1 超参数调试

1.1 超参数

目前已经讲到过的超参数中，重要程度依次是（仅供参考）：

最重要：

- 学习率 α；

其次重要：

- β：动量衰减参数，常设置为 0.9；
- #hidden units：各隐藏层神经元个数；
- mini-batch 的大小；

再次重要：

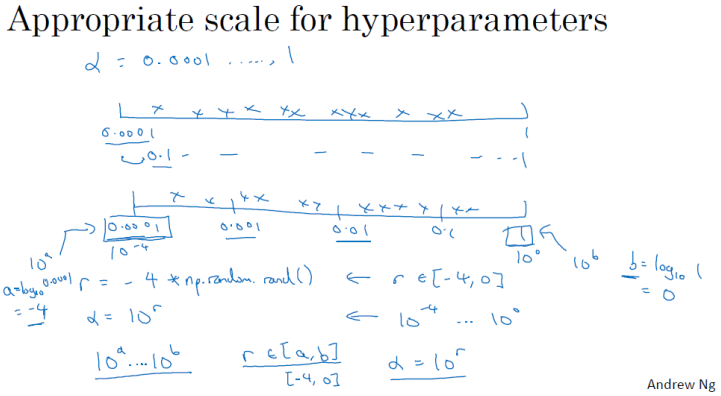
- β_1，β_2，ϵ：Adam 优化算法的超参数，常设为 0.9、0.999、10^{-8}；
- #layers：神经网络层数；
- decay_rate：学习衰减率；

1.2 调参技巧

系统地组织超参调试过程的技巧：

- 随机选择点（而非均匀选取），用这些点实验超参数的效果。这样做的原因是我们提前很难知道超参数的重要程度，可以通过选择更多值来进行更多实验；
- 由粗糙到精细：聚焦效果不错的点组成的小区域，在其中更密集地取值，以此类推；

1.3 选择合适的范围

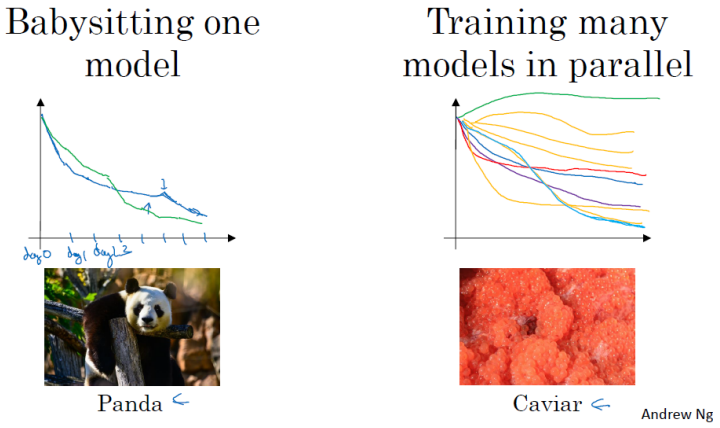


- 对于学习率 α，用对数标尺而非线性轴更加合理：0.0001、0.001、0.01、0.1 等，然后在这些刻度之间再随机均匀取值；
- 对于 β，取 0.9 就相当于在 10 个值中计算平均值，而取 0.999 就相当于在 1000 个值中计算平均值。可以考虑给 1-β 取值，这样就和取学习率类似了。

上述操作的原因是当 β 接近 1 时，即使 β 只有微小的改变，所得结果的灵敏度会有较大的变化。例如，β 从 0.9 增加到 0.9005 对结果（1/(1-β)）几乎没有影响，而 β 从 0.999 到 0.9995 对结果的影响巨大（从 1000 个值中计算平均值变为 2000 个值中计算平均值）。

1.4 一些建议

- 深度学习如今已经应用到许多不同的领域。不同的应用出现相互交融的现象，某个应用领域的超参数设定有可能通用于另一领域。不同应用领域的人也应该更多地阅读其他研究领域的 paper，跨领域地寻找灵感；
- 考虑到数据的变化或者服务器的变更等因素，建议每隔几个月至少一次，重新测试或评估超参数，来获得实时的最佳模型；
- 根据你所拥有的计算资源来决定你训练模型的方式：



