1 神经网络的实践层面

1.1 数据划分：训练 / 验证 / 测试

应用深度学习是一个高度迭代的过程

需要多次循环往复，才能为应用程序找到一个称心的神经网络，因此循环该过程的效率是决定项目进展速度的一个关键因素。而创建高质量的训练数据集，验证集和测试集也有助于提高循环效率

对于一个需要解决的问题的样本数据，在建立模型的过程中，数据会被划分为以下几个部分：

- 训练集（train set）：用训练集对算法或模型进行训练过程；
- 验证集（development set）：利用验证集（又称为简单交叉验证集，hold-out cross validation set）进行交叉验证，选择出最好的模型；
- 测试集（test set）：最后利用测试集对模型进行测试，获取模型运行的无偏估计（对学习方法进行评估）

在**小数据量**的时代，如 100、1000、10000 的数据量大小，可以将数据集按照以下比例进行划分：

- 无验证集的情况：70% / 30%；
- 有验证集的情况：60% / 20% / 20%；

而在如今的**大数据**时代，对于一个问题，我们拥有的数据集的规模可能是百万级别的，所以验证集和测试集所占的比重会趋向于变得更小。

- 100 万数据量：98% / 1% / 1%；
- 超百万数据量：99.5% / 0.25% / 0.25%（或者99.5% / 0.4% / 0.1%）

验证集的目的是为了验证不同的算法哪种更加有效，所以验证集只要足够大到能够验证大约 2-10 种算法哪种更好，而不需要使用 20% 的数据作为验证集。如百万数据中抽取 1 万的数据作为验证集就可以了。

测试集的主要目的是评估模型的效果，如在单个分类器中，往往在百万级别的数据中，我们选择其中 1000 条数据足以评估单个模型的效果。

交叉验证的基本思想是重复地使用数据；把给定的数据进行切分，将切分的数据集组合为训练集与测试集，在此基础上反复地进行训练、测试以及模型选择。

建议验证集要和训练集来自于**同一个分布（数据来源一致）**，可以使得机器学习算法变得更快并获得更好的效果。如果不需要用无偏估计来评估模型的性能，则可以不需要测试集。

1.2 模型估计：偏差、方差

1.2.1 直观理解

偏差-方差分解（bias-variance decomposition）是解释学习算法泛化性能的一种重要工具。

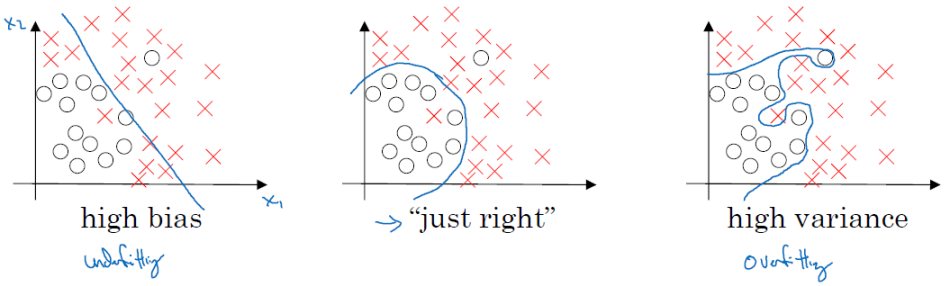
泛化误差可分解为偏差、方差与噪声之和：

- **偏差：**模型在样本上的输出与真实值之间的误差，即模型本身的精准度，反应出**算法的拟合能力**
- **方差：**模型每一次输出结果与模型输出期望之间的误差，即**模型的稳定性**，反应出预测的波动情况
- **噪声：**表达了在当前任务上任何学习算法所能够达到的期望泛化误差的下界，即刻画了**学习问题本身的难度**

$$Err(\boldsymbol{x}) = \mathbb{E}_D\left[\left(f(\boldsymbol{x};D) - \bar{f}(\boldsymbol{x})\right)^2\right] + \left(\bar{f}(\boldsymbol{x}) - y\right)^2 + \mathbb{E}_D\left[\left(y_D - y\right)^2\right]$$

