深度学习训练的常用技巧

0 网络模型验证与评价

训练误差和泛化误差

评价一个网络模型的好坏可以利用之前讲到的损失函数，损失值越小，则模型越好。不过，这里需要注意区分训练误差（training error）和泛化误差（generalization error），通俗来讲，前者指模型在训练数据集上表现出的误差，后者指模型在任意一个测试数据样本上表现出的误差的期望，并常常通过测试数据集上的误差来近似。

以高考为例来直观地解释训练误差和泛化误差这两个概念。训练误差可以认为是做往年高考试题（训练题）时的错误率，泛化误差则可以通过真正参加高考（测试题）时的答题错误率来近似。假设训练题和测试题都随机采样于一个未知的依照相同考纲的巨大试题库。如果让一名未学习中学知识的小学生去答题，那么测试题和训练题的答题错误率可能很相近。但如果换成一名反复练习训练题的高三备考生答题，即使在训练题上做到了错误率为 0，也不代表真实的高考成绩会如此。

在机器学习里，我们通常假设训练数据集（训练题）和测试数据集（测试题）里的每一个样本都是从同一个概率分布中相互独立地生成的。基于该独立同分布假设，给定任意一个机器学习模型（含参数），它的训练误差的期望和泛化误差都是一样的。例如，如果我们将模型参数设成随机值（小学生），那么训练误差和泛化误差会非常相近。但我们从前面几节中已经了解到，模型的参数是通过在训练数据集上训练模型而学习出的，参数的选择依据了最小化训练误差（高三备考生）。所以，训练误差的期望小于或等于泛化误差。也就是说，一般情况下，由训练数据集学到的模型参数会使模型在训练数据集上的表现优于或等于在测试数据集上的表现。由于无法从训练误差估计泛化误差，一味地降低训练误差并不意味着泛化误差一定会降低。

机器学习模型应关注降低泛化误差。

模型选择

在机器学习中，通常需要评估若干候选模型的表现并从中选择模型。这一过程称为模型选择（model selection）。可供选择的候选模型可以是有着不同超参数的同类模型。以多层感知机为例，我们可以选择隐藏层的个数，以及每个隐藏层中隐藏单元个数和激活函数。为了得到有效的模型，我们通常要在模型选择上下一番功夫。下面，我们来描述模型选择中经常使用的验证数据集（validation data set）。

验证数据集

从严格意义上讲，测试集只能在所有超参数和模型参数选定后使用一次。不

可以使用测试数据选择模型，如调参。由于无法从训练误差估计泛化误差，因此也不应只依赖训练数据选择模型。鉴于此，我们可以预留一部分在训练数据集和测试数据集以外的数据来进行模型选择。这部分数据被称为验证数据集，简称验证集（validation set）。例如，我们可以从给定的训练集中随机选取一小部分作为验证集，而将剩余部分作为真正的训练集。

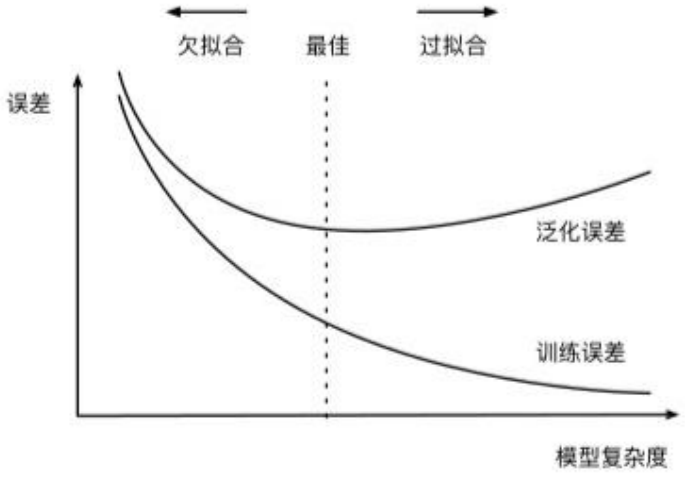
然而在实际应用中，由于数据不容易获取，测试数据极少只使用一次就丢弃。因此，实践中验证数据集和测试数据集的界限可能比较模糊。从严格意义上讲，除非明确说明，否则本书中实验所使用的测试集应为验证集，实验报告的测试结果（如测试准确率）应为验证结果（如验证准确率）。

