

数据规模越大，深度学习的效果越好

无人驾驶：1.物体检测 2.行人检测 3.标志识别 4.速度识别

计算机视觉

图像分类：一张图片被表示成三维数组的形式，每个像素的值为0-255 例：300\*100\*3

[R,G,B]

情况改变 ： 照射角度 光照强度 形状改变（核心）部分遮蔽（核心）背景混入

常规套路

1. 收集数据并给定标签
2. 训练一个分类器（神经网络）
3. 测试，评估

def train(train\_imgs,train\_labels):

…

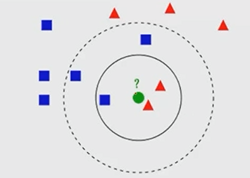
return model

def predict(model,test\_imgs):

…

return test\_labels

# k近邻 KNN

  
1.计算已知类别数据集中的点与当前点的距离

2.按照距离依次排序

3.选取与当前点距离最小的k个点

4.选取前k个点所在类别的出现概率

5.返回前k个点出现频率最高的类别作为当前点预测分类

KNN算法本身简单有效，lazy-learning

分类器不需要使用训练集进行训练，训练时间复杂度为0

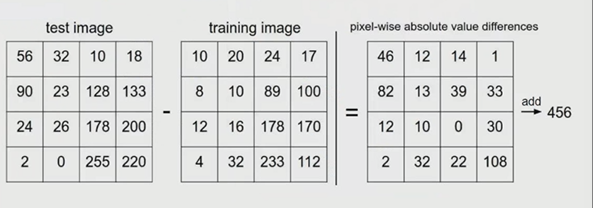
KNN分类的计算复杂度和训练集中的文档数目成正比，即：如果训练集中文档总数为n,那么KNN的分类时间复杂度为O(n)

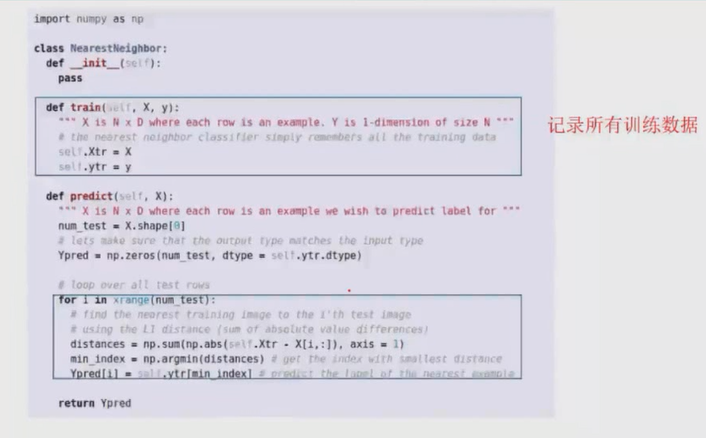
( 每一个样本的复杂度都为O(n),如果有n个样本，那么复杂度就是O() )

CIFAR-10数据集 <https://www.cs.toronto.edu/~kriz/cifar.html>  
10类标签，50000个训练数据 10000个测试数据 大小均为32\*32（图片比较小，模型的学习压力较小）

（还有更大的CIFAR-100 ）

L1距离：

矩阵情况：对应位置的差值的绝对值，再累加



# 超参数 · 交叉验证

超参数：

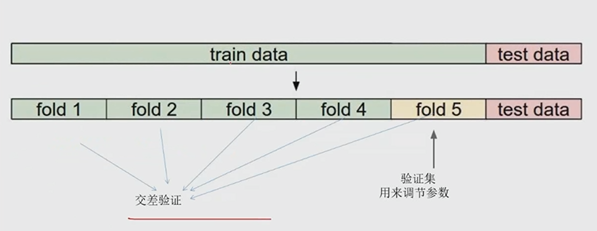
L1 distance (Manhattan distance): 

L2 distance (Euclidean distance 欧氏距离): 