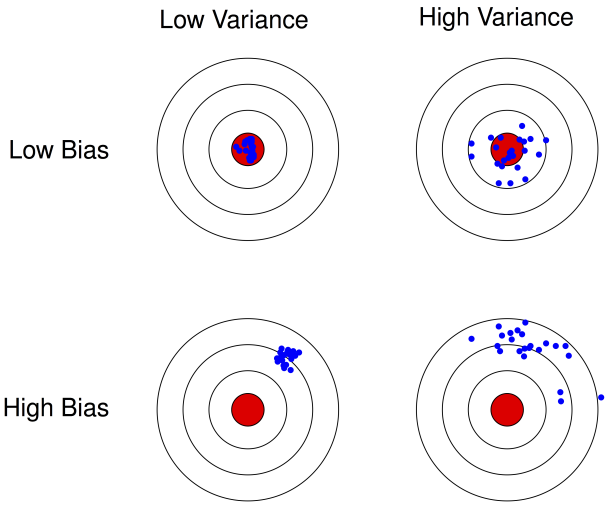
- variance
- bias^2
- noise

我们以一张图来说明：



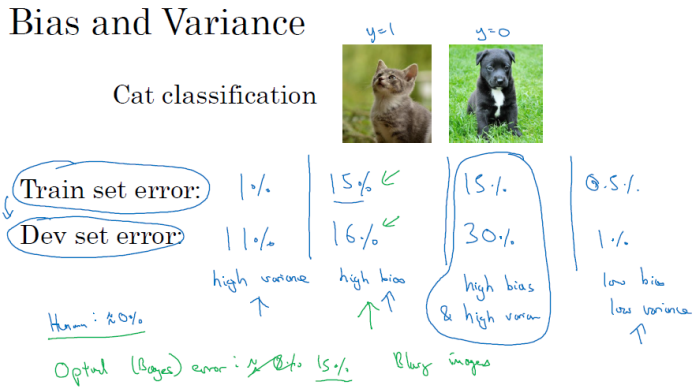
如图中的左图，对图中的数据采用一个**简单的模型**，例如线性拟合，并不能很好地对这些数据进行分类，分类后存在较大的偏差（Bias），称这个分类模型**欠拟合（Underfitting）**。右图中，采用**复杂的模型**进行分类，例如深度神经网络模型，当模型复杂度过高时变容易出现**过拟合（Overfitting）**，使得分类后存在较大的方差（Variance）。中间的图中，采用一个恰当的模型，才能对数据做出一个差不多的分类。

- 如果模型太简单，换句话说就是**用低维去拟合高维**，容易产生偏差，也就易出现欠拟合；
- 如果模型太复杂，换句话说就是**用高维去拟合低维**，容易产生方差，也就易出现过拟合；



- 低偏差低方差：表示模型既准确又稳定，效果最好，但是现实中这种情形很少遇见。
- 低偏差高方差：表示模型准确但是稳定性差，对验证数据&测试数据的拟合能力差，即是模型的泛化能力差，产生了过拟合(Overfitting)。
- 高偏差低方差：表示模型的准确度差，对数据的拟合能力弱，产生了欠拟合(Underfitting)。
- 高偏差高方差：表示模型既不准确又不稳定。

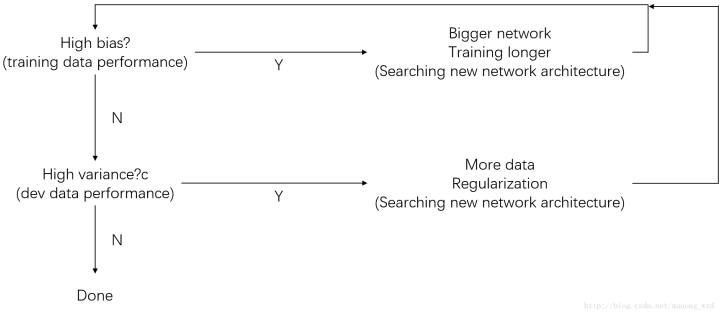
下面以一张图来说明问题：



当训练出一个模型以后，如果：

- 训练集的错误率较小，而验证集的错误率却较大，说明模型存在较大方差，可能出现了过拟合；
- 训练集和开发集的错误率都较大，且两者相当，说明模型存在较大偏差，可能出现了欠拟合；
- 训练集错误率较大，且开发集的错误率远较训练集大，说明方差和偏差都较大，模型很差；
- 训练集和开发集的错误率都较小，且两者的相差也较小，说明方差和偏差都较小，这个模型效果比较好。

