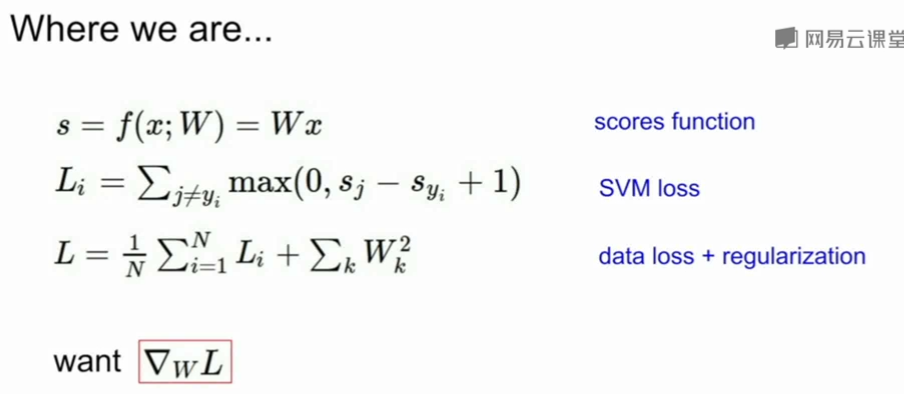
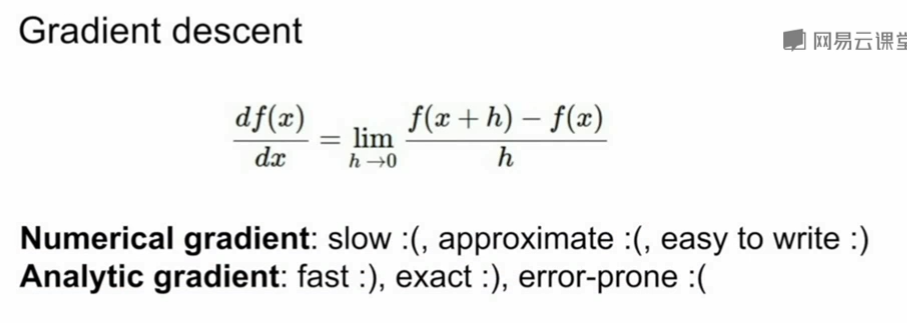
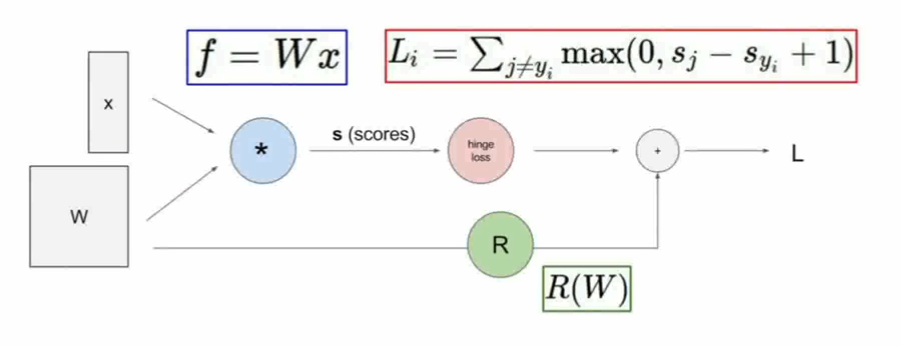
# 反向传播

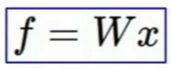
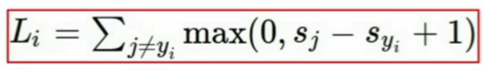


