神经网络基础

目 录

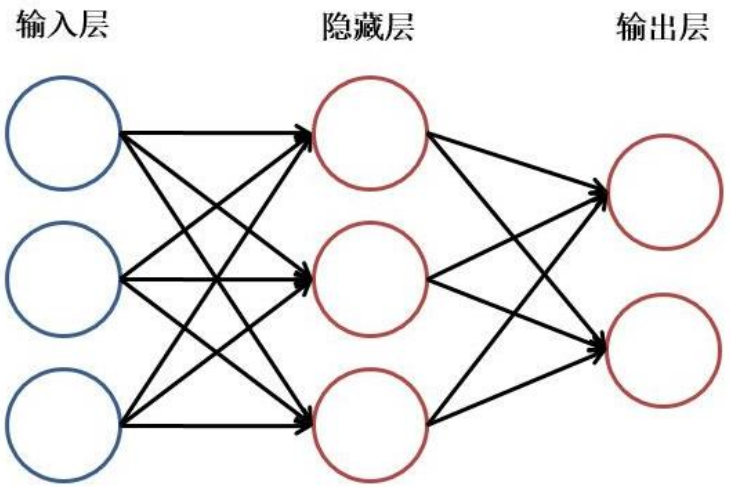
一.什么是神经网络？

（A）神经网络（neural network，缩写NN），是一种模仿生物神经网络(动物的中枢神经系统，特别是大脑)的结构和功能的数学模型或计算模型，**用于对函数进行估计或近似**。

（B）神经网络主要由：**输入层，隐藏层，输出层**构成。隐藏层层数以及隐藏层神经元个数是由人工设定。

（C）网络输入层的每个神经元代表了一个特征，一般来说，输出层个数代表了**分类标签**的个数。

一个基本的神经网络如右图：



二.二分类与逻辑回归

二分类问题

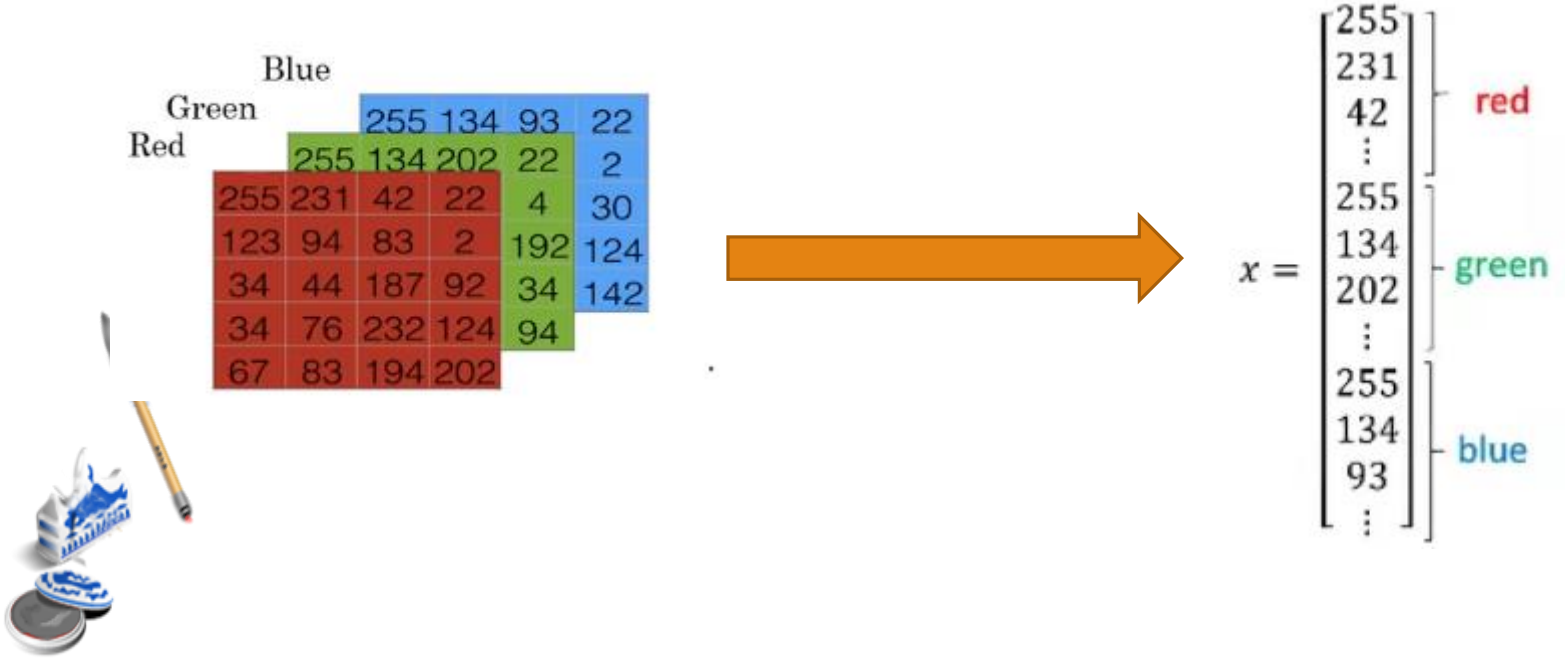
目的：训练出一个分类器，以图片的特征向量x作为输入，预测输出结果标签y是1还是0

1 (cat) vs 0 (non cat)

二.二分类与逻辑回归

二分类问题

特征向量 x：保存红、绿、蓝颜色通道的三个独立矩阵，如果输入图片的像素为5x4，则有三个5x4的矩阵分别对应红、绿、蓝三种像素的亮度。此时特征向量的维度 n_x=5x4x3=60。



二.二分类与逻辑回归

逻辑回归

逻辑回归（Logistic Regression）一般用于二分类问题中，给定一些输入，输出结果是离散值。例如用逻辑回归实现一个猫分类器，输入一张图片x，预测图片是否为猫，输出该图片中存在猫的概率结果y 。

逻辑回归是一种用于解决监督学习问题的学习算法。其目的，是使训练数据的标签值与预测出来的值之间的误差最小化

二.二分类与逻辑回归

逻辑回归

实现一个猫分类器，需要准备数量相似的猫图以及非猫图，取其中大部分组成该分类器的训练样本，少部分组成测试样本。将这些样本表示为特征向量的形式，则一个样本由一对(x, y)进行表示，其中x为n_x维的特征向量，y是该特征向量的标签（Label），根据该特征向量表示的是猫或非猫，值为0或1。m个训练样本对将被表示为：

