激活函数

ReLU 激活函数是你可以使用的最简单非线性激活函数。当输入是正数时，导数是 1，所以没有 S 型函数的反向传播错误导致的消失效果。研究表明，对于大型神经网络来说，ReLU 的训练速度要快很多。TensorFlow 和 TFLearn 等大部分框架使你能够轻松地在隐藏层使用 ReLU，你不需要自己去实现这些 ReLU。

不足
有时候一个非常大的梯度流过一个 ReLU 神经元，更新过参数之后，会使ReLU神经元始终为 0（这个神经元再也不会对任何数据有激活现象了）。这些"无效"的神经元将始终为 0，很多计算在训练中被浪费了。

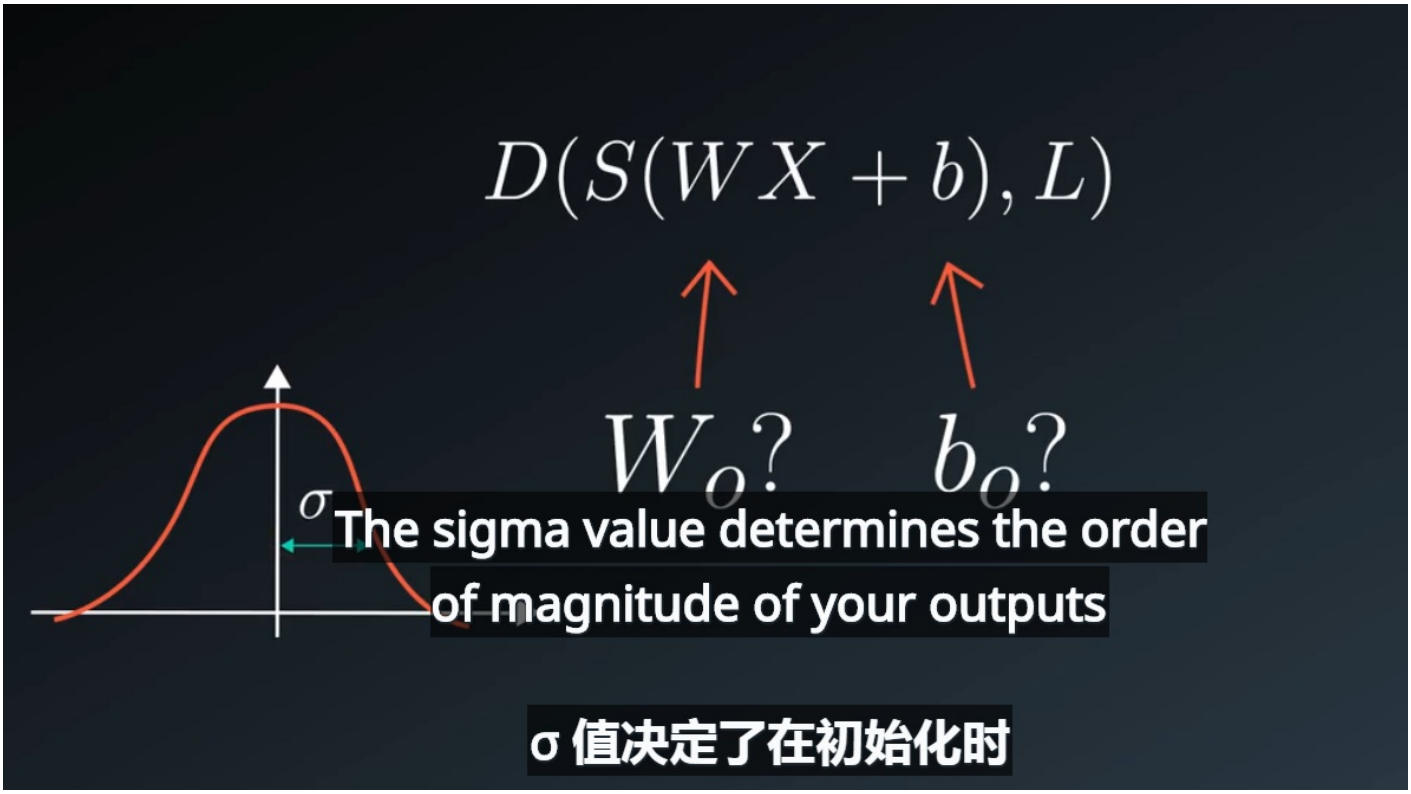
摘自 Andrej Karpathy 的 CS231n 课程:

遗憾的是，**ReLU 单元在训练期间可能会很脆弱并且会变得"无效"**。例如，流经 ReLU 神经元的大型梯度可能会导致权重按以下方式更新：神经元将再也不会在任何数据点上激活。如果发生这种情况，那么流经该单元的梯度将自此始终为零。也就是说，ReLU 单元会在训练期间变得无效并且不可逆转，因为它们可能会不再位于数据流形上。例如，学习速度（learning rate）设置的太高，你的网络可能有高达 40% 的神经元处于"无效"状态（即神经元在整个训练数据集上从未激活）。如果能正确地设置学习速度，那么该问题就不太容易出现。

权重初始化

对于权重的初始化的问题,通常的思路是按照高斯分布来决定其初始值，但是其权重初始方差大小也需要考虑：

- 首先方差决定了初始化训练中概率分布的陡峭程度
- 方差越大概率分布越集中于峰值区域，表示确定性很强,方差小代表高斯分布平坦，集中率和确定性较弱。
- 通常选择确定性较弱的分布开始初始化权重，然后让梯度训练算法来不断迭代增强其确定性
- 所以通常采用小方差的高斯分布，即 **输入节点的反比** 来确定其方差大小



关于验证集和测试集

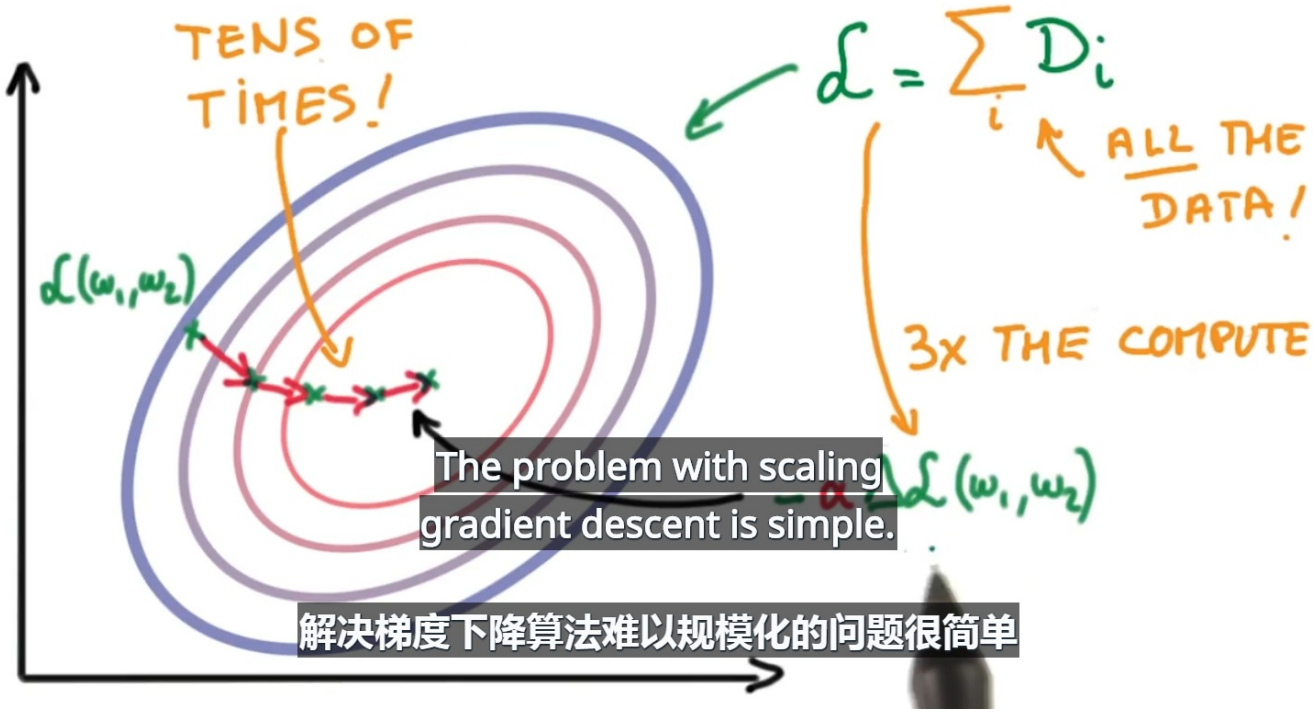
- **验证集**：用来调整分类器超参数的样本集，如在神经网络中选择隐藏层神经元的数量，**相对独立于训练** ,虽然验证集是独立于训练过程的的，但是因为人为根据验证集的表现调整了超参数，所以本质上训练数据还是拟合了验证集，即通过人的行为验证集和训练过程产生了交互，所以需要一个完全独立的测试集。
- **测试集**：仅用于对已经训练好的分类器进行性能评估的样本集，**完全独立**

