权重/参数初始化

G-kdom
不敢说无悔，起码能死心

一、参数初始化的重要性

参数初始化又称为**权重初始化**（weight initialization）或**权值初始化。**深度学习模型训练过程的本质是对weight（即参数 W）进行更新，这需要每个参数有相应的初始值。说白了，神经网络其实就是对权重参数w不停地迭代更新，以达到较好的性能。

模型权重的初始化对于网络的训练很重要，不好的初始化参数会导致梯度传播问题，降低训练速度；而好的初始化参数能够加速收敛，并且更可能找到较优解。如果权重一开始很小，信号到达最后也会很小；如果权重一开始很大，信号到达最后也会很大。

不合适的权重初始化会使得隐藏层数据的方差过大（例如，随着输入数据量的增长，随机初始化的神经元的输出数据的分布中的方差也增大），从而在经过sigmoid这种非线性层时离中心较远(导数接近0)，因此过早地出现梯度消失。

但是对于其他的方法，输入的数据经过多层神经网络，数据分布的方差可能会越来越大，也可能会越来越小。这主要取决于当前层的参数量和初始权重的分布情况。对于一个多层的网络，某一层的方差可以用累积的形式表达：

$$Var[z^i] = Var[x] \prod_{i'=0}^{i-1} n_{i'} Var[W^{i'}],$$

在深度学习中，**神经网络的权重初始化方法对对模型的收敛速度和性能有着至关重要的影响。**在深度神经网络中，随着层数的增多，我们在梯度下降的过程中，极易出现梯度消失或者梯度爆炸。一个好的权重初始化虽然不能完全解决梯度消失和梯度爆炸的问题，但是对于处理这两个问题是有很大的帮助的，并且十分有利于模型性能和收敛速度，在某些网络结构中甚至能够提高准确率。因此，对权重w的初始化则显得至关重要。

二、常见的参数（权重）初始化方法

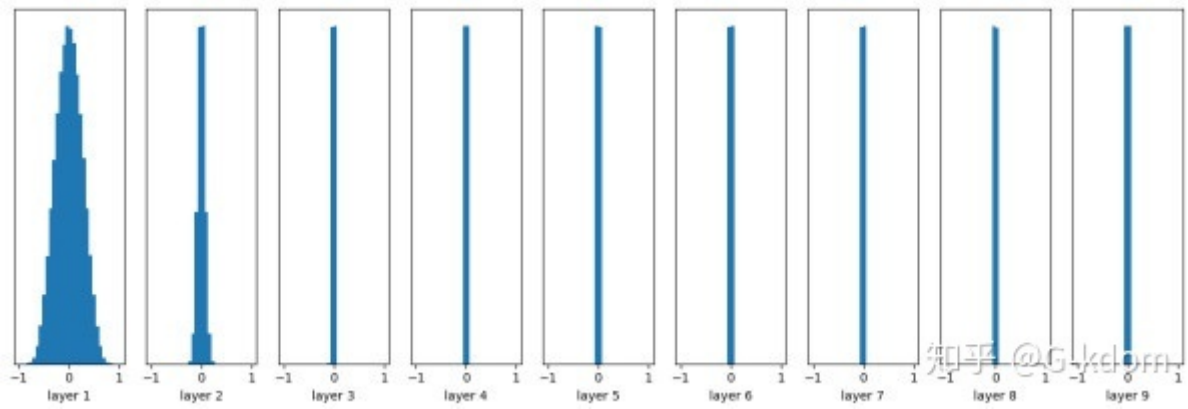
层的神经元学到的东西都是一样的（输出是一样的），而且在BP的时候，每一层内的神经元也是相同的，因为他们的gradient相同，weight update也相同。这显然是一个不可接受的结果。