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$$

将训练集的中的特征向量$x^{(1)}$、$x^{(2)}$...以及它们的标签$y^{(1)}$、$y^{(2)}$...分别堆叠起来，组成两个矩阵X、Y：

$$X = [x^{(1)}, x^{(2)}, \dots, x^{(m)}]$$

$$Y = [y^{(1)}, y^{(2)}, \dots, y^{(m)}]$$

则X会是个大小为$n_x \times m$的矩阵，Y是个大小为$1 \times m$的矩阵。

二.二分类与逻辑回归

逻辑回归

分类器要实现的，是给定以一个 n_x 维特征向量x表示，标签为y的一张图片，估计这张图片为猫图的概率 $\hat{y}$，即

$$\hat{y} = p(y = 1|x), 0 \leq \hat{y} \leq 1$$

有大量猫图的数据X时，希望能用一个函数，来表示出$\hat{y}$。所以考虑使用**线性拟合**的方法，从大量数据中寻找规律，从而实现这个猫分类器。规定一个n_x维向量w和一个值b作为参数，可得到线性回归的表达式：

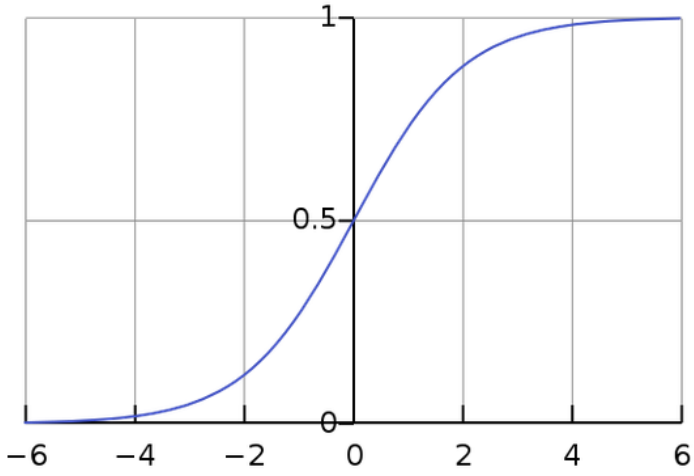
$$\hat{y} = w^T X + b$$

二.二分类与逻辑回归

逻辑回归

由于$\hat{y}$为概率值，取值范围为$[0,1]$，简单地进行线性拟合，得出的$\hat{y}$可能非常大，还可能为负值。这时，便需要一个**逻辑回归单元**来对其值域进行约束。以**sigmoid函数**为逻辑回归单元，而sigmoid函数的表达式为：

$$\sigma(z)=\frac{1}{1+e^{-z}}$$



由函数图像可知，sigmoid函数有以下几个很好的性质：

* 当z趋近于正无穷大时，$\sigma(z)=1$;
* 当z趋近于负无穷大时，$\sigma(z)=0$;
* 当$z=0$时，$\sigma(z)=0.5$。

所以可以用sigmoid函数来约束$\hat{y}$的值域，此时：

$$\hat{y}=\sigma(w^TX+b)=\frac{1}{1+e^{-(w^TX+b)}}$$

二.二分类与逻辑回归

逻辑回归的损失函数

为了训练逻辑回归模型中的参数w和b，使得输出值 $\hat{y}$ 与真实值y尽可能一致，即尽可能准确地判断一张图是否为猫，需要定义一个损失函数（Loss Function）作为衡量的标准。

用**损失函数（Loss Function）**来衡量单个样本预测值（$\hat{y}^{(i)}$）与真实值（$y^{(i)}$）之间的差异，换句话说，损失函数用来计算单个训练样本的预测值与真实值之间的误差。**平方误差（Square Loss）**是一种常用的损失函数：

$$\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$

二.二分类与逻辑回归

逻辑回归的损失函数

但在逻辑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）**将无法获得最优解。对逻辑回归模型，希望满足条件概率：

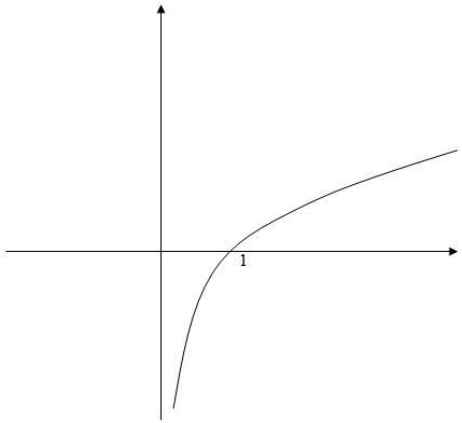
$$p(y|x) = \begin{cases} \hat{y}, & (y=1) \\ 1-\hat{y}, & (y=0) \end{cases}$$

将上下两个式子合二为一，可写成：

$$p(y|x) = \hat{y}^{y}(1-\hat{y})^{(1-y)}$$

对两边取对数，进一步化简为：

$$log\, p(y|x) = y log\, \hat{y} + (1-y) log\, (1-\hat{y})$$



Log函数曲线

二.二分类与逻辑回归

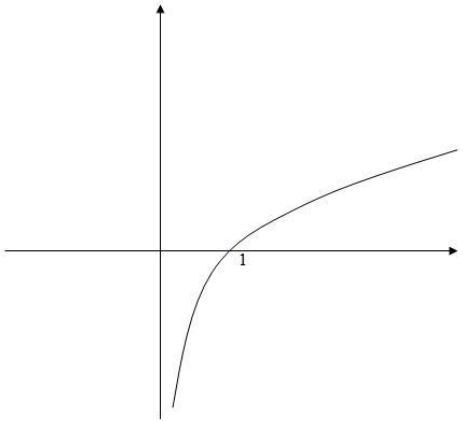
逻辑回归的损失函数

$p(y|x)$的值需要最大化，而损失函数需要最小化，所以在原来的式子中填上负号，就可以将它作为一个损失函数。在上式右边添加一个负号，就推导出了应用很广的**交叉熵（Cross Entropy）**损失函数，表达式为：

$$\mathcal{L}(\hat{y}, y) = -(y log\, \hat{y} + (1-y) log\, (1-\hat{y}))$$

交叉熵损失函数有如下性质：

* 当$y^{(i)} = 1$时，$\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -log\, \hat{y}^{(i)}$
* 当$y^{(i)} = 0$时，$\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -log\, (1-\hat{y}^{(i)})$