计算梯度，数值/解析



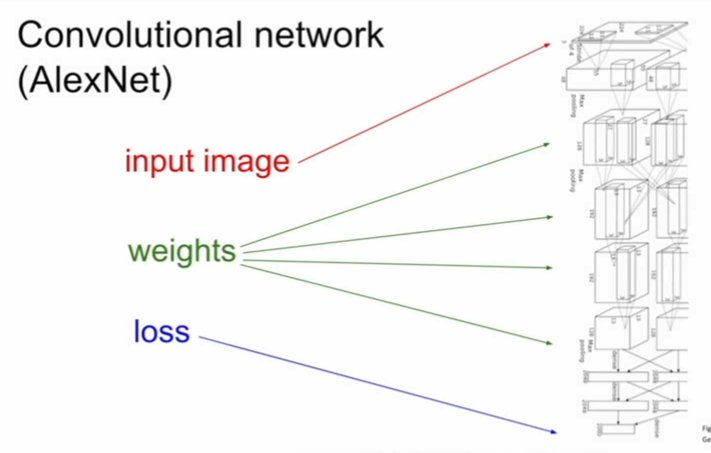
Computational Graph 计算图



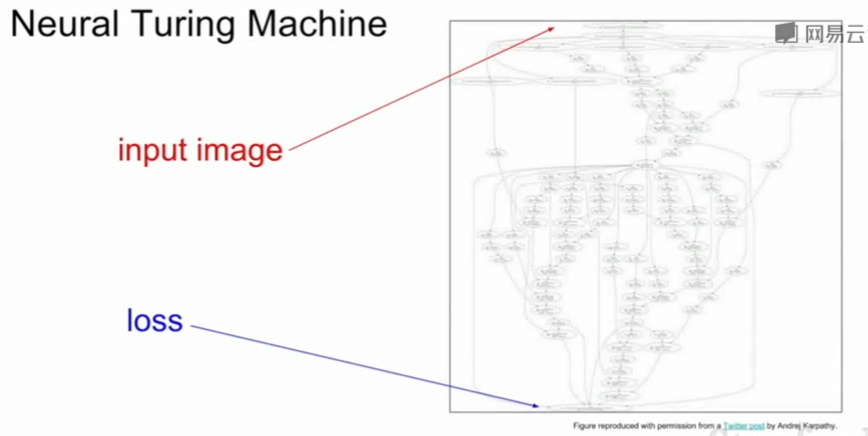
1. Score 计算：W\*X 
2. => 合页损失函数 
3. => 正则化 
4. 最后得出损失值 L

