深度学习笔记

视频名称	【斯坦福大学】深度学习（全192讲）吴恩达
视频链接	https://www.bilibili.com/video/BV1ev4y1U7j2/
作业链接	https://blog.csdn.net/u013733326/article/details/79827273
笔记作者	刘天翼

附录

本章将对后续编程作业中常用的函数和语句做出解释

需求：查看矩阵的维度，以及某一维度的长度和数据

查看矩阵的维度，以及某一维度的长度 Python 复制代码

```python
import numpy as np
# 二维矩阵
c = array([[1,1],
           [1,2],
           [1,3],
           [1,4]])
c.shape  # (4, 2)
c.shape[0]  # 4
c.shape[1]  # 2
# 一维矩阵
b =array([1,2,3,4])
b.shape  # (4,)
b.shape[0] # 4
# 也可以简写成
shape([1,2,3,4])  # (4,)
# 获取第一行数据
d = c[0][:]   # [1,1]
```

需求：按元素的矩阵运算：加法，减法，乘法，除法，指数运算，对数运算

按元素的矩阵运算

Python | 复制代码

```python
import numpy as np
A = np.array([[1,1,1,1],
              [2,2,2,2],
              [3,3,3,3]
             ])
A = A-1
A = A+1
A = A/2
A = A*2
A = np.exp(A)
A = np.log(A)
A = np.array([1,2,3,4])
B = np.divide(A,4)
```

需求：矩阵的升维

矩阵的升维

Python | 复制代码

```python
import numpy as np
A = np.array([1,1])
print(A.shape)           # (2,)
# 增加一个列维度
B = A.reshape(2,1)  # (2, 1)
print(B.shape)
print(B)
# 增加一个行维度
C = A.reshape(1,2)   # (1, 2)
print(C.shape)
print(C)
```

需求：矩阵的降维

矩阵的降维

Python | 复制代码

```python
import numpy as np
A = np.arange(64).reshape(4,4,4)
print(A)
# 矩阵降维
B = A.reshape(-1)   # (64,)
print(B.shape)
print(B)
C = A.reshape(4,-1)  # (4, 16)
print(C.shape)
print(C)
```

需求：矩阵默认值

矩阵默认值 Python 复制代码

```python
import numpy as np
# 构造特定形状的全0矩阵
A = np.zeros((2,3))
print(A)
# 构造特定形状的全1矩阵
B = np.ones((2,3))
print(B)


```

需求：矩阵计算

矩阵计算 Python 复制代码

```python
import numpy as np
A = np.array([[1,1],
              [1,1]
             ])
B = np.array([[1,0],
              [0,1]
             ])
# 矩阵点乘
C = np.dot(A,B)
print(C)
# 矩阵求和
print(np.sum(A))    # 4
print(np.sum(B))    # 2
# 求均值
A = np.arange(1,10).reshape(3,3)
print(A)
mean = np.mean(A)
print(mean)
mean_row = np.mean(A,axis = 1)
print(mean_row)    # 2. 5. 8.
mean_col = np.mean(A,axis = 0)
print(mean_col)    # 4. 5. 6.
```

需求：矩阵索引

矩阵索引 Python 复制代码

```python
import numpy as np
# 矩阵索引  从0开始
Test = np.array([
    [1,2,3,4],
    [5,6,7,8],
    [9,10,11,12],
    [13,14,15,16]
])
# 获取第三列的数据
A = Test[:,3]
print(A)
# 获取第二行的数据
B = Test[2,:]
print(B)
```

神经网络和深度学习

深度学习引言

神经网络和监督学习

Supervised Learning

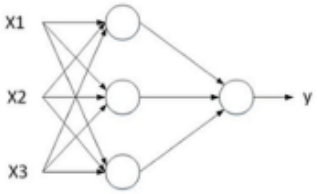
Input(x)	Output(y)	Application
Home features	Price	Real Estate
Ad,user info	Click on ad?(0/1)	Online Advertising
Image	Object(1,⋯,1000)	Photo tagging
Audio	Text transcript	Speech recognition
English	Chinese	Machine translation
Image,Radar info	Position of other cars	Autonomous driving

房价预测和线上广告问题：使用标准的神经网络模型；

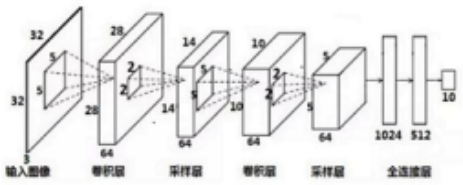
图像识别处理问题：使用**卷积神经网络（Convolution Neural Network），即CNN；**