Log函数曲线

二.二分类与逻辑回归

逻辑回归的成本函数

成本函数(Cost function)是定义在整个训练集上面的，也就是所有样本的误差的总和的平均，也就是损失函数的总和的平均。

对m个训练样本整体的成本函数，可以使用数理统计中的**极大似然估计法（Maximum Likelihood Estimation）**推导出来的。

假设所有训练样本独立同分布，则联合概率为所有样本概率的乘积:

$$P = \prod_{i=1}^{m} p(y^{(i)}|x^{(i)})$$

对上式进行极大似然估计，还是两边取对数得：

$$log\ P = \sum_{i=1}^{m} log\ p(y^{(i)}|x^{(i)}) = -\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

对上式再取平均值，得成本函数为：

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

二.二分类与逻辑回归

逻辑回归的梯度下降

找到使成本函数的值最小的参数w和b的值，一般采用梯度下降法。
顺着梯度的方向，下降速度最快，用数学公式表达即是：

$$w := w - \alpha \frac{\partial J(w,b)}{\partial w}$$

$$b := b - \alpha \frac{\partial J(w,b)}{\partial b}$$

其中的“:=”为赋值，α为学习率，通常为一个小于1的数，用来控制梯度下降过程中每一次移动的大小，相当于迈的步子大小。
α的不宜太小也不宜过大：太小会使迭代次数增加，容易陷入局部最优解；太大则会容易错过最优解。

二.二分类与逻辑回归

逻辑回归的梯度下降

找到使成本函数的值最小的参数w和b的值，一般采用梯度下降法。
顺着梯度的方向，下降速度最快，用数学公式表达即是：

$$w := w - \alpha \frac{\partial J(w,b)}{\partial w}$$

$$b := b - \alpha \frac{\partial J(w,b)}{\partial b}$$

其中的“:=”为赋值，α为学习率，通常为一个小于1的数，用来控制梯度下降过程中每一次移动的大小，相当于迈的步子大小。
α的不宜太小也不宜过大：太小会使迭代次数增加，容易陷入局部最优解；太大则会容易错过最优解。

二.二分类与逻辑回归

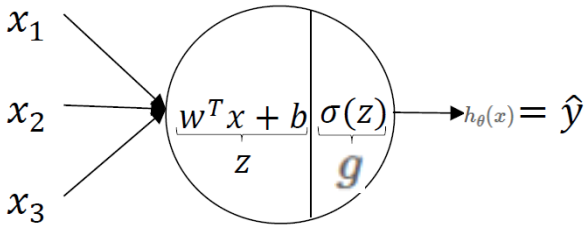
逻辑回归的总结

实际上，可以将逻辑回归看做是仅含有一个神经元的单层的神经网络。一般用于二分类网络，激活函数为Sigmoid。

逻辑回归的一个优点是其成本函数是一个凸函数，可以求得全局最小值，可以用极大似然估计求解。

$$h_\theta(x) = g(\theta^T x)$$

$$g(z) = \frac{1}{1+e^{-z}}$$

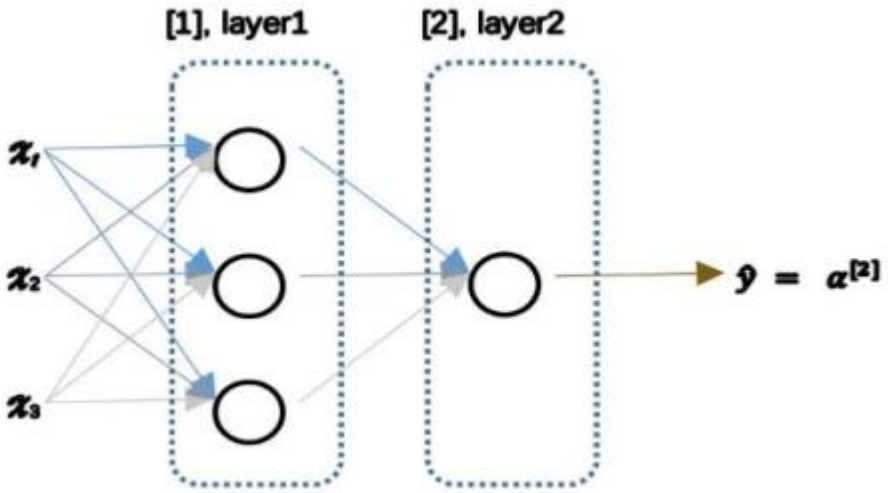


神经网络是一个对于处理复杂的非线性模型很优秀的算法。

三. 浅层神经网络

神经网络概述

神经网络每个单元相当于一个逻辑回归，神经网络由逻辑回归的堆叠起来。中间的层称为隐藏层或中间层。下图是网络结构：



三. 浅层神经网络

神经网络概述

（1）神经网络的正向传播和反向传播过程只是比逻辑回归多了一次重复的计算。正向传播过程分成两层，第一层是输入层到隐藏层，用上标[1]来表示：

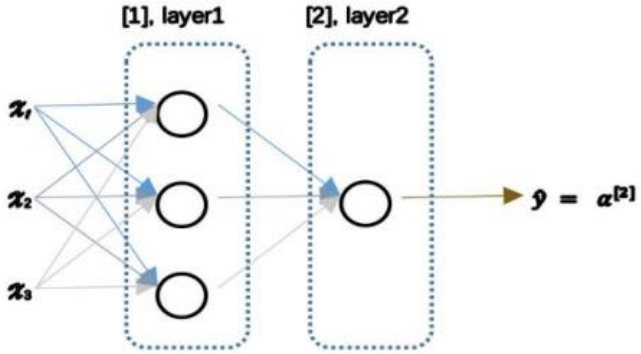
$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = \sigma(z^{[1]})$$

