蒋豆芽

深度学习——初始化方法 已购

来自【机器学习面试题汇总与解析（蒋豆芽面试题总结）】 | 54 浏览 | 0 回复 | 2021-05-03

蒋豆芽

+关注

机器学习面试题汇总与解析——初始化方法

1. **说说初始化方法有哪些？** ☆☆☆☆☆
2. **理想的参数初始化方法是什么？** ☆☆☆☆☆
3. **说说你用过的初始化方法，都有哪些优缺点** ☆☆☆☆☆
4. **网络参数初始化为0可以吗？** ☆☆☆☆☆
5. **随机初始化参数有什么问题？** ☆☆☆☆☆
6. **手推梯度消失和梯度爆炸问题** ☆☆☆☆☆
7. **怎么缓解梯度消失** ☆☆☆☆☆
8. **梯度消失的根本原因** ☆☆☆☆☆
9. **说说归一化方法** ☆☆☆☆☆

==

- 本专栏适合于**Python已经入门**的学生或人士，有一定的编程基础。
- 本专栏适合于**算法工程师、机器学习、图像处理求职**的学生或人士。
- 本专栏针对面试题答案进行了**优化，尽量做到好记、言简意赅。这才是一份面试题总结的正确打开方式。这样才方便背诵**
- 如专栏内容有错漏，欢迎在评论区指出或私聊我更改，一起学习，共同进步。
- **相信大家都有着高尚的灵魂，请尊重我的知识产权，未经允许严禁各类机构和个人转载、传阅本专栏的内容。**

==

1. **说说初始化方法有哪些？** ☆☆☆☆☆

   参考回答

   1. **全0初始化**，

      就是将所有权重置0。当然是不能这样的，神经网络通过梯度更新参数，参数都是0，梯度也就是0，神经网络就停止学习了。

   2. **随机初始化**

蒋豆芽

3. **Xavier初始化**

   随机初始化没有控制方差，所以对于深层网络而言，**随机初始化方法**依然可能失效。**理想的参数初始化还得控制方差**，对w进行一个规范化：

$$w = \frac{0.001 * randn(n_{in}, n_{out})}{\sqrt{n}} \tag{.}$$

   “**Xavier初始化**”维持了**输入输出**数据分布**方差一致性。**

4. **He初始化**

   He初始化由何凯明提出，为了解决“**Xavier初始化**”的缺点。对于非线性激活函数ReLU，“Xavier初始化”方法失效。He初始化基本思想是，当使用ReLU做为激活函数时，Xavier的效果不好，原因在于，当ReLU的输入小于0时，其输出为0，相当于该神经元被关闭了，影响了输出的分布模式。

   因此He初始化，在Xavier的基础上，**假设每层网络有一半的神经元被关闭，于是其分布的方差也会变小。**经过验证发现当对初始化值缩小一半时效果最好，故He初始化可以认为是**Xavier初始 / 2**的结果。

   所以原本的Xavier方差规范化的分母不再是$\sqrt{n_{in}}$而是$\sqrt{\frac{n_{in}}{2}}$

答案解析

什么是网络参数初始化

神经网络在训练时，**前向传播**和**反向传播**都涉及到每个神经元的权重更新w_i，也就是我们说的网络参数了，当然这些参数需要一个初始值。方法有很多，全0初始、随机初始等等，每个方法都有优缺点。

为什么需要合理的参数初始化

理想的网络参数初始化使得模型训练事半功倍，相反，糟糕的初始化可能导致网络**梯度消失**和**梯度爆炸。**举个例子，如网络使用sigmoid函数作为非线性激活函数，若参数初始化的值过大，前向运算时经过sigmoid函数后的输出结果几乎**全为0或1**，而反向传播时梯度全部为0，这就导致**梯度消失**了。再如ReLU，如果初始化不合理，前向运算的结果可能全部为负，发生“**死区**”现象。

再简单说，就是**参数又不能过大，又不能过小。**比如在前向传播过程中输出为$h(wx+b)$，因为w很小，所以输出很小，同时反向传播过程中梯度的变化也很小，那么参数的改变也很小，在不断的正向传播乘很小的数，反向传播又几乎不变的情况下，最后w会越来越小，趋近于0，出现**梯度消失**。反之同理。

