机器学习与深度学习面试系列十二（深度学习基础）

感知机模型是怎样的?

感知机是一种古老的**二分类**的线性分类模型，输入为实例的特征向量，输出为实例的类别（取+1和-1）。感知机对应于输入空间中将实例划分为两类的分离超平面。感知机旨在求出该超平面，为求得超平面导入了基于误分类的损失函数，利用梯**度下降法对损失函数进行最优化**。

在SVM那篇文章里，我们介绍过，样本点 x_i 到分离超平面 $w^Tx+b=0$ 的距离可以写作 $\frac{|w^Tx_i+b|}{||w||}$ ，所有的误分类样本都有 $-\frac{1}{||w||}y_i(w^Tx_i+b)>0$ 。感知机的目标是使误分类的样本数越少越好，所以最小化损失函数： $J=-\frac{1}{||w||}\sum_{i=1}^{N}y_i(w^Tx_i+b)$ 。忽略 $\frac{1}{||w||}$ (在SVM里说过，对于 w 和 b 同时放缩一定比例，不影响分离超平面的位置)，则损失函数写为 $J=-\sum_{i=1}^{N}y_i(w^Tx_i+b)$ 。利用梯度下降法，优化该损失函数：

$$w \leftarrow w-\lambda\nabla_w J=w-\lambda(-\sum_{i=1}^{N}y_ix_i)=w+\lambda\sum_{i=1}^{N}y_ix_i$$

$$b \leftarrow b-\lambda\nabla_b J=b-\lambda(-\sum_{i=1}^{N}y_i)=b+\lambda\sum_{i=1}^{N}y_i$$

其中 λ 为学习率，或者叫步长。通常利用随机梯度下降法来优化损失函数，即每次只选取一个样本点进行更新：

$$w \leftarrow w+\lambda y_ix_i$$

$$b \leftarrow b+\lambda y_i$$

感知机模型非常简单，只能处理线性可分的数据，而现实中大部分问题都不是线性可分的。感知机提出后，被嘲讽解决不了最简单的异或问题，导致神经网络几乎销声匿迹，人工智能进行了第一个低谷期。

神经网络与感知机关系?

我们将感知机看作单个的人工神经元结构，它的初衷是模拟一个生物神经细胞：

神经元结构

一个生物神经细胞的功能比较简单，而人工神经元只是生物神经细胞的理想化和简单实现，功能更加简单. 要想模拟人脑的能力，单一的神经元是远远不够的，需要通过很多神经元一起协作来完成复杂的功能.这样通过一定的连接方式或信息传递方式进行协作的神经元可以看作是一个网络，就是神经网络，又称作**多层感知机**。

神经网络最早是作为一种主要的连接主义模型. 20 世纪 80 年代中后期，最流行的一种连接主义模型是分布式并行处理(Parallel Distributed Processing， PDP)模型，其有 3 个主要特性:

1. 信息表示是分布式 的(非局部的);
2. 记忆和知识是存储在单元之间的连接上;
3. 通过逐渐改变单元之间的连接强度来学习新的知识.

神经网络和深度学习?

从传统机器学习来看，大部分有监督机器学习的本质就是去拟合从输入到输出的一个潜在的映射函数，这个函数可能很复杂，我们通过一些相对简单的模型通过“学习”（调整参数）去更好的拟合这个潜在的函数。神经网络也是一样，而且神经网络的成功就在于其极为强大的拟合能力。

万能逼近定理（Universal Approximation Theorem）告诉我们，仅具备单个隐藏层的神经网络，只要有足够的隐藏节点数量，它总能无限逼近任意复杂的函数。从这个定理出发，看起来我们不需要多层隐藏节点，单隐藏层的神经网络就已经满足学习能力了。

但实际上，如果仅采用单个隐藏层的神经网络，要想获得很高的拟合精度，隐藏节点的数量可能是呈指数上升的，对这么多参数的优化是灾难性的，并且它非常容易过拟合，所以我们选择使用更深的神经网络（更多的隐藏层），这就是深度学习。

深度学习和表示学习?