序列信号处理问题：使用**循环神经网络（Recurrent Neural Network），即RNN；**

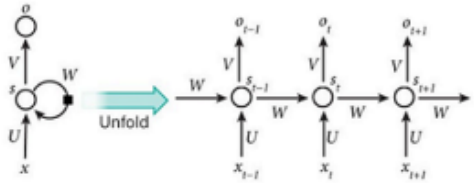
自动驾驶等复杂问题：使用**混合神经网络模型。**



Standard NN

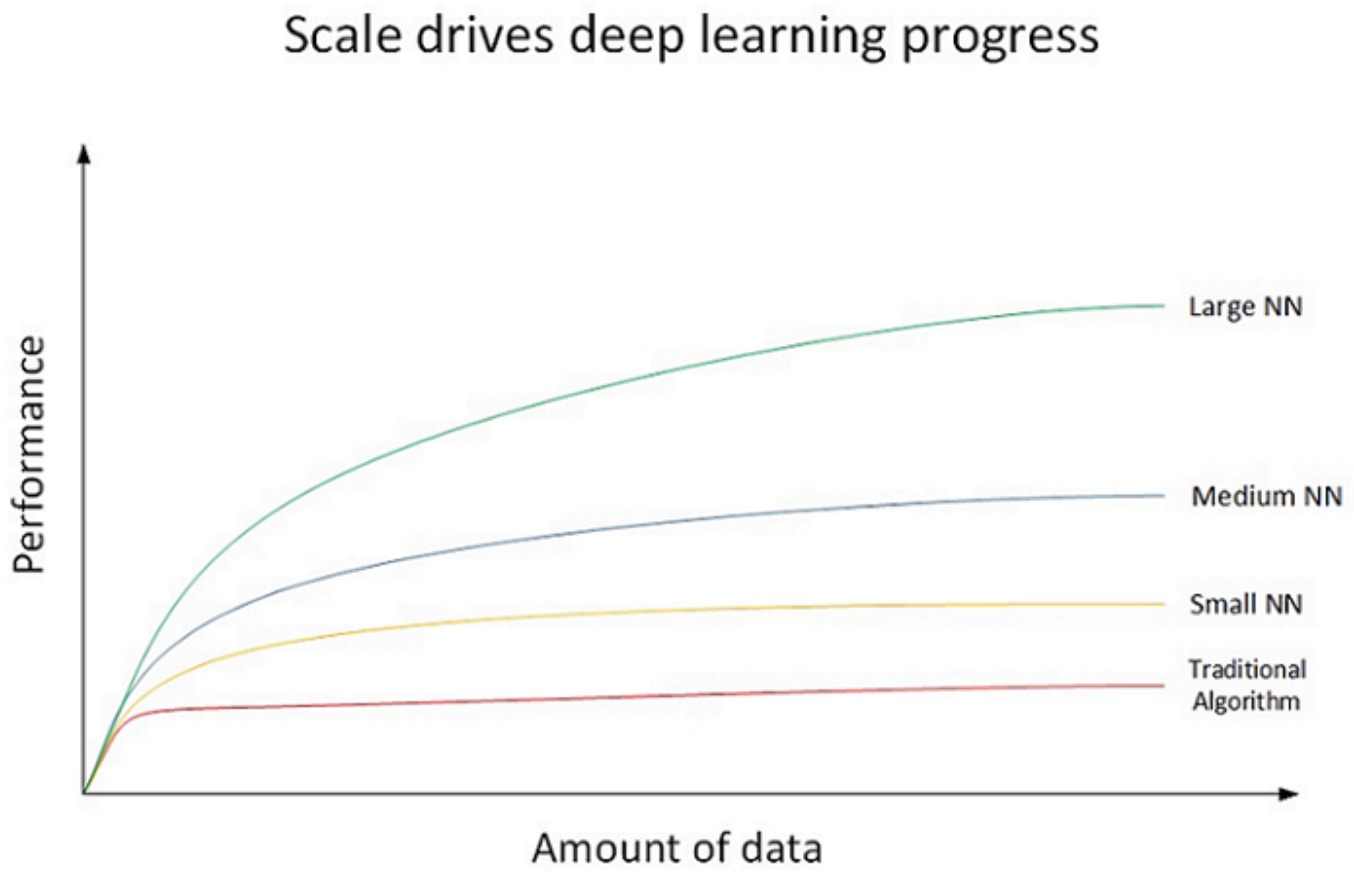


Convolutional NN



Recurrent NN

神经网络的应用场景



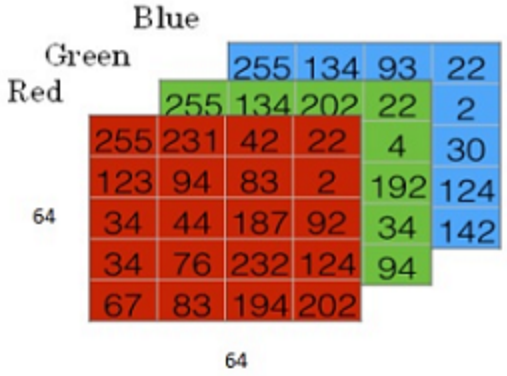
神经网络编程基础

二分类（Binary Classification）

二分类中输出值只有0和1两个离散值，通常使用逻辑回归模型解决二分类问题。

Binary Classification

$$\begin{cases} 1, & cat \\ 0, & not\ cat \end{cases}$$



X_orig = 64x64x3=12288

$$X = \begin{bmatrix} 255 \\ 231 \\ \cdots \\ 255 \\ 134 \\ \cdots \\ 132 \end{bmatrix} \qquad n = n_x = 12288$$

彩色图片包含RGB三个通道。以上述图片为例，输入值的维度是（64，64，3）。将每个通道一行一行取出，再拼接起来转化为一维的**特征向量（featurevector）**，此时输入特征向量维度为（12288，1）。通过该向量预测对应的输出是0还是1这就是一个二分类问题。

逻辑回归（Logistic Regression）

逻辑回归中，预测值用 $\hat{y} = P(y = 1 \mid x)$ 表示。

它的含义是：给定输入特征 x 时，预测标签 $y = 1$ 的概率，其取值范围在[0,1]之间。

当引入线性模型，逻辑回归的预测输出可以完整写成：

$$\hat{y} = Sigmoid(w^T x + b) = \sigma(w^T x + b)$$

Sigmoid函数的一阶导数可以用其自身表示：

$$\sigma\prime(z) = \sigma(z)(1 - \sigma(z))$$

损失函数（Loss Function）

单个样本的Cost Function用Loss Function来表示，使用**平方误差（squared error）**：

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

逻辑回归一般不使用平方误差来作为Loss Function，因为平方误差是非凸的，而非凸函数在使用梯度下降算法时，容易得到局部最小值（local minimum）即局部最优化。而最优化的目标是得到全局最优化（Global optimization），因此Loss Function应该是凸的。

构建另外一种Loss function：

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

当 $y=1$ 时， $L(\hat{y},y)=-log\hat{y}$ ；

如果 $\hat{y}$ 越接近1， $L(\hat{y},y)$ 近似为0，则表示预测效果越好；

如果 $\hat{y}$ 越接近0， $L(\hat{y},y)$ 近似为 $+\infty$ ，则表示预测效果越差；

当 $y=0$ 时， $L(\hat{y},y)=-log(1-\hat{y})$ ；

如果 $\hat{y}$ 越接近1， $L(\hat{y},y)$ 近似为 $+\infty$ ，则表示预测效果越差；

如果 $\hat{y}$ 越接近0， $L(\hat{y},y)$ 近似为0，则表示预测效果越好；

代价函数（Cost Function）

Cost function是m个样本的**Loss function**的平均值，反映了m个样本的预测输出与真实输出的平均接近程度:

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

梯度下降（Gradient Descent）

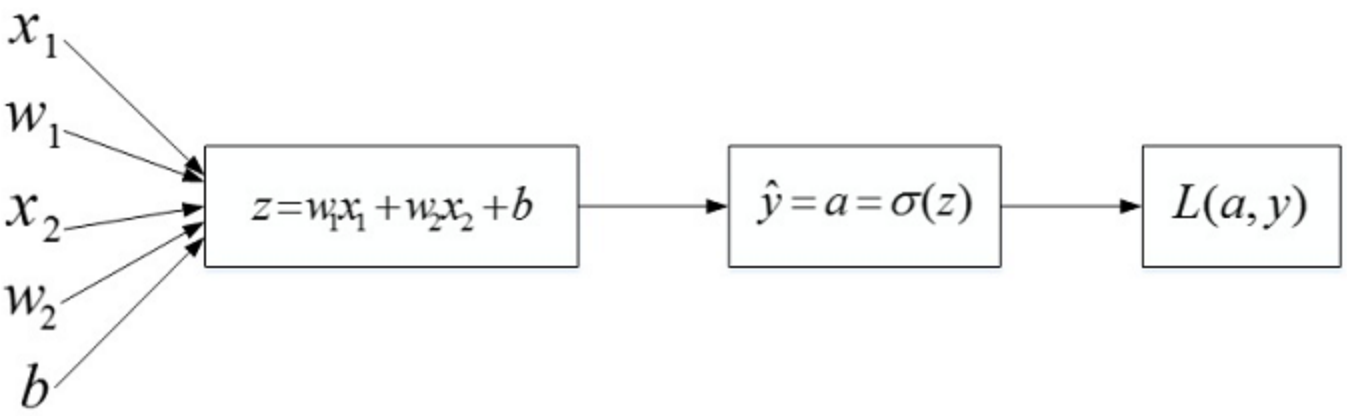
对单个样本而言，**逻辑回归Loss function**表达式如下：

$$z=w^Tx+b$$

$$\hat{y}=a=\sigma(z)$$

$$L(a,y)=-(ylog(a)+(1-y)log(1-a))$$

Logistic regression recap



计算该逻辑回归的反向传播过程:

$$da = \frac{\partial L}{\partial a} = -\frac{y}{a} + \frac{1-y}{1-a}$$

$$dz = \frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} = (-\frac{y}{a} + \frac{1-y}{1-a}) \cdot a(1-a) = a - y$$

$$dw_1 = \frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial w_1} = x_1 \cdot dz = x_1(a-y)$$

$$dw_2 = \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial w_2} = x_2 \cdot dz = x_2(a-y)$$

$$db = \frac{\partial L}{\partial b} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial b} = dz = a - y$$

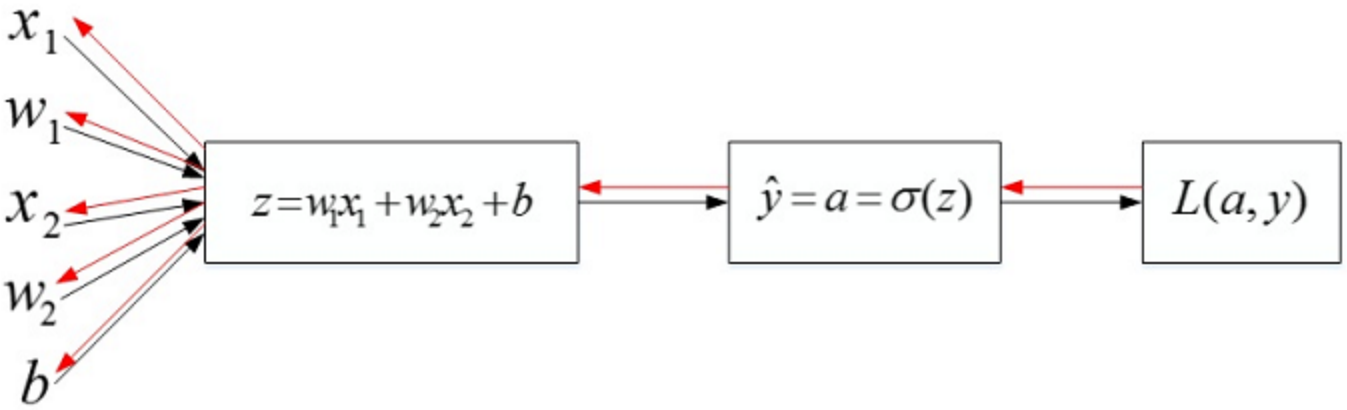
则梯度下降算法可表示为:

$$w_1 := w_1 - \alpha dw_1$$

$$w_2 := w_2 - \alpha dw_2$$

$$b := b - \alpha db$$

Logistic regression derivatives



对m个样本而言，**逻辑回归Cost function**表达式如下：

$$z^{(i)} = w^T x^{(i)} + b$$

$$\hat{y}^{(i)} = a^{(i)} = \sigma(z^{(i)})$$

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

计算该逻辑回归的反向传播过程:

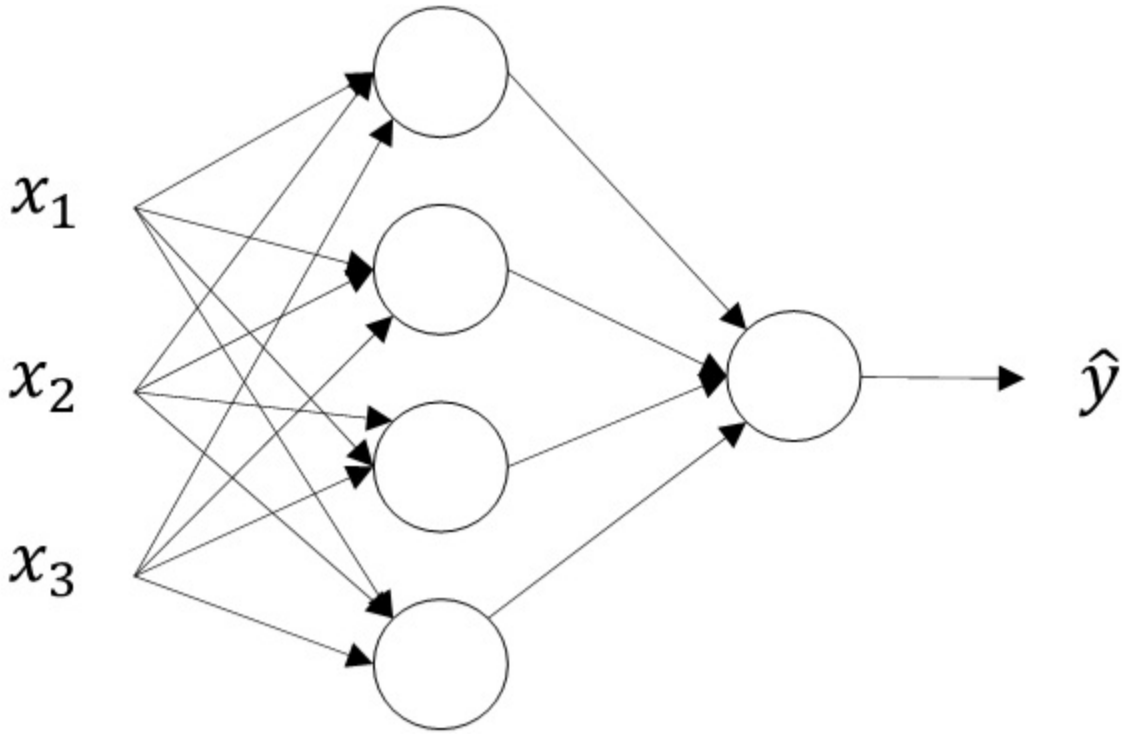
$$dw_m = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial w_m}L(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{m}\sum_{i=1}^{m} x_m^{(i)}(a^{(i)} - y^{(i)})$$