最理想化的参数初始化

通过我们上面的叙述，当然最理想化的参数初始化就是，经过多层网络后，**信号不被过分放大或过分减弱。**那么如何保证？数学化的方法就是使**每层网络的输入和输出的方差一致**。然后我们还要尽

蒋豆芽

参数初始化

1. **全0初始化**，

   就是将所有权重置0。当然是不能这样的，神经网络通过梯度更新参数，参数都是0，梯度也就是0，神经网络就停止学习了。

2. **随机初始化**

   将参数随机化，不过随机参数服从**高斯分布**或**均匀分布**，假设网络输入神经元个数为n_{in}，输出神经元个数为n_{out}，则服从高斯分布的参数随机初始化为：

$$w = 0.001 * randn(n_{in}, n_{out}) \tag{.}$$

   其中高斯分布均值为0，方差为1。0.001为控制因子，这样使得参数期望尽量接近0。

3. **Xavier初始化**

   随机初始化没有控制**方差**，所以对于深层网络而言，随机初始化方法依然可能失效。我们上面已经说了，**理想的参数初始化还得控制方差**，对w进行一个规范化：

$$w = \frac{0.001 * randn(n_{in}, n_{out})}{\sqrt{n}} \tag{.}$$

   其中，n为n_{in}或$\frac{n_{in}+n_{out}}{2}$，这便是“Xavier初始化”，维持了**输入输出数据**分布**方差一致性**。注意这里是**正态分布**。如果是**均匀分布**，则参数的随机取值范围为[$-\sqrt{\frac{6}{n_{in}+n_{out}}}$，$\sqrt{\frac{6}{n_{in}+n_{out}}}$]。

   这里我们来具体分析一下“Xavier初始化”如何做到了输入输出数据分布方差的一致性。假设s为未经非线性变换的该层网络的输出结果，$\vec{w}$为该层参数，$\vec{x}$为该层输入数据，则有：

$$\begin{aligned} Var(s) &= Var(\sum_i^n w_i x_i) \\ &= \sum_i^n Var(w_i x_i) \\ &= \sum_i^n [E^2(w_i)Var(x_i) + E^2(x_i)Var(w_i) + Var(x_i)Var(w_i)] \end{aligned} \tag{.}$$

   对于理想情况下处于稳定状态的神经网络参数和输入数据**均值应为0**，则$E(w_i) = E(x_i) = 0$，所以上面的式子简化为：

$$\begin{aligned} Var(s) &= \sum_i^n Var(x_i)Var(w_i) \\ &= (nVar(\vec{w}))var(\vec{x}) \end{aligned} \tag{.}$$

   我们希望输入输出数据分布方差的一致，即$Var(s) = var(\vec{x})$，那么令$(nVar(\vec{w})) = 1$，则$Var(\vec{w}) = \frac{1}{n_{in}} = \frac{2}{n_{in}+n_{out}}$。也就是规范化后的$\vec{w}$的方差为原来的$\frac{1}{n}$，那么规范化后的$\vec{w}$

蒋豆芽

我们更多会采用$\sqrt{\frac{2}{nin+nout}}$，**因为实际当中输入与输出的个数往往不相等，于是为了均衡考量**。从上面推导我们可以知道，**"Xavier初始化"需要配合BN层，而且针对线性激活函数才有用。**

4. **He初始化**

   He初始化由何凯明提出，为了解决"Xavier初始化"的缺点。对于非线性激活函数ReLU，"Xavier初始化"方法失效。He初始化基本思想是，当使用ReLU做为激活函数时，Xavier的效果不好，原因在于，当ReLU的输入小于0时，其输出为0，相当于该神经元被关闭了，影响了输出的分布模式。

   因此He初始化，在Xavier的基础上，**假设每层网络有一半的神经元被关闭，于是其分布的方差也会变小**。经过验证发现当对初始化值缩小一半时效果最好，故He初始化可以认为是**Xavier初始 / 2**的结果。

   所以原本的Xavier方差规范化的分母不再是$\sqrt{n_{in}}$而是$\sqrt{\frac{n_{in}}{2}}$，注意这里是正态分布。如果是均匀分布，则参数的随机取值范围为$[-\sqrt{\frac{6}{n_{in}}}, \sqrt{\frac{6}{n_{in}}}]$

