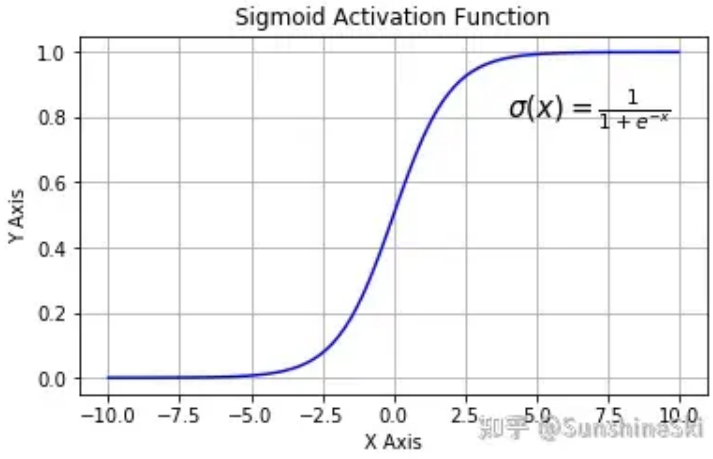
深度学习通用面试

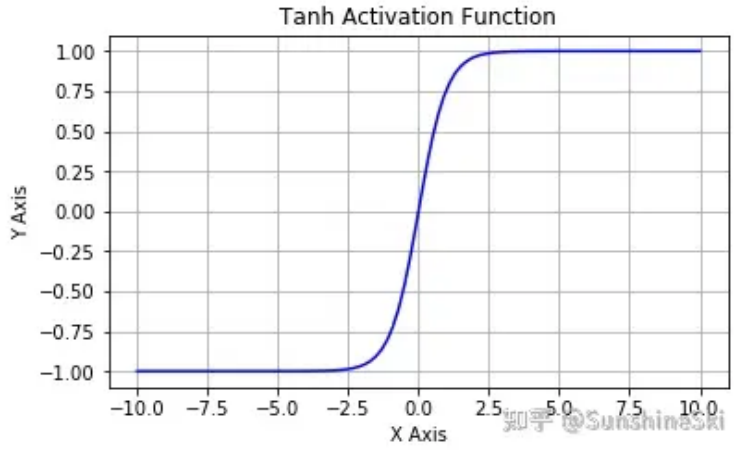
- 基础
  - 激活函数
    - 激活函数（Activation Function）是一种添加到人工神经网络中的函数，旨在帮助网络学习数据中的复杂模式。在神经元中，输入的input经过一系列加权求和后作用于另一个函数，这个函数就是这里的激活函数。
    - 为什么要有激活函数
      - 增加非线性
        - 神经网络中每一层的输入输出都是一个线性求和的过程，下一层的输出只是承接了上一层输入函数的线性变换，所以如果没有激活函数，那么无论你构造的神经网络多么复杂，有多少层，最后的输出都是输入的线性组合，纯粹的线性组合并不能够解决更为复杂的问题。而引入激活函数之后，我们会发现常见的激活函数都是非线性的，因此也会给神经元引入非线性元素，使得神经网络可以逼近其他的任何非线性函数，这样可以使得神经网络应用到更多非线性模型中。
    - 要求
      - 连续并可导（允许少数点上不可导），可导的激活函数可以直接利用数值优化的方法来学习网络参数；
      - 激活函数及其导数要尽可能简单一些，太复杂不利于提高网络计算率；
      - 激活函数的导函数值域要在一个合适的区间内，不能太大也不能太小，否则会影响训练的效率和稳定性。
    - 常见的激活函数
      https://zhuanlan.zhihu.com/p/3646205...
      - sigmoid
        - $$f(x) = \frac{1}{1+e^{-x}}$$