- Panda（熊猫方式）：在在线广告设置或者在计算机视觉应用领域有大量的数据，但受计算能力所限，同时试验大量模型比较困难。可以采用这种方式：试验一个或一小批模型，初始化，试着让其工作运转，观察它的表现，不断调整参数；
- Caviar（鱼子酱方式）：拥有足够的计算机去平行试验很多模型，尝试很多不同的超参数，选取效果最好的模型；

2 批标准化

2.1 介绍

批标准化（Batch Normalization，经常简称为 BN）会使参数搜索问题变得很容易，使神经网络对超参数的选择更加稳定，超参数的范围会更庞大，工作效果也很好，也会使训练更容易。

之前，我们对输入特征 X 使用了标准化处理。我们也可以用同样的思路处理隐藏层的激活值 $a^{[l]}$，以加速 $W^{[l+1]}$ 和 $b^{[l+1]}$ 的训练。在实践中，经常选择标准化 $Z^{[l]}$：

$$\mu = \frac{1}{m}\sum_i z^{(i)} \tag{1}$$

$$\sigma^2 = \frac{1}{m}\sum_i (z_i - \mu)^2$$

$$z^{(i)}_{norm} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

其中，m 是单个 mini-batch 所包含的样本个数，ϵ 是为了防止分母为零，保证数值稳定，通常取 10^{-8}。

这样，我们使得所有的输入 $z^{(i)}$ 均值为 0，方差为 1。但我们不想让隐藏层单元总是含有平均值 0 和方差 1，也许隐藏层单元有了不同的分布会更有意义。因此，我们计算：

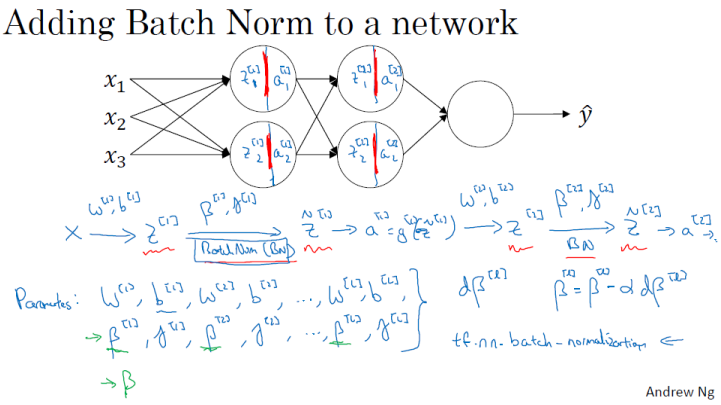
$$\tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta \tag{2}$$

其中，γ 和 β 都是模型的学习参数，所以可以用各种梯度下降算法来更新 γ 和 β 的值，如同更新神经网络的权重一样。

通过对 γ 和 β 的合理设置，可以让 $\tilde{z}^{(i)}$ 的均值和方差为任意值。这样，我们对隐藏层的 $z^{(i)}$ 进行标准化处理，用得到的 $\tilde{z}^{(i)}$ 替代 $z^{(i)}$ 。

设置 γ 和 β 的原因是，如果各隐藏层的输入均值在靠近 0 的区域，即处于激活函数的线性区域，不利于训练非线性神经网络，从而得到效果较差的模型。因此，需要用 γ 和 β 对标准化后的结果做进一步处理。例如当 $\gamma = \sqrt{\sigma^2 + \epsilon}, \beta = \mu$，就抵消掉了之前的正则化操作。

2.2 BN与神经网络



对于 L 层神经网络，经过 Batch Normalization 的作用，整体流程如下：

$$X \xrightarrow{W^{[1]},b^{[1]}} Z^{[1]} \xrightarrow[\text{BN}]{\beta^{[1]},\gamma^{[1]}} \tilde{Z}^{[1]} \rightarrow A^{[1]} \rightarrow \cdots \rightarrow A^{[L-1]} \xrightarrow{W^{[L]},b^{[L]}} Z^{[L]} \xrightarrow[\text{BN}]{\beta^{[L]},\gamma^{[L]}} \tilde{Z}^{[L]} \rightarrow A^{[L]}$$

