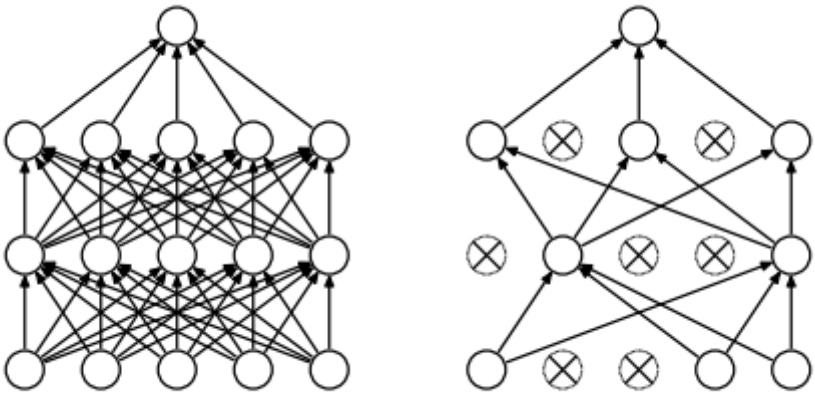
深度学习两大基础Tricks：Dropout和BN详解

Dropout

dropout作为目前神经网络训练的一项必备技术，自从被Hinton提出以来，几乎是进行深度学习训练时的标配。就像做菜时必须加料酒一样，无论何时，大家在使用全连接层的时候都会习惯性的在后面加上一个dropout层。通常情况下，dropout被作为一种防止神经网络过拟合的正则化方法，对神经网络的泛化性能有很大的帮助。每个人都会用dropout，但你真的理解它吗？本节我们就来看看dropout里的一些关键细节问题。

dropout的概念相信大家都已熟稔在心了，是指在神经网络训练中，以一定的概率随机地丢弃一部分神经元来简化网络的一项操作。本质上来说，dropout就是在正常的神经网络基础上给每一层的每一个神经元加了一道概率流程来随机丢弃某些神经元以达到防止过拟合的目的。



(a) Standard Neural Net (b) After applying dropout.

在keras中，dropout的实现只需要一行代码：

```
from keras.layers impo
x = Dropout(0.5)(x)
```

但作为深度学习从业人员的你，在使用keras敲下这样一行代码时，你需要心里对其实现细节无比清晰才对。我们先来看一下包含dropout的神经网络训练过程，主要是前向传播和反向传播。先来看包含dropout的前向传播过程。假设标准神经网络前向传播过程的数学描述为：

$$
\begin{aligned}
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)}\mathbf{y}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}),
\end{aligned}
$$

使用一个参数为p的服从Bernoulli二项分布的随机变量r，将这个随机变量加入到标准神经网络输入中，那么带有dropout的神经网络可以描述为：

$$
\begin{aligned}
r_j^{(l)} &\sim \text{Bernoulli}(p), \\
\tilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \\
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)}\tilde{\mathbf{y}}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}).
\end{aligned}
$$

这样的数学描述简洁明了，好像也没什么特别注意的细节。我们以一层全连接网络为例来看dropout的具体实现过程。

```
D1 = np.random.rand(A1.shape[0
D1 = D1 < prob
A1 = np.multiply(D1, A1)
A1 = A1 / prob
```

其中A1为上一层的输出，D1为用随机数生成的一组dropout向量，然后将其与保留概率prob做比较得到一个布尔向量，再将其与A1做乘积即可得到失活后的A1，按理说dropout到这里应该也就完成了，但最后还有一个将

A1除以保留概率的操作。所以这里有个疑问，为什么在dropout之后还要做个rescale的除法？

其实，这种实现dropout的方法也叫**Inverted Dropout**，是一种经典的dropout实现方法。先不说Inverted Dropout，我们来看正常dropout应该是怎样的：当我们使用了dropout后，在模型训练阶段只有占比为p部分的神经元参与了训练，那么在预测阶段得到的结果会比实际平均要大1/p，所以在测试阶段我们需要将输出结果乘以p来保持输出规模不变。这种原始的dropout实现方式也叫**Vanilla Dropout**。Vanilla操作有一个重大缺陷，那就是预测过程需要根据训练阶段所使用的dropout策略做调整，比较麻烦，所以一般情况下都不会使用这种方法。