深度学习其实包含了先验信念：真实世界的数据结构通常认为是由（相对）简单的低维潜在过程产生的，即数据变化可以通过简单的潜在因素的层次结构来解释。要学习到一种好的高层语义表示，通常需要从底层特征开始，经过多步非线性转换才能得到（多个线性转换可以合并成一个线性转换）。 深度学习优点是可以增加特征的重用性，从而指数级地增加表示能力。

在传统的机器学习中，也有很多有关特征学习的方法，比如主成分分析、Fisher线性判别分析等。但是，传统的特征学习一般是通过人为地设计一些准则，然后根据这些准则来选取有效的特征。特征的学习是和最终预测模型的学习分开进行的，因此学习到的特征不一定可以提升最终模型的性能。而深度学习模型让模型自动学习出好的特征表示(从底层特征，到中层特征，再到高层特征)，从而最终提升预测模型的准确率。所谓“深度”是指原始数据进行非线性特征转换的次数（也就是神经网络的隐藏层数）。大部分深度学习模型可以看作是一种端到端的学习，在学习过程中不进行分模块或分阶段训练，直接优化任务的总体目标，中间过程不需要人为干预。

常用的激活函数有哪些?

- Sigmoid激活函数。 $f(z)=\frac{1}{1+e^{-z}}$ ，其导函数为： $f'(z)=f(z)(1-f(z))$
- Tanh激活函数。 $f(z)=tanh(z)=\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$ ，其导函数为： $f'(z)=1-(f(z))^2$
- ReLU激活函数。 $f(z)=max(0,z)$ ，其导函数 $f'(z)=\begin{cases}1, & \text{if z} > 0\\ 0, & \text{if z} \le 0\end{cases}$

Sigmoid激活函数将输入z映射到区间(0，1)，当z很大时，f(z)趋近于1，当z很小时，f(z)趋近于0。其导数在z很大或很小时都会趋近于0，造成梯度消失的现象。实际上，Tanh激活函数相当于Sigmoid的平移： $tanh(z)=2sigmoid(2z)-1$ ，所以Tanh激活函数导数在z很大或很小时也会趋近于0，存在梯度消失。

ReLU激活函数的优越点分析?

优点：

1. 从计算的角度上，Sigmoid和Tanh激活函数均需要计算指数，复杂度高，而ReLU只需要一个阈值即可得到激活值。
2. ReLU的非饱和性可以有效地解决梯度消失的问题，提供相对宽的激活边界。
3. ReLU的单侧抑制提供了网络的稀疏表达能力。

为什么模型稀疏表达能力重要？考虑一下人工神经网络试图模仿的生物神经网络，这在直觉上是有意义的。尽管人类有数十亿个神经元，但并非所有时间都为我们所做的所有事情激发。相反，它们具有不同的作用，并由不同的信号激活。稀疏性导致简洁的模型，这些模型通常具有**更好的预测能力和更少的过拟合**。在稀疏网络中，神经元更有可能实际上正在处理问题的有意义的方面。例如，在检测图像中猫的模型中，可能存在可以识别耳朵的神经元，如果图像是关于建筑物的，则显然不应激活该神经元。稀疏网络比密集网络更快，因为要计算的东西更少。

局限性：

Dying ReLU。ReLU的训练过程中会导致神经元死亡的问题。这是由于ReLU函数导致负梯度在经过该ReLU单元时被置为0，且在之后也不被任何数据激活，即流经该神经元的梯度永远为0，不对任何数据产生响应。在实际训练中，如果学习率设置较大，会导致超过一定比例的神经元不可逆死亡，进而参数梯度无法更新，整个训练过程失败。

解决：Leaky ReLU。$f(z) = \begin{cases} z, & \text{if z} > 0 \\ az, & \text{if z} \leq 0 \end{cases}$，LReLU与ReLU的区别在于，当 $z \leq 0$ 时其值不为0，而是一个斜率为a的线性函数，一般a为一个很小的常数， 这样既实现了单侧抑制，又保留了部分负梯度信息以致不完全丢失。但另一方面，a值的选择增加了问题难度，需要较强的人工先验或多次重复训练以确定合适的参数值。因此还有众多ReLU的变形来解决这个问题，例如：PReLU(Parametric ReLU)，将a作为一个参数，随神经网络一起学习。再如Random ReLU(RReLU)，在训练过程中，斜率a作为一个满足某种分布的随机采样;测试时再固定下来。

反向传播算法?