下图是利用偏差和方差来训练神经网络的基本方法：



1.2.2 应对方法

存在高偏差（欠拟合）：

- 扩大网络规模，如添加隐藏层或隐藏单元数目；
- 寻找合适的网络架构，使用更大的 NN 结构；
- 花费更长时间训练。

存在高方差（过拟合）：

- 获取更多的数据；
- 正则化（regularization）；
- 寻找更合适的网络结构。

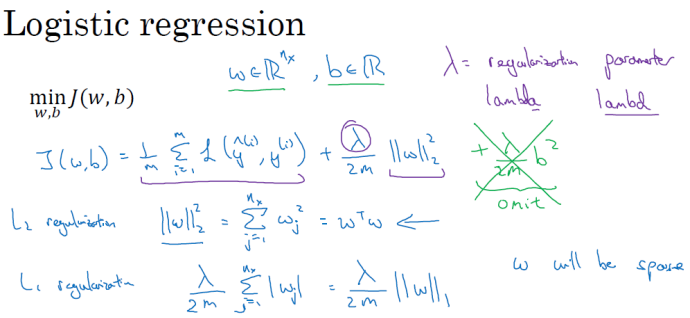
不断尝试，直到找到低偏差、低方差的框架。

1.3 正则化

正则化是在成本函数中加入一个正则化项，惩罚模型的复杂度。正则化可以用于解决高方差的问题。

1.3.1 L2 正则化

Logistic 回归中的正则化



由于L1正则化最后得到w向量中将存在大量的0，使模型变得稀疏化,所以一般都使用L2正则化。其中的参数 λ 称为正则化参数，这个参数通常通过开发集来设置。注意，lambda 在 Python 中属于保留字，所以在编程的时候，用 lambd 代替这里的正则化因子。

神经网络中的正则化

对于神经网络，加入正则化的成本函数：

$$J(w^{[1]}, b^{[1]}, \dots, w^{[L]}, b^{[L]}) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L}||w^{[l]}||_F^2 \tag{1}$$

因为 w 的大小为 $n^{[l]} \times n^{[l-1]}$，因此：

$$||w^{[l]}||_F^2 = \sum_{i=1}^{n^{[l-1]}}\sum_{j=1}^{n^{[l]}}(w_{ij}^{[l]})^2 \tag{2}$$

这被称为**弗罗贝尼乌斯范数（Frobenius Norm）**,所以神经网络的中的正则化项被称为弗罗贝尼乌斯范数矩阵。

在加入正则化项后，梯度变为（反向传播要按这个计算）：

$$dW^{[l]} = \frac{\partial L}{\partial w^{[l]}} + \frac{\lambda}{m}W^{[l]} \tag{3}$$

更新参数时：

$$\begin{aligned} W^{[l]} &:= W^{[l]} - \alpha[\frac{\partial L}{\partial w^{[l]}} + \frac{\lambda}{m}W^{[l]}] \\ &= W^{[l]} - \alpha\frac{\lambda}{m}W^{[l]} - \alpha\frac{\partial L}{\partial w^{[l]}} \\ &= (1 - \frac{\alpha\lambda}{m})W^{[l]} - \alpha\frac{\partial L}{\partial w^{[l]}} \end{aligned} \tag{4}$$

其中，因为 $1 - \frac{\alpha\lambda}{m} < 1$，会给原来的 $W^{[l]}$ 一个衰减的参数，因此 L2 正则化项也被称为**权重衰减（Weight Decay）**。