（2）第二层是隐藏层到输出层，用上标[2]来表示：

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(z^{[2]})$$



在写法上值得注意的是：
方括号上标[i]表示当前所处的层数；
圆括号上标(i)表示第i个样本。

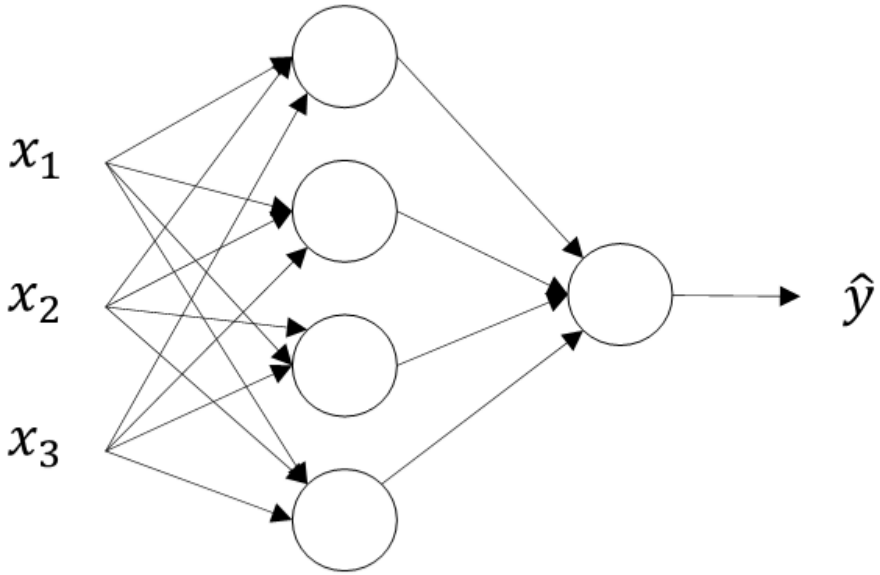
三. 浅层神经网络

神经网络表示

（1）以图示的方式来介绍单隐藏层的神经网络结构。如下图所示，单隐藏层神经网络就是典型的浅层（shallow）神经网络。

（2）从左到右，可以分成三层：输入层（Input layer），隐藏层（Hidden layer）和输出层（Output layer）。

（3）输入层和输出层，对应着训练样本的输入和输出。隐藏层是抽象的非线性的中间层

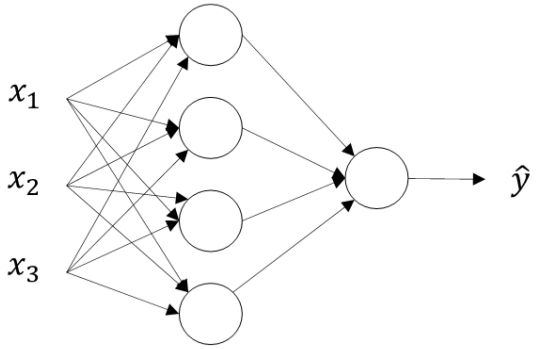


三. 浅层神经网络

神经网络表示

在写法上，我们通常把输入矩阵X记为$a^{[0]}$，把隐藏层输出记为$a^{[1]}$，上标从0开始。用下标表示第几个神经元，注意下标从1开始。例如$a_1^{[1]}$表示隐藏层第1个神经元，$a_2^{[1]}$表示隐藏层第2个神经元，，等等。这样，隐藏层有4个神经元就可以将其输出$a^{[1]}$写成矩阵的形式：

$$\boldsymbol{a}^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}$$

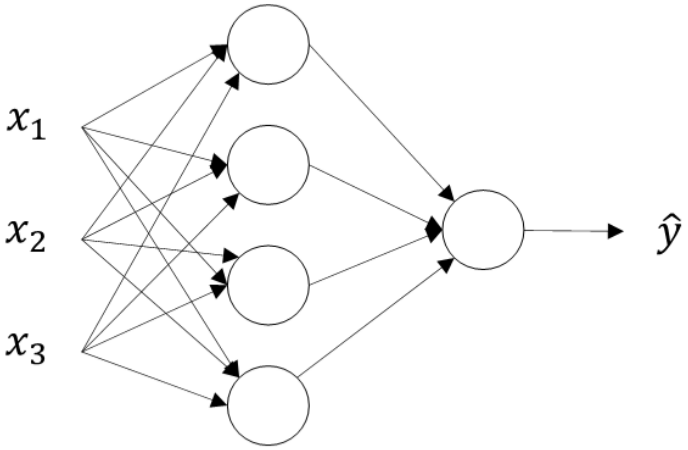


最后，相应的输出层记为$a^{[2]}$，即$\hat{y}$。这种单隐藏层神经网络也被称为两层神经网络（2 layer NN）。之所以叫两层神经网络是因为，通常我们只会计算隐藏层输出和输出层的输出，输入层是不用计算的。这也是我们把输入层层数上标记为0的原因（$a^{[0]}$）。

三. 浅层神经网络

神经网络表示

关于隐藏层对应的权重$W^{[1]}$和常数项$b^{[1]}$，$W^{[1]}$的维度是（4,3）。这里的4对应着隐藏层神经元个数，3对应着输入层x特征向量包含元素个数。常数项$b^{[1]}$的维度是（4,1），这里的4同样对应着隐藏层神经元个数。关于输出层对应的权重$W^{[2]}$和常数项$b^{[2]}$，$W^{[2]}$的维度是（1,4），这里的1对应着输出层神经元个数，4对应着输出层神经元个数。常数项$b^{[2]}$的维度是（1,1），因为输出只有一个神经元。总结一下，第i层的权重$W^{[i]}$维度的行等于i层神经元的个数，列等于i-1层神经元的个数；第i层常数项$b^{[i]}$维度的行等于i层神经元的个数，列始终为1。



