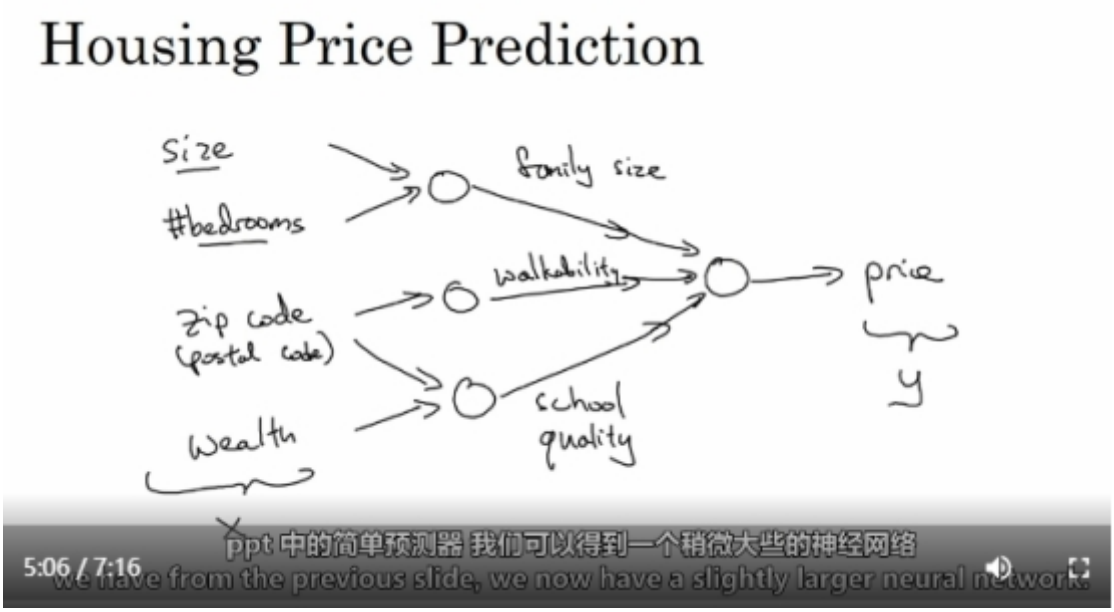
吴恩达视频笔记

一、神经网络与深度学习

1.深度学习介绍

1.1 神经网络介绍

神经网络是一种计算模型，由大量的节点(或神经元)直接相互关联而构成；每个节点(除输入节点外)代表一种特定的输出函数(或者认为是运算)，称为激励函数；每两个节点的连接都代表该信号在传输中所占的比重(即认为该节点的"记忆值"被传递下去的比重)，称为权重；网络的输出由于激励函数和权重的不同而不同，是对于某种函数的逼近或是对映射关系的近似描述；



1.2 监督学习

利用一组已知类别的样本调整分类器的参数，使其达到所要求性能的过程，也称为监督训练或有教师学习。

1.3 数据分类

①结构化数据：数据以固定格式存在，简单的说是指可以使用关系数据库表示和存储，即二维形式的数据，一般特点是：数据以行为单位，一行数据表示一个实体的信息，每一行数据的属性是相同的。

①结构化数据
需要事先预定义好属性，假设只有姓名，年龄，性别，工号四个属性信息，那么结构化数据表示为：

	姓名	年龄	性别	工号
员工一	张三	29	男	123456
员工二	李四	30	女	654321
员工三	王五	28	男	888888

②半结构化数据：半结构化数据是结构化数据的一种形式，虽然有结构，但它与关系型数据库的数据模型结构不同，不方便模式化（表示数据时有伸缩性），属于同一类实体可以有不同的属性，即使他们被组合在一起，这些属性的顺序并不重要。最典型的为JSON数据。

②半结构化数据

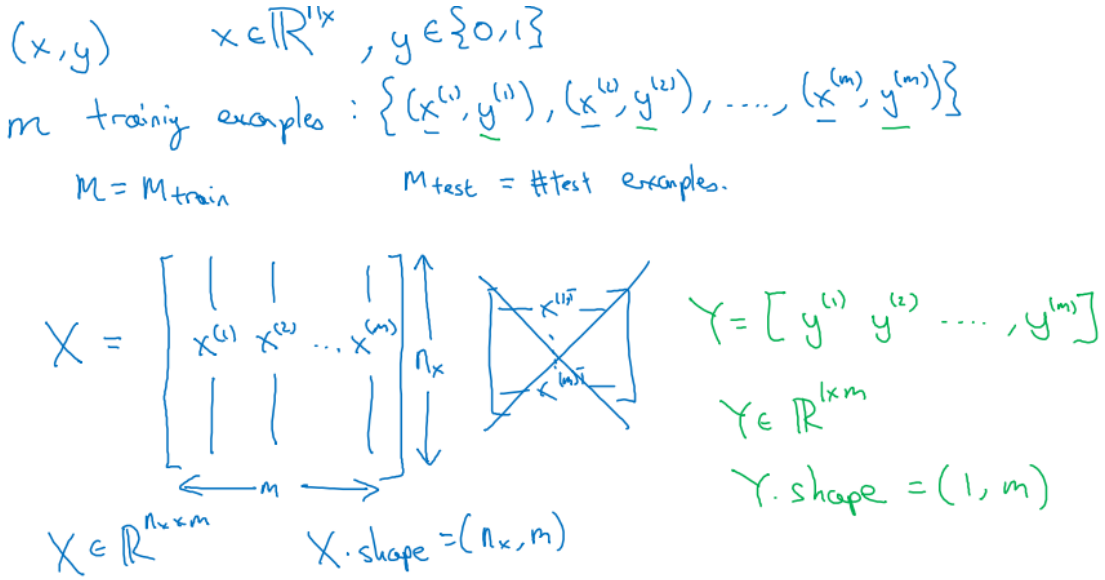
可能有时候要存放员工的简历，而员工的信息大不相同，有些员工的简历简单，只包括教育经历等等，而有些员工的简历丰富，不仅包括上述的情况，可能还包括工作经历、党籍情况、外语水平等等、掌握技能等等，可能还有一些我们不知道的情况。

③非结构化数据：非结构化数据是指数据没有一个预先定义好的组织方式，一般指文字型数据，其中也会包含数字等信息 。

2.神经网络基础

2.1 二元分类

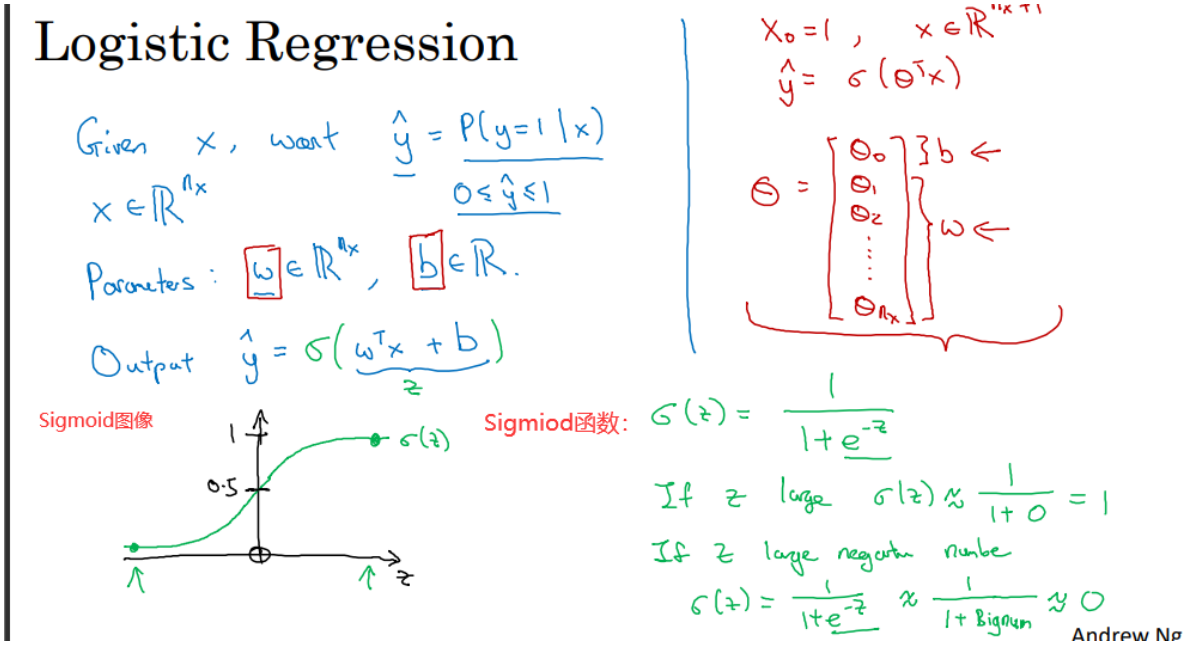
二元分类又称逻辑回归，是将一组样本划分到两个不同类别的分类方式。 在具体编程实现过程中，通常得到的输出矩阵为[n(x),m]维矩阵，n(x)为训练的样本个数，m为分类数；标签矩阵为[1,m]大小。



2.2 逻辑回归

逻辑回归也称作Logistic回归分析，是一种广义的线性回归分析模型，属于机器学习中的监督学习。其推导过程与计算方式类似于回归的过程，但实际上主要是用来解决二分类问题（也可以解决多分类问题）。

我们通过公式计算得出的数值，可能会超过0-1的范围，而当我们处理二分类问题时，通常希望输出的表示一个概率，例如：图片为猫的概率为0-1之间的某个数。因此通过引入sigmoid函数来将结果转换到0-1之间的数：



