1.把w初始化为0

在线性回归 logistics回归的时候，基本上都是把参数初始化为0，模型也能够很好的工作。在神经网络中，不可以把w初始化为0。

2.对w随机初始化

目前最常用的是随机初始化

```
def initialize_parameters_random(layers_dims):
    """
    Arguments:
    layer_dims -- python array (list) containing the size of each layer.
    Returns:
    parameters -- python dictionary containing your parameters "W1", "b1", ..., "WL", "bL":
                    W1 -- weight matrix of shape (layers_dims[1], layers_dims[0])
                    b1 -- bias vector of shape (layers_dims[1], 1)
                    ...
                    WL -- weight matrix of shape (layers_dims[L], layers_dims[L-1])
                    bL -- bias vector of shape (layers_dims[L], 1)
    """
    np.random.seed(3)  # This seed makes sure your "random" numbers will be the as ours
    parameters = {}
    L = len(layers_dims)  # integer representing the number of layers
    for l in range(1, L):
        parameters['W' + str(l)] = np.random.randn(layers_dims[l], layers_dims[l - 1])*0.01
        parameters['b' + str(l)] = np.zeros((layers_dims[l], 1))
    return parameters
```

乘0.01是因为要把W随机初始化到一个相对较小的值，因为如果X很大的话，W又相对较大，会导致Z非常大，这样如果激活函数是sigmoid，就会导致sigmoid的输出值1或者0，然后会导致一系列问题（比如cost function计算的时候，log里是0，这样会有点麻烦）。

但是随机初始化也有缺点，np.random.randn()其实是一个均值为0，方差为1的高斯分布中采样。当神经网络的层数增多时，会发现越往后面的层的激活函数（使用tanH）的输出值几乎都接近于0

3.Xavier初始化

为了解决随机初始化的问题提出来的，思想是尽可能的让输入和输出服从相同的分布，这也能避免后面层的激活函数的输出值趋向于0，

Xavier初始化可以帮助减少梯度弥散问题， 使得信号在神经网络中可以传递得更深。是最为常用的神经网络权重初始化方法。

算法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n,输出维度为m,那么参数将从$[-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}]$均匀分布中采样。

公式推导

假设输入一层X，输出一层Y，那么

$$Y = W_1X_1 + W_2X_2 + \cdots + W_nX_n$$

按照独立变量相乘的方差公式，可以计算出：

$$\mathrm{Var}(W_iX_i) = E[X_i]^2\mathrm{Var}(W_i) + E[W_i]^2\mathrm{Var}(X_i) + \mathrm{Var}(W_i)\mathrm{Var}(X_i)$$

我们期望输入X和权重W都是零均值，因此简化为

$$\mathrm{Var}(W_iX_i) = \mathrm{Var}(W_i)\mathrm{Var}(X_i)$$

进一步假设所有的X_i, W_i都是独立同分布，则有：

$$\mathrm{Var}(Y) = \mathrm{Var}(W_1X_1 + W_2X_2 + \cdots + W_nX_n) = n\mathrm{Var}(W_i)\mathrm{Var}(X_i)$$

即输出的方差与输入有关，为使输出的方差与输入相同，意味着使$n\mathrm{Var}(W_i) = 1$.因此$\mathrm{Var}(W_i) = \frac{1}{n} = \frac{1}{n_{in}}$.

如果对反向传播的梯度运用同样的步骤，可得：$\mathrm{Var}(W_i) = \frac{1}{n_{\text{out}}}$.

由于n_{in}, n_{out}通常不相等，所以这两个方差无法同时满足，作为一种折中的方案，可使用介于$\frac{1}{n_{in}}, \frac{1}{n_{out}}$之间的数来代替：简单的选择是$\mathrm{Var}(W_i) = \frac{2}{n_{\text{in}}+n_{\text{out}}}$.

可以根据均匀分布的方差，反推出W的均匀分布：

由于 [a,b] 区间的均匀分布的方差为：$\mathrm{Var} = \frac{(b-a)^2}{12}$，使其零均值，则b=-a,所以$\mathrm{Var} = \frac{(2b)^2}{12} = \frac{2}{n_{\text{in}}+n_{\text{out}}}$，可得$b = \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}$.

因此，**Xavier**初始化的就是按照下面的均匀分布（uniform distribution）：

$$W \sim U[-\frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}}}]$$

xavier权重初始化的作用，使得信号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）。

4.MSRA

方法来自于何凯明paper 《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：

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

bias初始化

通常初始化为0(若初始化为0.01等值,可能并不能得到好的提升,反而可能下降)

Batch Normalization

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. 参考论文 Batch Normalization。

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均可.
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. 另外, 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.