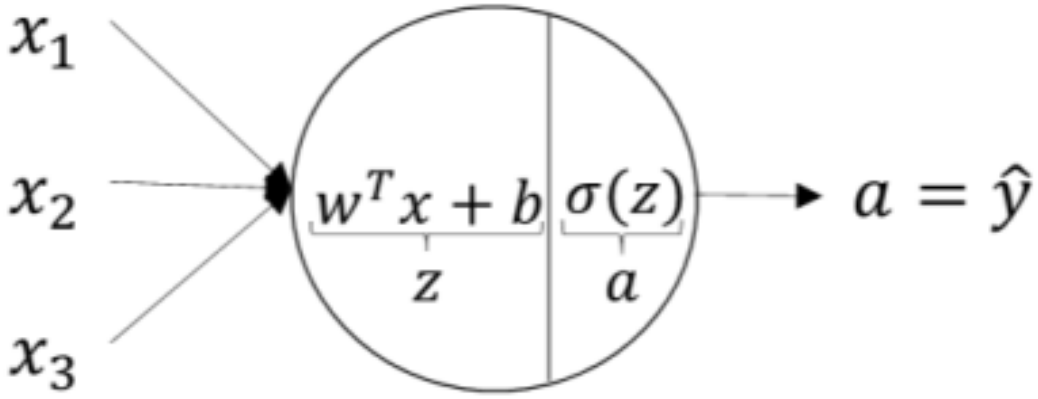
$$db = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial b}L(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{m}\sum_{i=1}^{m}(a^{(i)} - y^{(i)})$$

浅层神经网络

单隐藏层神经网络

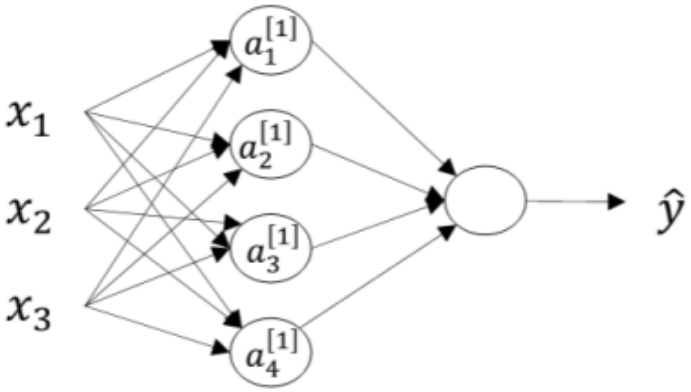
单隐藏层神经网络就是典型的**浅层（shallow）神经网络**





两层神经网络

两层神经网络可以看成是逻辑回归再重复计算一次



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

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

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

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

常见的非线性激活函数

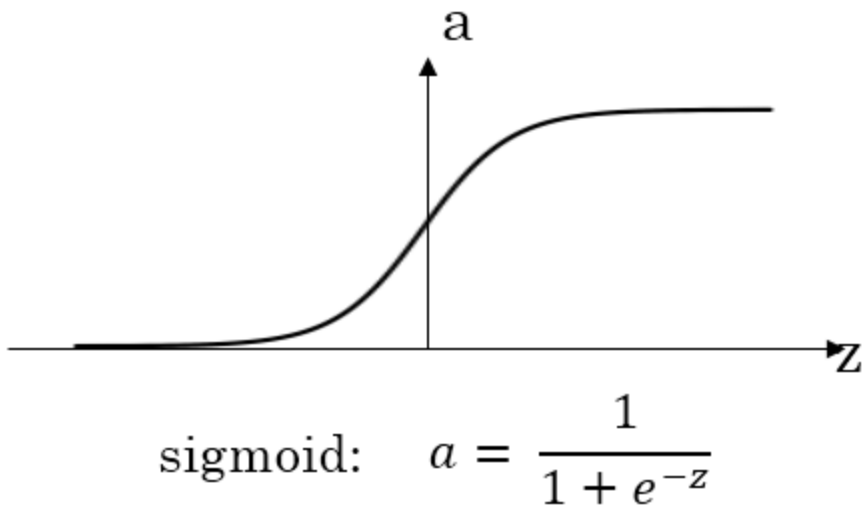
多层隐藏层的神经网络，如果使用线性函数作为激活函数，最终的输出仍然是输入的线性模型：