2. 随机初始化

随机初始化是很多人经常使用的方法，一般初始化的权重为高斯或均匀分布中随机抽取的值。然而这是有弊端的，一旦随机分布选择不当，就会导致网络优化陷入困境。

把权重初始化为较小的值，如均值为0，方差为0.01的高斯分布：

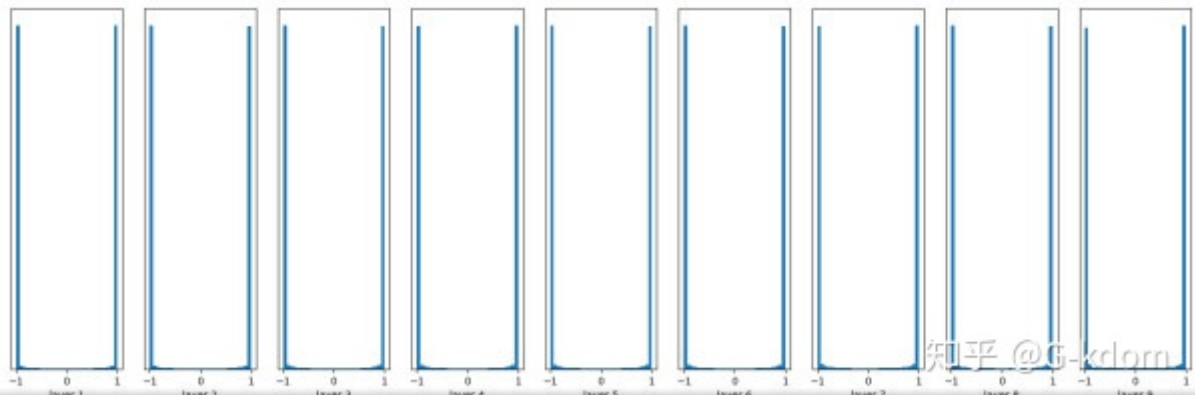
```
w = np.random.randn( node_in, node_out) * 0.01
```



随着层数的增加，我们看到输出值迅速向0靠拢，在后几层中，几乎所有的输出值 x 都很接近0。高斯初始化，给权重较小的值，这种更新方式在小型网络中很常见，然而当网络deep的时候，会出现**梯度消失**的情况。

但是如果把**权重初始成一个比较大的值**，如均值为0，方差为1的高斯分布：

```
w = np.random.randn(node_in, node_out) * 1.0
```



输出值几乎都接近于0。梯度更新时，梯度非常接近于0，就会导致**梯度消失**。

3. Xavier初始化

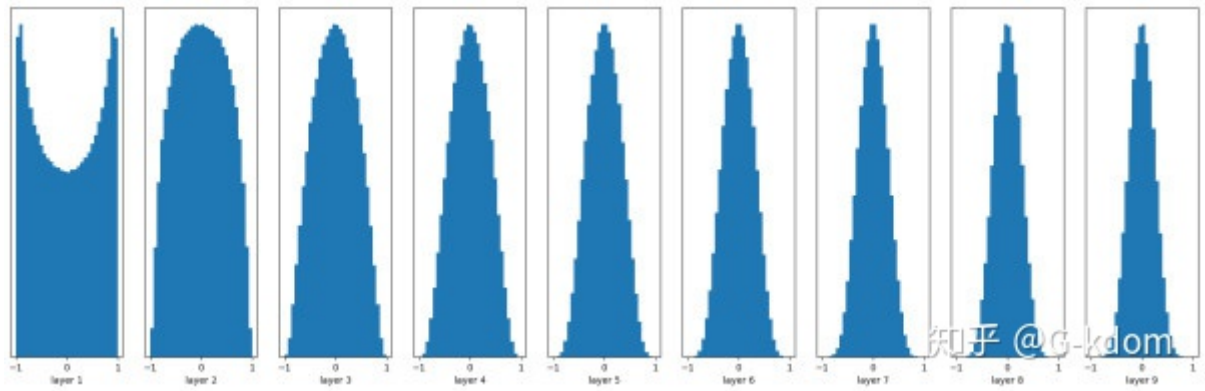
Xavier初始化可以帮助减少梯度消失的问题，使得信号在神经网络中可以传递得更深，在经过多层神经元后保持在合理的范围（不至于太小或太大）。