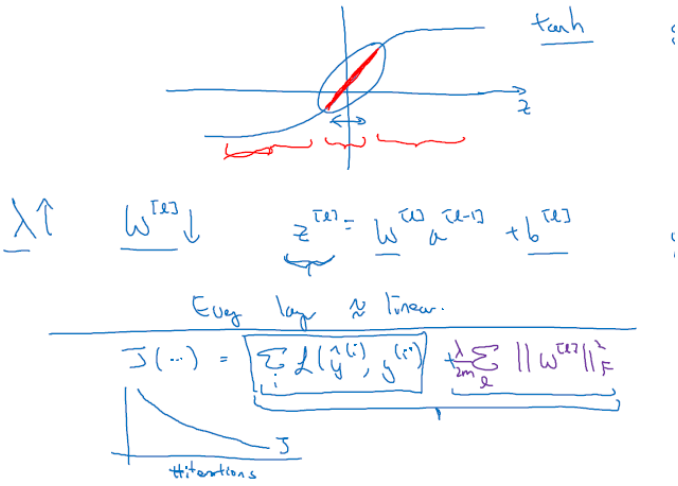
正则化预防过拟合解释

直观解释：

正则化因子设置的足够大的情况下，为了使成本函数最小化，权重矩阵 W 就会被设置为接近于 0 的值，直观上相当于消除了很多神经元的影响，那么大的神经网络就会变成一个较小的网络。当然，实际上隐藏层的神经元依然存在，但是其影响减弱了，便不会导致过拟合。

数学解释：

假设神经元中使用的激活函数为g(z) = tanh(z)（ sigmoid 同理）：



在加入正则化项后，当 λ 增大，导致 $W^{[l]}$ 减小，$Z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]}$便会减小。由上图可知，在 z 较小（接近于 0）的区域里，tanh(z) 函数近似线性，所以每层的函数就近似线性函数，整个网络就成为一个简单的近似线性的网络，因此不会发生过拟合。

上图中还有一个是代价函数随着迭代次数变化的曲线：在调试梯度下降时，它代表梯度下降的调幅数量。可以看到，不加正则项的代价函数对于梯度下降的每个调幅都单调递减。如果添加正则化函数，代价函数已经有一个全新的定义，函数可能不会在所有调幅范围内都单调递减。

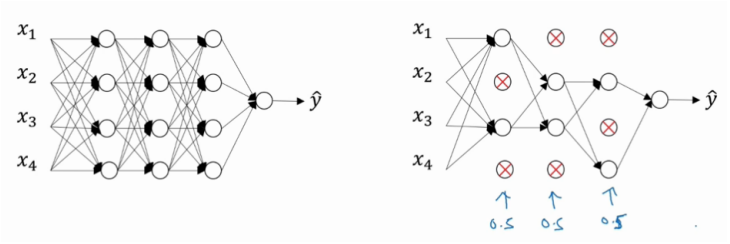
其他解释：

在权值 $W^{[l]}$ 变小之下，输入样本 X 随机的变化不会对神经网络模造成过大的影响，神经网络受局部噪音的影响的可能性变小。这就是正则化能够降低模型方差的原因。

1.3.2 dropout 正则化

随机失活

dropout（随机失活）是在神经网络的隐藏层为每个神经元结点设置一个随机消除的概率，保留下来的神经元形成一个结点较少、规模较小的网络用于训练。dropout 正则化较多地被使用在计算机视觉（Computer Vision）领域。



使用Python编程时可以用**反向随机失活（Inverted DropOut）**来实现DropOut正则化：

```
# 对第l层进行 dropout：
keep_prob = 0.8   # 设置神经元保留概率
dl = np.random.rand(al.shape[0], al.shape[1]) < keep_prob
al = np.multiply(al, dl)
al /= keep_prob
```

最后一步 al /= keep_prob 是因为 $a^{[l]}$ 中的一部分元素失活（相当于被归零），为了在下一层计算时不影响 $Z^{[l+1]} = W^{[l+1]}a^{[l]} + b^{[l+1]}$的期望值，因此除以一个keep_prob。