后面的复杂神经网络也是基于以上结构

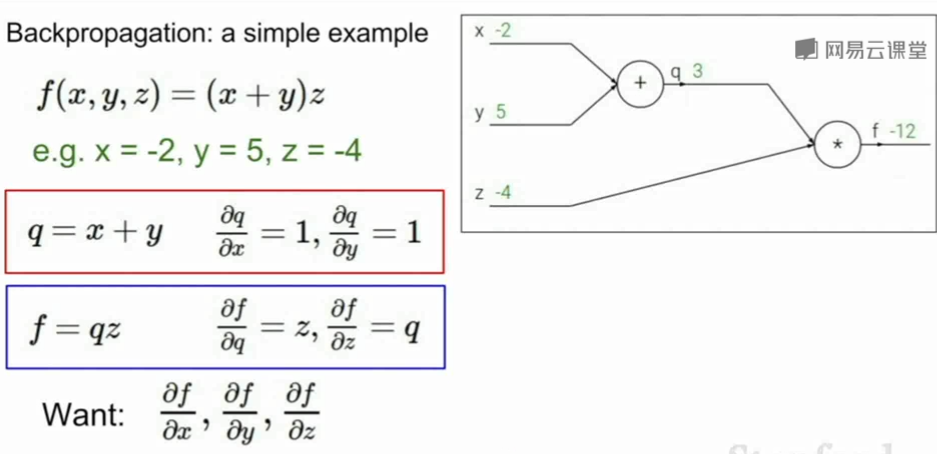
如卷积：



神经图灵机

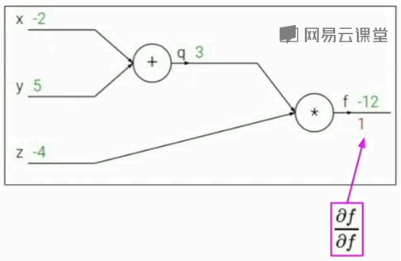


1. 前向传播

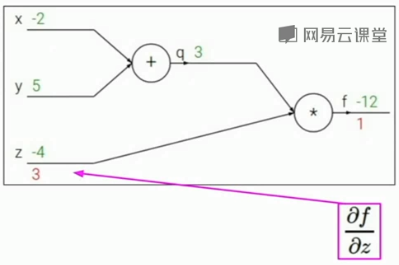


1. 反向传播

(1)对f求导，就是1



(2)在z方向上求导，就是q，这里q=x+y = -2 + 5 = 3



(3)同理下面计算= z, 而z = -4

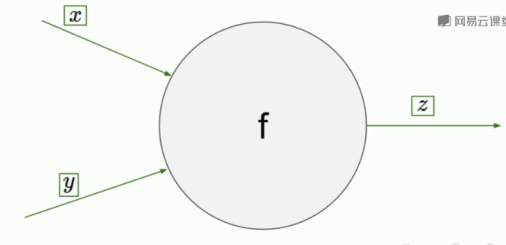
(4)再往前求，而y没有和f直接相连，所以有

同理 

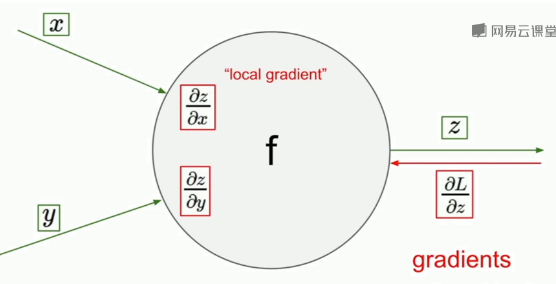
\*\*\* 为什么不直接求，而是转尔求？

因为实际的f函数会非常非常复杂，所以转而求 和 ， 会容易，而且节点只知道当前节点的前后函数，不一定知道 所有函数

在反向传播中，我们有很多这样的节点，在计算图中，每个节点只知道与他连接的上一个和下一个节点

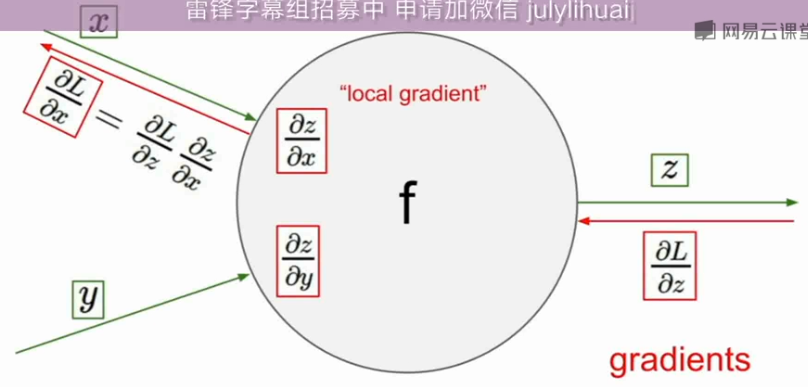


**Local gradient:**



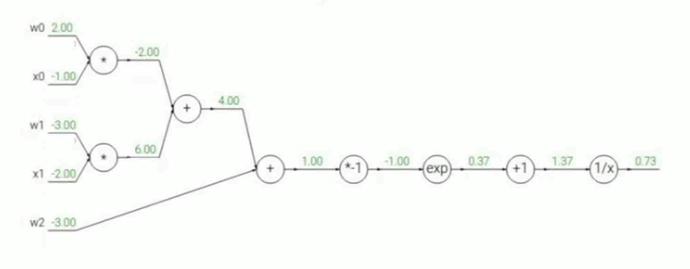
相对于节点的输入，也可以计算它们相对应的梯度是节点的直接输出，在返向传播梯度时，每个节点都会获得一个上一个节点到它的梯度，到最后的节点时，就计算了输出对应输入的梯度。