> 在对机器学习算法进行学习和实践的时候，我们经常会遇到"验证集"和"测试集"，通常的机器学习书籍都会告诉我们，验证集和测试集不相交，验证集和训练集不相交，测试集和训练集不相交。也就是验证集与测试集似乎是同一级的东西，那么我们自然而然会有一个困惑为什么还要分测试集和验证集呢？ 其实问题的答案是：训练集用于训练模型参数，测试集用于估计模型对样本的泛化误差，验证集用于"训练"模型的超参数。

什么是模型的参数 我们知道一个机器学习模型通常包括两个部分的参数：模型参数和超参数。其中超参数是用于控制模型行为的参数，这些参数不是通过模型本身学习而来的。例如多项式回归模型里面，多项式的次数，学习速率是超参数。这些超参数不能由模型本身训练得到是因为模型会倾向于把超参数训练的过大或者过小，从而极易导致过拟合。例如多项式回归模型里面，如果让模型本身去训练多项式的次数，那么模型会选择高次多项式，因为这样做误差可以取到特别小，极端情况下，N个点的多项式回归会选择次数N>

随机梯度下降

- 梯度下降算法会导致一个规模太大的问题，导致计算量太大，所以采用随机梯度下降，虽然随机梯度下降是一个非常差的优化器，因为他只去了小批次的样本做梯度下降，但是通过多次小批次的迭代，他快速的到达了近似的全局最优。



.

- 有很多超参数可以调节，初始学习率、学习率衰减、动量、batch_size等等
- ADAGRAD方法类似于SGD，但是可以自动选择超参数

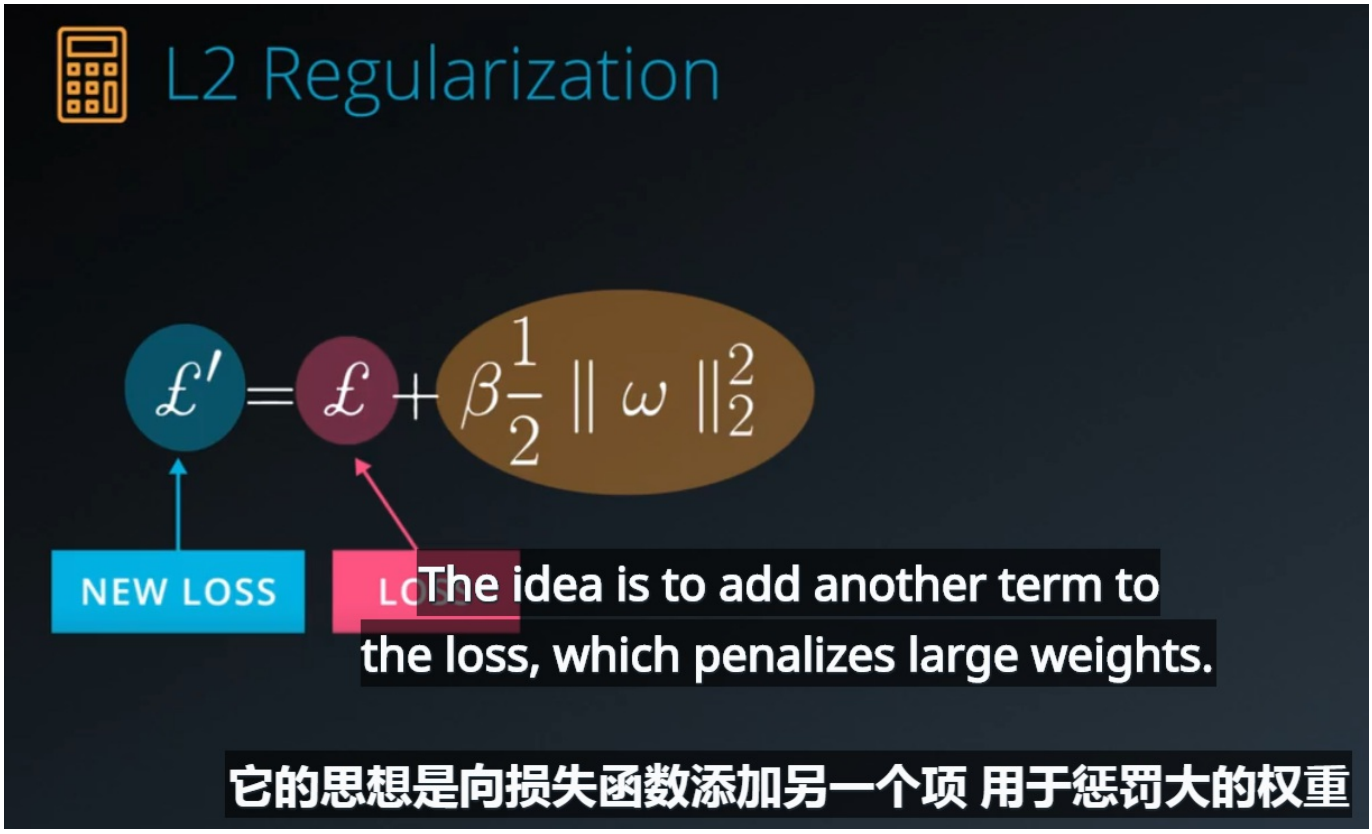
。Mini-batching 是一个一次训练数据集的一小部分，而不是整个训练集的技术。它可以使内存较小、不能同时训练整个数据集的电脑也可以训练模型。