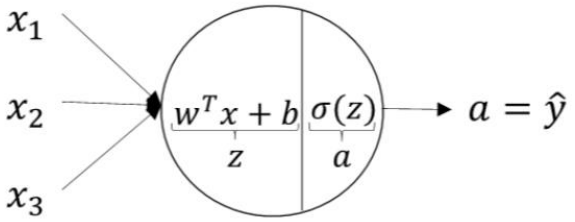
三. 浅层神经网络

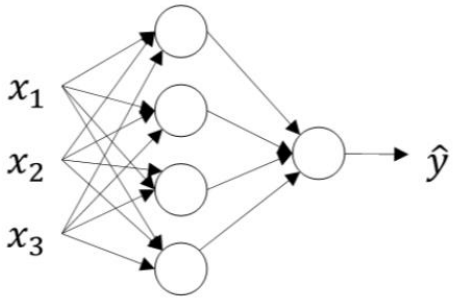
计算神经网络的输出

前面讲过两层神经网络可以看成是逻辑回归再重复计算一次。如下图所示，逻辑回归的正向计算可以分解成计算z和a的两部分：

$$z = w^T x + b$$

$$a = \sigma(z)$$





$z = w^T x + b$

$a = \sigma(z)$

三. 浅层神经网络

计算神经网络的输出

对于两层神经网络，从输入层到隐藏层对应一次逻辑回归运算；从隐藏层到输出层对应一次逻辑回归运算。每层计算时，要注意对应的上标和下标，一般我们记上标方括号表示层数，下标表示第几个神经元。

例如$a_i^{[l]}$表示第l层的第i个神经元。注意，i从1开始，l从0开始。

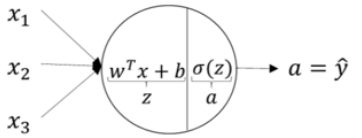
从输入层到隐藏层的计算公式：

$$z_1^{[1]} = w_1^{[1]T}x + b_1^{[1]},\ a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = w_2^{[1]T}x + b_2^{[1]},\ a_2^{[1]} = \sigma(z_2^{[1]})$$

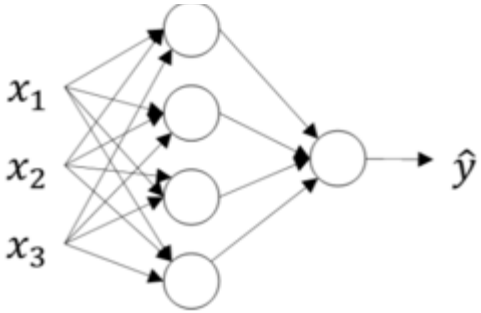
$$z_3^{[1]} = w_3^{[1]T}x + b_3^{[1]},\ a_3^{[1]} = \sigma(z_3^{[1]})$$

$$z_4^{[1]} = w_4^{[1]T}x + b_4^{[1]},\ a_4^{[1]} = \sigma(z_4^{[1]})$$



$z = w^Tx + b$

$a = \sigma(z)$

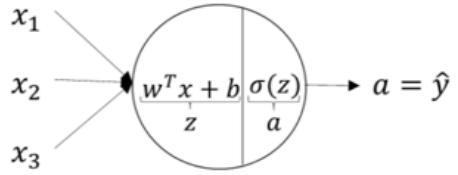


三. 浅层神经网络

计算神经网络的输出

从隐藏层到输出层的计算公式：

$$z_1^{[2]} = w_1^{[2]T} a^{[1]} + b_1^{[2]},\ a_1^{[2]} = \sigma(z_1^{[2]})$$

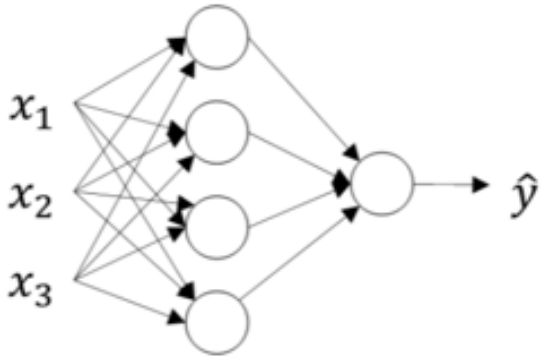


其中：

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}$$

$z = w^T x + b$

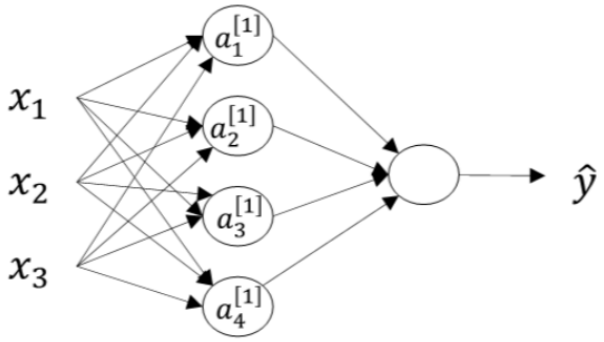
$a = \sigma(z)$



三. 浅层神经网络

计算神经网络的输出

为了提高程序运算速度，引入向量化和矩阵运算的思想，将上述表达式转换成矩阵运算的形式



$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = \sigma(z^{[1]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(z^{[2]})$$

三. 浅层神经网络

多样本向量化

前面介绍了单个样本的神经网络正向传播矩阵运算过程。而对于m个训练样本，也可以使用矩阵相乘的形式来提高计算效率。而且它的形式与上一部分单个样本的矩阵运算十分相似，比较简单。

使用for循环来求解单样本正向输出：

for i = 1 to m:

$$z^{[1](i)} = W^{[1]} x^{(i)} + b^{[1]}$$
$$a^{[1](i)} = \sigma(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]} a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = \sigma(z^{[2](i)})$$

三. 浅层神经网络

多样本向量化

利用矩阵运算的思想，输入矩阵X的维度为（n_x, m）。
可以把单样本的for循环写成矩阵运算的形式：

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$A^{[1]} = \sigma(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = \sigma(Z^{[2]})$$

向量化：

$$x = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

$$Z^{[1]} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

其中，$Z^{[1]}$的维度是（4,m），4是隐藏层神经元的个数；$A^{[1]}$的维度与$Z^{[1]}$相同；$Z^{[2]}$和$A^{[2]}$的维度均为（1,m）。对上面这四个矩阵来说，均可以这样来理解：行表示神经元个数，列表示样本数目m。