如上图，我们希望的就是找到

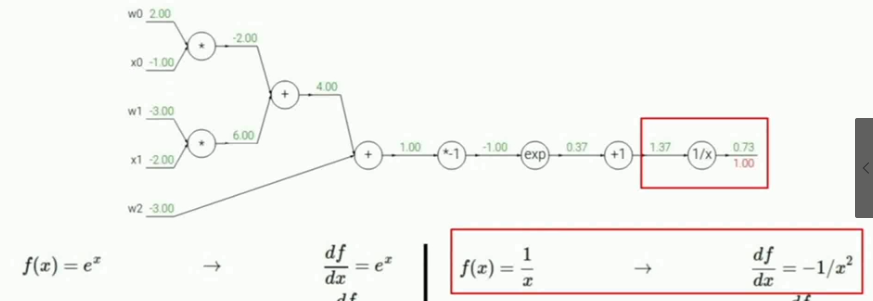


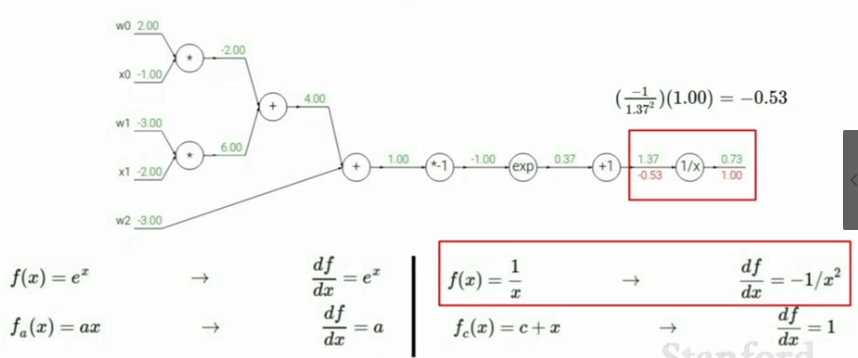
梯度传递的方式就是上游的梯度过来，再乘以每节点输出对每节点输入的梯度

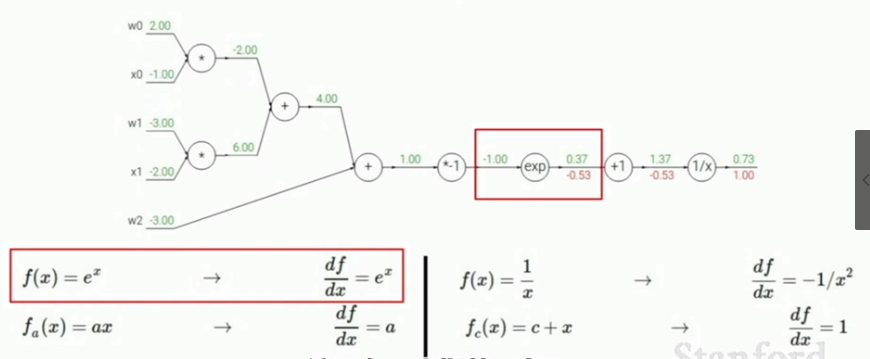
## 例 ，一共有5个输入：w0,x0,w1,x1,w2

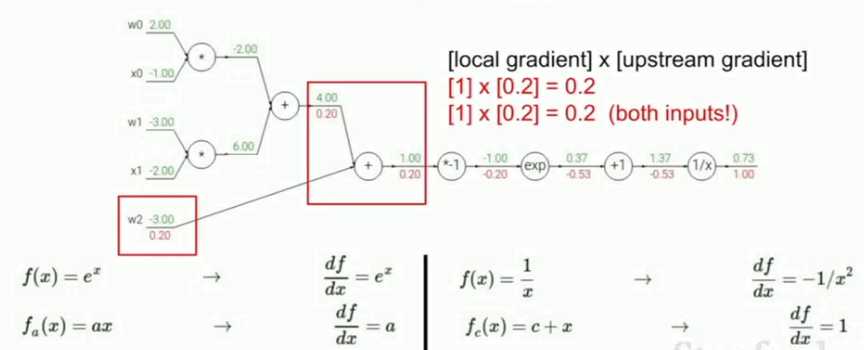


传播过程：

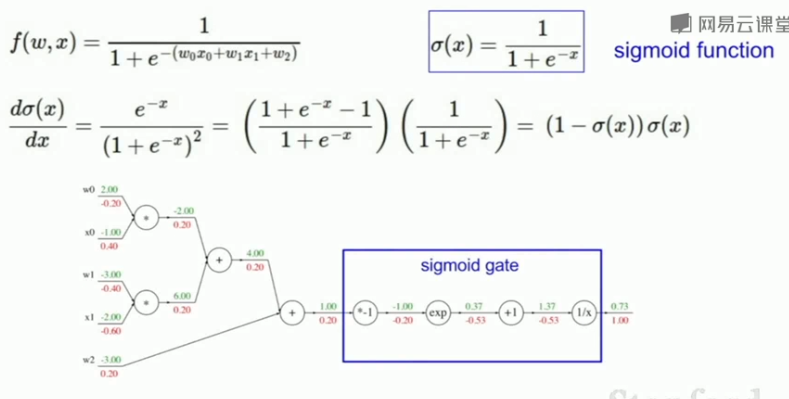








## Sigmoid function & Sigmoid gate



节点合并的原则是一个大的节点最多有两个输入，

计算图的好处， 计算过程中，总是找到一个梯度，当计算一个复杂的函数梯度时，也可以使用computational graph, 把他分成若干的节点，通过反向传播 和 链式法则 最终完成计算梯度的工作