2.3 逻辑回归损失函数

损失函数（Loss Function ）是定义在单个样本上的，算的是一个样本的误差。

代价函数（Cost Function ）是定义在整个训练集上的，是所有样本误差的平均，也就是损失函数的平均。

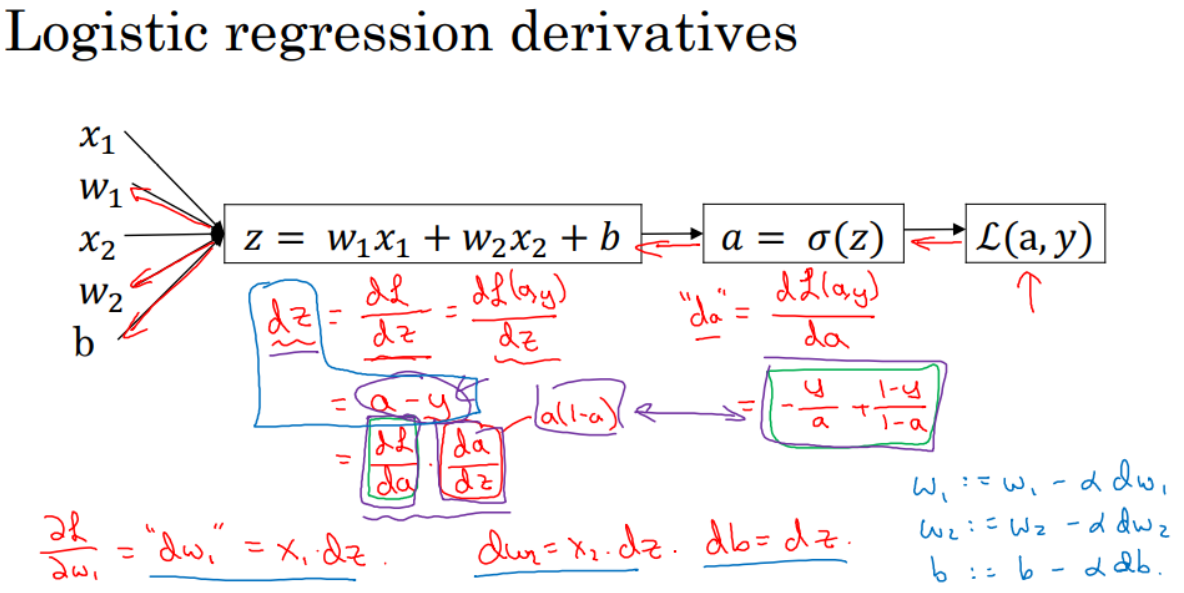
2.4 梯度下降

梯度下降， 拆解为梯度+下降，那么梯度可以理解为导数（对于多维可以理解为偏导），那么合起来变成了：导数下降。梯度下降就是用来求损失函数最小值时自变量对应取值。一般过程如下：

- 第一步，明确自己现在所处的位置(初始化自变量w，b)
- 第二步，找到相对于该位置而言下降最快的方向(通过求导的方式)
- 第三步， 沿着第二步找到的方向走一小步，到达一个新的位置，此时的位置肯定比原来低(通过学习率来控制步长)
- 第四部， 回到第一步
- 第五步，终止于最低点

2.5 逻辑回归梯度下降

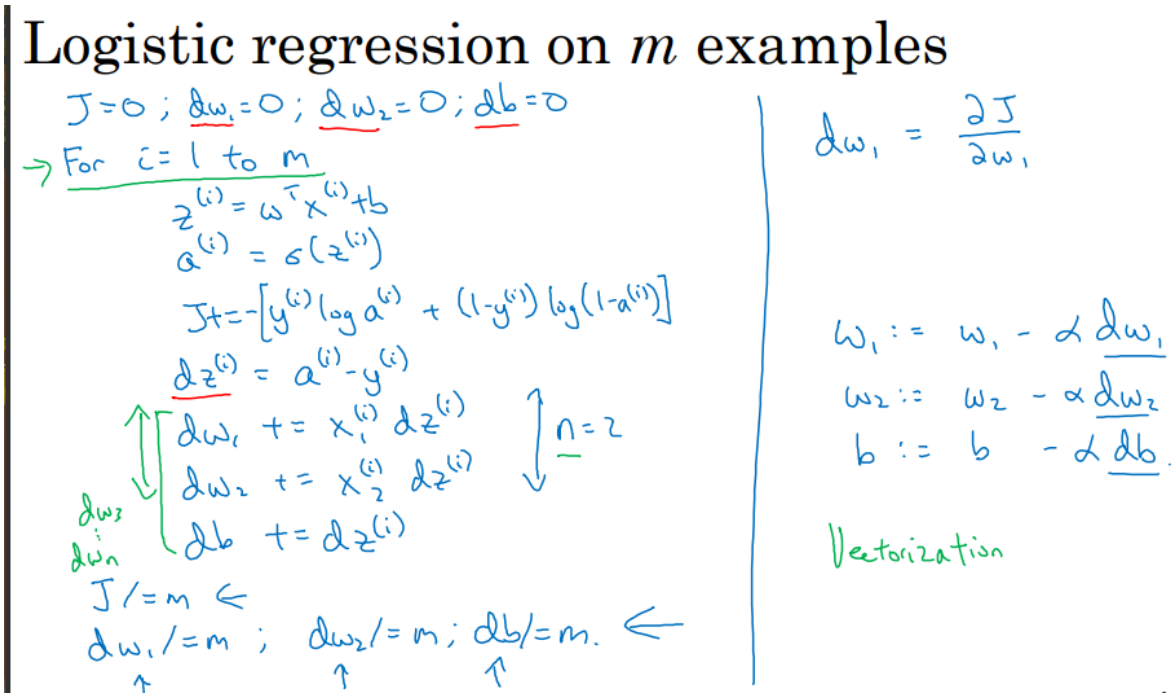
前向传播过程中，首先通过输入，结合w和b参数计算出z，z通过sigmoid等激活函数输出a，然后根据a以及真实值y计算出损失函数。



反向传播求导过程如红色所示，先求出da，再求出dz，然后可以选择求关于w和b的偏导，来判断出参数对最终结果的影响。

2.6 m示例上的梯度下降

在有m个样本的情况下，我们需要先计算出这些量的总和，然后处于样本个数计算平均值



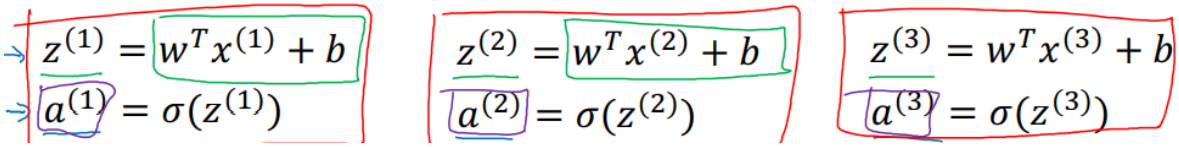
课程中提到，在该方式中，存在**两个循环**，一个是m个样本的循环，一个是特征个数w的循环。而**循环会导致代码运行速度减慢**，在深度学习时代便不能够运行在数据集庞大的情况下，因此提出了**矢量化**(向量化)的方式。