三. 浅层神经网络

多样本向量化的实现解释

矩阵乘列向量得到列向量：

$$W^{[1]}x = \begin{bmatrix} \cdots \\ \cdots \\ \cdots \end{bmatrix} \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ x^{(1)} & x^{(2)} & x^{(3)} & \vdots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ w^{(1)}x^{(1)} & w^{(1)}x^{(2)} & w^{(1)}x^{(3)} & \vdots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} =$$

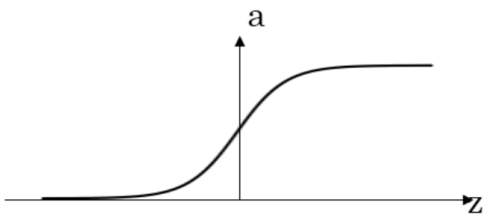
$$\begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} & \vdots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} = Z^{[1]}$$

上面式子中省略了$b^{[1]}$,$b^{[1]}$的维度与$Z^{[1]}$相同，再加上python具有广播的功能，所以可以使得向量b与每一列相加。

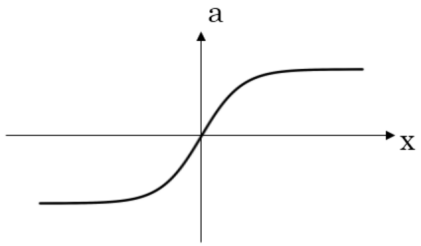
三. 浅层神经网络

激活函数

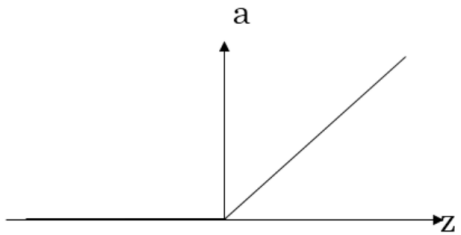
神经网络隐藏层和输出层都需要激活函数（activation function），在之前的内容中默认使用Sigmoid函数σ(x)作为激活函数。其实，还有其它激活函数可供使用，不同的激活函数有各自的优点。
下面介绍几个不同的激活函数g(x)。



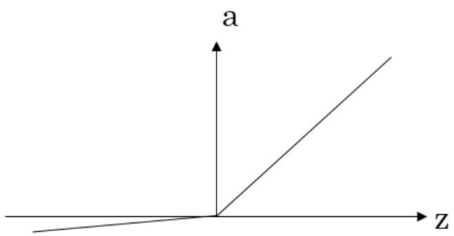
sigmoid: $a = \frac{1}{1+e^{-z}}$



tanh: $a = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$



ReLU: $a = \max(0, z)$

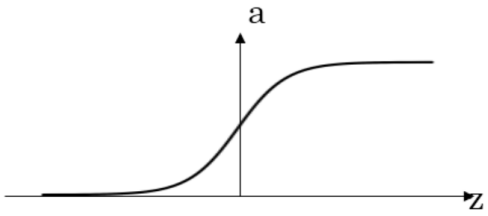


Leaky ReLU: $a = \max(0.01z, z)$

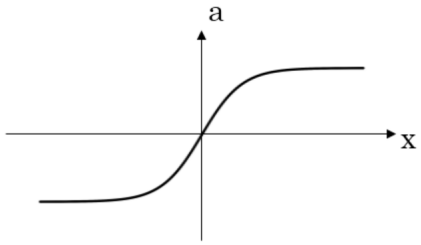
三. 浅层神经网络

激活函数比较

比较sigmoid函数和tanh函数。对于隐藏层的激活函数，一般来说，tanh函数要比sigmoid函数表现更好一些。因为tanh函数的取值范围在[-1,+1]之间，隐藏层的输出被限定在[-1,+1]之间，可以看成是在0值附近分布，均值为0。这样从隐藏层到输出层，数据起到了归一化（均值为0）的效果。因此，隐藏层的激活函数，tanh比sigmoid更好一些。而对于输出层的激活函数，因为二分类问题的输出取值为{0,+1}，所以一般会选择sigmoid作为激活函数。



sigmoid: $a = \frac{1}{1+e^{-z}}$

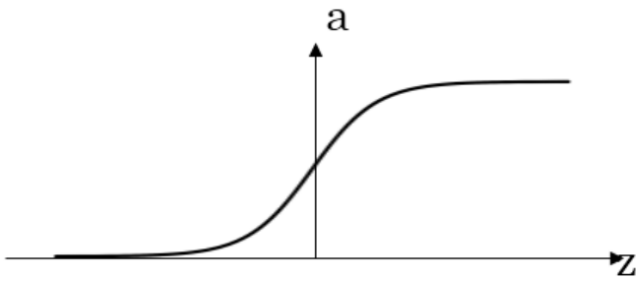


tanh: $a = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$

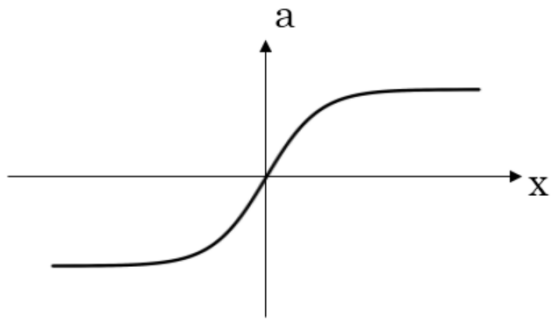
三. 浅层神经网络

激活函数比较

观察sigmoid函数和tanh函数，发现有这样一个问题，就是当|z|很大的时候，激活函数的斜率（梯度）很小。因此，在这个区域内，梯度下降算法会运行得比较慢。在实际应用中，应尽量避免使z落在这个区域，使|z|尽可能限定在零值附近，从而提高梯度下降算法运算速度。