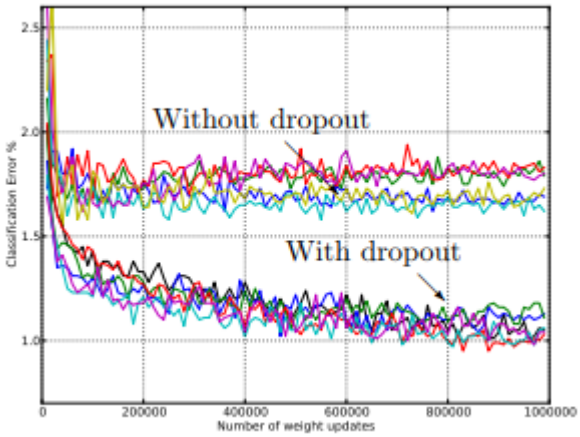
既如此，相必大家也知道了，我们目前用的都是Inverted Dropout方法，为了能够在神经网络训练完成后安安心心的做预测，我们可以把全部心思都放在训练阶段，所有的设置都在训练阶段完成。所以为了保证神经网络在丢弃掉一些神经元之后总体信号强度不变和预测结果稳定，也有一种说法叫保证**Bernoulli**二项分布的数学期望不变，我们在**Inverted Dropout**方法中对**dropout**之后的做了除以**p**的**rescale**操作。

反向传播时同理，梯度计算时需要除以保留概率：

```
dA1 = np.multiply(dA1, D1)
dA1 = dA1 / prob
```

这就是dropout最关键的一个细节问题，一般在深度学习岗位面试时面试官喜欢追着问其中的细节。值得注意一下。

另一个细节问题在于dropout有一种类似集成学习的boosting思想在里面。神经网络以1-p的概率丢弃某些神经元，当进行多次训练时，因为随机性每次训练的都是不同的网络，dropout使得神经网络不依赖于某些独立的特征，最后的结果也就是几次训练之后的平均结果，这些都使得神经网络具备更好的泛化性能，也正是dropout能够防止过拟合的主要原因。

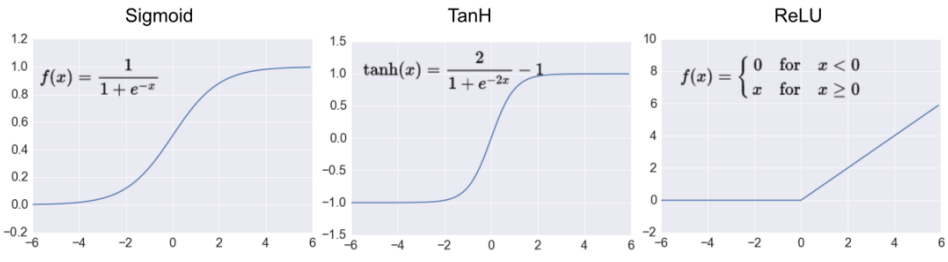


dropout作为一项主流的神经网络训练trick，大家在使用时要知其然，更要知其所以然。

BatchNormalization

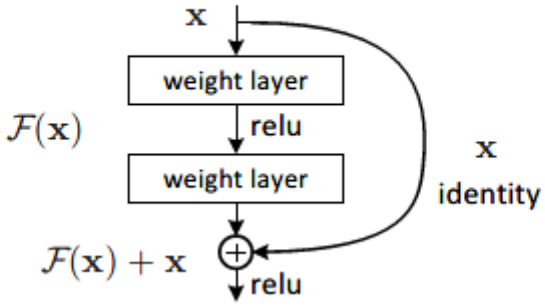
深度神经网络一直以来就有一个特点：随着网络加深，模型会越来越难以训练。所以深度学习有一个非常本质性的问题：为什么随着网络加深，训练会越来越困难？为了解决这个问题，学界业界也一直在尝试各种方法。

sigmoid作为激活函数一个最大的问题会引起梯度消失现象，这使得神经网络难以更新权重。使用ReLu激活函数可以有效的缓解这一问题。



对神经网络使用正则化方法也能对这个问题有所帮助，使用dropout来对神经网络进行简化，可以有效缓解神经网络的过拟合问题，对于深度网络的训练也有一定的帮助。ResNet使用残差块和skip connection来解决这个问题，使得深度加深时网络仍有较好的表现力。

BN本质上也是一种解决深度神经网络难以训练问题的方法。



机器学习的一个重要假设就是IID(Independent Identically Distributed)假设，即独立同分布假设。所谓独立同分布，就是指训练数据和测试数据是近似于同分布的，如若不然，机器学习模型就会很难有较好的泛化性能。

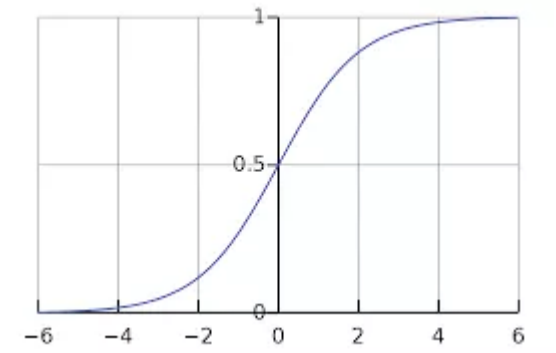
一个重要的观点就是深度神经网络在训练过程中每一层的输入并不满足独立同分布假设，当叠加的网络层每一层的输入分布都发生变化时，这使得神经网络训练难以收敛。这种神经网络隐藏层输入分布的不断变化的现象就叫Internal Covariate Shift(ICS)。ICS问题正是导致深度神经网络难以训练的重要原因之一。