K 折交叉验证

由于验证数据集不参与模型训练，当训练数据不够用时，预留大量的验证数据显得太奢侈。一种改善的方法是 K 折交叉验证（K-fold cross-validation）。在 K 折交叉验证中，我们把原始训练数据集分割成个不重合的子数据集，然后我们做 K 次模型训练和验证。每一次，我们使用一个子数据集验证模型，并使用其他个子数据集来训练模型。在这次训练和验证中，每次用来验证模型的子数据集都不同。最后，我们对这 K 次训练误差和验证误差分别求平均。

欠拟合和过拟合

接下来，我们将探究模型训练中经常出现的两类典型问题：一类是模型无法得到较低的训练误差，我们将这一现象称作欠拟合（underfitting）；另一类是模型的训练误差远小于它在测试数据集上的误差，我们称该现象为过拟合（overfitting）。在实践中，我们要尽可能同时应对欠拟合和过拟合。虽然有很多因素可能导致这两种拟合问题，在这里我们重点讨论两个因素：模型复杂度和训练数据.



如上图，给定训练数据集，如果模型的复杂度过低，很容易出现欠拟合；如果模型复杂度过高，很容易出现过拟合。应对欠拟合和过拟合的一个办法是针对数据集选择合适复杂度的模型。

影响欠拟合和过拟合的另一个重要因素是训练数据集的大小。一般来说，如果训练数据集中样本数过少，特别是比模型参数数量（按元素计）更少时，过拟

合更容易发生。此外，泛化误差不会随训练数据集里样本数量增加而增大。因此，在计算资源允许的范围之内，我们通常希望训练数据集大一些，特别是在模型复杂度较高时，例如层数较多的深度学习模型。

1.1 参数初始化

前面我们学习了前向传播和反向传播，知道在网络训练的时候需要先给定初始的参数，得出预测值，然后计算与真实值的误差，再通过反向传播计算误差对参数的导数，利用不同的优化算法更新参数。

参数初始化是深度学习网络训练的关键步骤，可根据网络结构，激活函数和具体的任务灵活选择乃至组合不同的参数初始化方法。

如若初始化方法选择不当，可能会导致以下一些问题：

1）对称性问题：假如参数都初始化相同的值，那么在前向传播过程中，后面的神经元计算得到的输出是相同的，也就失去了学习和表示不同特征的能力。以卷积神经网络为例，所有的卷积核参数值都一样，不同的卷积核提取的特征也都是一样的。

2）梯度不稳定：参数初始化不合适的话，可能会出现梯度消失或梯度爆炸的问题；

3）信号传播不稳定：参数初始化方法不合适，信号传播过程方差变化可能过大，特征信息会失真；

常用的参数初始化方法

在神经网络中，有几种常用的参数初始化方法，包括：

1）零初始化（Zero Initialization）：将所有的权重和偏置初始化为零。这种方法简单直接，但在实践中这种方法很少使用，因为它会导致所有的神经元具有相同的更新，并且会带来梯度消失问题等等。

2）随机初始化（Random Initialization）：将权重和偏置随机地初始化为较小的随机值。这可以打破对称性，并为神经元提供不同的起点，促进网络的多样性和学习能力。常见的随机初始化方法包括从均匀分布或高斯分布中随机采样。但是会带来训练不稳定和梯度消失或爆炸的问题。

3）Xavier 初始化方法，概率分布的进化版，核心思想就是分布的方差根据前一层和后一层的神经元数量来设置，以保持方差在不同层之间大致相等，避免在深层网络中产生梯度消失或梯度爆炸的问题。这种初始化方法有助于提供合适的梯度范围，促进网络的稳定训练和收敛。

具体而言，对于具有线性激活函数（如 sigmoid 和 tanh）的网络层，Xavier 初始化将权重初始化为均匀分布或高斯分布。

对于均匀分布的 Xavier 初始化（均匀版）：

从均匀分布中随机初始化权重矩阵 W，范围为[-a, a]，其中 a = sqrt(6 / (n + m))。

来看一个具有 5 层的神经网络的例子，以解释 Xavier 初始化是如何工作的。假设我们有一个具有以下结构的神经网络：输入层（100 个神经元） - 隐藏层 1

（80 个神经元） - 隐藏层 2（60 个神经元） - 输出层（10 个神经元）。现在，我们将使用 Xavier 初始化来初始化每一层的权重。

对于隐藏层 1，前一层是输入层，有 100 个神经元，后一层是隐藏层 1 本身，有 80 个神经元。根据公式，可以计算权重初始范围 a：a = sqrt(6 / (100 + 80)) ≈ 0.136。现在，可以从均匀分布[-0.136, 0.136]中随机初始化隐藏层 1 的权重矩阵。