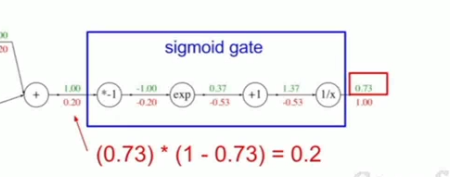
Sigmoid 函数及其导数：





如下图，假设整个节点的输出 sigmoid函数，值为0.73，即=0.73，

求导时因为有，即直接算出=0.73\*（1-0.73）=0.2，则反向传播为0.2 \* 1 = 0.2

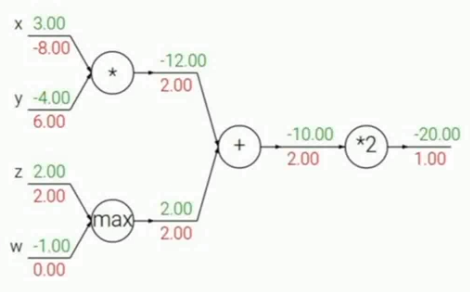


# Patterns in backward flow

Gate 可以是不同的属性与逻辑，加减乘除 等

加法门是一个梯度分布

Add gate: gradient distributor



**Max gate**: 较大者输出

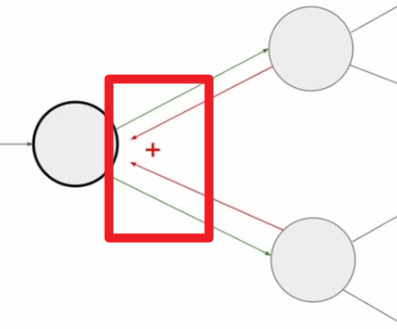
对max gate求导，对其中一个输入的梯度为1（最大的那个，就是输出的那个），其他输入的梯度为0