3.Python和向量化

尽量通过Python和numpy的内置函数，来替代显式的for循环运算，计算速度会大幅度增长。

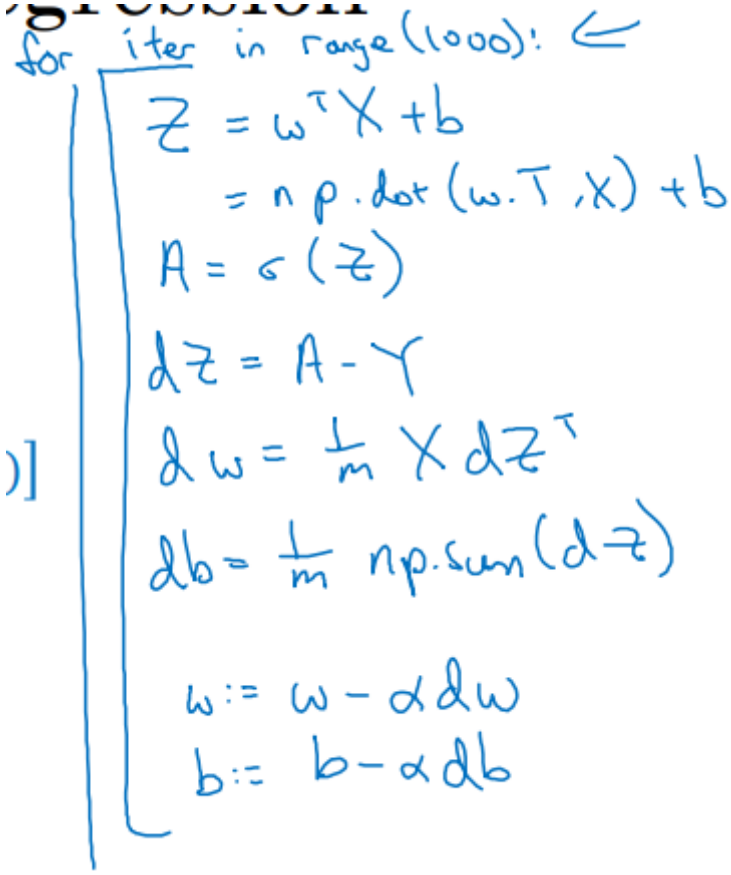
3.1 向量化逻辑回归

在计算前向传播时：



一批样本的z和a，可以通过z=np.dot(W.T,X)+b来进行计算，得出z后通过激活函数后便可得到矩阵A。

整体计算过程如下所示：



3.2 Python中的广播

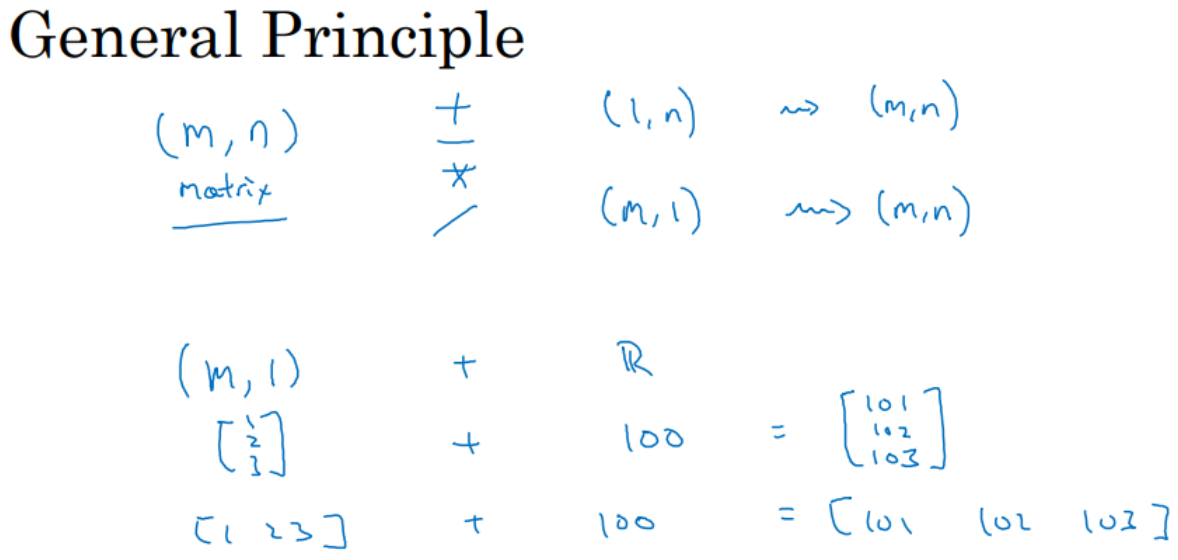
小知识点：

①axis=0表示垂直求和，axis=1表示水平求和。

②reshape是一个时间复杂度为O(1)的方法，因此并不会占用很多的程序运行时间。

Python中的广播，意在对 缺失维度进行补充，可以减小代码量 。 两个数组的后缘维度相同，或者在其中一方的维度为1。广播在缺失或者长度为1的维度上补充。

例如[m,n]矩阵加上[1,n]矩阵，那么[1,n]矩阵会**水平**地复制自己，将第一行复制成m行，达到[m,n]的效果；再与[m,1]矩阵进行操作时，后者则会复制n次。



3.3 Python中的向量注释

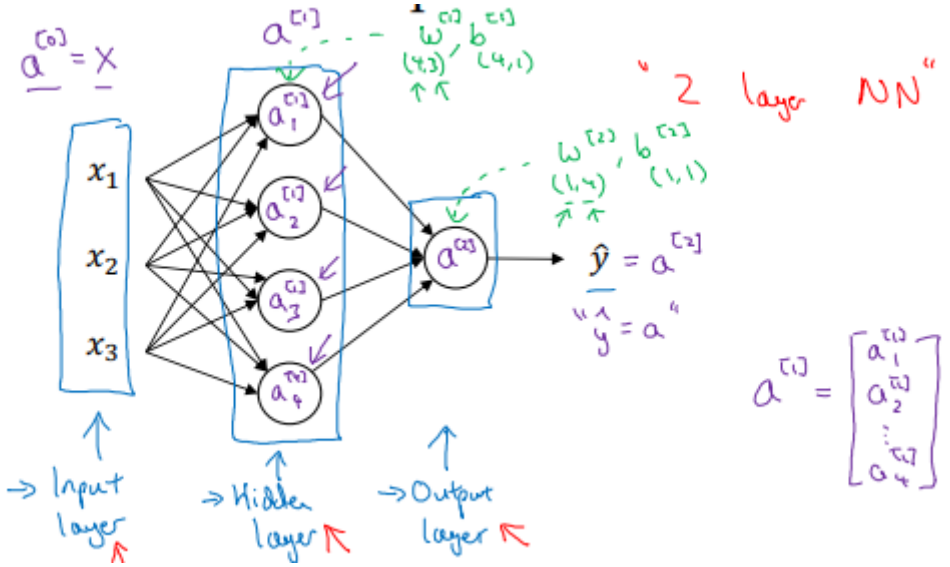
在课程中，举了np.random.randn(5)的例子，这段语句会产生(5,)格式的矩阵，而且转置以及内积的运算结果都与我们所认知的结果有些许不同：

建议不要使用形如(5,)(n,)这样秩为1的数组，而是通过np.random.randn(5，1)生成一个真正意义上的5*1的矩阵。对于秩为1的矩阵，可以通过reshape方法来进行格式的转化，同时通过设置断言assert语句来进行及时的检查。

4.浅层神经网络

4.1 神经网络的表现形式

神经网络主要由输入层、隐含层、输出层来构成，一个简单的神经网络如图所示：

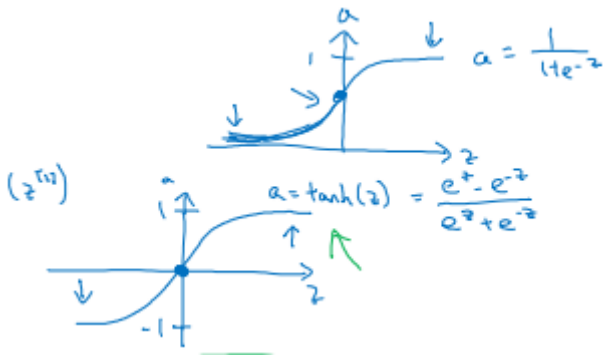


隐含层的含义主要是指： 隐藏层不直接接受外界的信号,也不直接向外界发送信号。隐藏层在神经网络中相当于一个盒子。并且神经网络通常从隐含层开始对网络层数进行计数，将输入层看作是第0层。

4.2 激活函数

sigmoid和tanh：

sigmoid函数比tanh函数收敛饱和速度慢，函数梯度收敛更为平滑，函数值域范围更窄。而tanh的平均值为0，更适用于神经网络。这两种激活函数的缺点为，如果z很大或者很小，函数的斜率将会很小，接近于0，这将会减缓梯度下降的速度。



整流线性单元函数relu：

relu 是一个分段线性函数，如果输入为正，它将直接输出，否则，它将输出为零。它已经成为许多类型神经网络的默认激活函数，因为使用它的模型更容易训练，并且通常能够获得更好的性能。

缺点为，当z < 0时，导数为0，因此提出了leaky relu函数。

优点为，对于大部分的z，激活函数的斜率不会为0，且训练速度更快

激活函数选择规则：

①输出为0，1，在使用二元分类任务时，选择sigmoid函数

②除此之外的其他情况，大多使用relu函数(默认函数)

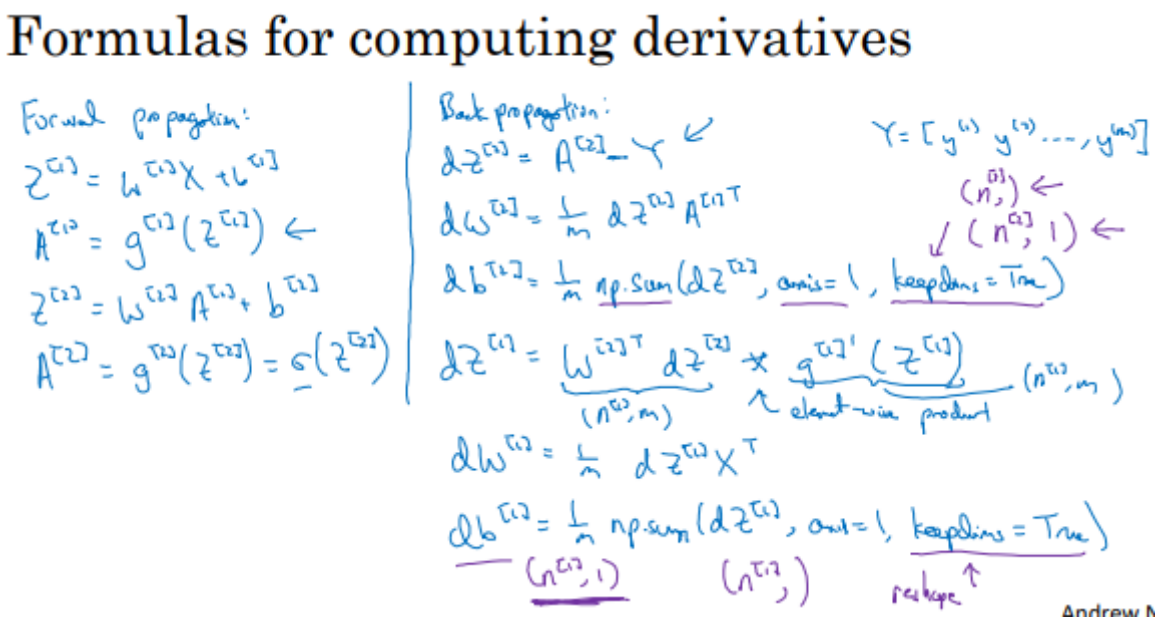
4.3 为什么需要非线性激活函数

如果不使用激活函数，那么神经网络的输出，就仅仅是输入函数的线性变化。在多层的神经网络中，如果这样，那么它所做的就仅仅是计算一个线性激活函数，隐含层便没有意义。因为线性关系无论怎么组合，结果还是线性函数。

可能使用线性激活函数的位置在于输出层，而在隐含层除过数据压缩外，使用线性激活函数都是极其罕见的。

4.4 神经网络的梯度下降

为了训练神经网络中的种种参数，需要进行梯度下降；在编程中，设置keepdims=True，来防止产生[n,]结构的数组。所利用的公式如下：



4.5 随机初始化

在对参数进行初始化时，若全设置为0，则隐藏神经元就是“对称”的，所以无论进行多少次迭代，这两个隐藏单元仍然在使用同样的功能进行计算。

在实际编程过程中，通过np.random.randn((m,n))*0.01，便可以对一个[m,n]的矩阵进行初始化。并且乘以0.01是因为习惯于用非常非常小额度随机初始值，大的数，在运用sigmoid或者tanh时，会来到图像的“平坦”部分，导致梯度下降十分缓慢，学习进度缓慢。

5.深层神经网络

5.1 神经网络前向传播-矩阵维数

在检查神经网络L层的维数是否正确时，通常使用公式：

参数w的矩阵维度：[nL,n(L-1)]