注意，在测试阶段不要使用 dropout，因为那样会使得预测结果变得随机。

随机失活的解释

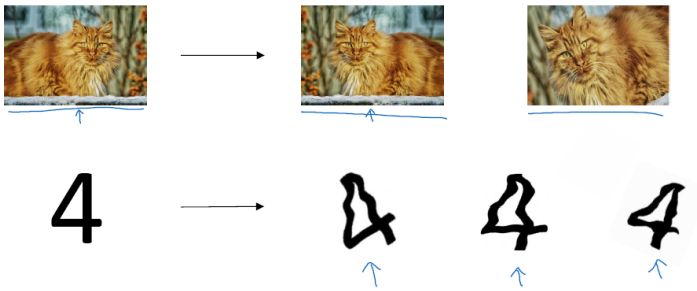
对于单个神经元，其工作是接收输入并产生一些有意义的输出。但是加入了 dropout 后，输入的特征都存在被随机清除的可能，所以该神经元**不会再特别依赖**于任何一个输入特征，即不会给任何一个输入特征设置太大的权重。

因此，通过传播过程，dropout 将产生和 L2 正则化相同的**收缩权重**的效果。

对于不同的层，设置的 keep_prob 也可以不同。一般来说，神经元较少的层，会设 keep_prob 为 1.0，而神经元多的层则会设置比较小的 keep_prob。

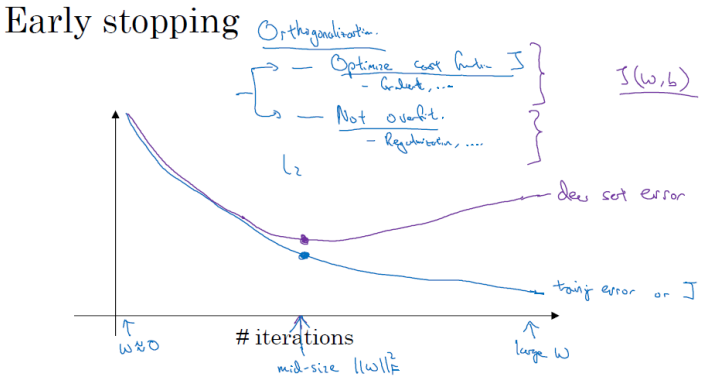
dropout 的一大**缺点**是成本函数无法被明确定义。因为每次迭代都会随机消除一些神经元结点的影响，因此无法确保成本函数单调递减。因此，使用 dropout 时，先将 keep_prob 全部设置为 1.0 后运行代码，确保 J(w,b) 函数单调递减，再打开 dropout。

1.3.3 数据扩增



在无法获取额外的训练样本下，对已有的数据做一些简单的变换。例如对一张图片进行翻转、放大扭曲，以此引入更多的训练样本。

1.3.4 早停止法



将训练集和验证集进行梯度下降时的成本变化曲线画在同一个坐标轴内，当训练集误差降低但验证集误差升高，两者开始发生较大偏差时及时停止迭代，并返回具有最小验证集误差的连接权和阈值，以避免过拟合。Early stopping的优点是，只运行一次梯度下降，你可以找出 w 的较小值，中间值和较大值，而无需尝试正则化超级参数 λ 的很多值。这种方法的缺点是无法同时达成偏差和方差的最优，此外同时训练代价函数和验证集误差的话，考虑的东西会变得很复杂。