交叉验证：

测试集只能最终的时候才能用

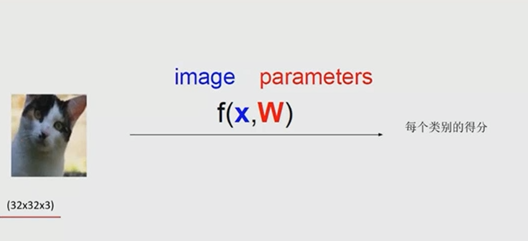


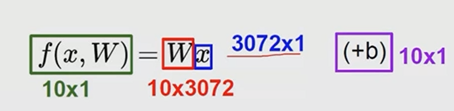


背景主导：一张图片中，主题只占一部分，背景对结果的影响应该去除，比如在K近邻中，计算距离时应该把背景部分的距离直去掉。



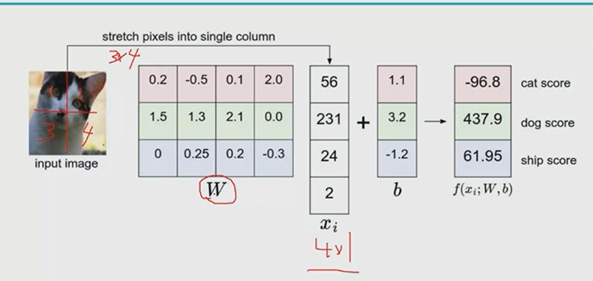
# 线性分类 Linear Classify

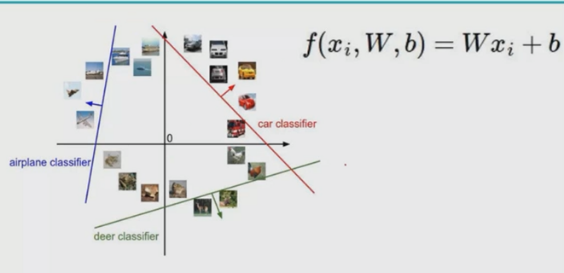




如下， f = WX+b将图片变换成4x1的向量，输出f为3x1, W为3x4,

输出f为对应不同类别的得分值，如图中dog 的得分最大，则输出的结果判定为dog的可能性最大







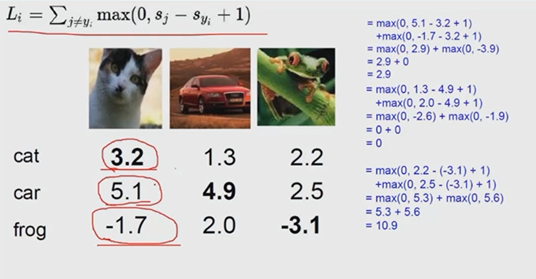
线性分类 – 线性边界（f = Wx） 得到分类结果

# 损失函数 – 对结果的修正

当判断对时保留，判断错时进行修正，使损失值减少（以趋于0）



（）



蓝色代表正确值，

Score值越高越好，

如果在绿色区域，说明损失值与正确值的差值大于,偏差较大，

处于红色区域，偏差较小，

处于蓝色区域，判断正确



## 损失函数计算





排除样本个数对结果的影响



# 正则化惩罚

x=[1,1,1,1]

模型1: =[1,0,0,0]

模型2: =[0.25, 0.25, 0.25, 0.25]

则：

 与 **** 相等，不同的模型结果相同！，但是的泛化能力明显差于，（,对x的后三个变量不起作用） 为了避免这种问题，加入正则化惩罚项 ,用来惩罚，的权重参数



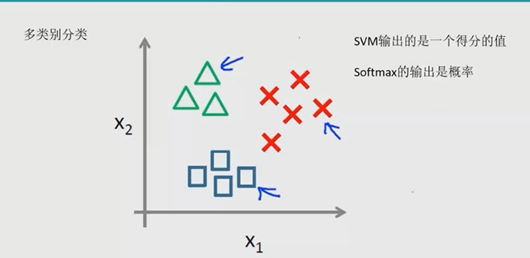
一般惩罚项为：

则=1，=

损失函数终极版：



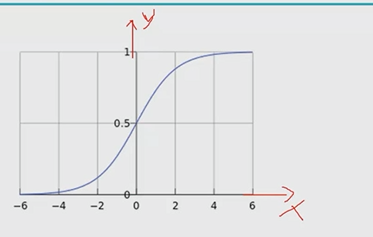
# Softmax



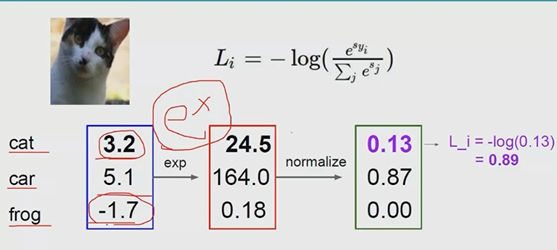
Sigmoid函数:



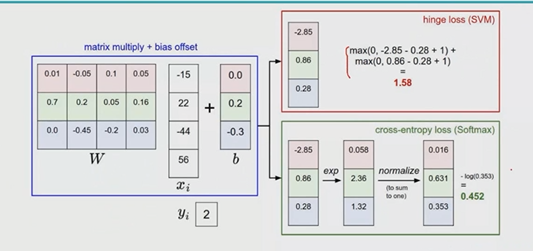
 定义域为R, 值域为概率值(0,1)



得分值变换成概率，经过计算，正确分类的得分对应的概率值是0.13，求-log 得到损失值0.89



## SVM与SOFTMAX对比



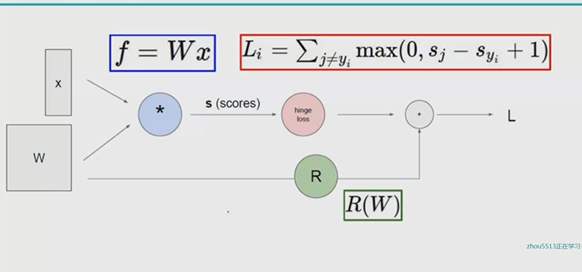
Softmax 永远会得到一个损失值

# 优化：找到参数 – 优化

例如优化如下函数：求能使最优化的：（如最小化）

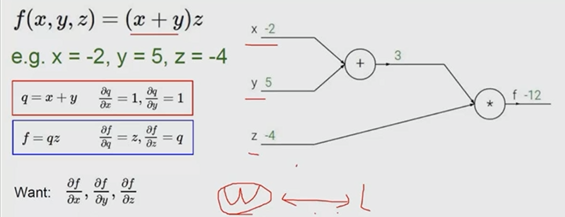


# 反向传播



得分值 >>> loss值 >>> 更新参数(一般是权重值)

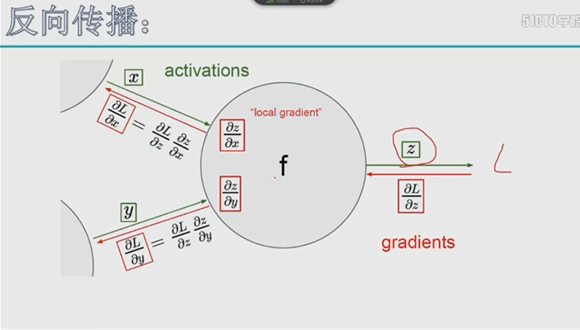
>>> 获取新的得分值 >>> …



原理：计算每个权重参数对结果产生多大贡献，

如果对结果贡献为正向，则加大，

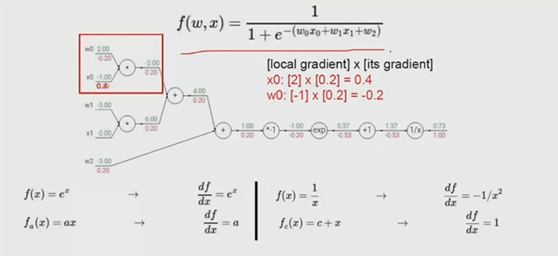
如果对结果贡献为负向，则减少



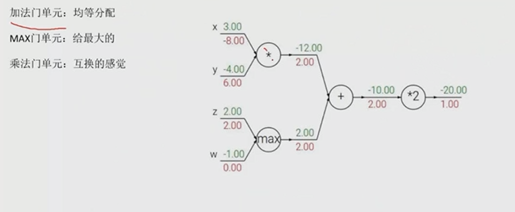
L传给x 是先传给z， 再由z传给x: ,

即：x对L的贡献为 x对z的贡献 乘 z对L的贡献

例：一个sigmod函数的值传导



门单元：

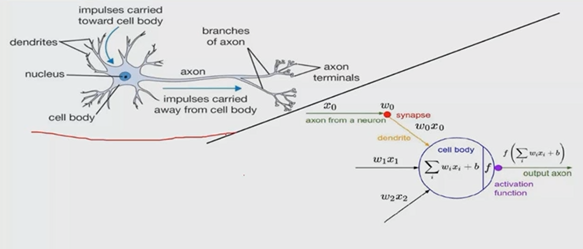


加法门单元 ：梯度传递均等分配, 求导都是为1

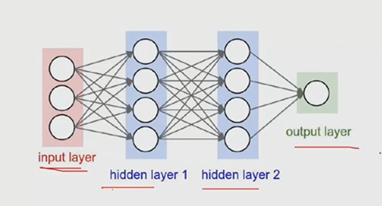
MAX门单元 ：梯度直接传给最大的值，不会传给另外一个值，所有最大的求导是1，其他是0