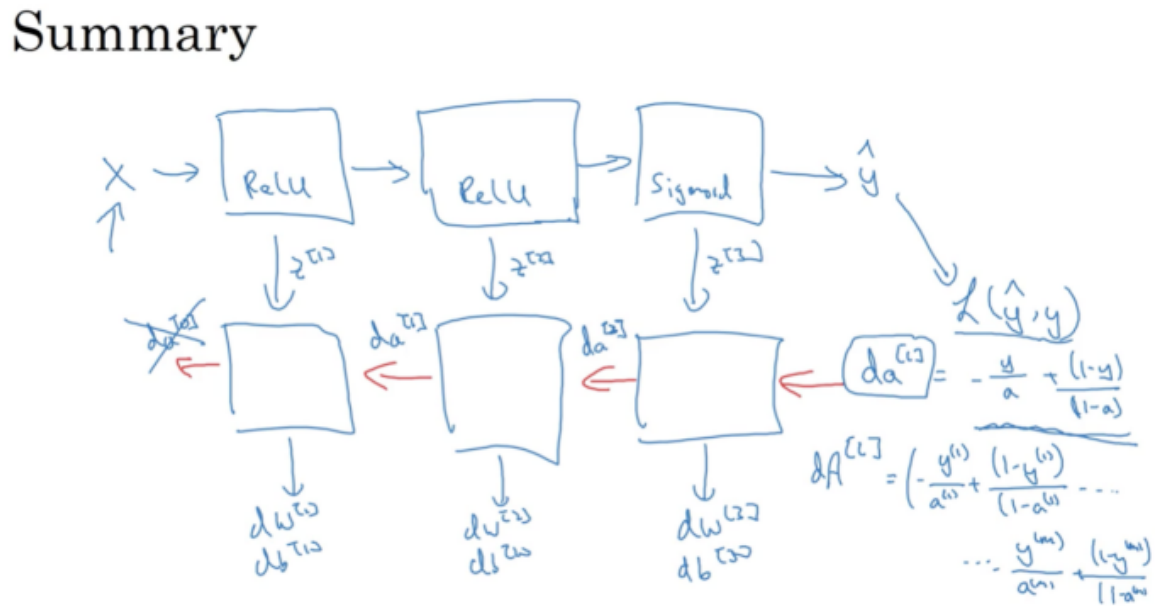
参数b的矩阵维度：[nl,1]

反向传播时，dw和w矩阵的维度相同，db和b矩阵的维度相同。

而在多样本同时进行运算时，第一层网络的输出z1的维度为[n1,m]，且m的数量为我们训练集的大小。也就是说，参数矩阵的格式不会发生变化，但是z和a的矩阵的**列数，从1变为训练集大小m。**

5.2 模块构建以及正反向传播

输入x，经过数层神经网络，且在每一层都保留z[i]，最后输出y^，通过y^计算出损失函数L，再由公式计算出dA，以及每一层的dw和db。



将每一层的z[i]保留下来对计算反向传播十分方便。

5.3 参数和超参数

诸如学习率，迭代次数，隐含层结点个数，隐含层层数，激活函数的选择，这些人为设置的参数最终也会对W和b带来一定的影响。并且在不同的任务场景中，超参数的设置也有不同的规律。

二、改进深度神经网络

1.深度学习的实践领域

1.1 训练/开发/测试集

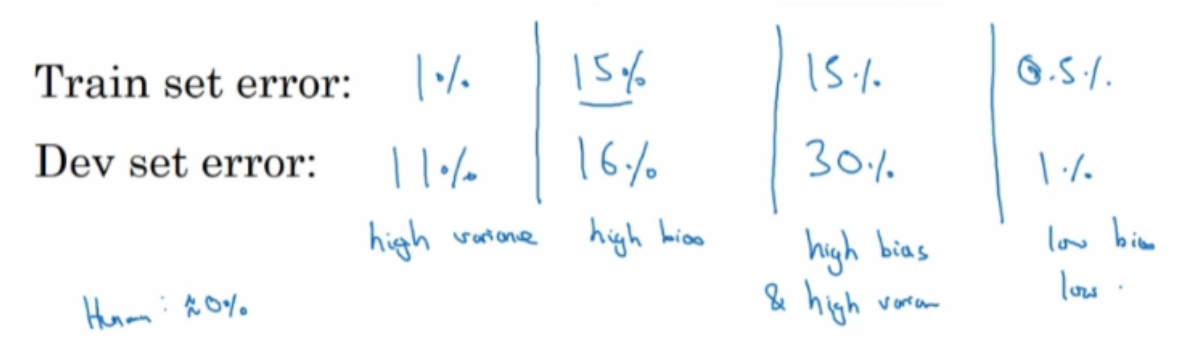
合理地设置这些数据集，能够使我们的网络迭代效率更高。

训练集，顾名思义它是用来训练模型的，为了减少泛化误差，我们需要通过训练集不断的训练来使得我们的模型能够更好的接近真实数据。 测试集，用来测试模型的准确性，我们将测试集应用于训练集训练好的模型，会得到一个模型的得分，验证集主要提供**无偏估计**，查看模型训练的效果是否朝着坏的方向进行。**验证集的作用是体现在训练的过程。**通过查看训练集和验证集的损失值随着epoch的变化关系可以看出模型是否过拟合，如果是可以及时停止训练，然后根据情况调整模型结构和超参数，大大节省时间。

在分割时，如果数据集比较小，就可以采用传统的60/20/20分割比例；而当数据集比较大时，可以将验证集和测试集的数量设置的远远小于20%。并且要确保开发集和测试集中的**数据分布**相同。

1.2 偏差和方差

在课程中，举例如下：



当基误差非常低而且训练集和验证集同分布时：

①训练集结果好，验证集结果差：高方差(过拟合)

②训练集结果差，验证集训练集相仿：高偏差

③训练集结果差，验证集结果差：高偏差，高方差

④训练集结果好，验证集训练集相仿：低偏差，低方差

1.3 机器学习的基本原则

当训练完毕后：

①首先判断网络是否有高偏差，若有，则考虑通过加深网络层数，更换优化器，或者是更换网络结构来进行解决，直到网络能够很好地拟合训练集

②解决高偏差后，解决高方差问题，通过获得更多的数据，正则化，或者是更换网络结构的方法

1.4 正则化

为了解决高方差，即过拟合问题，引入了正则化，在模型中加入一个惩罚项，惩罚模型的复杂度，防止过拟合。逻辑回归中的正则化：

- 逻辑回归的损失函数中增加L1正则化

$$J(w,b)=\frac{1}{m}\sum_{i=1}^{m}L(\hat{y}^{(i)},y^{(i)})+\frac{\lambda}{2m}||w||_1$$

L1正则化会使w变得稀疏，

- 逻辑回归的损失函数中增加L2正则化

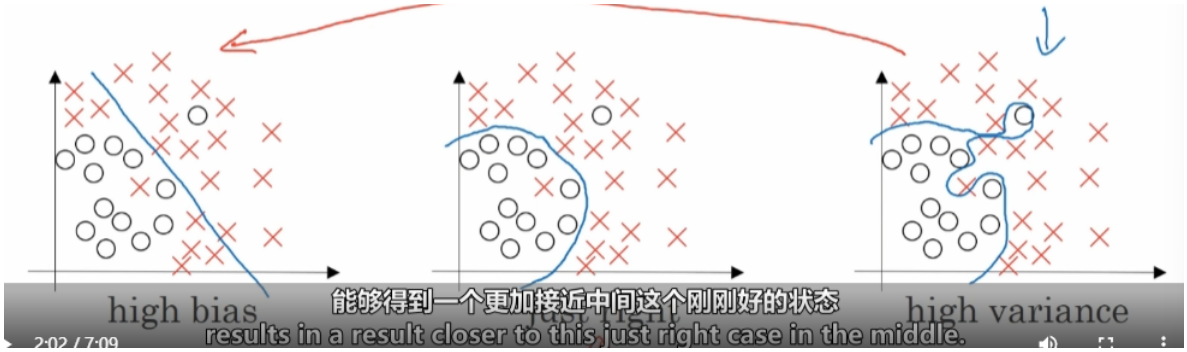
$$J(w,b)=\frac{1}{m}\sum_{i=1}^{m}L(\hat{y}^{(i)},y^{(i)})+\frac{\lambda}{2m}||w||_2^2$$

而L2正则化(梯度缩减)则使用的更多，且lambda作为正则化参数，也是需要人为调整的超参数。这里矩阵的范数，也被称为佛罗贝尼乌斯范数，即**所有w参数的平方和的结果。**

其中只对参数w进行正则化是因为w是一个非常高维的参数矢量，而b只是单个数字，是众多参数中的一个。

1.5 正则化解决过拟合原理

通过不断加深网络的规模和深度，可以使网络从高偏差->正常->高误差的方式进行转变：



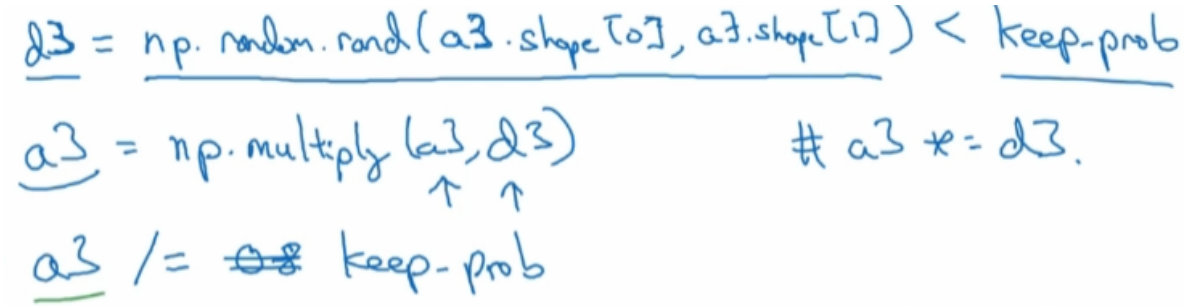
正则化的原理如下：

①通过调整正则化参数->使权重矩阵的数值减少->隐藏层节点的影响减小->“休眠”节点->网络变简单->拟合能力变弱

②加入正则化使z变小->使激活函数向简单的线性函数转变->拟合能力变差，不能拟合复杂函数

1.6 随机失活正则化

通过设置概率的方式，能够以“掷硬币”的方式，概率地将一些隐含层节点的矢量d设置为0，即不起作用。并且在实际运行过程中，针对每一个样本，网络都会随机地抛弃不同的节点，因此不会一直丢弃相同的隐藏单元。