反向传播其实就是高中数学所学的函数链式求导法则。考虑一个复合函数：$f(x,y,z) = (x+y)z$。将公式分成两部分：$q = x + y$ 和 $f = qz$。对这分开的两个公式进行计算，因为 f 是 q 和 z 相乘，所以 $\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$，又因为 q 是 x 加 y，所以 $\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$。然而，神经网络并不需要关心中间量 q 的梯度，因为 $\frac{\partial f}{\partial q}$ 没有用。相反，函数 f 关于 x, y, z 的梯度才是需要关注的。**链式法则**指出将这些梯度表达式链接起来的正确方式是相乘，比如 $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$。

将这个过程可视化如上图。**前向传播**从输入计算到输出（绿色），**反向传播**从尾部开始，根据链式法则递归地向前计算梯度（显示为红色），一直到网络的输入端。可以认为，梯度是从计算链路中回流。

目前的深度学习框架其都是实现了自动求梯度的功能，这使得我们只需要关注模型架构的设计，而不必关注模型背后的梯度是如何计算的。自动求梯度的实现主要分为两种：前向模式和反向模式。

前向模式是借助“二元数”（dual number）在前向计算过程可以同时得到函数值与其导数。例如我们要计算 $f(x) = 2x + x^2$ 在 x=2 处的函数值与导数。

$$
\begin{aligned}
f(2+\varepsilon) &= 2\times(2+\varepsilon)+(2+\varepsilon)\times(2+\varepsilon) \\
&= 4+2\varepsilon+4+(2+2)\varepsilon+\varepsilon^2 \\
&= 8+6\varepsilon
\end{aligned}
$$

直接就能算出x=2处，函数值为8，导数值是6。但是前向模式在一次计算中，只能计算一个参数的梯度值，而深度神经网络里往往存在大量的参数，如果采用前向模式，需要多次计算整个网络，效率很低。所以深度学习框架中，基本都采用反向模式来做实现。反向模式就是我们所说的反向传播算法，框架将自动生成整个计算图，从神经网络的输出端依次反向求梯度，直到计算出所有参数的梯度。

深度学习与可微编程?

LeCun曾在facebook的文章里说:"Deep Learning Is Dead. Long Live Differentiable Programming!" (深度学习已死，可微编程永生)。这个概念严格定义我也说不好，但是我们可以从目前深度学习发展的角度来谈谈看法。其实任何神经网络都可以像搭积木一样拼出来，这个积木只需要一个条件，就是可以微分，这种可微分是做BP的必要条件, 而且可微分有一个很好的特点，就是链式法则，这样决定了所有的微分可以组合起来，而自动求导机制正是为这样一个可微的模块化组装提供了可能。

目前大部分的深度学习系统都是基于一阶梯度优化器进行优化，当然也存在其他风格的深度学习系统，例如：Hebbian learning。

深度学习的"三朵乌云"?

上文中提到，我们不会无限加宽单隐藏层神经网络，而是选择更深的神经网络。其实深度神经网络也给我们带来了新的问题：

- **梯度爆炸和梯度消失**。深度学习中采用反向传播的方式来进行自动求导，梯度会层层进行累积，试想如果每一层或大部分层的梯度都大于1或者小于1（如sigmoid/tanh激活函数存在的梯度饱和问题，当z很大或者很小的时候，梯度值几乎为0），即便只是大一点点或者小一点点 $(0.99^{1000} = 0.000043, 1.01^{1000} = 20959.16)$ ，都将给神经网络的训练带来灾难性的结果。梯度值太大，导致很快就计算溢出了，梯度值太小，参数几乎不更新，形象的称之为"没学到东西"。
- **网络退化问题**。梯度消失/爆炸问题导致模型训练难以收敛，这个问题很大程度上可以被标准初始化和中间层正规化方法有效控制了，这些方法使得深度神经网络可以收敛。在神经网络可以收敛的前提下，随着网络深度增加，网络表现先是逐渐增加至饱和，然后迅速下降[1]。这一点并不符合常理：如果存在某个 K 层的网络 f 是当前最优的网络，那么可以构造一个更深的网络，其最后几层仅是该网络 f 第 K 层输出的恒等映射(Identity Mapping)，就可以取得与 f 一致的结