接下来继续计算隐藏层 2 的权重初始范围。前一层是隐藏层 1，有 80 个神经元，后一层是隐藏层 2 本身，有 60 个神经元。使用相同的公式来计算权重初始范围 a：a = sqrt(6 / (80 + 60)) ≈ 0.153。然后，从均匀分布[-0.153, 0.153]中随机初始化隐藏层 2 的权重矩阵。

对于高斯分布的 Xavier 初始化（高斯版）：

从高斯分布中随机初始化权重矩阵 W，均值为 0，方差为 variance，其中 variance = 2 / (n + m)。

通过这样的递归计算和初始化过程，Xavier 初始化确保了每一层的权重都与前一层和后一层的神经元数量相关联。这有助于平衡信号和梯度的传播，避免梯度消失或梯度爆炸问题，从而提高神经网络的训练稳定性和收敛性能。

4）He 初始化方法针对使用 Rectified Linear Units (ReLU)激活函数的神经网络进行了优化，其核心思想是，使用了前一层神经元数量来计算权重的初始范围，以确保输入信号的方差与输出信号的方差保持一致。

具体而言，对于具有 n 个前一层神经元的全连接层，He 初始化使用以下公式来计算权重的初始范围：a = sqrt(2 / n)。在这个公式中，n 是前一层神经元的数量。

在使用 ReLU 激活函数时，输入为负时输出为 0，而输入为正时输出等于输入。这意味着在前向传播过程中，大约一半的神经元输出为 0。考虑一层具有 n 个输入神经元的全连接层，如果权重的方差为 1/n，那么通过该层的输出的方差为 1/n * n/2 = 1/2（每个神经元权重独立同分布，所以为 n/2）。因此，为了保持信号的方差在前向传播中的稳定性，我们设置权重的初始范围为 sqrt(2/n)，希望权重的初始范围能够使输出的方差保持在 1。这样，每一层的输出都具有合适的范围，有助于提供良好的梯度传播和训练稳定性。

参数初始化的使用位置

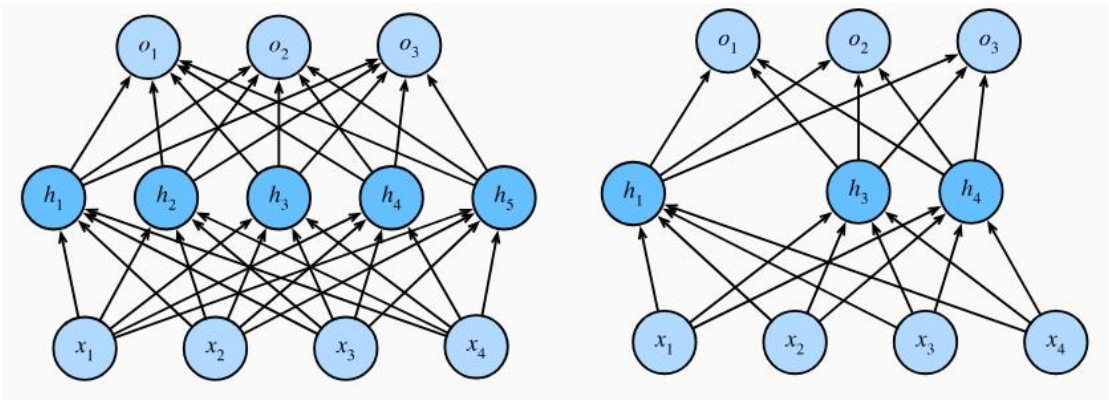
一般在网络框架初始化完成后，使用
if init_weights:
self._initialize_weights()
来初始化，可选择不同的初始化方法。