乘法门单元 : 梯度求导“互换” ，如xy ，对x求导得y，对y求导得x

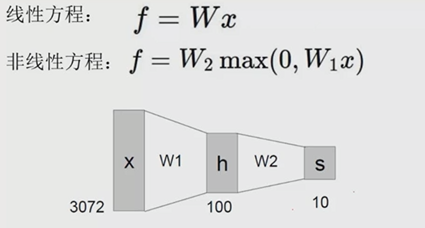
# 神经网络整体架构



特点1：层次结构



特点2：非线性结构



## 激活函数（神经网络的核心）

激活函数 ：，对，激活函数决定是否起作用

激活函数使得model变为非线性

每加一个隐层，就对输入进行一次非线性加工

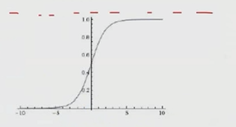
单隐层神经网络



2隐层神经网络

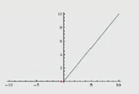


### 各种激活函数的比较

1. Sigmod函数：  

一般隐层数较大，（深度残差网络已经达到152层）

梯度消失

1. ReLU函数：  

x>0 f=x

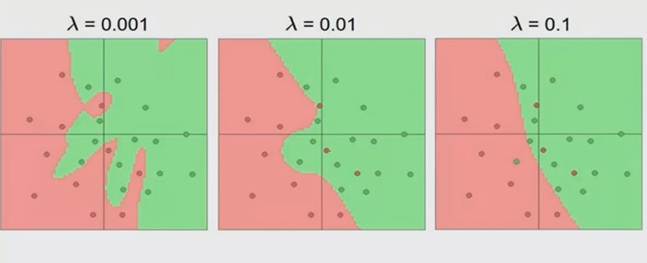
x<0 f=0

解决梯度消失问题（求导是不一定都小于1），求导简单

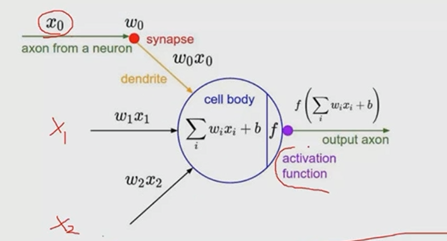
神经网络页面模拟

<https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html>

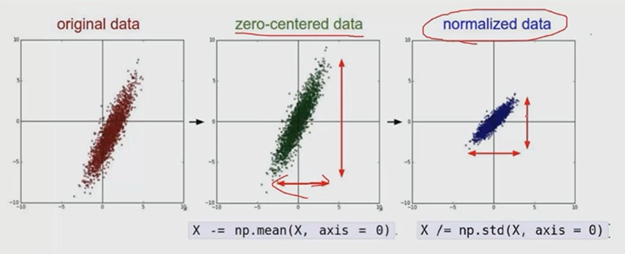
# 过拟合问题解决方案



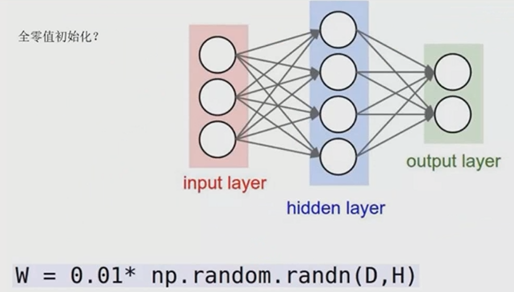
神经元越多，过拟合风险越大



## 数据预处理，-归一化，把浮动控制在0-1内，以0为中心



## 权重初始化：w

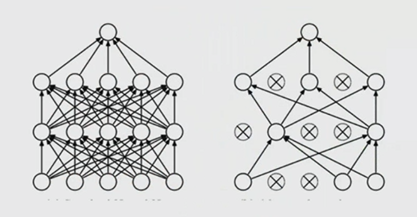


区间随机赋值：

C:\Users\Owner\AppData\Local\Temp\1543461876(1).png

随机初始化 / 高斯初始化

## Drop out



左侧为全连接，过于复杂，容易过拟合，

右侧drop out 为在更新时避掉一些神经元，一般是随机的，每次迭代drop out的神经元都不一样，一方面计算量少了，一方面减少过拟合