Xavier初始化的基本思想：保持输入和输出的方差一致（服从相同的分布），这样就避免了所有输出值都趋向于0。

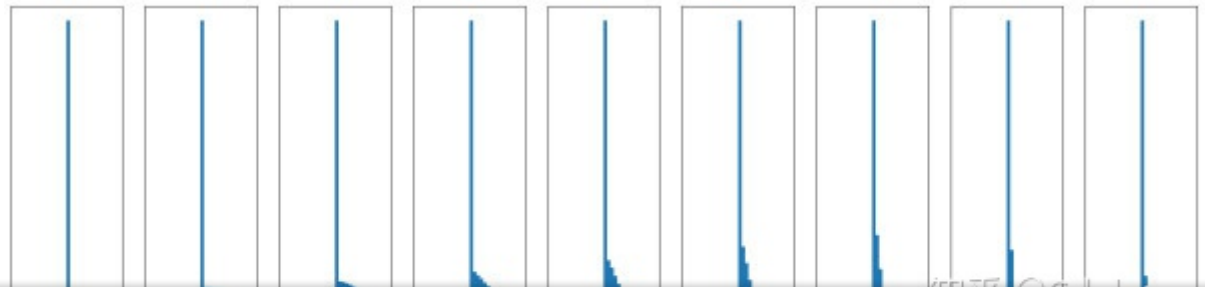
根据输入和输出神经元的数量自动决定初始化的范围：定义参数所在的层的输入维度为 m ，输出维度为 n ，那么参数将从 $\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right)$ 均匀分布中采样。

注意，为了问题的简便，**Xavier初始化的推导过程是基于线性函数的（假设激活函数是线性的）。**

```
w = np.random.randn(node_in, node_out) / np.sqrt(node_in)
```



能够看出，深层的激活函数输出值还是非常漂亮的服从标准高斯分布。虽然Xavier initialization能够很好的 tanh 激活函数，但是对于目前神经网络中最常用的 ReLU 激活函数，还是无能能力，请看下图：



4. He initialization（MSRA）

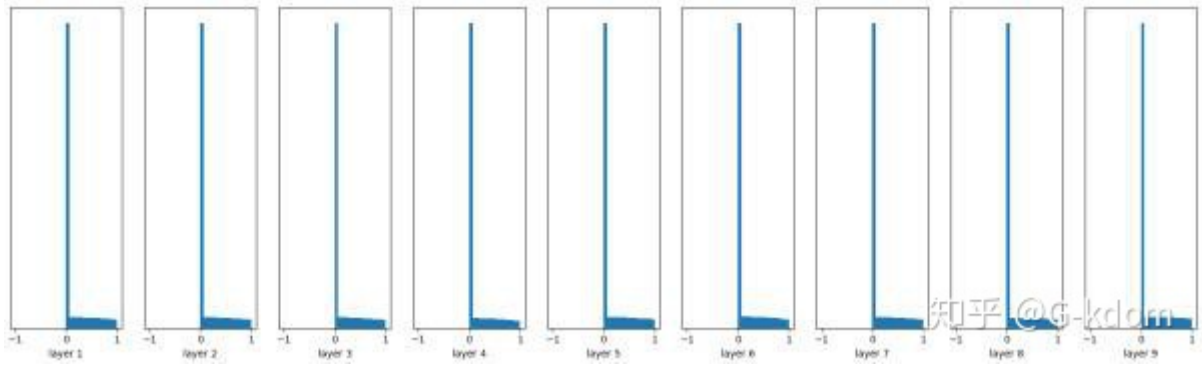
为了解决上面的问题，何恺明大神提出了一种针对ReLU的初始化方法，一般称作 He initialization 或 MSRA。论文链接：Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification

主要想要解决的问题是由于经过relu后，方差会发生变化，因此我们初始化权值的方法也应该变化。只考虑输入个数时，MSRA初始化是一个均值为0，方差为2/n的高斯分布：

$$w \sim G\left[0, \sqrt{\frac{2}{n}}\right]$$

He initialization的思想是：在ReLU网络中，假定每一层有一半的神经元被激活，另一半为0（x负半轴中是不激活的），所以要保持variance不变，只需要在Xavier的基础上再除以2：

```
w = np.random.randn(node_in, node_out) / np.sqrt(node_in/2)
```