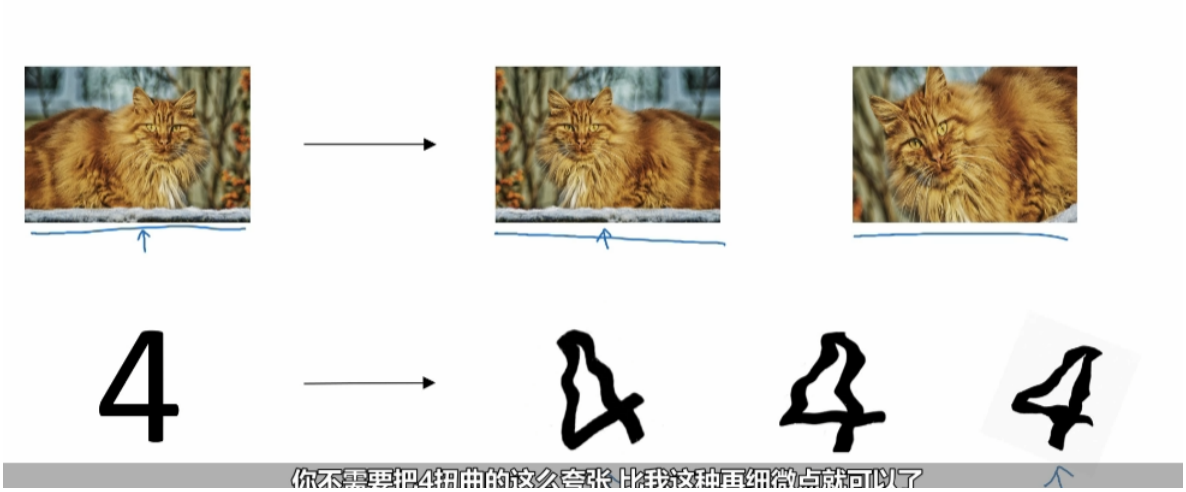


1.7 理解随机失活

在实际使用过程中，对整个网络，可以对不同的层设置不同的dropout残留值，对参数多的层，抛弃更多的节点，而对参数少的层，抛弃少的活着不进行抛弃。同时，采用dropout之后，因为每一次迭代都会随机的失活不同的节点，因此**代价函数J不是单调递减的。**

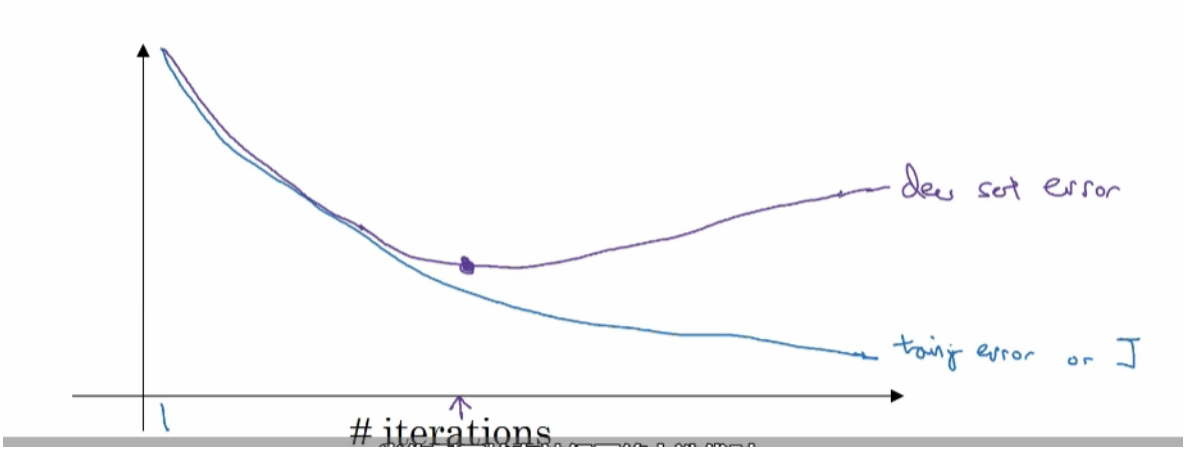
1.8 其他的正则化方法

①数据集扩充，通过对图片进行反转和旋转，以及扭曲的方式，可以得到新的图片



②早终止法

当发生过拟合时，验证集的误差总是先下降后上升，而早终止法的目的在于，将训练停止在验证集误差最小的那一次迭代：



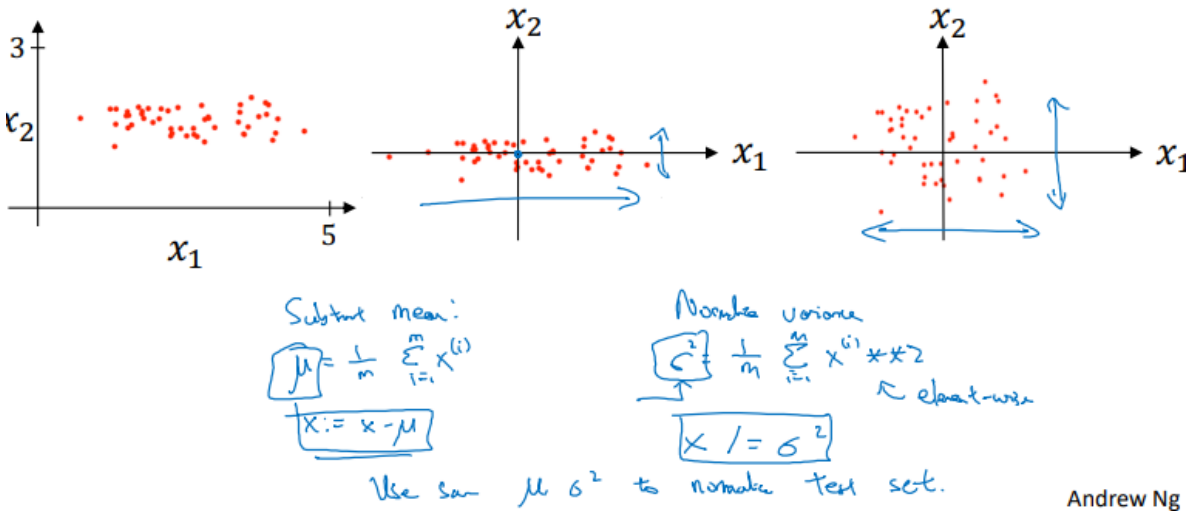
通过停在中间某处，能够得到一个不大不小，效果较好的w值。但缺点是，早终止法将①减少损失函数J②避免过拟合，这两个任务交叉在了一起，在实际编程过程中，这种一个工具解决两种问题的方式通常会使问题变得更加复杂。