1.4 归一化输入

使用归一化处理输入 X 能够有效加速收敛

对训练及测试集进行归一化的过程为：

- 零均值
- 归一化方差

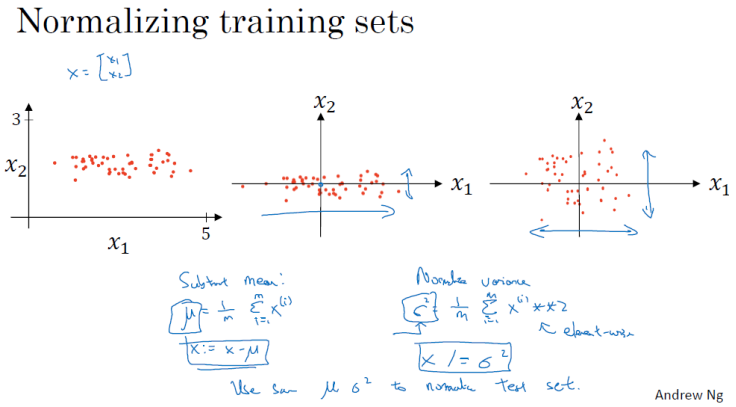
$$\bar{x} = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} \tag{5}$$

$$x^{(i)} := x^{(i)} - \bar{x} \tag{6}$$

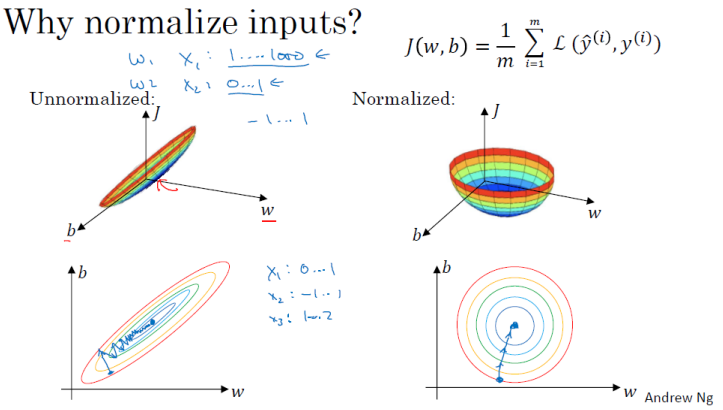
$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} x^{(i)2} \tag{7}$$

$$x^{(i)} := \frac{x^{(i)}}{\sigma^2} \tag{8}$$

下面是一张图直观理解：



归一化解释



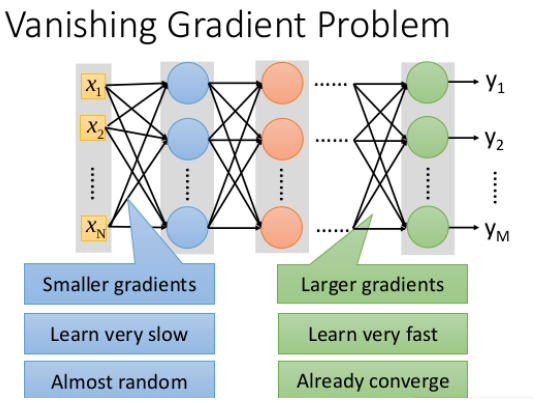
由图可知，使用归一化前后，成本函数的形状有较大差别。

在不使用归一化的成本函数中，如果设置一个较小的学习率，可能需要很多次迭代才能到达全局最优解；而如果使用了归一化，那么无论从哪个位置开始迭代，都能以相对较少的迭代次数找到全局最优解。