果；也许 K 还不是所谓“最佳层数”，那么更深的网络就可以取得更好的结果。总而言之，与浅层网络相比，更深的网络的表现不应该更差。因此，一个合理的猜测就是，对神经网络来说，恒等映射 $H(x) = x$ 并不容易拟合。一个直观的解释就是由于非线性激活函数Relu的存在，每次输入到输出的过程都几乎是不可逆的，这也造成了许多不可逆的信息损失。一个特征的一些有用的信息损失了，那么恒等映射显然不容易拟合。

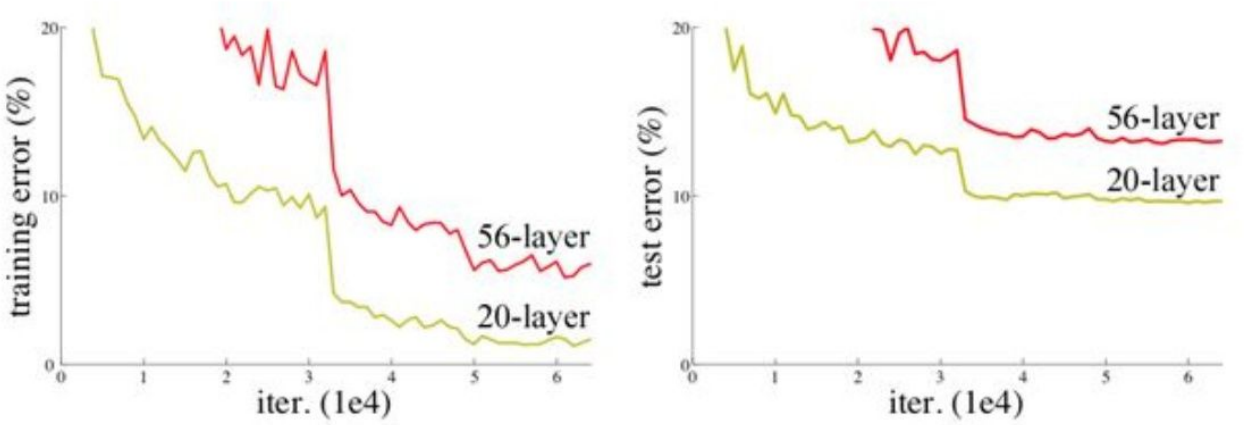


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

- **过拟合**。过拟合的情况我们已经不陌生了，在传统机器学习里也很常见。深度学习里，由于参数众多，模型高度复杂，过拟合的情况非常常见，除了传统的L1、L2正则化方法外，早停止、Dropout是非常有用的手段来解决深度学习过拟合问题。

参数初始化有哪些方法?

神经网络中，参数的初始化可以帮助我们有限的缓解梯度消失/爆炸问题，适当的初始化还可以打破对称权重现象。

1. **预训练初始化**。通常采用一个已经在大规模数据上训练过的模型来做初始化的参数，然后在目标任务上进一步学习，又称为精调(Fine-tuning)。
2. **固定值初始化**。对于一些特殊的参数，我们可以根据经验用一个特殊的固定值来进行初始化。比如偏置(Bias)通常用 0 来初始化，再如在LSTM网络的遗忘门中，偏置通常初始化为 1 或 2，使得时序上的梯度变大。
3. **随机初始化**。在神经网络里，如果初始化参数都相同，在第一遍前向计算时，所有的隐藏层神经元的激活值都相同。在反向传播时，所有权重的更新也都相同. 这样会导致隐藏层神经元没有区分性。这种现象也称为对称权重现象。为了打破这个平衡，我们采用随机初始化的方法。随机初始化有三类常用的方法：基于固定方差的参数初始化、基于方差缩放的初始化方法和正交初始化方法。

参数随机初始化有哪些方法?