Mini-batching 从运算角度来说是低效的，因为你不能在所有样本中计算 loss。但是这点小代价也比根本不能运行模型要划算。

它跟随机梯度下降（SGD）结合在一起用也很有帮助。方法是在每一代训练之前，对数据进行随机混洗，然后创建 mini-batches，对每一个 mini-batch，用梯度下降训练网络权重。因为这些 batches 是随机的，你其实是在对每个 batch 做随机梯度下降（SGD）。

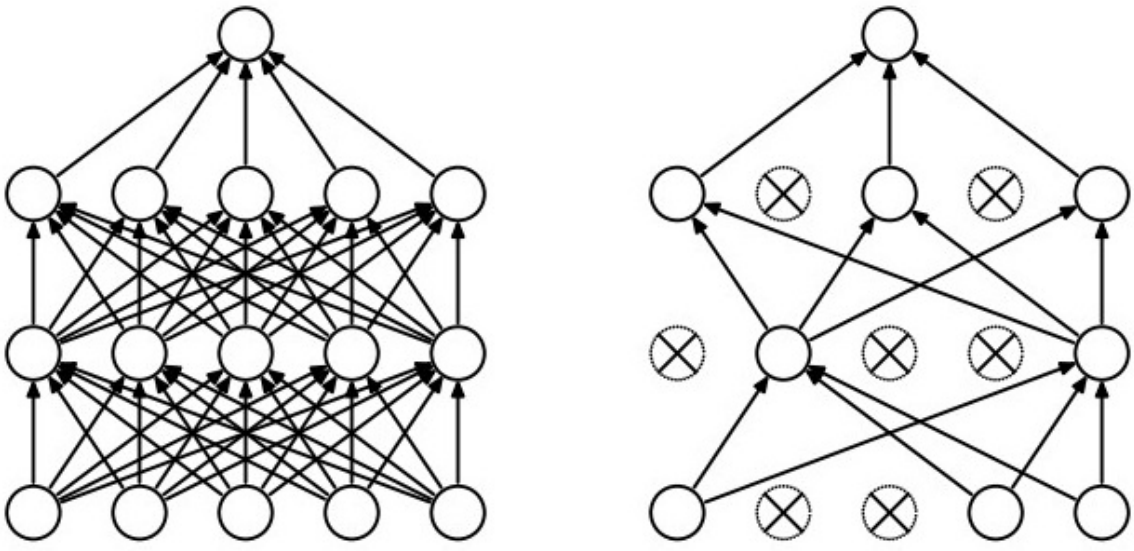
关于正则化

- 为了防止过拟合的现象，即模型特异性太强，只针对被训练的数据才有效，类似于紧身裤，不适合其他数据，泛化能力不够，紧身裤很难被其他数据穿上。
- 正则化则指人为的约束神经网络，隐式地减少自由参数的数量，同时也不会让其更难以优化，即将紧身裤变成弹力裤。即L2正则化，向损失函数添加了一个惩罚项，用于惩罚过大的权重。即某节点的权重越大代表其越重要，可能会产生特异性，正则化通过惩罚这种分布规律，让所有权重分布更加均匀，一定程度上增强了模型的泛化能力。
- L2正则化，即权重的L2范数（即Wi的平方和）来惩罚较大权重项。



关于dropout

Dropout 是一个降低过拟合的正则化技术。它在网络中暂时的丢弃一些单元（神经元），以及与它们的前后相连的所有节点。图 1 是 dropout 的工作示意图。



TensorFlow 提供了一个 tf.nn.dropout() 函数，你可以用来实现 dropout。 让我们来看一个 tf.nn.dropout()的使用例子。

```
keep_prob = tf.placeholder(tf.float32) # probability to keep units

hidden_layer = tf.add(tf.matmul(features, weights[0]), biases[0])
hidden_layer = tf.nn.relu(hidden_layer)
hidden_layer = tf.nn.dropout(hidden_layer, keep_prob)

logits = tf.add(tf.matmul(hidden_layer, weights[1]), biases[1])
```