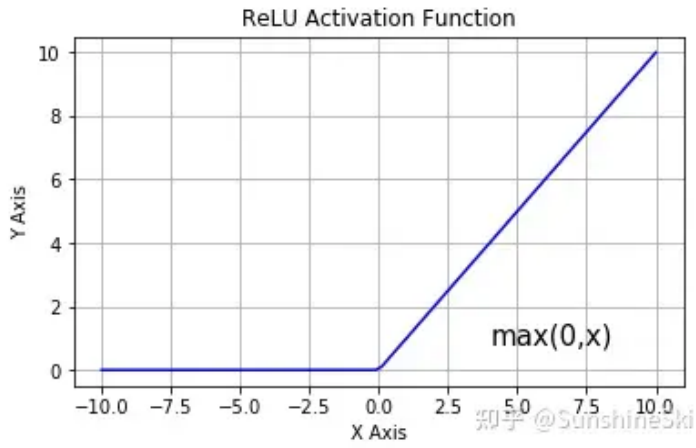
- 优点
  - Sigmoid 函数的输出范围是 0 到 1。由于输出值限定在 0 到1，因此它对每个神经元的输出进行了归一化；
  - 用于将预测概率作为输出的模型。由于概率的取值范围是 0 到 1，因此 Sigmoid 函数非常合适；
  - 梯度平滑，避免「跳跃」的输出值；
  - 函数是可微的。这意味着可以找到任意两个点的 sigmoid 曲线的斜率；
- 缺点
  - 梯度消失
    - Sigmoid 函数趋近 0 和 1 的时候变化率会变得平坦，也就是说，Sigmoid 的梯度趋近于 0。神经网络使用 Sigmoid 激活函数进行反向传播时，输出接近 0 或 1 的神经元其梯度趋近于 0。这些神经元叫作饱和神经元。因此，这些神经元的权重不会更新。此外，与此类神经元相连的神经元的权重也更新得很慢。该问题叫作梯度消失。
  - 不以零为中心
    - Sigmoid 输出不以零为中心的，，输出恒大于0，非零中心化的输出会使得其后一层的神经元的输入发生偏置偏移（Bias Shift），并进一步使得梯度下降的收敛速度变慢。
  - 计算成本高昂
    - exp() 函数与其他非线性激活函数相比，计算成本高昂，计算机运行起来速度较慢。

- Tanh/双曲正切激活函数
  - $$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{2}{1+e^{-2x}} - 1$$

- 可以看作放大平移的sigmoid



  - 优点
    - tanh的输出间隔为1，整个函数以0。
    - 输出的正负性不变
  - 缺点
    - 也有梯度小时的问题

- ReLU/修正线性单元
  - $$f(x) = \begin{cases} x & , x >= 0 \\ 0 & , x < 0 \end{cases} \\ = \max(0, x)$$
  - 

    - 优点
      - 当输入为正时，导数为1，一定程度上改善了梯度消失问题，加速梯度下降的收敛速度；
      - 计算速度快得多。ReLU 函数中只存在线性关系，因此它的计算速度比 sigmoid 和 tanh 更快。

- 被认为具有生物学合理性（Biological Plausibility）,比如单侧抑制、宽兴奋边界（即兴奋程度可以非常高）

- 缺点

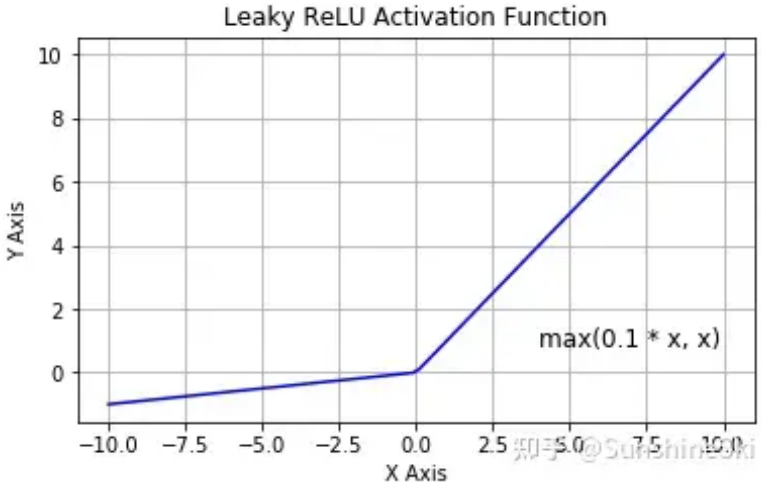
  - Dead ReLU 问题。当输入为负时，ReLU 完全失效，在正向传播过程中，这不是问题。有些区域很敏感，有些则不敏感。但是在反向传播过程中，如果输入负数，则梯度将完全为零；