1.2 Dropout

丢弃法是为了应对过拟合问题。

全连接层的每一个节点都和上一层全部节点相连接，这样可以让每一个输出都综合考虑前面所有被提取到的特征，但是会使参数量增加产生大量计算，延缓网络的训练速度，同时也容易产生过拟合现象。

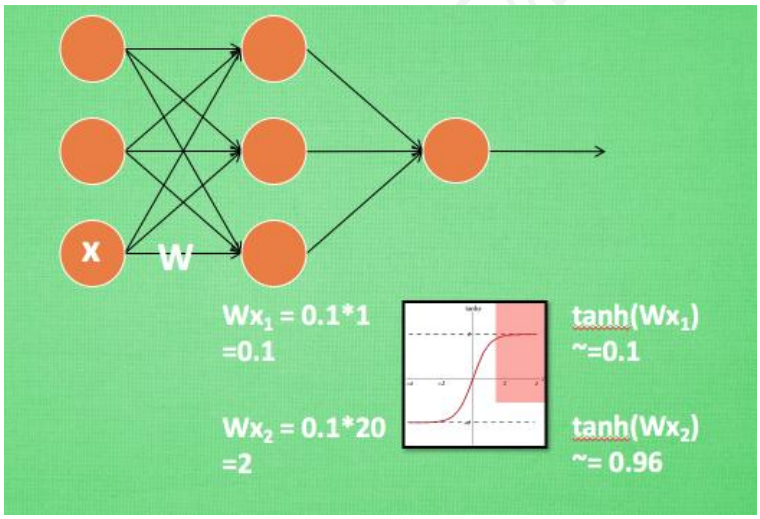
通常我们可以通过加入 Dropout 的方式缓解过拟合问题，如下图中的隐藏层，在每次前向传播的时候以一定的比例抑制部分神经元，之后在反向传播阶段只需要更新未被抑制的神经元参数即可。



1.3 Batch normalization

为什么做标准化

具有统一规格的数据，能让机器学习更容易学习到数据之中的规律。而在神经网络中的标准化的核心目的是为了让数据经过激活函数后不处于饱和阶段（不然相当于激活函数失效了，梯度也反向传播不过去）



在神经网络中，数据分布对训练会产生影响。比如某个神经元 x 的值为 1，某个 Weights 的初始值为 0.1，这样后一层神经元计算结果就是 Wx = 0.1；又或者 x = 20，这样 Wx 的结果就为 2。

现在还不能看出什么问题，但是,当我们使用像 tanh 的激励函数后，Wx 的激活值就变成了 ~0.1 和 ~1，接近于 1 的部已经处在了激励函数的饱和阶段，也就是如果 x 无论再怎么扩大，tanh 激励函数输出值也还是接近 1. 换句话说，神

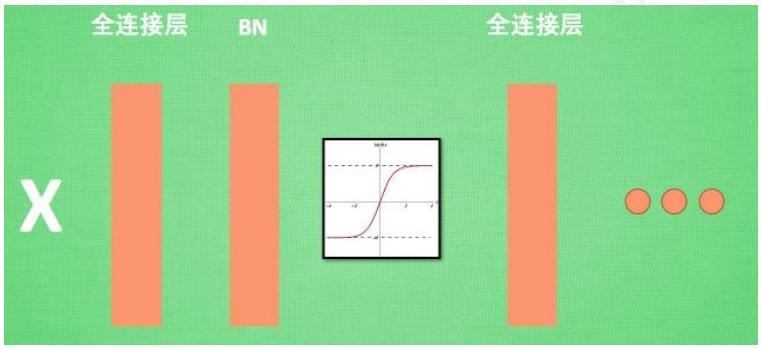
经网络在初始阶段已经不对那些比较大的 x 特征范围敏感了。 这样很糟糕，想象我轻轻拍自己的感觉和重重打自己的感觉居然没什么差别，这就证明我的感官系统失效了.

