注意**：需配合He初始化（`kernel_initializer='he_normal'`）。

(4) Leaky ReLU

- **用例**：对抗生成网络（GAN）

```python
model.add(Dense(128, activation=LeakyReLU(alpha=0.2)))  # alpha通常取0.01~0.2
```

- **改进**：负区间斜率缓解神经元死亡。

(5) ELU

- **用例**：自编码器（Autoencoder）

```python
model.add(Dense(64, activation='elu'))  # 负区间指数平滑
```

- **效果**：比ReLU更鲁棒，但计算稍慢。

(6) GELU

- **用例**：Transformer/BERT

```python
# Hugging Face Transformers默认使用
from transformers import BertModel
bert = BertModel.from_pretrained('bert-base-uncased')  # 内部使用GELU
```

- **数学近似**：可用 `0.5x(1 + tanh[√(2/π)(x + 0.044715x³)])` 加速计算。

4. 梯度对比（反向传播影响）

函数	梯度公式	梯度范围
Sigmoid	$\sigma'(x) = \sigma(x)(1 - \sigma(x))$	(0, 0.25]
Tanh	$1 - \tanh^2(x)$	(0, 1]
ReLU	$1 \text{ if } x > 0 \text{ else } 0$	{0, 1}
Leaky ReLU	$1 \text{ if } x > 0 \text{ else } \alpha$	{α, 1}
ELU	$1 \text{ if } x > 0 \text{ else } \alpha e^x$	(α, 1]
GELU	$\Phi(x) + x\phi(x)$ （φ为PDF）	(0, ~1.1)

5. 选择指南

场景	推荐激活函数	理由
二分类输出层	Sigmoid	输出概率解释性强
RNN/LSTM隐藏层	Tanh	零中心化缓解梯度偏移

5. 选择指南

场景	推荐激活函数	理由
二分类输出层	Sigmoid	输出概率解释性强
RNN/LSTM隐藏层	Tanh	零中心化缓解梯度偏移
CNN/全连接隐藏层	ReLU/Leaky ReLU	计算高效，缓解梯度消失
对抗性训练（GAN）	Leaky ReLU	避免判别器过早死亡
Transformer类模型	GELU	平滑随机性提升泛化能力
自监督学习（如BERT）	GELU	近似ReLU但更平滑
需要负区间响应	ELU	负值处理更鲁棒

6. 最新趋势

- **Transformer架构**：普遍采用GELU（如GPT、BERT）。
- **稀疏模型**：ReLU变体（如Swish $x \cdot \sigma(x)$）在MobileNet中表现优异。
- **量子化感知训练**：使用分段线性激活（如Hard-Sigmoid）便于部署。