$$\hat{y} = W_2(W_1x + b_1) + b_2 = W_2W_1x + W_2b_1 + b_2$$

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。

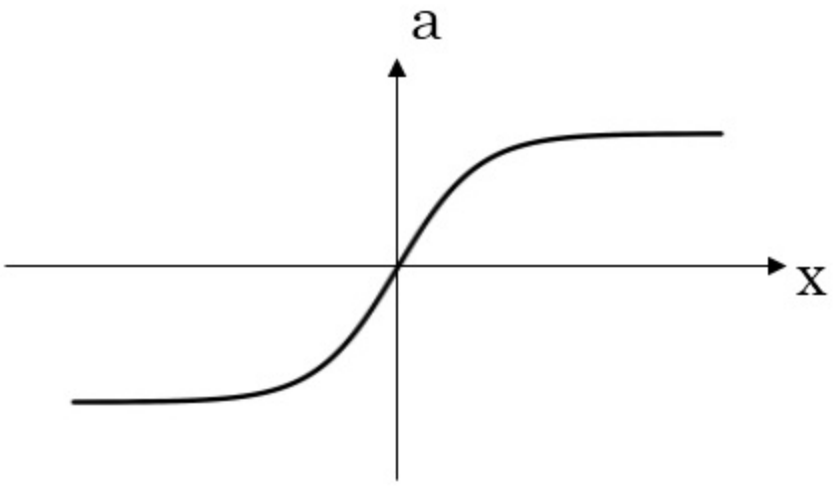
特殊的，对于预测房价等问题可以使用线性激活函数。

sigmoid函数



该函数适合作为输出层的激活函数，因为二分类问题的输出取值为 $\{0, 1\}$ 。

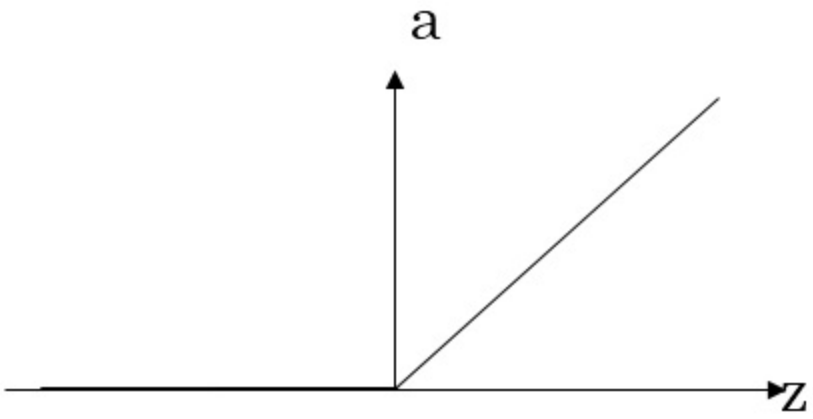
tanh函数



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

该函数适合作为隐藏层的激活函数，因为该函数的取值范围在 $[-1,+1]$ 之间，隐藏层的输出被限定在 $[-1,+1]$ 之间，即有归一化的作用。

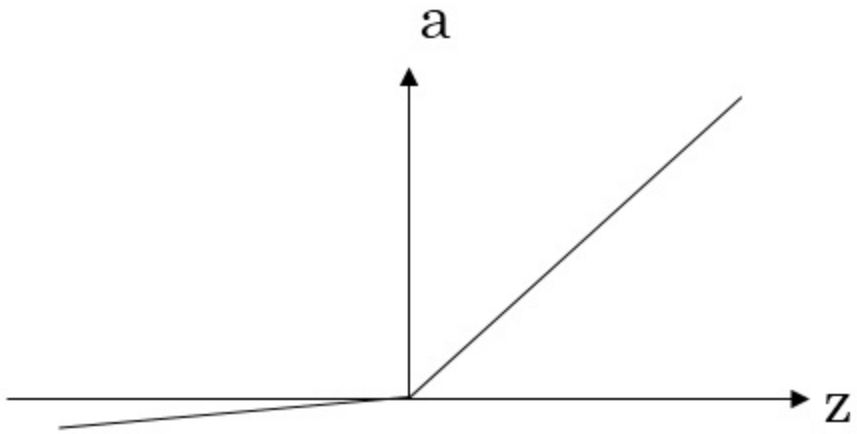
ReLU函数



$$\text{ReLU:} \quad a = \max(0, z)$$

选择该函数作为激活函数能够保证 $z > 0$ 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 $z < 0$ 时，存在梯度为 0 的缺点。如果输出值恒为正值，可以考虑使用该函数作为激活函数。

Leaky ReLU函数

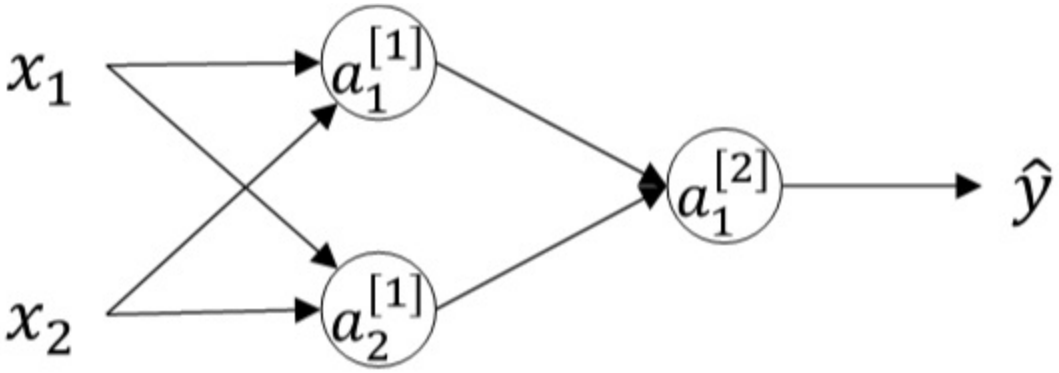


$$\text{Leaky ReLU: } a = \max(0.01z, z)$$

该函数保证了 $z < 0$ 时，梯度不为 0 。

随机初始化

神经网络模型中的参数权重 W 不能全部初始化为零



如果权重 $W^{[1]}$ 和 $W^{[2]}$ 都初始化为零，即：

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

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

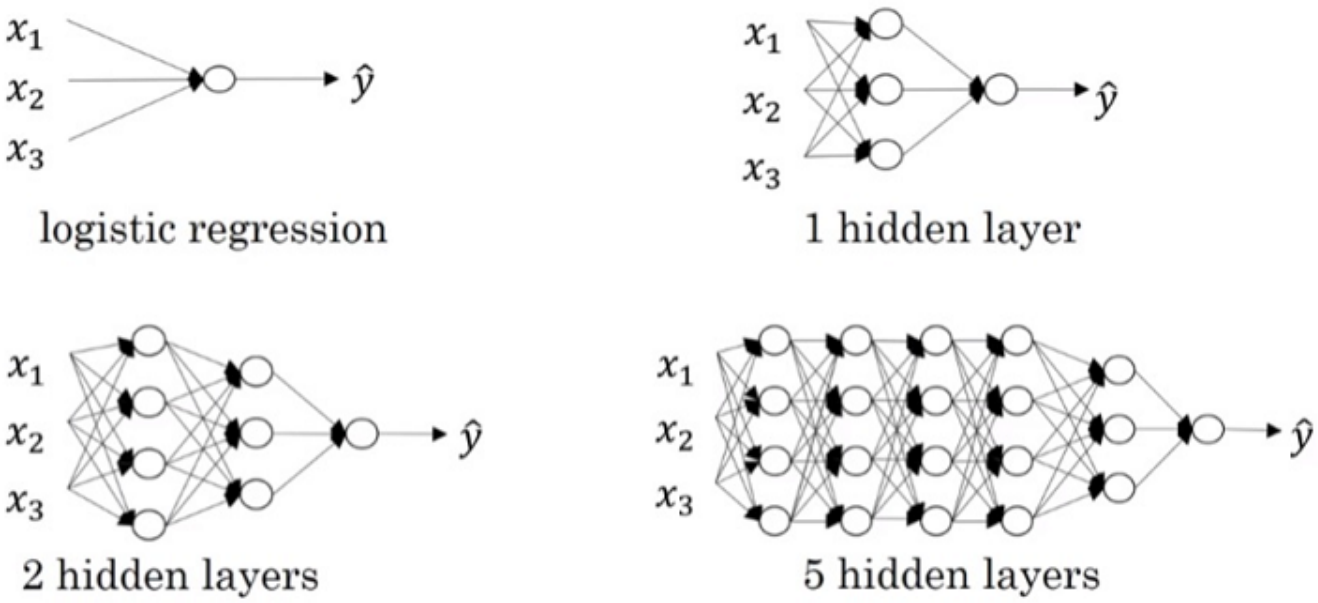
这样会使得隐藏层第一个神经元的输出等于第二个神经元的输出，也会使得 $dW_1^{[1]} = dW_2^{[1]}$ ，最终 $W_1^{[1]}$ 和 $W_2^{[1]}$ 每次迭代更新都会得到完全相同的结果，因此引出随机初始化：