5. **其他初始化方法**

   其他初始化方法使用较少，这里不再赘述。

类似的问题还有：

2. **理想的参数初始化方法是什么？** ☆☆☆☆☆

   参考回答

   当然最理想化的参数初始化就是，经过多层网络后，信号不会产生**梯度消失**或**梯度爆炸**。数学化的方法就是使**每层网络的输入和输出的方差一致**。然后我们还要尽量保证**每层网络参数分布均值为0**，因为这如同归一化，归一化的好处是加快训练；另一个原因也是为了计算方便。

   答案解析

   无。

3. **说说你用过的初始化方法，都有哪些优缺点** ☆☆☆☆☆

   参考回答

   参考上面回答。

   答案解析

   无。

参考回答

就是将所有权重置0。当然是不能这样的，神经网络通过梯度更新参数，参数都是0，梯度也就是0，神经网络就停止学习了。

答案解析

无。

5. **随机初始化参数有什么问题?** ☆☆☆☆☆

   参考回答

   答案参考上面。

   答案解析

   无。

6. **手推梯度消失和梯度爆炸问题** ☆☆☆☆☆

   参考回答

   假设我们有四层隐藏层，每层隐藏层只包含一个神经元节点。激活函数为sigmoid，输入为x，则：

   第一层的前向运算为$z_1 = w_1 x + b_1$，经过激活函数后$a_1 = \sigma(z_1)$

   第二层的前向运算为$z_2 = w_2 a_1 + b_2$，经过激活函数后$a_2 = \sigma(z_2)$

   第三层的前向运算为$z_3 = w_3 a_2 + b_3$，经过激活函数后$a_3 = \sigma(z_3)$

   第四层的前向运算为$z_4 = w_4 a_3 + b_4$，经过激活函数后$a_4 = \sigma(z_4)$

   接着我们开始梯度反向求导：根据链式法则

   $$\begin{aligned}\frac{\partial C}{\partial b_1} &= \frac{\partial C}{\partial a_4} \cdot \frac{\partial a_4}{\partial z_4} \cdot \frac{\partial z_4}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial b_1} \\ &= \frac{\partial C}{\partial a_4} \cdot \sigma^{'}(z_4) \cdot w_4 \cdot \sigma^{'}(z_3) \cdot w_3 \cdot \sigma^{'}(z_2) \cdot w_2 \cdot \sigma^{'}(z_1) \cdot 1\end{aligned} \tag{.}$$

   当 $\sigma^{'}(z_i) < 1$，$\frac{\partial C}{\partial b_1} \to 0$，梯度消失

   当 $\sigma^{'}(z_i) > 1$，$\frac{\partial C}{\partial b_1} \to \infty$，梯度爆炸

   我们神经网络中的初始权值也一般是小于 1 的数，所以相当于公式中是多个小于 1 的数在不断的相乘，导致乘积和还很小。这只是有两层的时候，如果层数不断增多，乘积和会越来越趋近于 0，

除了这个情况以外，还有一个情况会产生梯度消失的问题，即当我们的权重设置的过大时候，较高的层的激活函数会产生饱和现象，如果利用 Sigmoid 函数可能会无限趋近于 1，这个时候斜率接近 0，最终计算的梯度一样也会接近 0， 最终导致无法更新。

答案解析

无。

7. **怎么缓解梯度消失**☆☆☆☆☆

参考回答

1. 预训练加微调
2. 梯度剪切
3. 使用合理的参数初始化方案，如He初始化
4. 使用 ReLU、LReLU、ELU、maxout 等激活函数

   sigmoid函数的梯度随着x的增大或减小和消失，而ReLU不会。
5. 使用批规范化BN
6. 残差结构

答案解析

无。

8. **梯度消失的根本原因**☆☆☆☆☆

参考回答

答案参考上面。

答案解析

无。

9. **说说归一化方法**☆☆☆☆☆

参考回答

1. min-max标准化

2. Z-score标准化方法（白化）

$$x' = \frac{x - mean(x)}{\sigma} \quad (.)$$

σ为方差。

答案解析

为什么要归一化

1）归一化后加快了梯度下降求最优解的速度

2）归一化有可能提高精度