基于固定方差的参数初始化。

1. 高斯分布初始化：使用一个高斯分布 $N(0,\sigma^2)$ 对每个参数进行随机初始化。
2. 均匀分布初始化：在一个给定的区间 $[-r,r]$ 内采用均匀分布来初始化参数。由均匀分布的基本知识：对于 $[a,b]$ 区间上的随机变量x， $var(x)=\frac{(a-b)^2}{12}$ ，所以要想在 $[-r,r]$ 上，满足方差为 σ^2 ，需要 r 取 $\sqrt{3\sigma^2}$ 。

在基于固定方差的随机初始化方法中，比较关键的是如何设置方差 σ^2 ， 如果取的太小，一是会导致神经元的输出过小，经过多层之后信号就慢慢消失了，二是还会使得Sigmoid型激活函数丢失非线性的能力。以Sigmoid型函 数为例，在0附近基本上是近似线性的。这样多层神经网络的优势也就不存在了。如果参数范围取的太大，会导致输入状态过大，对于Sigmoid型激活函数来说，激活值变得饱和，梯度接近于0，从而导致梯度消失问题。为了降低固定方差对网络性能以及优化效率的影响，基于固定方差的随机初始化方法一般需要配合逐层归一化来使用。

基于方差缩放的初始化方法。

要高效地训练神经网络，参数初始化的区间应该根据神经元的性质进行差异化的设置。如果一个神经元的输入连接很多，它的每个输入连接上的权重就应该小一些， 以避免神经元的输出过大(当激活函数为 ReLU 时)或过饱和(当激活函数为 Sigmoid 函数时)。初始化一个深度网络时，为了缓解梯度消失或爆炸问题，我们尽可能保持每个神经元的输入和输出的方差一致，根据神经元的连接数量进行自适应的调整初始化分布的方差，这类方法称为方差缩放(Variance Scaling).

初始化方法	激活函数	均匀分布 $[-r,r]$	高斯分布 $\mathcal{N}(0,\sigma^2)$
Xavier 初始化	Logistic	$r=4\sqrt{\frac{6}{M_{l-1}+M_l}}$	$\sigma^2=16\times\frac{2}{M_{l-1}+M_l}$
Xavier 初始化	Tanh	$r=\sqrt{\frac{6}{M_{l-1}+M_l}}$	$\sigma^2=\frac{2}{M_{l-1}+M_l}$
He 初始化	ReLU	$r=\sqrt{\frac{6}{M_{l-1}}}$	$\sigma^2=\frac{2}{M_{l-1}}$

正交初始化方法。

一种更加直接的方式是将 $w^{(l)}$ 初始化为正交矩阵，即 $w^{(l)}(w^{(l)})^T=I$ ，这种方法称为正交初始化(Orthogonal Initialization). 正交初始化的具体实现过程可以分为两步：首先，用均值为 0、方差

为 1 的高斯分布 初始化一个矩阵，其次，将这个矩阵用奇异值分解得到两个正交矩阵，并使用其中之一作为权重矩阵。

当在非线性神经网络中应用正交初始化时，通常需要将正交矩阵乘以一个缩放系数 ρ . 比如当激活函数为 ReLU 时，激活函数在 0 附近的平均梯度可以近似为0.5. 为了保持范数不变，缩放系数 ρ 可以设置为 $\sqrt{2}$ 。

逐层归一化是怎么做的？解决了什么问题？

在深度神经网络中，中间某一层的输入是其之前的神经层的输出。 因此，其之前的神经层的参数变化会导致其输入的分布发生较大的差异。在使用随机梯度下降来训练网络时，每次参数更新都会导致网络中间每一层的输入的分布发生改变。 越深的层，其输入的分布会改变得越明显。 就像一栋高楼，低楼层发生一个较小的偏移，都会导致高楼层较大的偏移，这就是梯度消失/爆炸的来源。为了使每一个神经层的输入的分布在训练过程中保持一致。最简单直接的方法就是对每一个神经层都进行归一化操作，使其分布保持稳定。这里的逐层归一化方法是指可以应用在深度神经网络中的任何一个中间层，实际上并不需要对所有层进行归一化。逐层归一化主要包括：批量归一化、层归一化。

批量归一化(Batch Normalization)。

令第 l 层的净输入 为 $z^{(l)}$ ，神经元的输出为 $a^{(l)}$

$$a^{(l)} = f(z^{(l)}) = f(Wa^{(l-1)} + b)$$

为了减少内部协变量偏移问题，就要使得净输入 $z^{(l)}$ 的分布一致，比如都归一化到标准正态分布。但是逐层归一化需要在中间层进行操作，要求效率比较高，因此复杂度比较高的白化方法就不太合适。为了提高归一化效率，一般使用标准归一化，将净输入 $z^{(l)}$ 的每一维都归一到标准正态分布：

$$\tilde{z}^{(l)} = \frac{z^{(l)} - E(z^{(l)})}{\sqrt{var(z^{(l)} + \epsilon)}}$$