随机初始化 Python 复制代码

```
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
```

深层神经网络

深层神经网络的结构



对于一个 L 层的神经网络，它包含了 $L-1$ 个隐藏层，最后的 L 层是输出层；

$a^{[l]}$ 和 $W^{[l]}$ 的上标 l 均从 1 开始计数 $l=1,\cdots,L$ ；

把输入 x 记为 $a^{[0]}$ ，输出层的 $\hat{y}$ 记为 $a^{[L]}$

第 L 层有： $Z^{[L]}=W^{[L]}A^{[L-1]}+b^{[L]}$ ，其中 $W^{[L]}$ 的维度是 $(n^{[L]},n^{[L-1]})$ ， $b^{[L]}$ 的维度是 $(n^{[L]},1)$

此外 $dW^{[L]}$ 和 $W^{[L]}$ 的维度一致， $db^{[L]}$ 和 $b^{[L]}$ 的维度一致.

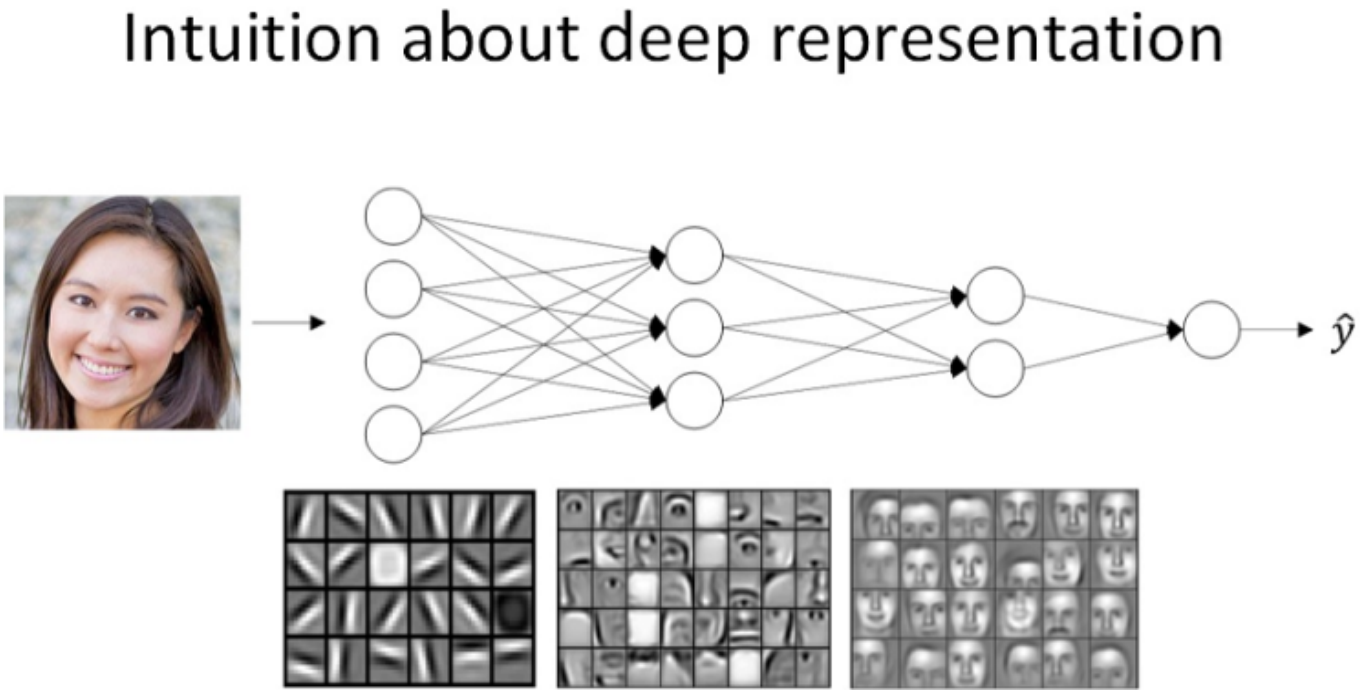
前向传播和反向传播

Summary of gradient descent

$dz^{[2]} = a^{[2]} - y$	$dZ^{[2]} = A^{[2]} - Y$
$dW^{[2]} = dz^{[2]}a^{[1]^T}$	$dW^{[2]} = \frac{1}{m} dZ^{[2]}A^{[1]^T}$
$db^{[2]} = dz^{[2]}$	$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$
$dz^{[1]} = W^{[2]T}dz^{[2]} * g^{[1]\prime}(z^{[1]})$	$dZ^{[1]} = W^{[2]T}dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$
$dW^{[1]} = dz^{[1]}x^T$	$dW^{[1]} = \frac{1}{m} dZ^{[1]}X^T$
$db^{[1]} = dz^{[1]}$	$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)$

深层神经网络的优势

1.人脸识别



神经网络第一层所做的事就是从原始图片中提取出人脸的轮廓与边缘，即**边缘检测**。这样每个神经元得到的是一些边缘信息。神经网络第二层所做的事情就是将前一层的边缘进行组合，组合成人脸一些局部特征，比如眼睛、鼻子、嘴巴等。再往后面，就将这些局部特征组合起来，融合成人脸的模样。