    - 【Dead ReLU问题】ReLU神经元在训练时比较容易“死亡”。在训练时，如果参数在一次不恰当的更新后，第一个隐藏层中的某个 ReLU 神经元在所有的训练数据上都不能被激活，那么这个神经元自身参数的梯度永远都会是0，在以后的训练过程中永远不能被激活。这种现象称为死亡ReLU问题，并且也有可能会发生在其他隐藏层。

  - 不以零为中心：和 Sigmoid 激活函数类似，ReLU 函数的输出不以零为中心，ReLU 函数的输出为 0 或正数,给后一层的神经网络引入偏置偏移，会影响梯度下降的效率。

- Leaky ReLU

  - $$\begin{aligned}\operatorname{LeakyReLU}(x) &= \begin{cases} x & \text{if } x > 0 \\ \gamma x & \text{if } x \leq 0 \end{cases} \\ &= \max(0, x) + \gamma \min(0, x),\end{aligned}$$

    - 

  - 优点

    - Leaky ReLU 通过把 x 的非常小的线性分量给予负输入（0.01x）来调整负值的零梯度（zero gradients）问题，当 x < 0 时，它得到 0.1 的正梯度。该函数一定程度上缓解了 dead ReLU 问题，
    - leak 有助于扩大 ReLU 函数的范围，通常 a 的值为 0.01 左右；
    - Leaky ReLU 的函数范围是（负无穷到正无穷）

  - 缺点

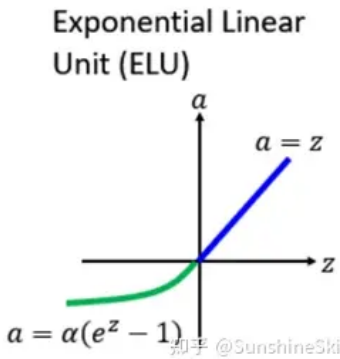
- 在实际使用中效果并不是总是比ReLU好

- Parametric ReLU
  - 将Leaky ReLU的参数从一个常数项变成了一个可学习参数，且允许每一个神经元有不同的参数。

$$\begin{aligned}\mathrm{PReLU}_i(x) &= \begin{cases} x & \text{if } x > 0 \\ \gamma_i x & \text{if } x \leq 0 \end{cases} \\ &= \max(0, x) + \gamma_i \min(0, x),\end{aligned}$$

- ELU
  -

$$g(x) = \mathrm{ELU}(x) = \begin{cases} x, x > 0 \\ \alpha\left(\mathrm{e}^{x} - 1\right), x \leqslant 0 \end{cases}$$

    - 

      - 优点
        - 没有 Dead ReLU 问题，输出的平均值接近 0，以 0 为中心；在0处可导。
        - ELU函数的输出在负数区域趋于-α，这可以帮助网络更容易适应不同的数据分布，减少训练过程中的偏差。
        - ELU 在较小的输入下会饱和至负值，从而减少前向传播的变异和信息。
      - 缺点
        - 计算量更大
        - 在实际使用中效果并不是总是比ReLU好