$E(z^{(l)})$ 和 $var(z^{(l)})$ 指当前参数下， $z^{(l)}$ 的每一维在整个训练集上的期望和方差。因为目前主要的训练方法是基于小批量的随机梯度下降方法，因此 $z^{(l)}$ 的期望和方差通常用当前小批量样本集的均值和方差近似估计。

给定一个包含 K 个样本的小批量样本集合，第 l 层神经元的净输入 $z^{(1,l)}, z^{(2,l)}, \ldots, z^{(k,l)}$ 的均值和方差为：

$$\mu_{(l)} = \frac{1}{K}\sum_{k=1}^{K} z^{(k,l)}$$

$$\sigma_{(l)}^2 = \frac{1}{K}\sum_{k=1}^{K} (z^{(k,l)} - \mu_{(l)}) \odot (z^{(k,l)} - \mu_{(l)})$$

对净输入 $z^{(l)}$ 的标准归一化会使得其取值集中在0附近，如果使用sigmoid型函数时，这个取值区间刚好接近线性变换区间，减弱了神经网络的非线性性质，因此，为了使得归一化不对网络的表示能力造成负面影响，我们可以通过一个附加的缩放和平移变换改变取值区间。

$\tilde{z}^{(l)} = \frac{z^{(l)} - \mu_{(l)}}{\sqrt{\sigma_{(l)}^2 + \epsilon}} \odot \gamma + \beta \Leftarrow BN_{\gamma,\beta}(z^{(l)})$ ， γ 和 β 是该层的学习参数。

层归一化(Layer Normalization)。

层归一化和批归一化不同的是，**层归一化是对一个中间层的所有神经元进行归一化**。

令第 l 层神经的净输入为 $z^{(l)}$ ，其均值和方差为：

$$\mu^{(l)} = \frac{1}{n^{(l)}}\sum_{i=1}^{n^{(l)}} z_i^{(l)}$$

$\sigma^{(l)^2} = \frac{1}{n^{(l)}}\sum_{k=1}^{n^{(l)}} (z_i^{(l)} - \mu^{(l)})^2$ ， $n^{(l)}$ 为第 l 层神经元的数量。

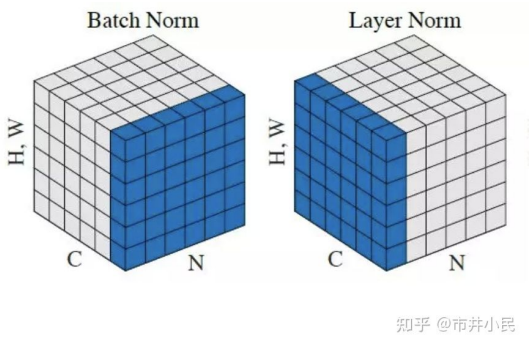
层归一化定义为：

$$\tilde{z}^{(l)} = \frac{z^{(l)} - \mu^{(l)}}{\sqrt{\sigma^{(l)} + \epsilon}} \odot \gamma + \beta \Leftarrow LN_{\gamma,\beta}(z^{(l)})$$

γ, β 代表缩放和平移的参数向量，和 $z^{(l)}$ 维数相同，是该层的学习参数。

批量归一化和层归一化的区别?

- 批量归一化 — 为每一个小batch计算每一层的平均值和方差
- 层归一化 — 独立计算每一层每一个样本的均值和方差



批量归一化有什么作用？有什么局限性?

作用：

1. **解决反向传播过程中的梯度问题。**Batch Normalization的通过规范化的手段,将越来越偏的分布拉回到标准化的分布,使得激活函数的输入值落在激活函数对输入比较敏感的区域，从而减缓梯度消失问题。
2. **加快训练和收敛速度的。**Batch Normalization把每层的数据转换到均值为0，方差为1的状态下，这样数据的分布是相同的，训练会比较容易收敛（和我们在特征工程说的归一化作用一样）。另一方面，均值为0，方差为1的状态下，在梯度计算时会产生比较大的梯度值，可以加快参数的训练，
3. **防止过拟合。**BN的使用使得一个mini-batch中的所有样本都被关联在了一起，因此网络不会从某一个训练样本中生成确定的结果。具体来说，同样一个样本的输出不再仅仅取决于样本本身，也取决于跟这个样本属于同一个mini-batch的其它样本。同一个样本跟不同的样本组成一个mini-batch，它们的输出是不同的（仅限于训练阶段，在inference阶段是没有这种情况的）。可以理解成一种数据增强。