实际上，Batch Normalization 经常使用在 mini-batch 上，这也是其名称的由来。

下面是使用 BN 的前向传播和后向传播流程图：

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

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

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

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

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

$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2}(\sigma_{\mathcal{B}}^2 + \epsilon)^{-3/2}$$

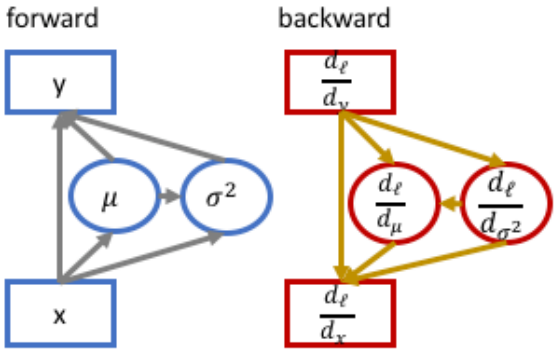
$$\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}\right) + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{\sum_{i=1}^{m} -2(x_i - \mu_{\mathcal{B}})}{m}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}$$

$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \widehat{x}_i$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}$$

整体对比如下：



Batch Normalization [5] in training mode.

使用 Batch Normalization 时，因为标准化处理中包含减去均值的一步，因此 b 实际上没有起到作用，其数值效果交由 β 来实现。因此，在 Batch Normalization 中，可以省略 b 或者暂时设置为 0。

在使用梯度下降算法时，分别对 $W^{[l]}$，$\beta^{[l]}$ 和 $\delta^{[l]}$ 进行迭代更新。除了传统的梯度下降算法之外，还可以使用之前学过的动量梯度下降、RMSProp 或者 Adam 等优化算法。

2.3 BN 有效解释

Batch Normalization 效果很好的原因有以下两点：

1. 通过对隐藏层各神经元的输入做类似的标准化处理，提高神经网络训练速度；
2. 可以使前面层的权重变化对后面层造成的影响减小，整体网络更加健壮。

关于第二点，如果实际应用样本和训练样本的数据分布不同（例如，橘猫图片和黑猫图片），我们称发生了“Covariate Shift”。这种情况下，一般要对模型进行重新训练。Batch Normalization 的作用就是减小 Covariate Shift 所带来的影响，让模型变得更加健壮，鲁棒性（Robustness）更强。

即使输入的值改变了，由于 Batch Normalization 的作用，使得均值和方差保持不变（由 γ 和 β 决定），限制了在前层的参数更新对数值分布的影响程度，因此后层的学习变得更容易一些。Batch Normalization 减少了各层 W 和 b 之间的耦合性，让各层更加独立，实现自我训练学习的效果。

另外，Batch Normalization 也起到**微弱的正则化（regularization）**效果。因为在每个 mini-batch 而非整个数据集上计算均值和方差，只由这一小部分数据估计得出的均值和方差会有一些噪声，因此最终计算出的 $\tilde{z}^{(i)}$ 也有一定噪声。类似于 dropout，这种噪声会使得神经元不会再特别依赖于任何一个输入特征。

因为 Batch Normalization 只有微弱的正则化效果，因此可以和 dropout 一起使用，以获得更强大的正则化效果。通过应用更大的 mini-batch 大小，可以减少噪声，从而减少这种正则化效果。

最后，不要将 Batch Normalization 作为正则化的手段，而是当作加速学习的方式。正则化只是一种非期望的副作用，Batch Normalization 解决的还是反向传播过程中的梯度问题（梯度消失和爆炸）。

