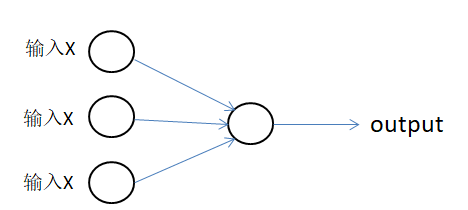
1.神经网络



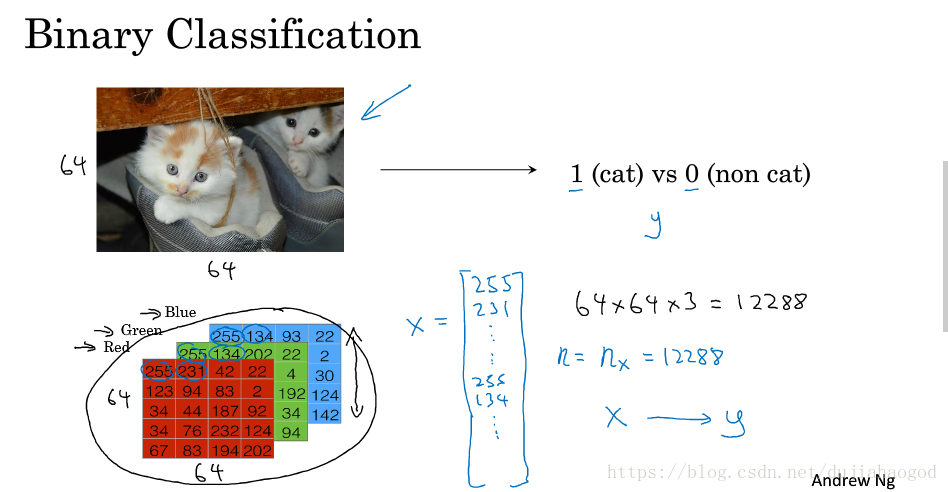
只要输入X,所有的中间过程它会自己完成，就能够得到输出结果output

2、在图像领域，经常使用的是卷积神经网络（CNN），对序列数据，经常使用RNN,语言、英语汉字、字母等则使用更为复杂的RNNs，对于图片等需要使用混合的神经网络结构。

3、机器学习也被用于结构化数据和非结构化数据，结构化数据：是设计的数据库；非结构化数据，比如音频、图像等，这里的特征可能是图像中的像素值或者是文本中的一个单词。

4、在神经网络的训练过程中，遍历数据集不应该用if来遍历，而应该用更好更快的方法。

5、二分分类



如图，例如，你有一张图片作为输入，如图所示。你想输出识别此图的标签。如果是猫，输出1.如果不是，则输出0。我们用y；来表示输出的结果标签。

计算机保存一张图片，要保存三个独立矩阵。分别对应图片中的红、绿、蓝三个颜色通道。如果输入图片是64×64像素的，就有三个64×64的矩阵，分别对应图片中红、绿、蓝三种像素的宽度。要把这些像素亮度值放进一个特征向量中，就要把这些像素值都提出来，放入一个特征向量x。为了把这些像素值取出放入特征向量，就要像下面这样定义一个特征向量x，以表示这张图片。我们把所有的像素值都取出来。例如255、231等等，直至列完所有的红色像素。接着是255、134、255、134等等，最后得到一个很长的特征向量，把图像中所有的红、绿、蓝像素强度值都列出来。如果图片是64×64的话，那么向量x的总维度就是64×64×3。因为这里三个矩阵的元素数量，对于这个例子，数字是12288，把它们乘起来，这就是结果。我们用nx=12288，来表示输入的特征向量x的维度。

在二分分类问题中，目标是训练出一个分类器，它以图片的特征向量x作为输入，预测输出的结果标签y是1还是0。也就是预测图片中是否有猫。

用一对（x，y）来表示一个单独的样本，x时nx维的特征向量。标签y值为0或1。训练集由m个训练样本构成（x^(1),y^(1)）表示样本一的输入和输出，（x^(2),y^(2)）表示样本二的输入和输出，（x^(m),y^(m)）表示最后一个样本m的输入和输出。这些一起就表示整个训练集。用小写的字母m来训练样本的个数。我们定义一个矩阵，用大写的X表示。它有训练集中的x1、x2这些组成。像这样写成矩阵的列。现在我们把x^(1)放进矩阵的第一列，x^(2)是第二列......x^m是第m列，最后得到矩阵x。这个矩阵有m列。m是训练集的样本数。这个矩阵的高度记为nx。

6.Logistic回归

Logistic 回归是一种用于解决监督学习（Supervised Learning）问题的学习算法，其输出y的值为0或1。Logistic回归的目的是使训练数据与其预测值之间的误差最小化。

规定一个nx维向量和一个值b作为参数，可得到：

è¿éåå¾çæè¿°

但由于y^为一个概率值，取值范围为[0,1]，简单地进行线性拟合，得出的y^可能非常大，也有可能为负值。这时，便需要一个sigmoid函数来对它的值域进行约束。

设Z=WT+b，则sigmoid函数为：

è¿éåå¾çæè¿°

用Sigmoid函数来约束y^：



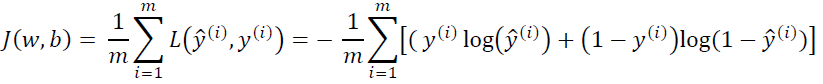
为了训练logistic回归模型中的参数w和b，需要定义一个成本函数，用损失函数来衡量算法的运行情况：

è¿éåå¾çæè¿°

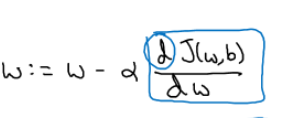
但在logistic回归中一般不使用这个损失函数，因为在训练参数过程中，使用这个损失函数将得到一个非凸函数，最终将存在很多局部最优解，这种情况下使用梯度下降（Gradient Descent）法无法找到最优解。所以在logistic回归中，一般采用log函数：

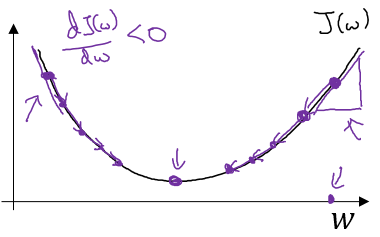
è¿éåå¾çæè¿°

成本函数J:



7.梯度下降法来训练或学习训练集上的W和b:





以上图为例，当w位于最优点右侧时，其随着宽度的增加，高度也在不断增加，因此斜率是正数，即dw>0。因此，在迭代过程中，w的值会不断减小（w:=w-adw）。

而当w位于最优点左侧时，其随着宽度的增加，高度确在不断减小，因此斜率是负数，即dw<0。因此，在迭代过程中，w的值会不断增加（w:=w-adw）。

8、向量化：尽量避免使用for循环，在深度学习安全领域、深度学习实践中，你会经常发现自己训练大数据集，因为深度学习算法处理大数据集效果很棒，所以你的代码运行速度非常重要，否则如果在大数据集上，你的代码可能花费很长时间去运行，你将要等待非常长的时间去得到结果, 所以在深度学习领域，运行向量化是一个关键的技巧。

在logistic回归中，我们需要去计算Z=WTX+b，其中w是列向量，x也是列向量，如果有很多的特征，他们就是非常大的向量，所以w和x都是R内的nx维度的向量。

非向量化的实现：

z=0;

for i in range(n\_x):

z+= w[i]\*x[i]

z+=b;

向量化的实现：

z = np.dot(w,x)+b;

其中np.dot(w,x)就是WTX

从而向量化的表示可以比非向量化具有更快的计算速度，加快代码的执行。