当然我们是可以用之前提到的对数据做 normalization 预处理，使得输入的 x 变化范围不会太大，让输入值经过激励函数的敏感部分。

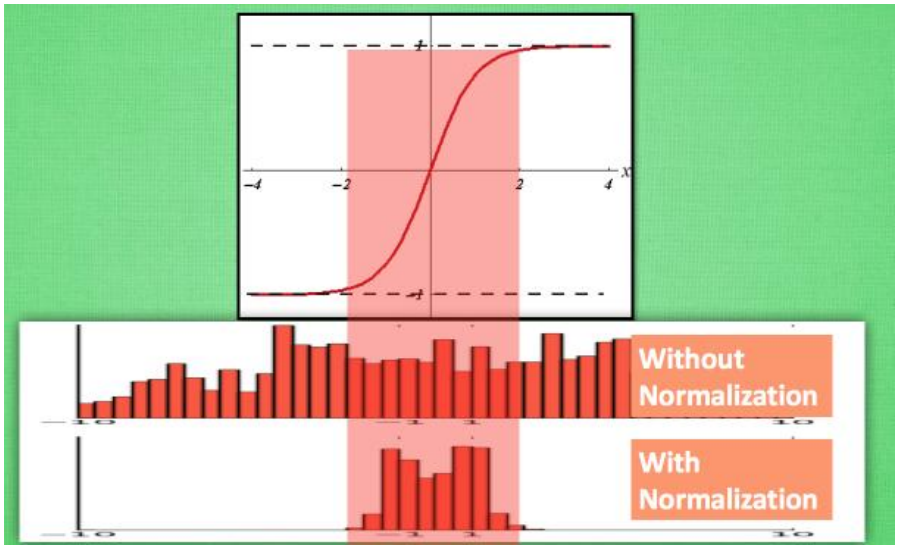
但刚刚这个不敏感问题不仅仅发生在神经网络的输入层，而且在隐藏层中也经常会发生。只是 x 换到了隐藏层当中，我们能不能对隐藏层的输入结果进行像之前那样的 normalization 处理呢？答案是可以的，batch normalization 正是处理这种情况.

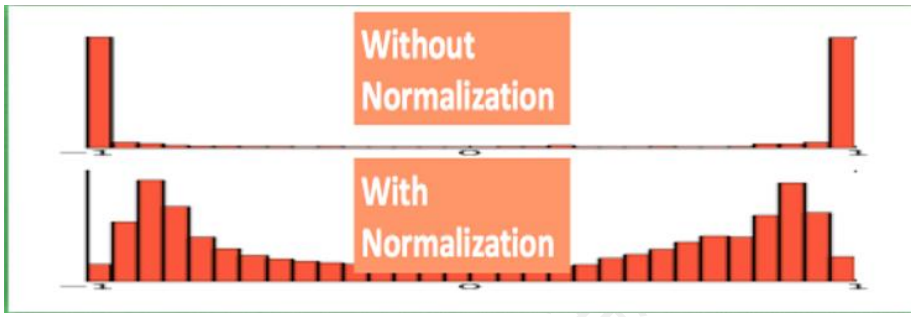
BN 添加位置

Batch Normalization，批标准化，和普通的数据标准化类似，是将分散的数据统一的一种做法，也是优化神经网络的一种方法。Batch normalization 也可以被看做一个层。在一层层的添加神经网络的时候，我们先有数据 X，再添加全连接层，全连接层的计算结果会经过激活函数成为下一层的输入，接着重复之前的操作. Batch Normalization (BN) 就被添加在每一个全连接和激活函数之间.