- softmax

-

$$S_i = \frac{e^i}{\sum_j e^j}$$

    - 优点
        - 用于多分类问题，将输入转化为概率分布，适合输出层。
        - 对于多类别分类问题的概率建模很有用。
    - 缺点
        - 不适合用于隐藏层的激活函数，因为它不具备非线性性质。
        - 在零点不可微
- 问题
    - 为什么ReLU更好
        - 采用sigmoid等函数，算激活函数时（指数运算），计算量大，反向传播求误差梯度时，求导涉及除法和指数运算，计算量相对大，而采用Relu激活函数，整个过程的计算量节省很多。
        - 对于深层网络，sigmoid函数反向传播时，很容易就会出现梯度消失的情况（在sigmoid接近饱和区时，变换太缓慢，导数趋于0，这种情况会造成信息丢失），这种现象称为饱和，从而无法完成深层网络的训练。而ReLU就不会有饱和倾向，不会有特别小的梯度出现。
        - Relu会使一部分神经元的输出为0，这样就造成了网络的稀疏性，并且减少了参数的相互依存关系，缓解了过拟合问题的发生。
- 正向传播
    - 神经网络从输入到输出一层层计算并保存中间变量的过程。
- 反向传播（Back-propagation）
    - 反向传播(back-propagation)指的是计算神经网络参数梯度的方法。总的来说，反向传播依据微积分中的链式法则，沿着从输出层到输入层的顺序，依次计算并存储目标函数有关神经网络各层的中间变量以及参数的梯度。

1. **前向传播**：首先，输入数据在神经网络中向前传播，经过每一层的处理（包括权重和偏置的线性组合和激活函数），直到输出层产生输出。
2. **计算损失**：在输出层，算法计算预测值与真实值之间的差异，这通常通过一个损失函数（如均方误差或交叉熵）来实现。
3. **反向传播误差**：接着，计算损失函数关于网络参数（权重和偏置）的梯度。这一步骤从输出层开始，逆向通过网络的每一层进行。这里的关键是链式法则，它用于有效地计算复合函数的导数。
   例如，对于某个权重 w，损失函数的梯度可以表示为 $\frac{\partial L}{\partial w}$。如果损失函数是 L，某层的输出是 y，而 y 是权重 w 的函数，那么根据链式法则，我们有：

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

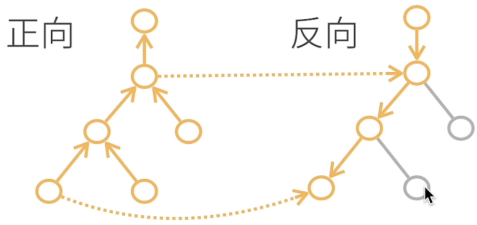
4. **更新参数**：最后，利用这些梯度通过梯度下降或其他优化算法来更新网络的权重和偏置。更新的方式通常是从当前值中减去梯度与学习率的乘积。

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot \frac{\partial L}{\partial w}$$

   其中，η 是学习率，一个决定更新步长大小的超参数。

反向传播是深度学习中最核心的概念之一，使得训练深层神经网络变得可行和有效。通过迭代地执行这个过程，网络可以逐渐学习到将输入映射到正确输出的方法。

- 构造计算图
- 前向：执行图，存储中间结果
- 反向：从相反方向执行图
  - 去除不需要的枝



- 为什么pytorch默认累加梯度
  - 如果累积多个batch算不出来，可以分批计算后累加
  - 多个模型share权重时，这样有好处
  - 可能会有多个损失函数
- 优化器
  - 随机梯度下降（SGD）

https://zhuanlan.zhihu.com/p/3579638...

- 随机梯度下降（Stochastic Gradient Descent，简称SGD）是一种用于优化机器学习算法的方法，特别是在大规模数据集上训练神经网络时非常有效。它是梯度下降算法的一种变体。在标准的梯度下降中，我们计算整个数据集的梯度，这在大数据集上可能非常耗时。而在SGD中，我们不是使用整个数据集，而是在每一步中随机选择一个样本（或一小批样本），来估计梯度并更新模型参数。

  - batch_size=1，就是SGD。
  - batch_size=n，就是mini-batch
  - batch_size=m，就是batch

  其中1<n<m，m表示整个训练集大小。

  - 优点
    - 效率高: 对于大型数据集和高维空间，SGD的效率比标准梯度下降要高得多，因为它每次更新只需要计算单个数据点的梯度。
    - 在线学习: SGD可以用于在线学习。在在线学习场景中，数据是顺序到来的，SGD可以在每次接收到新数据时立即更新模型，而不需要重新查看整个数据集。
    - 逃离局部最小值: 由于其更新的随机性，SGD在某些情况下能够从局部最小值中跳出，找到全局最小值或更好的局部最小值。
  - 缺点
    - 大部分时候SGD向着全局最小值靠近，有时候你会远离最小值，因为那个样本恰好给你指的方向不对，因此随机梯度下降法是有很多噪声的，平均来看，它最终会靠近最小值，不过有时候也会方向错误，因为随机梯度下降法永远不会收敛，而是会一直在最小值附近波动。一次性只处理了一个训练样本，这样效率过于低下。

- batch size对训练的影响

  https://zhuanlan.zhihu.com/p/4153328...