如果输入特征处于不同范围内，可能有些特征值从0到1，有些从1到1000，那么归一化特征值就非常重要了。如果特征值处于相似范围内，那么归一化就不是很重要了。

1.5 梯度消失 / 梯度爆炸

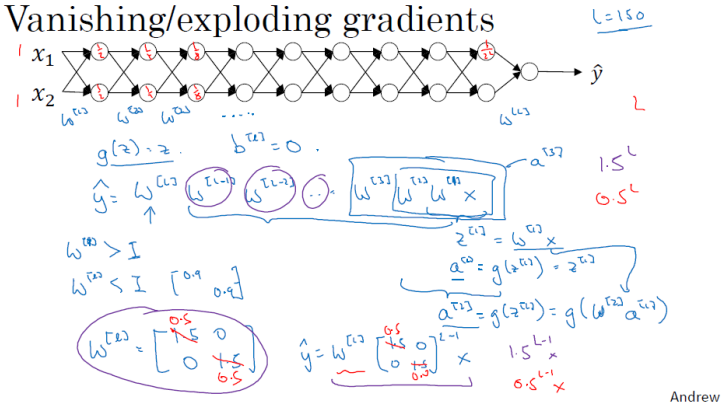
在梯度函数上出现的以指数级递增或者递减的情况分别称为梯度爆炸或者梯度消失



假定 $g(z)=z, b^{[l]}=0$，对于目标输出有：

$$\hat{y}=W^{[L]}W^{[L-1]}\ldots W^{[2]}W^{[1]}X \tag{9}$$

即下图所示：

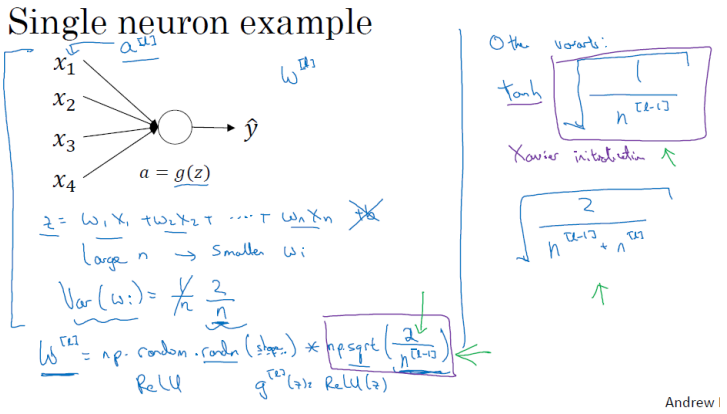


- 对于 W[l]的值大于 1 的情况，激活函数的值将以指数级递增；
- 对于 W[l]的值小于 1 的情况，激活函数的值将以指数级递减。

对于导数同理。因此，在计算梯度时，根据不同情况梯度函数会以指数级递增或递减，导致训练导数难度上升，梯度下降算法的步长会变得非常小，需要训练的时间将会非常长。

利用初始化来缓解梯度消失和梯度爆炸（不完整的解决方案）

我们先来研究只有一个神经元的情况，然后才是深度网络



根据：$z=w_1x_1+w_2x_2+\ldots+w_nx_n+b$

$b=0$，暂时忽略b，为了预防z值过大或过小，你可以看到n越大，你希望w_i越小，因为z是w_ix_i的和，如果你把很多此类项相加，希望每项值更小，最合理的方法就是设置$w_i=\frac{1}{n}$，n表示神经元的输入特征数量，实际上，你要做的就是设置某层权重矩阵$w^{[l]}=np.random.randn(\text{shape})*\text{np.sqrt}(\frac{1}{n^{[l-1]}})$，$n^{[l-1]}$就是我喂给第$l$层神经单元的数量（即第$l-1$层神经元数量）。

这样，激活函数的输入 x 近似设置成均值为 0，标准方差为 1，神经元输出 z 的方差就正则化到 1 了。如果激活函数的输入特征被零均值和标准方差化，方差是1，z也会调整到相似范围，这就没解决问题（梯度消失和爆炸问题）。但它确实降低了梯度消失和爆炸问题，因为它给权重矩阵w设置了合理值，你也知道，它不能比1大很多，也不能比1小很多，所以梯度没有爆炸或消失过快。

- **Relu**激活函数，**He** 初始化，$\sqrt{\frac{2}{n^{[l-1]}}}$
- **tanh**函数，**Xavier** 初始化，$\sqrt{\frac{1}{n^{[l-1]}}}$

1.6 梯度检验

1.6.1 梯度的数值逼近

使用双边误差的方法去逼近导数，精度要高于单边误差。

- 单边误差 $O(\varepsilon)$

$$f'(\theta) = \lim_{\varepsilon \to 0} = \frac{f(\theta+\varepsilon) - (\theta)}{\varepsilon} \tag{10}$$