BN 效果

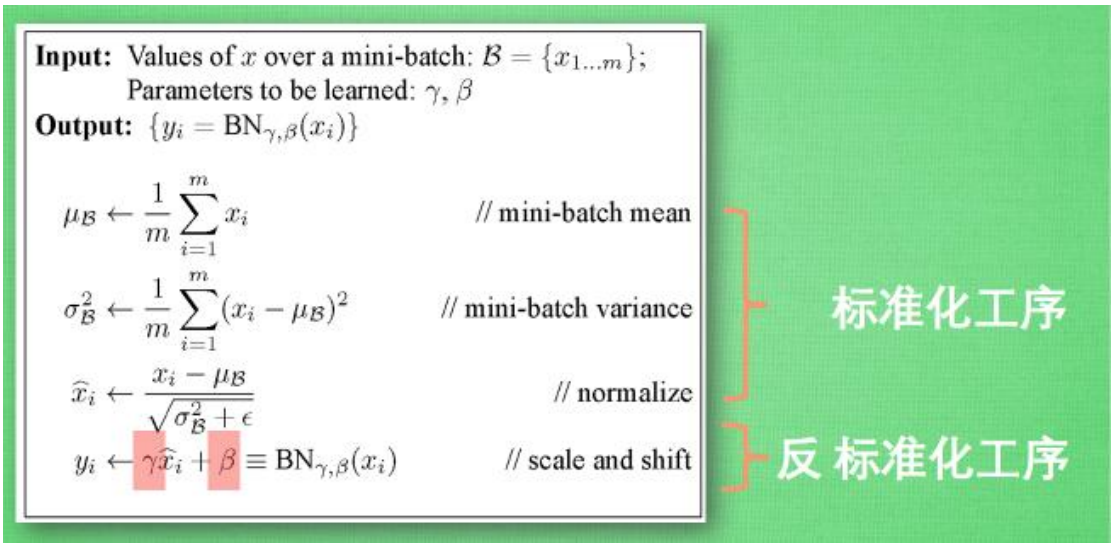




之前说过，计算结果在进入激活函数前的值很重要，如果我们不单单看一个值，我们可以说，计算结果值的分布对于激活函数很重要. 对于数据值大多分布在这个区间的数据，才能进行更有效的传递.

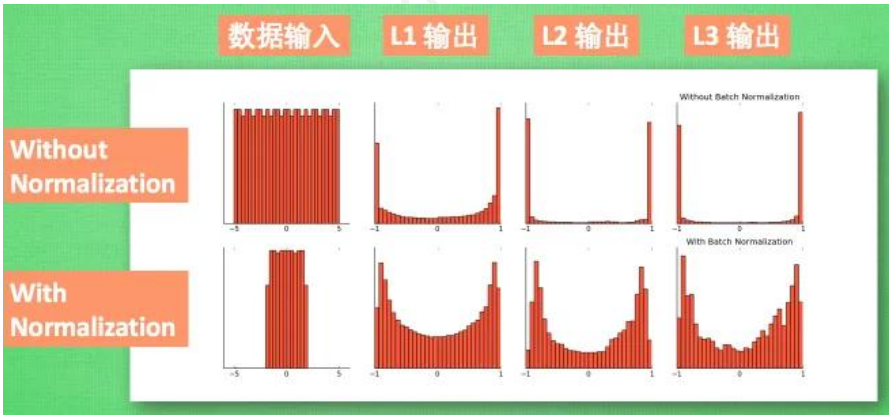
对比这两个在激活之前的值的分布，没有 normalize 的数据使用 tanh 激活以后，激活值大部分都分布到了饱和阶段，也就是大部分的激活值不是-1，就是1，而 normalize 以后，大部分的激活值在每个分布区间都还有存在. 再将这个激活后的分布传递到下一层神经网络进行后续计算，每个区间都有分布的这一种对于神经网络就会更加有价值。

BN 算法



我们引入一些 batch normalization 的公式. 这三步就是我们在刚刚一直说的 normalization 工序, 但是公式的后面还有一个反向操作, 将 normalize 后的数据再扩展和平移. 这是为了让神经网络自己去学着使用和修改这个扩展参数 gamma, 和平移参数 β, 这样神经网络就能自己慢慢琢磨出前面的 normalization 操作到底有没有起到优化的作用, 如果没有起到作用, 就使用 gamma 和 belt 来抵消一些 normalization 的操作.

最后我们来看看一张神经网络训练到最后, 代表了每层输出值的结果的分布图. 这样我们就能一眼看出 Batch normalization 的功效啦. 让每一层的值在有效的范围内传递下去.



代码实现

在 PyTorch 中，nn.BatchNorm2d(num_features=out_channels) 中的 num_features 参数设置为输出通道维度（out_channels），是由 Batch Normalization（批归一化）在卷积神经网络（CNN）中的**作用机制**决定的。

BatchNorm2d 需为卷积层输出的每个通道单独维护一组归一化参数，因此其通道数必须与卷积层的输出通道数完全一致。这一设计确保了批归一化能够正确作用于每个特征通道，稳定训练过程并加速收敛。

每个通道维护独立均值和方差的核心逻辑是：

1. **语义独立性**：不同通道对应不同特征检测器，分布特性差异显著，需单独处理；
2. **目标针对性**：批归一化旨在修正“单通道内”的分布偏移，需按通道计算统计量；
3. **维度匹配性**：与卷积输出的 (batch, channels, H, W) 结构对齐，确保操作可实现。