1.9 归一化输入

归一化操作通常包括两个步骤：

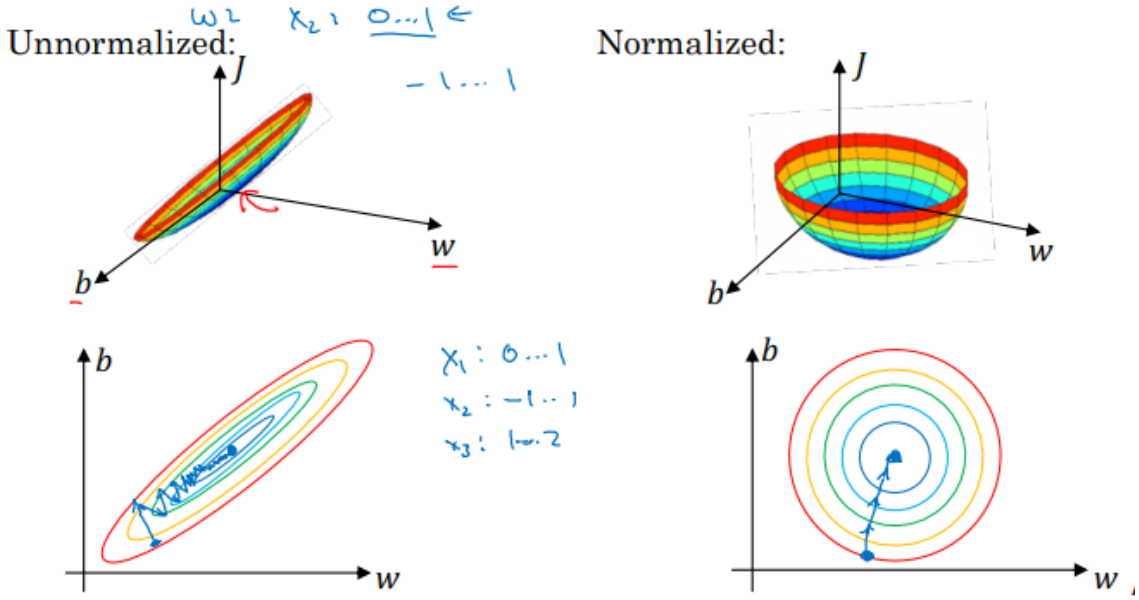
①减去均值，即均值归零

②对方差进行归一化



注意，要对训练集和测试集用一样的方式进行归一化，使用一样的μ和 σ 的平方。

当数据维度不同时，归一化前后的函数图像如下所示：

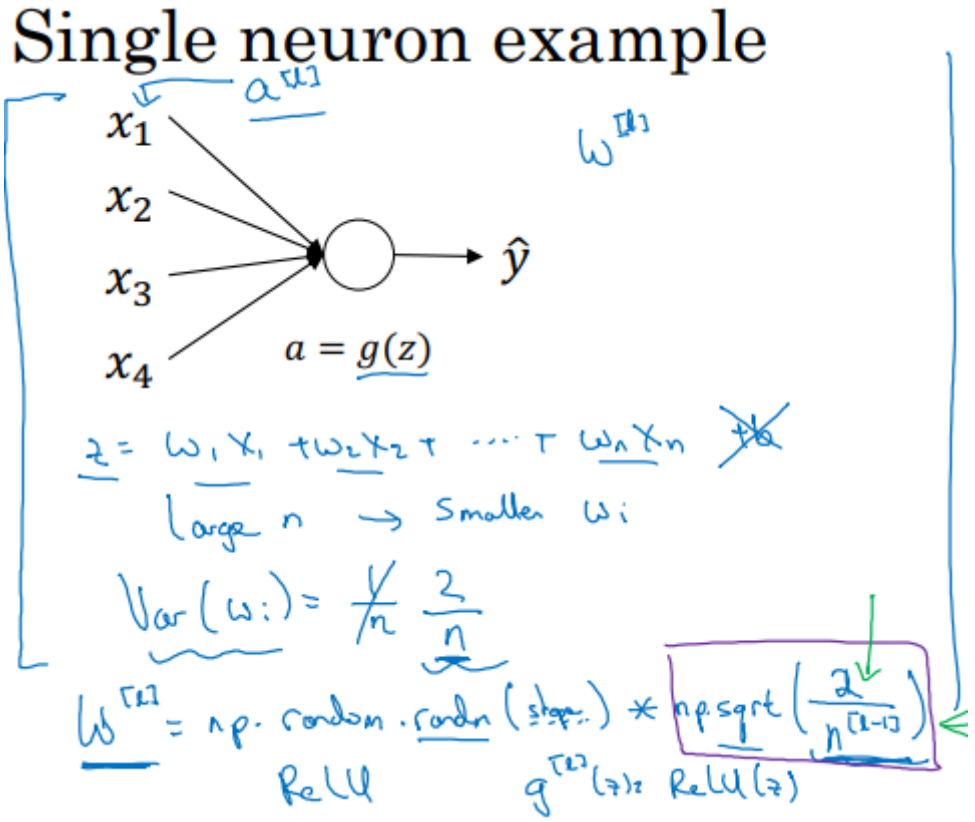


若不进行归一化，则在如此扁平的图上进行梯度下降，需要很小的学习率并且反复辗转，归一化后则可以采用更大的学习率来进行学习，效率更高。

1.10 梯度消失和爆炸

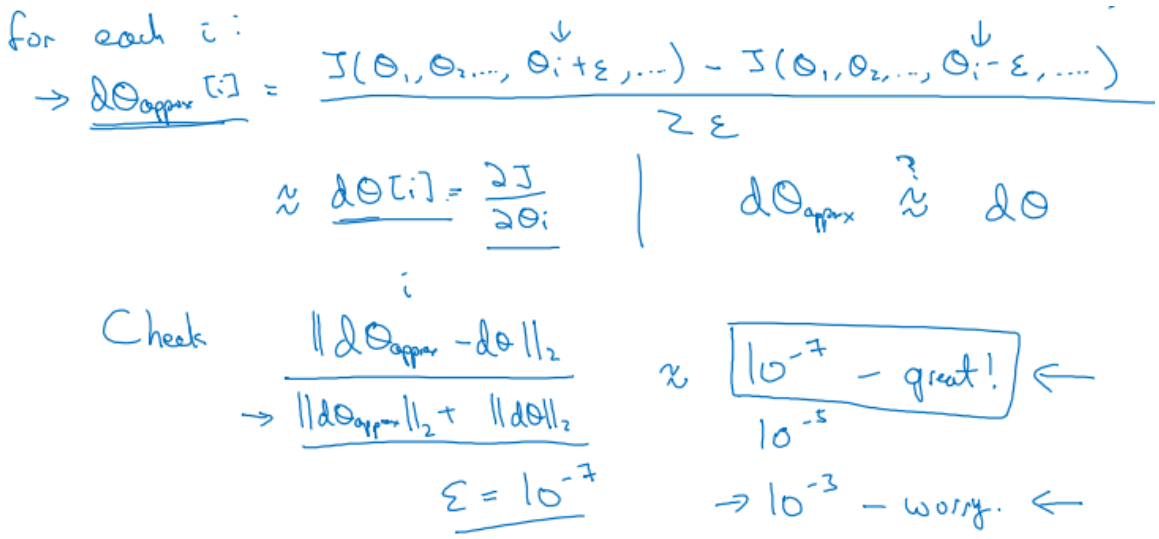
指在深层网络训练过程中，梯度变得非常小或者非常大的问题，可以通过深度网络权值初始化来进行部分解决。

由于节点的输入是多个wi*xi，所以在乘法与加法的作用下，会导致节点的输入变得很大，因此通过将权重矩阵缩小的方式，来减轻梯度爆炸和消失。



1.11 梯度检验

为了确认代码中反向传播计算的梯度是否正确，可以采用梯度检验（gradient check）的方法。通过计算数值梯度，得到梯度的近似值，然后和反向传播得到的梯度进行比较，若两者相差很小的话则证明反向传播的代码是正确无误的。 并且，双边检测误差更低，精度更高。



为了执行梯度检验，我们首先要把所有参数转化成一个巨大的向量数据。 然后执行双边误差检测， 这个双边误差应该近似于dθ。 且使用二范数来进行度量。最后我们将向量长度做归一化处理，得到上图中最下面的公式。如果我们得到的这个误差在10的负7次方的数量级，那么非常好，如果这个误差比较大，那么我们就要重新检验了。

在执行梯度检验时，要注意：

①仅在调试时使用

②查找相近的组成部分来查找bug

③不要忘记正则化对代价函数J的改变

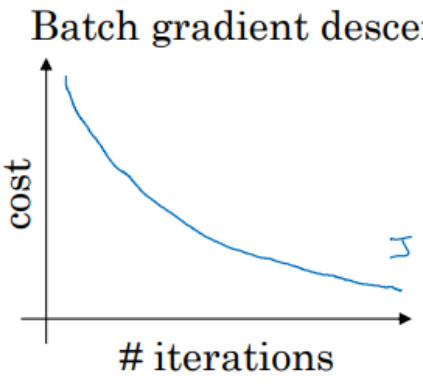
④在随机初始化时，即w和b有很多值为0时，运行梯度检验；在运行一段时间后，再次进行梯度检验

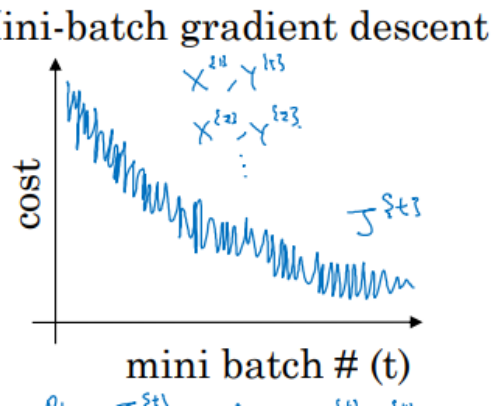
2.优化算法

2.1 小批量梯度下降

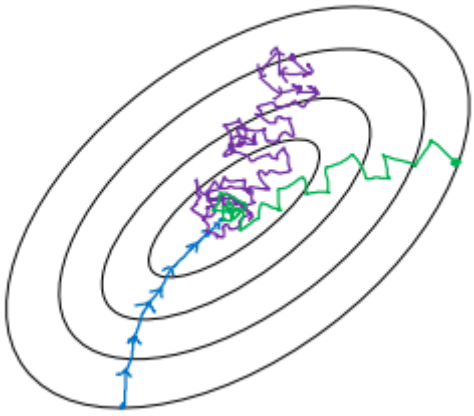
为了加快训练速度，将大数据集拆分为小数据集，每次只处理一个小批量数据集。这样小批量梯度下降对训练集的一轮处理，可以得到**小批量数据集数目的梯度逼近。**

与dropout算法类似，使用小批量梯度下降，就好像每次迭代都使用了不同的训练集，其代价函数J也不是单调递减的：





根据mini-batch size的大小不同，最大值取全样本数目则就成为批量梯度下降；最小值取1则就成为随机梯度下降。批量梯度下降的噪声小且步长较大，而随机梯度下降噪声很大且不会收敛到最低点，且会失去使用向量加速运算的机会。而小批量梯度下降可以兼顾这两种算法的优点。



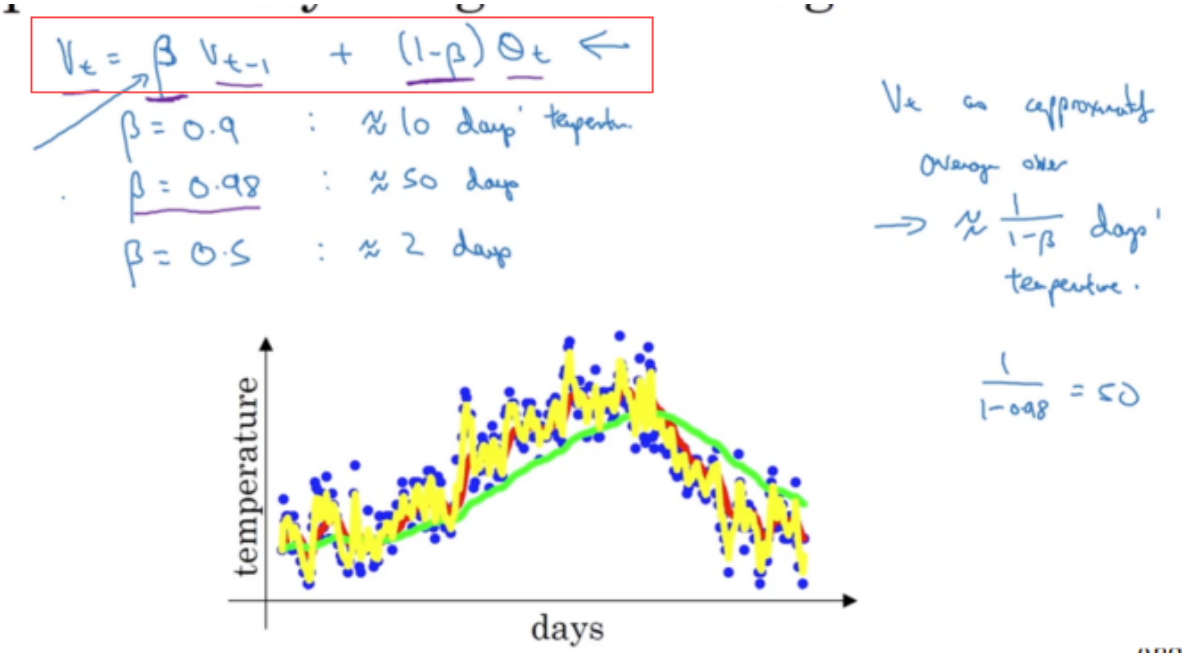
算法选择规则：

①如果训练集较小(<2000)，则使用批量梯度下降算法

②大数据集，则一般选择64-512作为mini-batch大小，**设置为2的幂数则代码运行更快**

2.2 指数加权平均

通过红色框中的公式，可以实现指数加权平均：



并且通过不同的β值，可以调整加权平均的窗口大小，用**1/(1-β)**公式可以计算出窗口大小。且窗口越大，则曲线越平滑，窗口越小，曲线越曲折但同时可以更灵敏地反应数据变化。