  - 在batch size小于显存的情况下，batch size越大则能提高内存利用率和并行率，跑完一次数据集的速度越快。
  - batch size太小时，误差会很大，训练比较糟糕。
  - batch size较小时，收敛速度更快。会引入更多噪声，提升泛化能力。（更容易找到距离较远的最优点）
  - batch size越大，梯度下降的方向就越稳定，但是损失下降地更慢，且更容易陷入局部最优。（因为单个样本梯度下降的方向可能是指向很不一样，平均之后根据三角不等式，下降距离就变小了，所以要多个样本的下降方向一样，大size的下降速度才能和小size一样。）

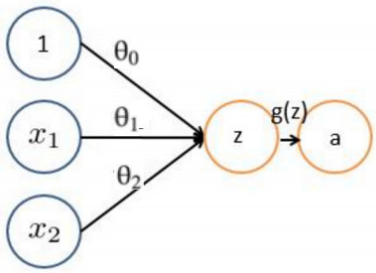
- softmax

- Softmax函数是在机器学习中，特别是在多分类的神经网络模型和注意力矩阵的归一化中常用的一个函数。它通常用于神经网络的最后一层，将实数输出转换为概率分布。

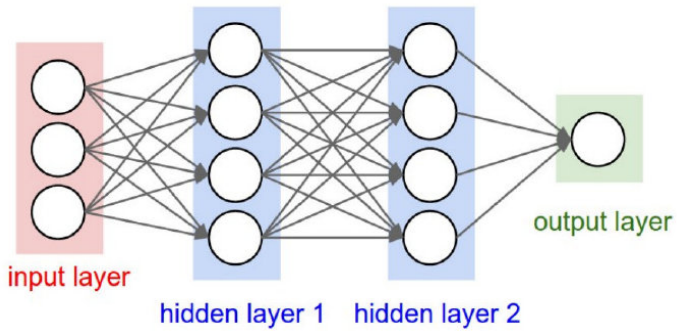
$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

  - 概率解释：将softmax用作输出层的激活函数，可以将输出解释为概率分布。每个输出值表示输入属于特定类别的概率。
  - 处理多分类问题：在多分类问题中，softmax能够处理多于两个类别的情况，使得模型能够区分多个不同的类别。
  - 数值稳定性：通过归一化，softmax可以处理较大或较小的输入值，减少数值计算中的不稳定性。
- 损失函数
  - 交叉熵

- 全连接层
  - 单层感知机
    - 由一个线性层和一个sigmoid组成



  - 隐藏层
    - 全连接层（Fully Connected Layer）：全连接层的每个神经元（节点）都与上一层的每个神经元相连接，每个连接都有一个权重，这意味着每个神经元都接收到上一层所有神经元的输出，并对这些输入进行加权求和，后再通过一个激活函数（sigmoid, ReLU)。



    - 多隐藏层的神经网络比 单隐藏层的神经网络工程效果好很多。

- 提升隐层层数或者隐层神经元个数，神经网络“容量”会变大，空间表达力会变强。
- 过多的隐层和神经元节点，会带来过拟合问题。
- 不要试图通过降低神经网络参数量来减缓过拟合，用正则化或者dropout。

- 向量化计算
  -

$$\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix} = \overbrace{\begin{bmatrix} \dots W_1^{[1]T} \dots \\ \dots W_2^{[1]T} \dots \\ \dots W_3^{[1]T} \dots \\ \dots W_4^{[1]T} \dots \end{bmatrix}}^{W^{[1]}} * \overbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}^{input} + \overbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}}^{b^{[1]}}$$