可以理解为 add gate 把梯度传递给每一个输入，而max gate把梯度传递给最大的分支，其他则归0

**Multiplication gate**:输出两个输入的乘积

对他求导，得对应的另一个输入，可以把他看成一个gradient switcher,通过对一个输入大小来调整另一个输入的梯度

## 梯度的叠加 gradients add at branch



改变一个节点，就会影响它后面的所有节点，而后面的这些节点又会通过反向传播回流影响这个节点

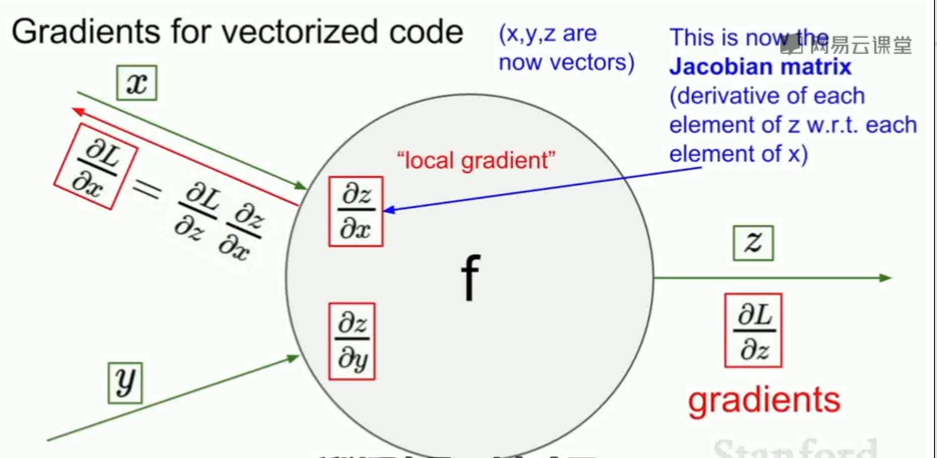


\*\*\*目前只是找到的每个节点的梯度，还没有更新这些节点的权重

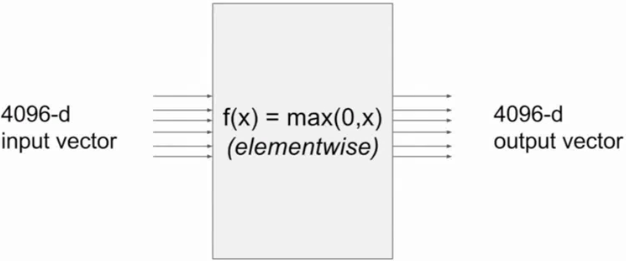
## Jacobian matrix

令，称它为函数f在点x处的jacobi矩阵（1xn阵），他的地位和作用相当于单变量函数的一阶导数

对于输入x不是变量，是向量的情况，梯度就变成了jacobian



举例：max gate,比较0和X



\*\*\*此例中，jacobian matrix 应该是4096x4096 每个元素都是4096x1

\*\*\*实际中，可能有多个输入同时处理的情况，比如100个输入变量，那么dimension就会变成409600， 要计算409600x409600的变量，这会是非常大的运算量

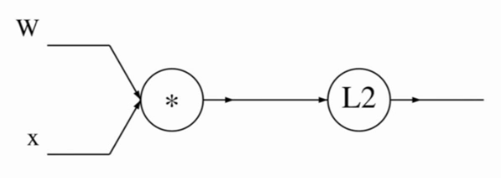
\*\*\*实际上不用按照这样方法计算，我们对每一个元素求偏导，再想想每个偏导的结果，which dimension of inputs affect which dimension of the output?

答案是对角阵diagonal，对x1求导，其他都是0，同理

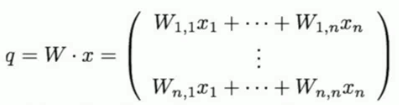
### 向量化的例子： （L2范数）

其中有 ，

1. 画出computational graph



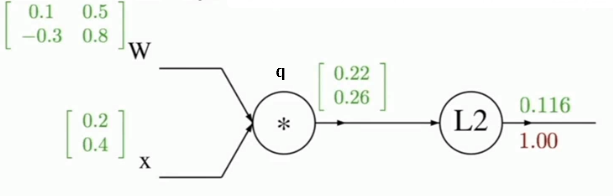
1. 太费钱前向传播 先计算乘法:







带入初始的W 和 x的值



1. 开始反向求导

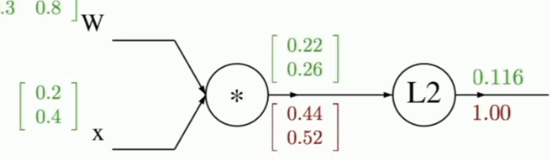
#### q点



Jacobian matrix: 

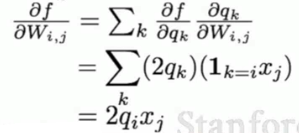
这里的q是2维向量，我们希望找到q的每个元素是如何影响f的最终值的，即：

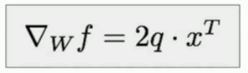
则q点的反向传播梯度为2\*q = 2\*[0.22,0.26]=[0.44,0.52]