一直在做铺垫，还没说到底什么是BN。Batch Normalization，简称BN，翻译过来就是批标准化，因为这个Normalization是建立在Mini-Batch SGD的基础之上的。BN是针对ICS问题而提出的一种解决方案。一句话来说，**BN**就是使得深度神经网络训练过程中每一层网络输入都保持相同分布。

既然ICS问题表明神经网络隐藏层输入分布老是不断变化，我们能否让每个隐藏层输入分布稳定下来？通常来说，数据标准化是将数据喂给机器学习模型之前一项重要的数据预处理技术，数据标准化也即将数据分布变换成均值为0，方差为1的标准正态分布，所以也叫0-1标准化。图像处理领域的数据标准化也叫白化(whiten)，当然，白化方法除了0-1标准化之外，还包括极大极小标准化方法。

所以一个很关键的联想就是能否将这种白化操作推广到神经网络的每一个隐藏层？答案当然是可以的。

ICS问题导致深度神经网络训练难以收敛，隐藏层输入分布逐渐向非线性激活函数取值区间的两端靠近，比如说sigmoid函数的两端就是最大正值或者最小负值。这里说一下梯度饱和和梯度敏感的概念。当取值位于sigmoid函数的两端时，即sigmoid取值接近0或1时，梯度接近于0，这时候就位于梯度饱和区，也就是容易产生梯度消失的区域，相应的梯度敏感就是梯度计算远大于0，神经网络反向传播时每次都能使权重得到很好的更新。

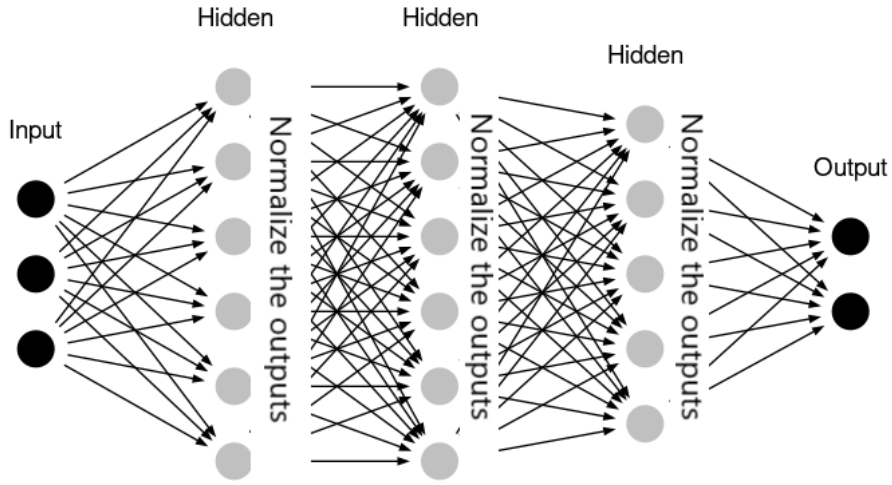


当梯度逐渐向这两个区域靠近时，就会产生梯度爆炸或者梯度消失问题，这也是深度神经网络难以训练的根本原因。**BN**将白化操作应用到每一个隐藏层，对每个隐藏层输入分布进行标准化变换，把每层的输入分布都强行拉回到均值为**0**方差为**1**的标准正态分布。这样一来，上一层的激活输出值**(**即当前层的激活输入值**)**就会落在非线性函数对输入的梯度敏感区，远离了原先的梯度饱和区，神经网络权重易于更新，训练速度相应加快。

那么具体到实际应用时，BN操作应该放在哪里？以一个全连接网络为例：

可以看到，BN操作是对每一个隐藏层的激活输出做标准化，即BN层位于隐藏层之后。对于Mini-Batch SGD来说，一次训练包含了m个样本，具体的BN变换就是执行以下公式的过程：

这里有个问题，就是在标准化之后为什么又做了个scale and shift的变换。从作者在论文中的表述看，认为每一层都做BN之后可能会导致网络的表征能力下降，所以这里增加两个调节参数(scale和shift)，对变换之后的结



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

果进行反变换，弥补网络的表征能力。

BN不仅原理上说的通，但关键还是效果好。BN大大缓解了梯度消失问题，提升了训练速度，模型准确率也得到提升，另外BN还有轻微的正则化效果。

参考资料：

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov; 15(Jun):1929−1958, 2014.

Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift[J]. arXiv preprint arXiv:1502.03167, 2015.