局限性：

1. BN 在每次训练迭代中计算mini-batch统计信息（Mini-batch 均值和方差），因此在训练时需要更大的mini-batch大小，以便它可以有效地逼近来自 mini-batch 的总体均值和方差。这使得 BN 更难用于训练对象检测、语义分割等应用的网络，因为它们通常在高输入分辨率（通常高达 1024x 2048）下工作，这样的样本使用更大的批量进行训练在计算上是不可行的。
2. BN 不适用于 RNN。这是 RNN 与之前的时间戳具有循环连接，并且在 BN 层中的每个时间步长都需要单独的 β 和 γ ，这反而增加了额外的复杂性并使 BN 与 RNN 一起使用变得更加困难。

残差网络的结构是怎样的?

通过上述深度网络退化问题的描述我们已经明白，要让之不退化，**根本原因就是如何做到恒等映射**。事实上，已有的神经网络很难拟合潜在的恒等映射函数 $H(x) = x$ 。但如果把网络设计为 $H(x) = F(x) + x$，即直接把恒等映射作为网络的一部分，就可以把问题转化为学习一个残差函

数 $F(x)=H(x)-x$.只要 $F(x)=0$，就构成了一个恒等映射 $H(x)=x$，拟合残差至少比拟合恒等映射容易得多。[2]

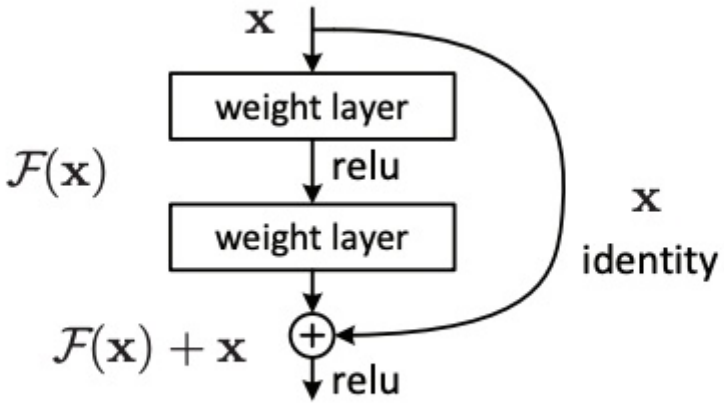


Figure 2. Residual learning: a building block.

残差网络为什么有效?

1. 加了残差结构后就是给了输入 x 多一个选择，在神经网络学习到这层的参数是冗余的时候它可以选择直接走这条"跳接"曲线，跳过这个冗余层，而不需要再去拟合参数使得输出 $H(x)$ 等于 x 。
2. 因为学习残差的计算量比学习输出等于输入小。假设普通网络为A，残差网络为B，输入为2，输出为2（输入和输出一样是为了模拟冗余层需要恒等映射的情况），那么普通网络就是 $A(2)=2$，而残差网络就是 $B(2)=F(2)+2=2$，显然残差网络中的 $F(2)=0$。我们知道网络中权重一般会初始化成0附近的数，那么我们就很容易理解，为什么让 $F(2)$（经过权重矩阵）拟合0会比 $A(2)=2$ 容易了。
3. 我们知道ReLU能够将负数激活为0，而正数输入等于输出。这相当于过滤了负数的线性变化，让 $F(x)=0$ 变得更加容易。
4. 我们知道残差网络可以表示成 $H(x)=F(x)+x$，这就说明了在求输出 $H(x)$ 对输入 x 的倒数（梯度），也就是在反向传播的时候， $H'(x)=F'(x)+1$，残差结构的这个常数1也能保证在求梯度的时候梯度不会消失。客观上，残差神经网络也缓解了梯度消失的问题。

参考

1. ^ Deep Residual Learning for Image Recognition http://arxiv.org/abs/1512.03385
2. ^ https://zhuanlan.zhihu.com/p/106764370