#### W(输入)节点



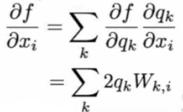


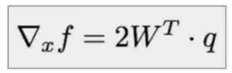
最终得到：

\*\*\*的维度和W的维度一定是一样的，这可以用来检测求导是否正确

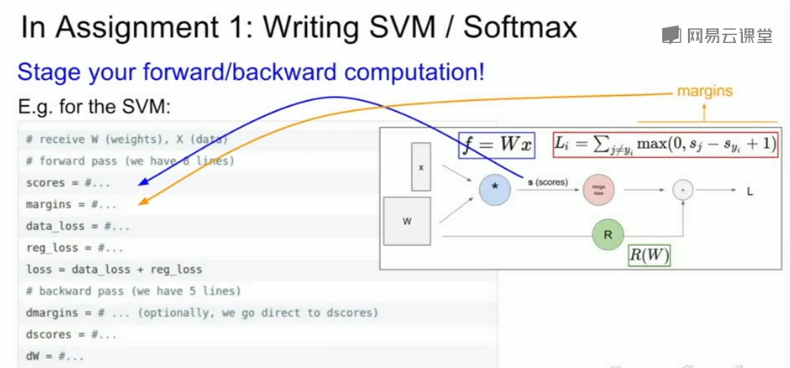
#### x(输入)节点



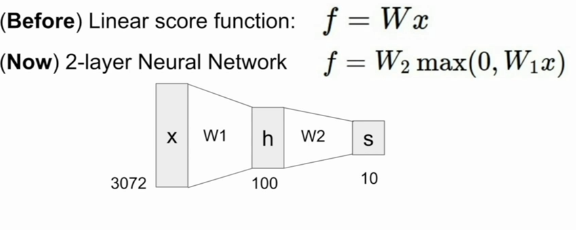


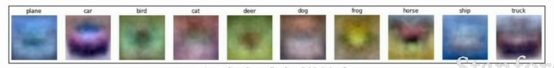
最终得到：

在后面的作业中：需要计算softmax 和 svm的梯度



# 2.神经网络





对于cifar10的分类例子， 权重矩阵W1的每一行相当于一个模板，W2是h里所有向量的权重，除了max( ) 还有很多非线性的函数可以选择，所以W1不一定是3072\*10，可以有很高的维度，再通过W2,W3 … 最终收敛到10个分类

h 是W1\*x 算出的得分，W2再对这些得分进行weighting 再得出最终的得分

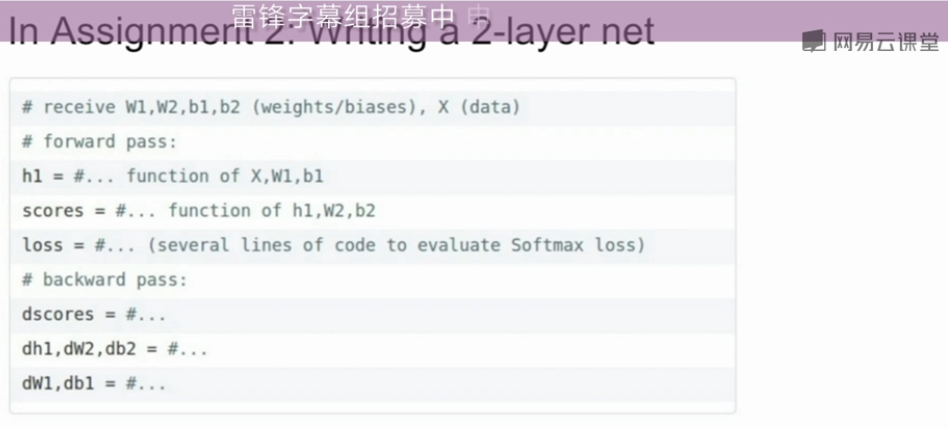
h 是非线性之后的值

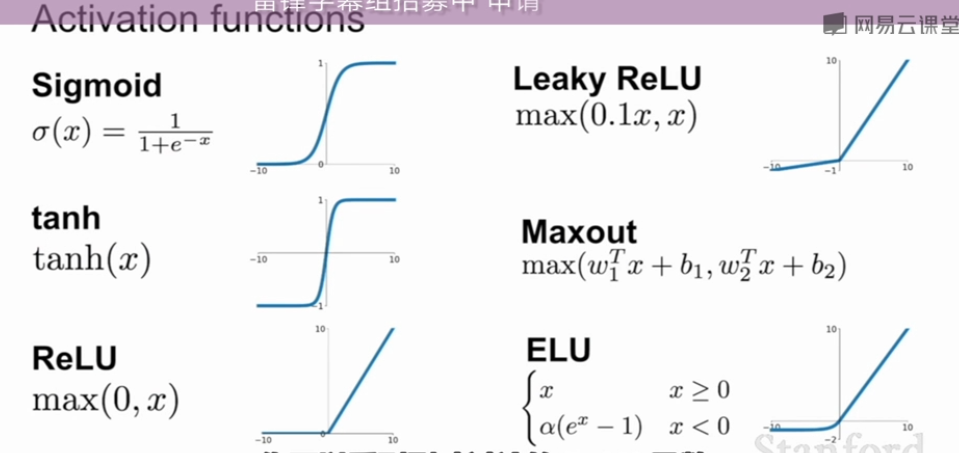
h=max(0,W1\*x)

