深度学习

1. 避免显示的for循环

Numpy等专门进行数值计算的类库，对矩阵的计算做了很多优化，比如说并行化等，所以在进行深度学习的编程时，要避免使用显示的for循环。我们应该调用numpy的库函数，来提高计算的效率。

2. 熟练运用python的广播机制

广播机制通常会发生在操作两个不同形状的矩阵的时候，比如一个3x3的矩阵加上一个1x1的矩阵，广播机制会自动把1x1的矩阵，扩展成3x3的矩阵，来使得加法计算成立。广播机制可以简化编写的代码，提升效率；但是不熟悉广播机制的话，也会发生很多令人很奇怪的，很难找出的bug。

3. 一个最简单的猫识别器

训练数据：209张，64x64x3的含有和不含有猫的图片

神经网络结构：最简单的逻辑回归模型，没有隐藏层，每条数据对应的输入层为64x64x3维度的特征向量。

训练流程：输入数据，向前传播得到损失函数的值，使用梯度下降的方法，向后传播计算导数，并更新w，b两个参数，重复向前传播与向后传播的循环，迭代2000次，学习率为0.5

训练结果：生成的模型，对于训练数据的识别准确度为94%，对于50张测试数据的识别率为70%

4. 为什么要用非线性的激活函数

通过证明可以得到，如果使用线性激活函数，或者干脆不使用激活函数，那么神经网络的输出永远是输入值的线性组合，那么无论有多少隐藏层，都和没有隐藏层没区别，和普通的逻辑回归没什么区别。

通常，只有在进行线性回归的时候（比如预测房价，预测成绩等），我们需要的输出值的范围是整个实数，这种情况下，也只会在最后的输出值那一层，使用线性的激活函数；在隐藏层使用线性激活函数是极其特殊的情况。

5. 随机初始化w,b

在神经网络中，不能将w，b初始化为0，因为如果w,b全是0，那么所有的隐藏层的输出永远都是一样的，在向后传播时，对于所有的w和b的改变也是相同的，那就导致了，隐藏层中所有节点功能都是相同的。

也不能将w，b初始化为比较大的数（比如两位数），这种情况下，最初进行梯度下降的位置会非常靠近整个函数的两个极端，如果使用的是sigmoid激活函数或者tanh激活函数，两端的导数都非常非常小，会导致整个梯度下降的过程非常缓慢。