随着层数由浅到深，神经网络提取的特征也是从边缘到局部特征到整体，由简单到复杂。如果隐藏层足够多，那么能够提取的特征就越丰富、越复杂，模型的准确率就会越高。

2.语音识别

浅层的神经元能够检测一些简单的音调，较深的神经元能够检测出基本的音素，更深的神经元就能够检测出单词信息。如果网络够深，还能对短语、句子进行检测。

神经网络从左到右，神经元提取的特征从简单到复杂。特征复杂度与神经网络层数成正相关。特征越来越复杂，功能也越来越强大。

3.神经元个数

处理同一逻辑问题，深层网络所需的神经元个数比浅层网络要少很多处理同一逻辑问题，深层网络所需的神经元个数比浅层网络要少很多。

参数和超参数

参数： $W^{[L]}$ 、 $b^{[L]}$

超参数：学习速率 α 、训练迭代次数、神经网络层数、各层神经元个数，激活函数等

叫做超参数的原因是它们决定了参数的值

如何设置最优的超参数：

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cost function随着迭代次数增加的变化，根据结果选择cost function最小时对应的超参数值

改善深层神经网络

深度学习基础

训练集，测试集，验证集

在机器学习发展的小数据量时代，采用的是：

①70%训练集，30%测试集；

②60%训练集，20%验证集和20%测试集。

在大数据时代，数据量可能是百万级别，采用的是：

①训练集99.5%，验证和测试集各占0.25%，

要确保验证集和测试集的数据来自同一分布

没有测试集也不要紧，测试集的目的是对最终所选定的神经网络系统做出无偏估计，如果不需要无偏估计，也可以不设置测试集。所以如果只有验证集，没有测试集，要做的就是在训练集上训练，尝试不同的模型框架，在验证集上评估这些模型，然后迭代并选出适用的模型。

偏差和方差

理解偏差和方差的两个关键数据是**训练集误差**和**验证集误差**

方差高：假定训练集误差是1%，验证集误差是11%（过拟合）

偏差高：假设训练集误差是15%，验证集误差是16%（欠拟合）

初始模型训练完成后，首先要知道算法的偏差高不高，如果偏差较高，要做的就是增加神经网络的隐藏层个数、神经元个数，训练时间延长，选择其它更复杂的NN模型等。

如果网络足够大，通常可以很好的拟合训练集，一旦偏差降低到可以接受的数值，检查一下方差有没有问题，为了评估方差，要查看验证集性能，从一个性能理想的训练集推断出验证集的性能是否也理想，如果方差高，最好的解决办法就是增加训练样本数据，进行正则化，选择其他更复杂的NN模型。

正则化

在深度学习模型中，L2 regularization的表达式为：

$$J(w^{[1]},b^{[1]},\cdots,w^{[L]},b^{[L]})=\frac{1}{m}\sum_{i=1}^{m}L(\hat{y}^{(i)},y^{(i)})+\frac{\lambda}{2m}\sum_{l=1}^{L}||w^{[l]}||^2$$

$$||w^{[l]}||^2=\sum_{i=1}^{n^{[l]}}\sum_{j=1}^{n^{[l-1]}}(w_{ij}^{[l]})^2$$

梯度更新：

$$dw^{[l]}=dw_{before}^{[l]}+\frac{\lambda}{m}w^{[l]}$$

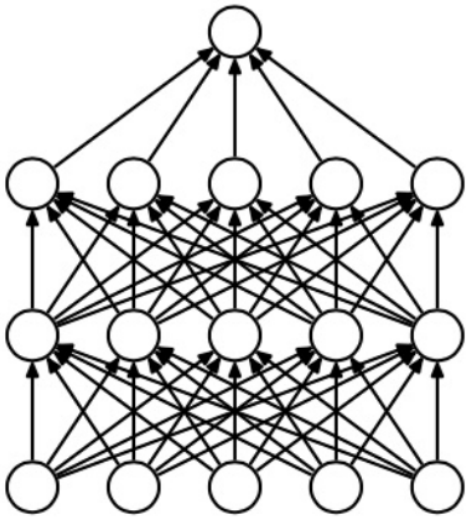
$$w^{[l]}:=w^{[l]}-\alpha\cdot dw^{[l]}$$

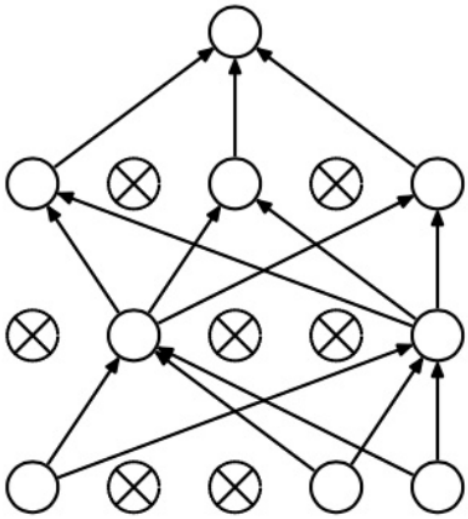
正则化为什么可以预防过拟合:

假设激活函数是$tanh$函数。$tanh$函数的特点是在z接近零的区域，函数近似是线性的，而当$|z|$很大的时候，函数非线性且变化缓慢。当使用正则化，λ较大，即对权重$w^{[l]}$的惩罚较大，$w^{[l]}$减小。因为$z^{[l]}=w^{[l]}a^{[l]}+b^{[l]}$。当$w^{[l]}$减小的时候，$z^{[l]}$也会减小。则此时的$z^{[l]}$分布在$tanh$函数的近似线性区域。那么这个神经元起的作用就相当于是linear regression。如果每个神经元对应的权重$w^{[l]}$都比较小，那么整个神经网络模型相当于是多个linear regression的组合，即可看成一个linear network。得到的分类超平面就会比较简单，不会出现过拟合现象

Dropout

Dropout是指在深度学习网络的训练过程中，对于每层的神经元，按照一定的概率将其暂时从网络中丢弃。即每次训练时，每一层都有部分神经元不工作，起到简化复杂网络模型的效果，从而避免发生过拟合