2.4 测试时的 BN

Batch Normalization 将数据以 mini-batch 的形式逐一处理，但在测试时，可能需要对每一个样本逐一处理，这样无法得到 μ 和 σ^2。

理论上，我们可以将所有训练集放入最终的神经网络模型中，然后将每个隐藏层计算得到的 $\mu^{[l]}$ 和 $\sigma^{2[l]}$ 直接作为测试过程的 μ 和 σ 来使用。但是，实际应用中一般不使用这种方法，而是使用之前学习过的指数加权平均的方法来预测测试过程单个样本的 μ 和 σ^2。

对于第 l 层隐藏层，考虑所有 mini-batch 在该隐藏层下的 $\mu^{[l]}$ 和 $\sigma^{2[l]}$，然后用指数加权平均的方式来预测得到当前单个样本的 $\mu^{[l]}$ 和 $\sigma^{2[l]}$。这样就实现了对测试过程单个样本的均值和方差估计。

3 SoftMax 回归

目前为止，介绍的分类例子都是二分类问题：神经网络输出层只有一个神经元，表示预测输出 $\hat{y}$ 是正类的概率$P(y=1|x)$，$\hat{y}>0.5$ 则判断为正类，反之判断为负类。

对于多分类问题，用 C 表示种类个数，则神经网络输出层，也就是第 L 层的单元数量 $n^{[L]}=C$。每个神经元的输出依次对应属于该类的概率，即 $P(y=c|x), c=0,1,..,C-1$。有一种 Logistic 回归的一般形式，叫做 Softmax 回归，可以处理多分类问题。

对于 Softmax 回归模型的输出层，即第 L 层，有：

$$Z^{[L]}=W^{[L]}a^{[L-1]}+b^{[L]} \tag{3}$$

激活函数使用的是softmax函数：

$$\sigma(z)_j=\frac{exp(z_j)}{\sum_{i=1}^{m}exp(z_i)} \tag{4}$$

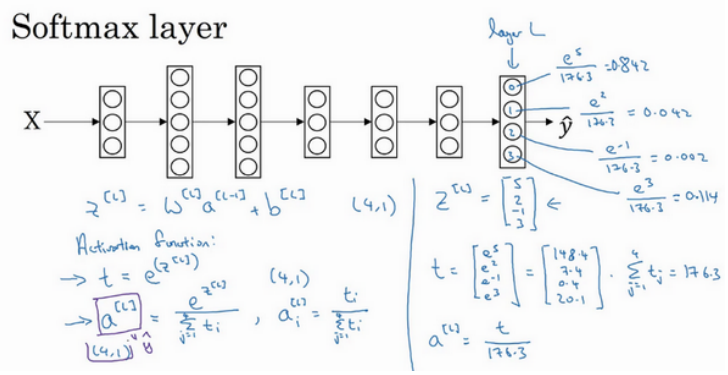
for i in range(L)，有：

$$a_i^{[L]}=\frac{e^{Z_l^{[L]}}}{\sum_{i=1}^{C}e^{Z_l^{[L]}}} \tag{5}$$

为输出层每个神经元的输出，对应属于该类的概率，满足：

$$\sum_{i=1}^{C}a_i^{[L]}=1 \tag{6}$$

下面是一个直观地例子：



代价函数

定义损失函数为：

$$L(\hat{y},y)=-\sum_{j=1}^{C}y_j log\hat{y}_j \tag{7}$$

当 i 为样本真实类别，则有：

$$y_j=0, j\neq i \tag{8}$$

因此，损失函数可以简化为：

$$L(\hat{y},y)=-y_i log\hat{y}_i=log\hat{y}_i \tag{9}$$

所有 m 个样本的成本函数为：

$$J=\frac{1}{m}\sum_{i=1}^{m}L(\hat{y},y) \tag{10}$$

4 深度学习框架

- Caffe / Caffe 2
- CNTK
- DL4J
- Keras
- Lasagne
- mxnet
- PaddlePaddle
- TensorFlow
- Theano
- Torch

选择框架的标准

- 便于编程：包括神经网络的开发和迭代、配置产品；
- 运行速度：特别是训练大型数据集时；
- 是否真正开放：不仅需要开源，而且需要良好的管理，能够持续开放所有功能。