sigmoid: $a = \frac{1}{1+e^{-z}}$

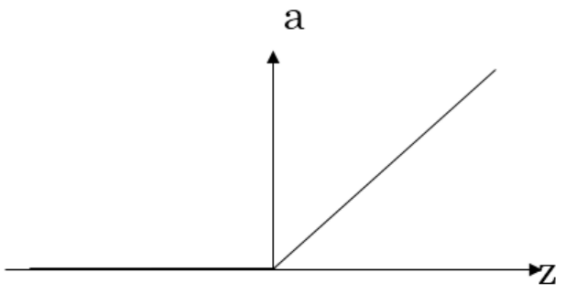


tanh: $a = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$

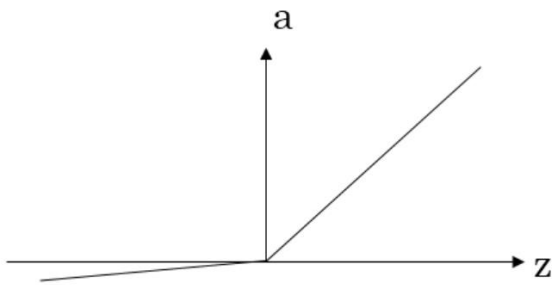
三. 浅层神经网络

激活函数比较

为了弥补sigmoid函数和tanh函数的这个缺陷，就出现了ReLU激活函数。
ReLU激活函数在z大于零时梯度始终为1；在z小于零时梯度始终为0；z等于零时的梯度可以当成1也可以当成0，实际应用中并不影响。
对于隐藏层，选择ReLU作为激活函数能够保证z大于零时梯度始终为1，从而提高神经网络梯度下降算法运算速度。但当z小于零时，存在梯度为0的缺点。
为了弥补这个缺点，出现Leaky ReLU激活函数，能保证z小于零时梯度不为0。



ReLU: $a = \max(0, z)$



Leaky ReLU: $a = \max(0.01z, z)$

三. 浅层神经网络

激活函数使用总结

（1）分类问题，输出层的激活函数一般会选择sigmoid函数。但是隐藏层的激活函数通常不会选择sigmoid函数，tanh函数的表现会比sigmoid函数好一些。
（2）实际应用中，通常会选择使用ReLU或者Leaky ReLU函数，保证梯度下降速度不会太小。
（3）其实，具体选择哪个函数作为激活函数没有一个固定的准确的答案，应该要根据具体实际问题进行验证。

三. 浅层神经网络

为什么需要非线性激活函数？

假设所有的激活函数都是线性的，为了简化计算，我们直接令激活函数g(z)=z，即a=z。那么，浅层神经网络的各层输出为：

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = z^{[1]}$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = z^{[2]}$$

对上式中$a^{[2]}$进行化简计算：

$$a^{[2]} = z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} = W^{[2]}(W^{[1]}x + b^{[1]}) + b^{[2]} = (W^{[2]}W^{[1]})x + (W^{[2]}b^{[1]} + b^{[2]}) = W'x + b'$$

经过推导发现$a^{[2]}$仍是输入变量x的线性组合。即便是包含多层隐藏层的神经网络，如果使用线性函数作为激活函数，最终的输出仍然是输入x的线性模型。这样的话神经网络就没有任何作用了。

因此，隐藏层的激活函数必须要是非线性的。

三. 浅层神经网络

为什么需要非线性激活函数？

（1）如果所有的隐藏层全部使用线性激活函数，只有输出层使用非线性激活函数，那么整个神经网络的结构就类似于一个简单的逻辑回归模型，而失去了神经网络模型本身的优势和价值。

（2）值得一提的是，如果是预测问题而不是分类问题，输出y是连续的情况下，输出层的激活函数可以使用线性函数。如果输出y恒为正值，则也可以使用ReLU激活函数，具体情况，具体分析。

三. 浅层神经网络

激活函数的导数

在梯度下降反向计算过程中少不了计算激活函数的导数即梯度：

Sigmoid函数：

$$g(z) = \frac{1}{1+e^{(-z)}}$$

$$g'(z) = \frac{d}{dz}g(z) = g(z)(1-g(z)) = a(1-a)$$

tanh函数：

$$g(z) = \frac{e^{(z)} - e^{(-z)}}{e^{(z)} + e^{(-z)}}$$

$$g'(z) = \frac{d}{dz}g(z) = 1 - (g(z))^2 = 1 - a^2$$

ReLU函数：

$$g(z) = max(0, z)$$

$$g'(z) = \begin{cases} 0, & z < 0 \\ 1, & z \geq 0 \end{cases}$$

Leaky ReLU函数：

$$g(z) = max(0.01z, z)$$

$$g'(z) = \begin{cases} 0.01, & z < 0 \\ 1, & z \geq 0 \end{cases}$$

三. 浅层神经网络

神经网络的梯度下降

在梯度下降反向计算过程中少不了计算激活函数的导数即梯度：

神经网络正向传播过程为：

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$A^{[1]} = g(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = g(Z^{[2]})$$

其中，g(⍰)表示激活函数。

反向传播是计算导数（梯度）的过程，这里列出成本函数对各个参数的梯度：

$$dZ^{[2]} = A^{[2]} - Y$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$

$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdim = True)$$

$$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g'(Z^{[1]})$$

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^{T}$$

$$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdim = True)$$

三. 浅层神经网络

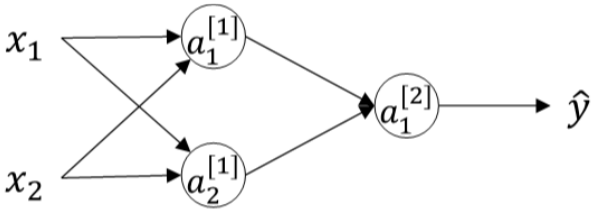
随机初始化的必要性

W不能初始化为零，否则一层中每个单元都做相同的计算，和一个单元没什么区别，b可以初始化为零。例如：

一个浅层神经网络，如下图，包含两个输入，隐藏层包含两个神经元。
如果权重$W^{[1]}$和$W^{[2]}$都初始化为零，即：

$$W^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$W^{[2]} = \begin{bmatrix} 0 & 0 \end{bmatrix}$$



这样使得隐藏层第一个神经元的输出等于第二个神经元的输出，即$a_1^{[1]} = a_2^{[1]}$。经过推导得到$dz_1^{[1]} = dz_2^{[1]}$，以及$dW_1^{[1]} = dW_2^{[1]}$。因此，这样的结果是隐藏层两个神经元对应的权重行向量$W_1^{[1]}$和$W_2^{[1]}$每次迭代更新都会得到完全相同的结果，$W_1^{[1]}$始终等于$W_2^{[1]}$，完全对称。这样隐藏层设置多个神经元就没有任何意义了。值得一提的是，参数b可以全部初始化为零，并不会影响神经网络训练效果。