在学习的初始阶段，由于V0的原因，导致起初几个预测值远低于实际值，因此引入了偏差修正

$$V_0 = 0$$
$$V_1 = 0.98 V_0 + 0.02 \theta_1$$
$$V_2 = 0.98 V_1 + 0.02 \theta_2$$
$$= 0.98 \times 0.02 \times \theta_1 + 0.02 \theta_2$$
$$= 0.0196 \theta_1 + 0.02 \theta_2$$

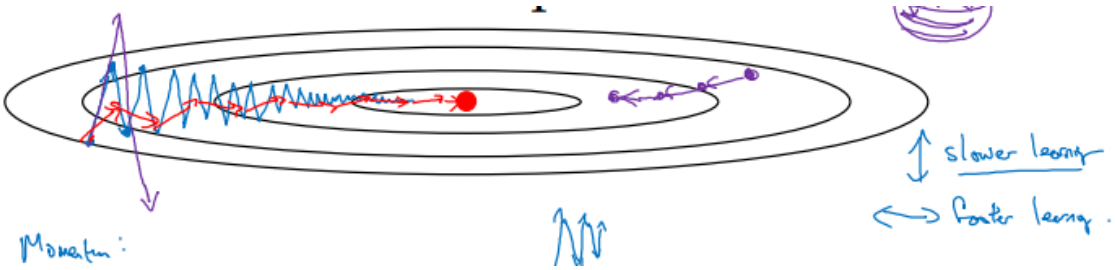
采用偏差修正公式对学习过程中的前几项进行修正：

$$V_i' = \frac{V_i}{1 - \beta^i}$$

2.3 动量梯度下降

通过计算梯度的指数加权平均，然后使用这个梯度来更新权重。

在通常的训练过程中，梯度震荡向最小值逼近，但是无法采用大步长，因为很可能震荡出去：



而使用动量梯度下降，可以减少在垂直方向上的震荡，且在水平方向上移动的更快

2.4 RMSprop

全称为均方根传递，在出现震荡的维度里，会计算出一个更大的和值(导数的加权平均)，最后抑制了出现震荡的方向。

$$S_{dW} = \beta_2 S_{dW} + (1-\beta_2) dW^2 \leftarrow \text{small}$$
$$S_{db} = \beta_2 S_{db} + (1-\beta_2) db^2 \leftarrow \text{large}$$
$$W := W - \alpha \frac{dW}{\sqrt{S_{dW}} + \varepsilon} \qquad b := b - \alpha \frac{db}{\sqrt{S_{db}} + \varepsilon}$$
$$\varepsilon = 10^{-8}$$

并且为了避免除以0的可能性发生，在实际编程中要在分母加上一个非常非常小的值。

2.5 Adam优化算法

将指数加权平均与RMSprop算法结合起来，同时需要对数值进行偏差修正，最后的结果展示为，将指数加权平均算法中的Vdw处于RMSprop中的偏差修正SdW的平方根+极小值。红色框中即为最终公式：

$V_{dw}=0$, $S_{dw}=0$. $V_{db}=0$, $S_{db}=0$

On iteration t:

Compute dw, db using current mini-batch

$V_{dw}=\beta_1 V_{dw}+(1-\beta_1)dw$, $V_{db}=\beta_1 V_{db}+(1-\beta_1)db$ ← "momentum" β_1

$S_{dw}=\beta_2 S_{dw}+(1-\beta_2)dw^2$, $S_{db}=\beta_2 S_{db}+(1-\beta_2)db$ ← "RMSprop" β_2

yhat = np.array([.9, 0.2, 0.1, .4, .9])

$V_{dw}^{corrected}=V_{dw}/(1-\beta_1^t)$, $V_{db}^{corrected}=V_{db}/(1-\beta_1^t)$

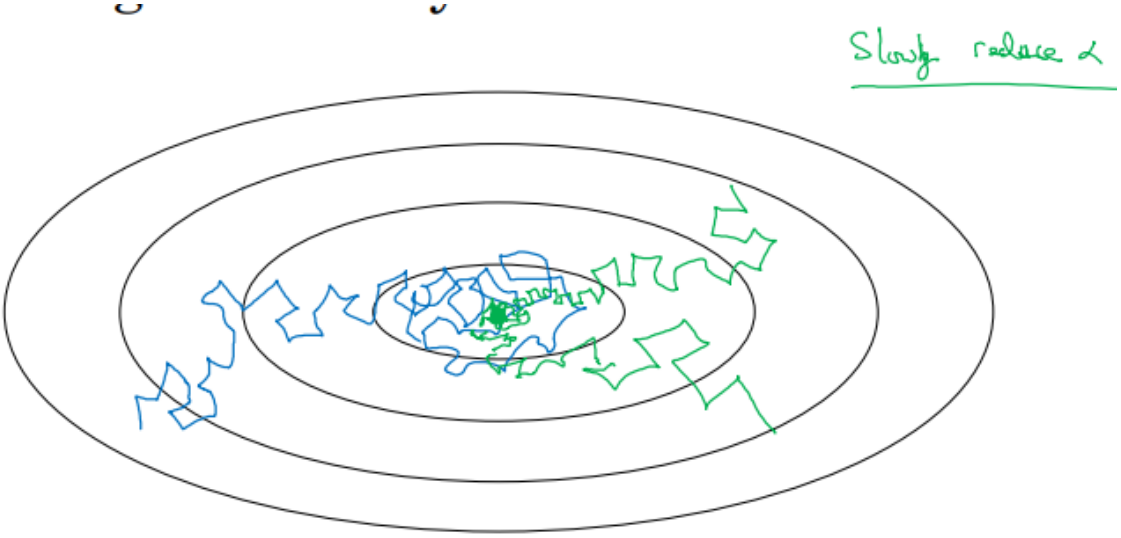
$S_{dw}^{corrected}=S_{dw}/(1-\beta_2^t)$, $S_{db}^{corrected}=S_{db}/(1-\beta_2^t)$

$$W:=W-\alpha\frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected}}+\varepsilon} \qquad b:=b-\alpha\frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}}+\varepsilon}$$

对于超参数的设置，将加权平均中的β默认设置为0.9，RMSProp中的参数β2设置为0.999，在RMSprop中的极小值设置为10^-8次方。并且在实际使用过程中，通常直接使用默认值。

2.6 学习速率衰减

在使用小批量梯度下降时，若采用固定学习率，那么最后可能会在最低点处震荡，因为每一批数据都有噪声，因此，引入学习率衰减，可以：**在初始阶段保持较高速度下降，在临近最低点时进行缓慢逼近，消除震荡。**