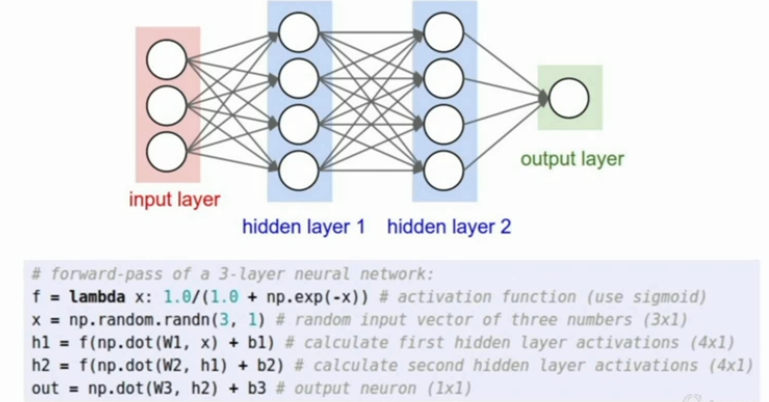
\*\*\*上面幻灯片里的图

上例出W1与输入x关联，W1 \* x: 可理解为可以得到这些输入的范本

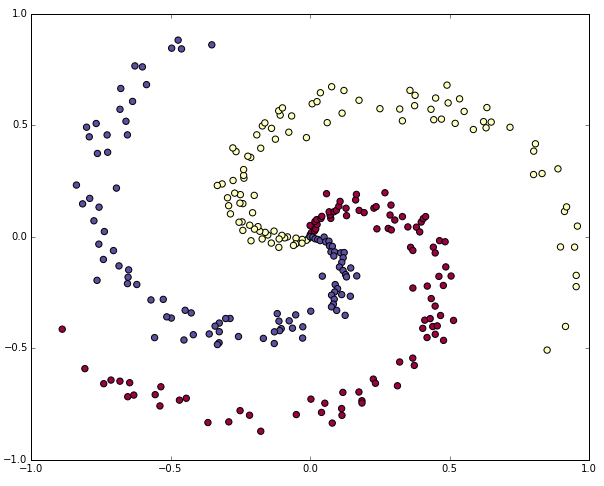
至于W2，h是符合多少范本的得分







# neural-networks-case-study



输入： X (N,D) (300,2)

W: (D,C) (2,3)

b: (N,1)

正向传播（输入变换）：



反向传播（求导）：

1）对score求导，

dscores **=** probs

dscores[range(num\_examples),y] **-=** 1

dscores **/=** num\_examples

2）对W,b 求导,因为有scores = X\*W+b

再加上正则化项的求导结果W



其中 (C,N) (N,C), dscores 与score是同样维度的，所以是

dW **=** np**.**dot(X**.**T, dscores)+ reg**\***W

db **=** np**.**sum(dscores, axis**=**0, keepdims**=**True)

### Performing a parameter update更新参数W,b

*# perform a parameter update*

W **+=** **-**step\_size **\*** dW

b **+=** **-**step\_size **\*** db

## Training a Neural Network



两层网络，

第一层 W(D,h) b(1,h) 输出：hidden

第二层W2(h,K) h2(1,K) 输出：scores

激活函数：ReLu

前向：

1）

*# evaluate class scores with a 2-layer Neural Network*

hidden\_layer **=** np**.**maximum(0, np**.**dot(X, W) **+** b) *# note, ReLU activation*

scores **=** np**.**dot(hidden\_layer, W2) **+** b2

后向（求梯度）：

*# backpropate the gradient to the parameters*

*# first backprop into parameters W2 and b2*

dW2 **=** np**.**dot(hidden\_layer**.**T, dscores)

db2 **=** np**.**sum(dscores, axis**=**0, keepdims**=**True)

dhidden **=** np**.**dot(dscores, W2**.**T)

dhidden[hidden\_layer **<=** 0] **=** 0#ReLU

**backpropagate the ReLU 对ReLU进行反向传播求导**