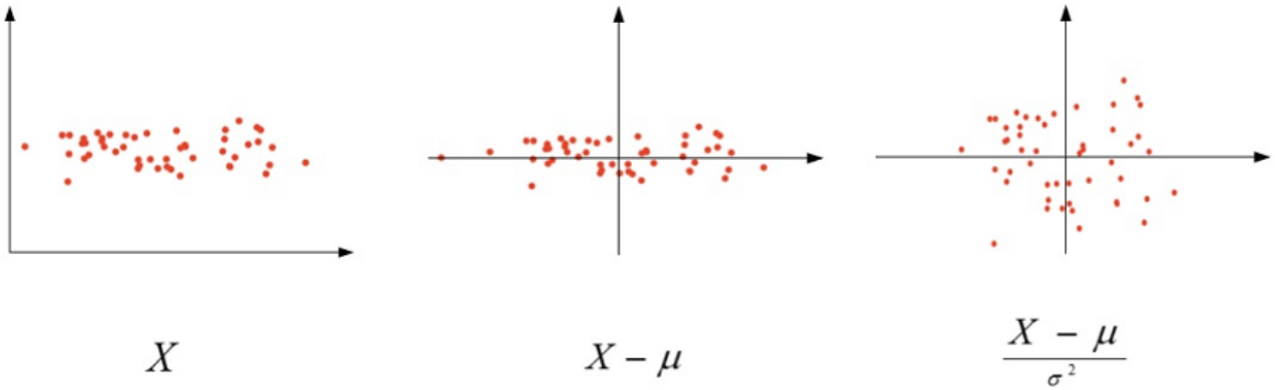
标准化

在训练神经网络时，标准化输入可以提高训练的速度

$$\mu = \frac{1}{m}\sum_{i=1}^{m} X^{(i)}$$

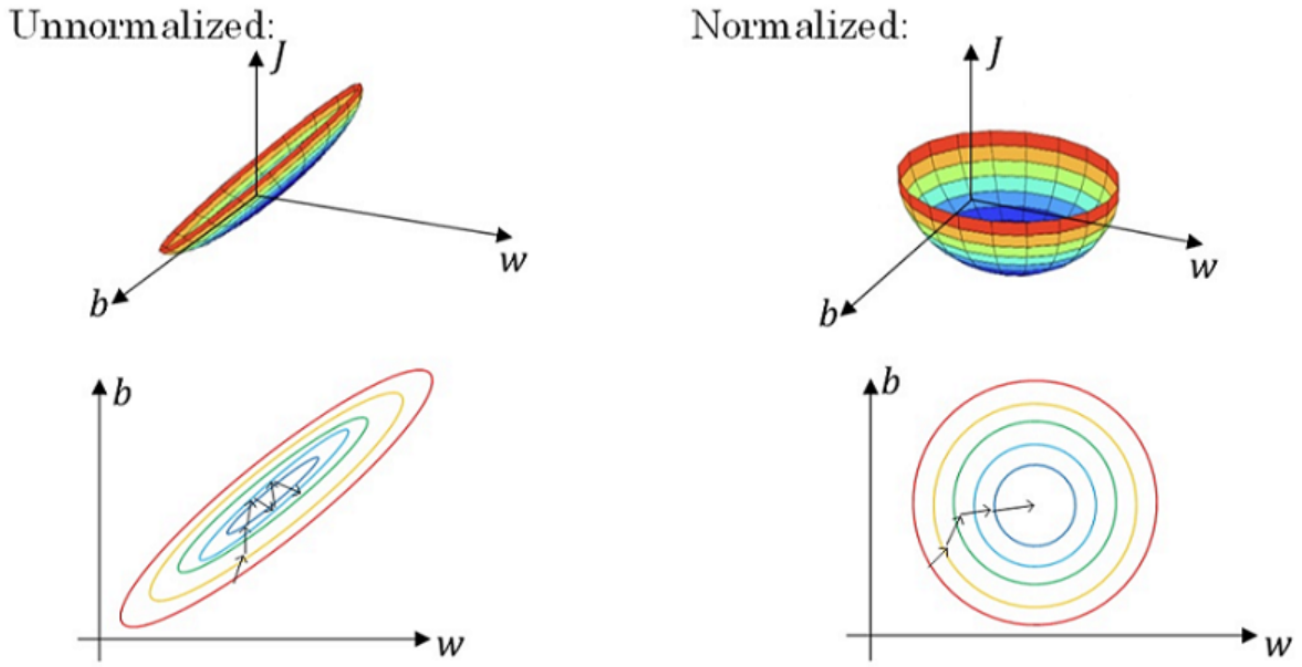
$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} (X^{(i)})^2$$

$$X := \frac{X - \mu}{\sigma^2}$$



由于训练集进行了标准化处理，测试集或在实际应用时，应该使用同样的μ和σ^2对其进行标准化处理。保证训练集和测试集的标准化操作一致

对输入进行标准化操作，是为了让所有输入归一化在同样的尺度上，方便进行梯度下降算法时能够更快更准确地找到全局最优解。假如输入特征是二维的，且x_1的范围是[1,1000]，x_2的范围是[0,1]。如果不进行标准化处理，x_1与x_2之间分布极不平衡，训练得到的w_1和w_2也会在数量级上差别很大。这样导致的结果是costfunction与w和b的关系可能是一个非常细长的椭圆形碗。对其进行梯度下降算法时，由于w_1和w_2数值差异很大，只能选择很小的学习因子α，来避免J发生振荡。一旦α较大，必然发生振荡，J不再单调下降。如果进行了标准化操作，x_1与x_2分布均匀，w_1和w_2数值差别不大，得到的cost function与w和b的关系是类似圆形碗。对其进行梯度下降算法时，α可以选择相对大一些，且J一般不会发生振荡，保证了J是单调下降



如果输入特征之间的范围比较接近，不进行标准化操作没有太大影响

初始化

为了让z不会过大或者过小，w应该越小才好。方法是在初始化w时，令其方差为$\frac{1}{n}$

激活函数是$tanh$相应的python伪代码为：

```
w[l] = np.random.randn(n[l],n[l-1])*np.sqrt(1/n[l-1])
```

如果激活函数是$ReLU$，权重w的初始化一般令其方差为$\frac{2}{n}$：

```
w[l] = np.random.randn(n[l],n[l-1])*np.sqrt(2/n[l-1])
```

另外一种初始化w的方法，令其方差为$\frac{2}{n^{[l-1]}+n^{[l]}}$：

```
w[l] = np.random.randn(n[l],n[l-1])*np.sqrt(2/(n[l-1] + n[l]))
```

结构化机器学习项目

卷积神经网络

自然语言处理