这种设计让批归一化既能有效稳定训练，又能保留不同通道的特征区分度，是其在 CNN 中发挥作用的关键。

具体原因如下：

一、BatchNorm2d 的作用对象：通道维度的统计特性

BatchNorm2d 用于对卷积层的输出进行归一化，其核心是**对每个通道的特征图单独计算均值和方差**，并进行标准化处理。对于卷积层的输出张量，其形状通常为 (batch_size, channels, height, width)，其中：

- batch_size：批次中的样本数；
- channels：特征图的通道数（即卷积层的输出通道数 out_channels）；
- height, width：每个通道的特征图尺寸。

BatchNorm2d 的操作逻辑是：

1. 对每个通道（共 out_channels 个），在**整个批次**和**特征图的空间维度（height×width）** 上计算均值和方差；
2. 使用这些统计量对该通道的所有元素进行归一化（减去均值，除以标准差）；
3. 通过可学习的缩放参数（gamma）和偏移参数（beta）调整归一化后的结果，保留特征表达能力。

二、为什么 num_features 必须等于输出通道数？

num_features 参数的作用是**告诉 BatchNorm2d 需要处理多少个通道**，每个通道对应一组独立的均值、方差、gamma 和 beta 参数。因此：

- 若卷积层的输出通道数为 out_channels，则该输出包含 out_channels 个特征图通道；
- BatchNorm2d 必须为每个通道维护一组统计参数（均值、方差）和可学习参数（gamma、beta），因此 num_features 必须等于 out_channels，才能与卷积层的输出通道一一对应。

例如：

- 若卷积层定义为 nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3)，其输出通道数为 64；

- 对应的 BatchNorm2d 必须设置为 `nn.BatchNorm2d(num_features=64)`，表示需要对 64 个通道分别进行归一化。

三、反推：若 num_features 不等于输出通道数会怎样？

如果 num_features 与卷积层的输出通道数不匹配（例如卷积输出 64 通道，但 num_features=32），PyTorch 会在运行时抛出维度不匹配的错误。因为 BatchNorm2d 会尝试按 num_features 个通道处理输入，而输入的实际通道数与之不符，导致无法正确计算每个通道的统计量。

在 Batch Normalization（批归一化）中，为每个通道维护独立的均值和方差，是由卷积神经网络（CNN）中**特征通道的语义独立性**和**批归一化的核心目标**共同决定的。具体原因可从以下三个维度理解：

一、特征通道的语义独立性：不同通道代表不同“特征检测器”

在 CNN 中，每个通道（channel）对应一个独立的 “特征检测器”，负责提取输入数据中特定类型的特征：

- 例如，在图像识别的第一层卷积中，不同通道可能分别检测“边缘”“纹理”“颜色块”等不同低级特征；
- 深层卷积的通道则可能检测“眼睛”“车轮”等更抽象的高级特征。

这些通道的特征分布（均值、方差）天然存在差异：

- 检测“边缘”的通道可能输出大量正负交替的激活值（均值接近 0，方差较大）；
- 检测“背景颜色”的通道可能输出集中在某个范围内的激活值（均值固定，方差较小）。

若对所有通道使用同一套均值和方差进行归一化，会强行 “拉平” 这种语义差异，破坏不同通道的特征表达能力。因此，必须为每个通道单独维护均值和方差，才能针对性地标准化每个 “特征检测器” 的输出。

二、批归一化的核心目标：稳定“单通道内”的分布偏移

批归一化的设计初衷是解决 **“内部协变量偏移”（Internal Covariate Shift）**——即网络深层的输入分布会因浅层参数更新而持续变化，导致训练不稳定。这种分布偏移是**以通道为单位发生的**：

- 每个通道的激活值会随着网络训练不断变化（例如，某通道的输出可能从“集中在 0 附近”逐渐变为“集中在 5 附近”）；
- 不同通道的分布偏移方向和幅度完全独立（如通道 A 的均值上升，通道 B 的均值下降）。