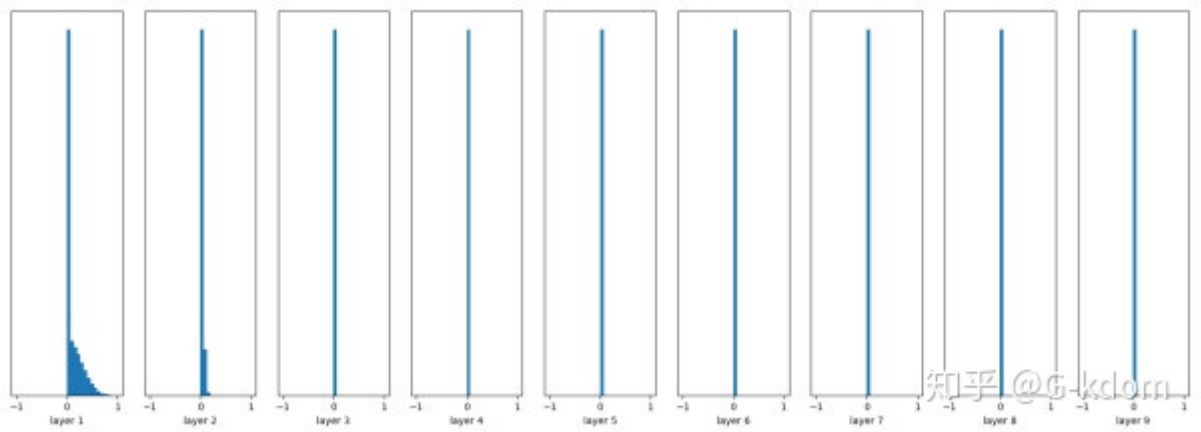


效果是比Xavier initialization好很多。现在神经网络中，隐藏层常使用ReLU，权重初始化常用He initialization这种方法。

5. Batch Normalization Layer

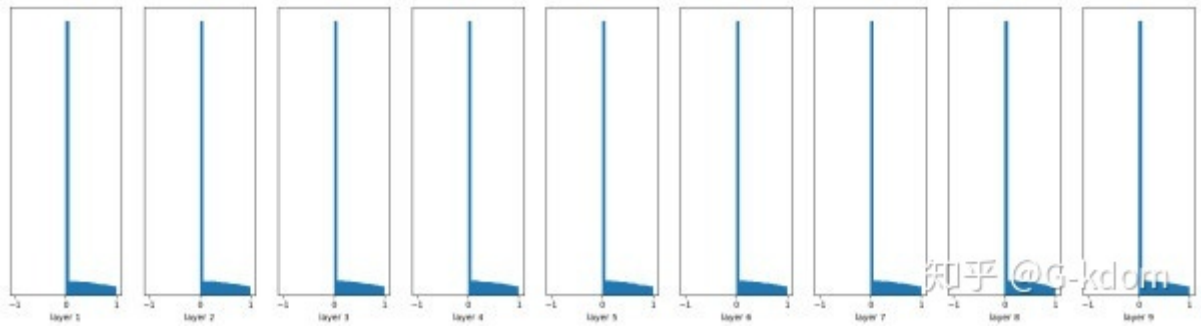
在网络中间层中使用 Batch Normalization 层一定程度上能够减缓对较好的网络参数初始化的依赖，使用方差较小的参数分布即可。

```
w = tf.Variable(np.random.randn( node_in, node_out)) * 0.01
...
fc = tf.nn.relu(fc)
```



【随机初始化，有Batch Normalization】

```
w = tf.Variable(np.random.randn( node_in, node_out)) * 0.01
...
fc = tf.contrib.layers.batch_norm(fc,center = True, scale = True,
                                  is_training = True)
fc = tf.nn.relu(fc)
```



很容易看到，Batch Normalization的效果非常好，推荐使用。

6. pre-training （“锅甩给别人，机智地一匹”）

初始化这个问题明显比较麻烦，不然大家也不会这么喜欢用**pre-training模型**了。

pre-training是早期训练神经网络的有效初始化方法，一个便于理解的例子是先使用greedy layerwise auto-encoder做unsupervised pre-training，然后再做fine-tuning。

fine-tuning阶段：将pre-train过的每一层放回神经网络，利用pre-train阶段得到的参数初始值和训练数据对模型进行整体调整。在这一过程中，参数进一步被更新，形成最终模型。

随着数据量的增加以及activation function 的发展，pre-training的概念已经渐渐发生变化。目前，从零开始训练神经网络时我们也很少采用auto-encoder进行pre-training，而是直奔主题做模型训练。**如果不想从零开始训练神经网络时，我们往往选择一个已经在任务A上训练好的模型（称为pre-trained model），将其放在任务B上做模型调整（称为fine-tuning）。**

三、总结

1. 当前的主流初始化方式 Xavier，MSRA 主要是为了保持每层的输入与输出方差相等, 而参数的分布采用均匀分布或高斯分布都行。

2. 在广泛采用 Batch Normalization 的情况下，使用普通的小方差的高斯分布即可。

3. 在迁移学习的情况下，优先采用预训练的模型进行参数初始化。