三. 浅层神经网络

随机初始化的操作

将W进行随机初始化（b可初始化为零），即：

```
W_1 = np.random.randn((2,2))*0.01
b_1 = np.zero((2,1))
W_2 = np.random.randn((1,2))*0.01
b_2 = 0
```

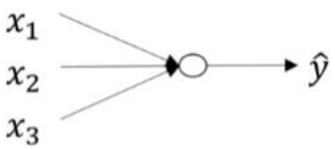
（1）以0.01的目的是尽量使得权重W初始化比较小的值。因为如果使用sigmoid函数或tanh函数作为激活函数，W比较小，得到的|z|也比较小（靠近零点），而零点区域的梯度比较大，这样能大大提高梯度下降算法的更新速度，尽快找到全局最优解。

（2）如果激活函数是ReLU或Leaky ReLU函数，则不需要考虑这个问题。但是，如果输出层是sigmoid函数，则对应的权重W最好初始化到比较小的值。

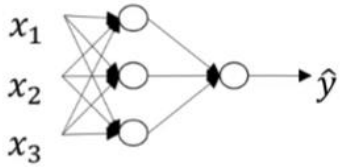
四. 为什么需要神经表示

深层神经网络

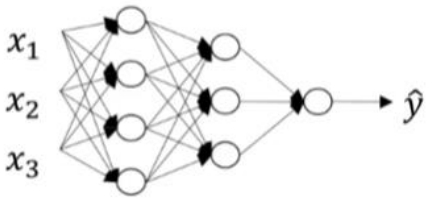
深层神经网络其实就是包含更多的隐藏层神经网络。如下图所示，分别列举了逻辑回归、1个隐藏层的神经网络、2个隐藏层的神经网络和5个隐藏层的神经网络它们的模型结构。



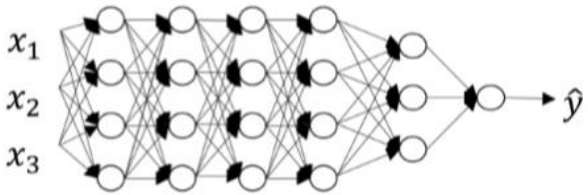
logistic regression



1 hidden layer



2 hidden layers

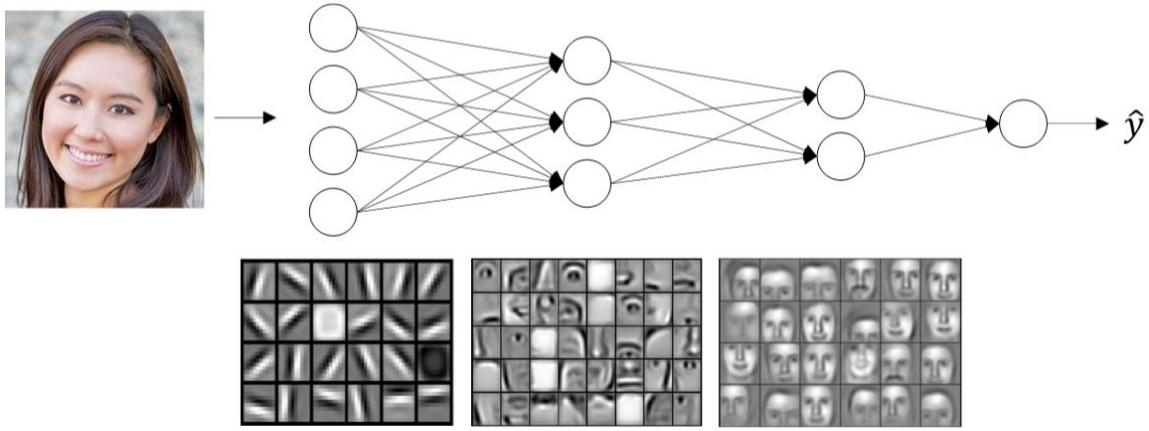


5 hidden layers

四. 为什么需要神经表示

为什么需要深层

（1）神经网络能处理很多问题，而且效果显著。其强大能力主要源自神经网络足够“深”，也就是说网络层数越多，神经网络就更加复杂和深入，学习也更加准确。



（2）深层网络还有另外一个优点，就是能够减少神经元个数，从而减少计算量

五. 参数 VS 超参数

参数和超参数的概念

（1）神经网络中的参数就是我们熟悉的$W^{[l]}$和$b^{[l]}$。

（3）超参数则是例如学习速率α，训练迭代次数N，神经网络层数L，各层神经元个数$n^{[l]}$，激活函数g(z)等。之所以叫做超参数的原因是它们决定了参数$W^{[l]}$和$b^{[l]}$的值。

（3）如何设置最优的超参数是一个比较困难的、需要经验知识的问题。通常的做法是选择超参数一定范围内的值，分别代入神经网络进行训练，测试成本函数随着迭代次数增加的变化，根据结果选择成本函数最小时对应的超参数值。这类似于验证的方法。

总结

一、介绍逻辑回归如何应用在二分类任务中；

二、介绍浅层神经网络。首先简单概述了神经网络的结构：包括输入层，隐藏层和输出层。然后推导了神经网络的正向输出，并以向量化的形式归纳出来；

三、接着介绍不同的激活函数并做了比较，实际应用中根据不同需要选择合适的激活函数。激活函数必须是非线性的，不然神经网络模型起不了任何作用；

四、然后介绍了神经网络的反向传播过程以及各个参数的导数推导，并以矩阵形式表示出来；

五、介绍权重随机初始化的重要性，必须对权重W进行随机初始化操作；

六、介绍为什么需要深层表示；

七、介绍参数与超参数的概念。

谢谢聆听

参考网页：

为什么用Python： https://www.cnblogs.com/lliuye/p/9183914.html
二分类问题：https://blog.csdn.net/weixin_43968778/article/details/88247281
逻辑回归：https://blog.csdn.net/sinat_35473930/article/details/77967260
浅层神经网络：https://blog.csdn.net/red_stone1/article/details/78018269
浅层神经网络2： https://www.cnblogs.com/ys99/p/9286301.html
深层神经网络：https://blog.csdn.net/red_stone1/article/details/78062345