- 双边误差 $O(\varepsilon^2)$

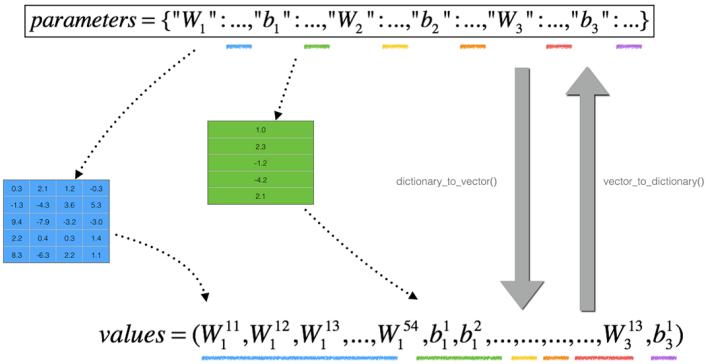
$$f'(\theta) = \lim_{\varepsilon \to 0} = \frac{f(\theta+\varepsilon) - (\theta-\varepsilon)}{2\varepsilon} \tag{11}$$

当 ε 越小时，结果越接近真实的导数，也就是梯度值。可以使用这种方法来判断反向传播进行梯度下降时，是否出现了错误。

1.6.2 梯度检验的实施

连接参数

将参数 $W^{[1]}$ 和 $b^{[1]}$ ..... $W^{[L]}$ 和 $b^{[L]}$ 全部连接出来，成为一个巨型向量 θ。这样：
$J(W^{[1]}, b^{[1]}, \ldots, W^{[L]}, b^{[L]}) = J(\theta)$。
同时，对 $dW^{[1]}$ 和 $db^{[1]}$ ..... $dW^{[L]}$ 和 $db^{[L]}$ 执行同样的操作得到巨型向量 $d\theta$，它和 θ 有同样的维度。



现在，我们需要找到 $d\theta$ 和代价函数 J 的梯度的关系。

进行梯度检验

对成本函数的每个参数 $\theta_{[i]}$ 加入一个很小的 ϵ，求得一个梯度逼近值$d\theta_{approx[i]}$

$$d\theta_{approx}[i] = \frac{J(\theta_1, \theta_2, \ldots, \theta_i + \varepsilon, \ldots) - J(\theta_1, \theta_2, \ldots, \theta_i - \varepsilon, \ldots)}{2\varepsilon} \tag{12}$$
$$\approx d\theta[i] = \frac{\partial J}{\partial \theta_i}$$

以解析方式求得 $J'(\theta)$ 在 θ 时的梯度值 dθ, 进而再求得它们之间的欧几里得距离：

$$\frac{||d\theta_{approx} - d\theta||_2}{||d\theta_{approx}||_2 + ||d\theta||_2} \tag{13}$$

检验反向传播的实施是否正确。其中，

$$||x||_2 = \sum_{i=1}^{N} |x_i|^2 \tag{14}$$

表示向量 x 的 2-范数（也称“欧几里德范数”）。

当计算的距离结果与ϵ的值相近时，即可认为这个梯度值计算正确，否则就需要返回去检查代码中是否存在bug。

1.6.3 注意事项

- 不要在训练中使用梯度检验，它只用于调试（debug）。使用完毕关闭梯度检验的功能
- 如果算法的梯度检验失败，要检查所有项，并试着找出 bug，即确定哪个 $d\theta_{approx}[i]$ 与 dθ 的值相差比较大
- 当成本函数包含正则项时，也需要带上正则项进行检验
- 梯度检验不能与 dropout 同时使用。因为每次迭代过程中，dropout 会随机消除隐藏层单元的不同子集，难以计算 dropout 在梯度下降上的成本函数 J。建议关闭 dropout，用梯度检验进行双重检查，确定在没有 dropout 的情况下算法正确，然后打开 dropout
- 在随机初始化过程中，运行梯度检验，然后再训练网络，w 和 b 会有一段时间远离0，如果随机初始化值比较小，反复训练网络之后，再重新运行梯度检验。这是比较微妙的一点，现实中几乎不会出现这种情况。