因此，批归一化需要**针对每个通道单独计算均值和方差**，才能准确捕捉并修正该通道的分布偏移。例如：

- 对通道 A，用其自身的均值（如 5）和方差（如 2）进行归一化；
- 对通道 B，用其自身的均值（如 -3）和方差（如 1）进行归一化。

只有这样，才能确保每个通道的激活值都被标准化到 “均值 0、方差 1”（或通过可学习参数调整的目标分布），真正实现稳定训练的目的。

三、数学操作的维度匹配：与卷积输出的张量结构对齐

从张量维度看，卷积层的输出形状为 (batch_size, channels, height, width)，其中：

- 批归一化的操作对象是**每个通道内的所有空间位置和所有样本**（即对每个通道，计算 batch_size × height × width 个元素的均值和方差）。

这种操作天然要求为每个通道维护独立的统计量：

- 若通道数为 C，则需要 C 个均值（每个通道 1 个）和 C 个方差（每个通道 1 个）；
- 这些统计量与输入张量的通道维度一一对应，才能在归一化时正确匹配（通道 i 的元素使用第 i 组均值和方差）。

例如，对于 (32, 64, 28, 28) 的输出（32 个样本，64 个通道，28×28 特征图），批归一化会计算 64 个均值（每个对应 1 个通道），并对 64 个通道分别执行归一化。

Batch Normalization VS Layer Normalization

batch normalization 是对一批样本的同一纬度特征做归一化。如下图我们想根据这个 batch 中的三种特征（身高、体重、年龄）数据进行预测性别，首先我们进行归一化处理，如果是 Batch normalization 操作则是对每一列特征进行归一化，如下图求一列身高的平均值。

学生编号	身高	体重	年龄
1	177	61	21
2	169	62	22
3	165	52	19
4	173	60	20
5	171	54	21
Batch平均	171	57.8	20.6

BN 特点：强行将数据转为均值为 0，方差为 1 的正态分布，使得数据分布一致，并且避免梯度消失。而梯度变大意味着学习收敛速度快，能够提高训练速度。

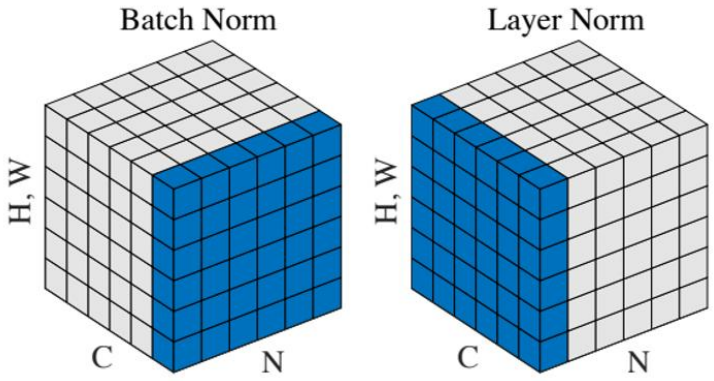
而 layer normalization 是对单个样本的所有维度特征做归一化。如下表中，如果是 Layer normalization 则是对每一行（该条数据）的所有特征数据求均值。

学生编号	身高	体重	年龄	Layer平均
1	177	61	21	86.33333333
2	169	62	22	84.33333333
3	165	52	19	78.66666667
4	173	60	20	84.33333333
5	171	54	21	82

Layer normalization 更多的是用到自然语言处理上，比如著名的 Transformer 架构。Transformer 为什么用 Layer Normalization 呢？

从操作上看：BN 是对同一个 batch 内的所有数据的同一个特征数据进行操作；而 LN 是对同一个样本进行操作。

从特征维度上看：BN 中，特征维度数=均值 or 方差的个数；LN 中，一个 batch 中有 batch_size 个均值和方差。



如在 NLP 中上图的 C、N、H,W 含义：

N：N 句话，即 batchsize；

C：一句话的长度，即 seqlen；

H,W：词向量维度 embedding dim。

BN 不适合 RNN、transformer 等序列网络，不适合文本长度不定和 batchsize 较小的情况，适合于 CV 中的 CNN 等网络；

而 LN 适合用于 NLP 中的 RNN、transformer 等网络，因为 sequence 的长度可能是不一致的。

栗子：如果把一批文本组成一个 batch，BN 就是对每句话的第一个词进行操作，BN 针对每个位置进行缩放就不符合 NLP 的规律了。而 LN 则是对一句话的所有词进行操作。

参考资料

https://zh.d2l.ai/index.html

https://zhuanlan.zhihu.com/p/24810318 BN