衰减方法：

①通过公式，设置不同的超参数进行衰减：

$$\alpha=\frac{1}{1+\text{decay-rate}*\text{epoch-num}}\alpha_0$$

$$\alpha=\frac{k}{\sqrt{\text{epoch-num}}}\cdot\alpha_0$$

②指数衰减，设置一个<1的数值，通过指数快速衰减

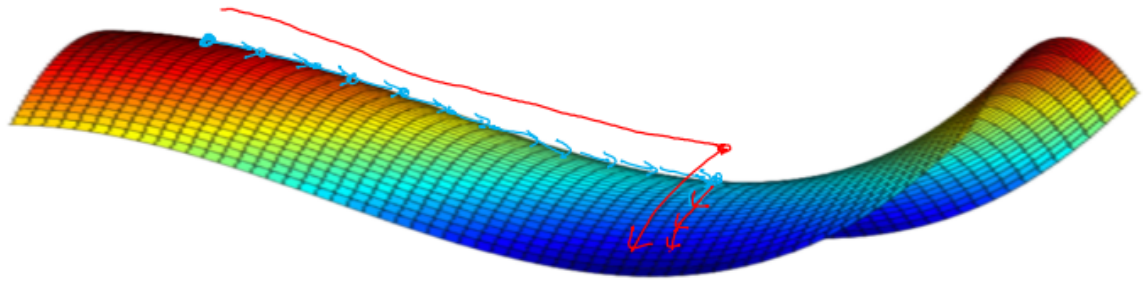
③离散式阶梯衰减

④手动衰减

2.7 局部最优解

在高维空间中，碰到鞍点的概率要比局部最优的概率大很多(因为局部最优需要多维向量都呈现凹函数的样子)。

由此而引出的概念：**停滞区**。停滞区会使网络的更新更加缓慢，在停滞区中缓慢下降，然后由于随机的数值扰动，走出停滞区：



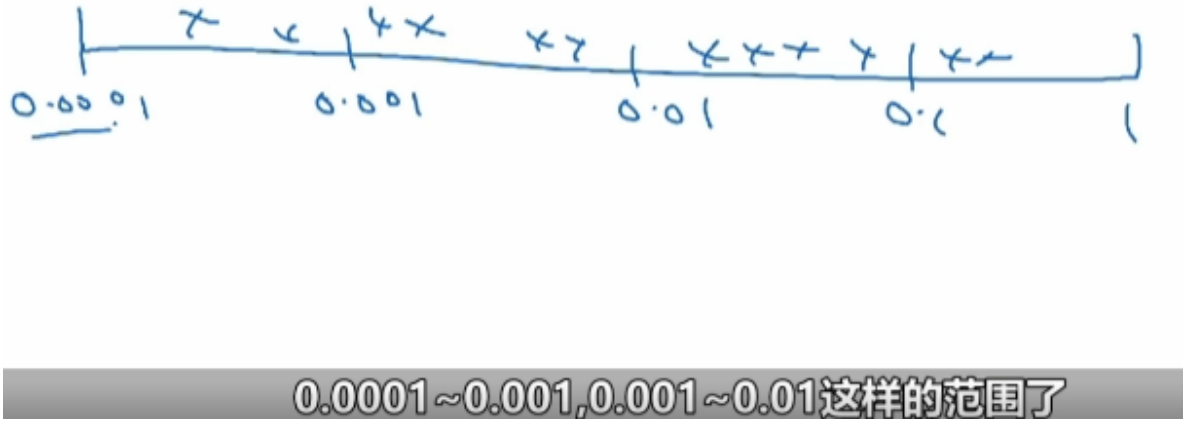
3.超参数调整、批量归一化和编程框架

3.1 超参数调整

参数重要性排序：**学习率>动量项，Mini-Batch，隐藏层单元数量>网络层数，学习率衰减**

通过在参数维度中随机取样，我们可以取得想要的随机参数组合。

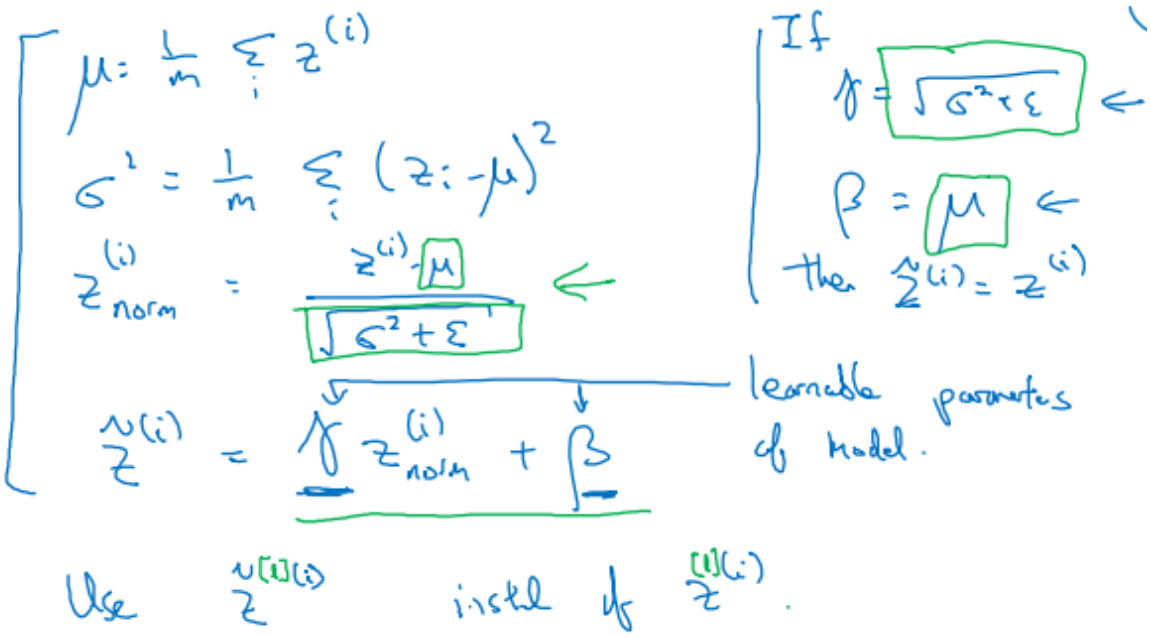
在选择超参数的具体数值时，对于从0.001到1的学习率参数，通常采用对数方式来进行搜索和选择，而不采用限线性搜索：



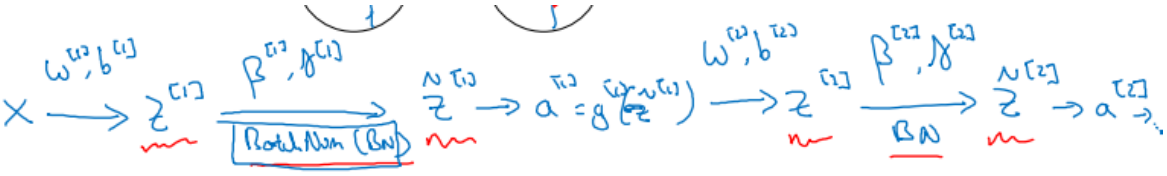
如果自己的计算资源有限，则针对单个模型的，以时间为轴的调参过程更实用；若数据量小并且计算资源充足，则可以同时并行运行多个模型来观察最后的变化。

3.2 批量归一化

batch norm批量归一化可以让网络对于超参数的设置不那么敏感，同时更容易训练，通过在隐含层上也进行归一化。并且通过如下公式(参数γ和β通过网络迭代学习)，来可控方差和均值：



在网络中，输出的z，通过BN操作后转化后，再交由下一层，且参数β通过梯度下降算法来进行优化：



归一化有效原因：

①通过批量归一化，让数据具有相同的变化范围将加速学习

②减少了隐藏单元值得分布的不稳定性，确保无论z因前面网络的何种影响而变，这一层的z的均值和方差不变；相当于，**BN算法消除了不同层数间参数的耦合**，相当于独立的层，有效增加网络运行速度。

③具有一定正则化效果，但同时应用大规模的batch size，则会削弱正则化效果。

在测试时，例如使用单个样本，这时的均值和方差则要通过训练集来进行估算μ和sigma平方，通过在训练集中，使用指数加权平均来记录μ和sigma平方值，以此为依据来进行估算。

3.4 多类别分类

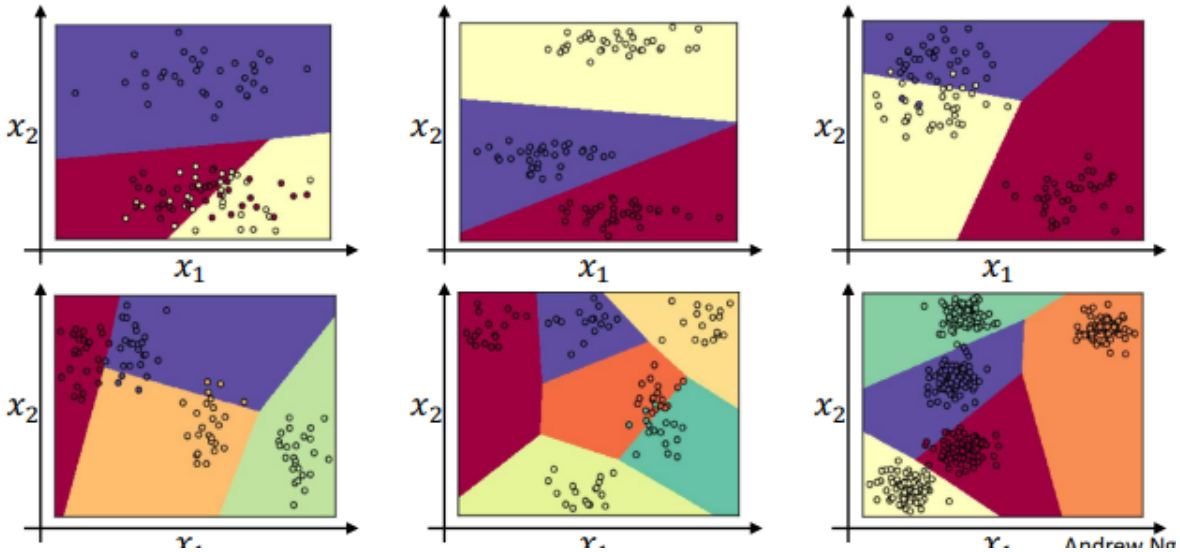
Softmax激活函数的作用过程如下：

$$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \qquad t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix}$$

$$g^{[L]}(z^{[L]}) = \begin{bmatrix} e^5/(e^5+e^2+e^{-1}+e^3) \\ e^2/(e^5+e^2+e^{-1}+e^3) \\ e^{-1}/(e^5+e^2+e^{-1}+e^3) \\ e^3/(e^5+e^2+e^{-1}+e^3) \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$$

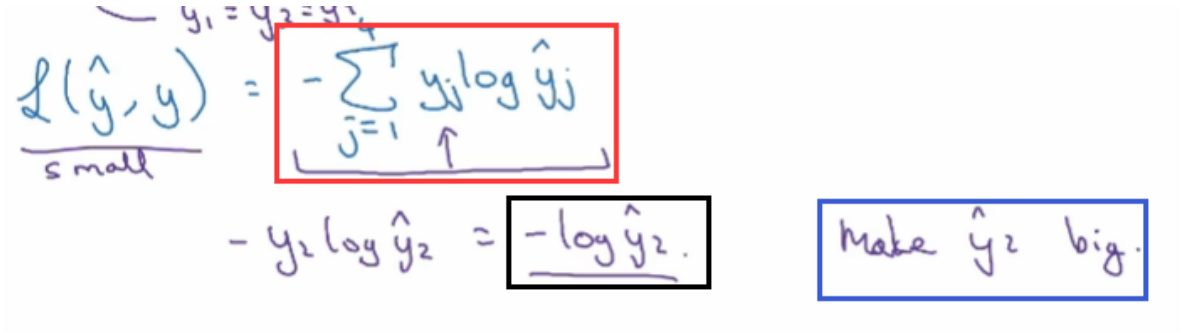
例如输入的向量为[5,2,-1,3]，则首先转化为[e^5,e^2,e^-1,e^3]，即[148.4,7.4,0.4,20.1]，再将这四个数加起来得m，再用向量中的每一个元素除以m，得到[0.842,0.042,0.002,0.114]，即一个表示概率的向量。

Softmax可以线性划分节点的类别，如下所示：



而通过中间加入隐含层，我们可以将上述线性结果转化为非线性的划分方式，以此来达到分类目的。

分类数C=2时，Softmax分类就简化为了逻辑回归。在训练Softmax网络时，代价函数的目的总是使**真实分类所对应的概率尽可能大**，如图所示：



红色框为基础公式，黑色框所展示的部分，是因为真实标签为[0,1,0,0]所简化后的公式，要使黑色框公式展示内容减小，则应该使y2的预测概率越大。

3.5 编程框架介绍

通过①编程的简单性②运行速度③是否开源，这些原则，来选择所使用的框架

- Caffe/Caffe2
- CNTK
- DL4J
- Keras
- Lasagne
- mxnet
- PaddlePaddle
- TensorFlow
- Theano
- Torch

参考资料

[1] 结构化数据、半结构化数据、非结构化数据*SU_ZCS的博客-CSDN博客*结构化、半结构化和非结构化数据

[2] 神经网络之激活函数_Sigmoid和Tanh探索 - 知乎 (zhihu.com)

[3] 【直观详解】什么是正则化*清晨的光明的博客-CSDN博客*正则化

[4] 通俗理解 Adam 优化器 - 知乎 (zhihu.com)