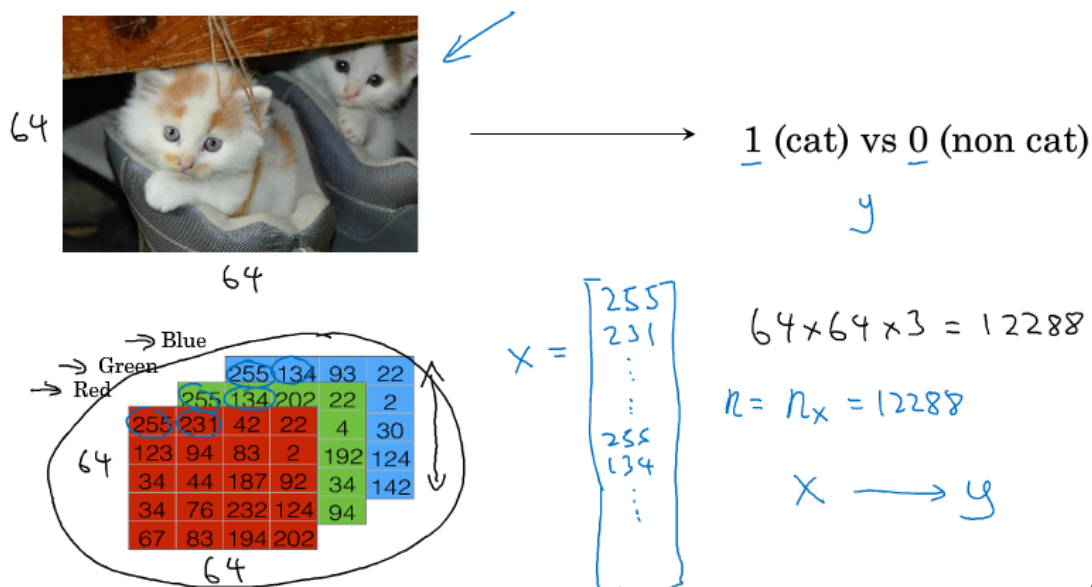


# 神经网络和深度学习--神经网络基础

## 2-1 二分类

### Binary Classification



Andrew Ng

对于二分类问题，大牛给出了一个小的Notation。

- 样本： $(x,y)$ ，训练样本包含 $m$ 个；
- 其中 $x \in \mathbb{R}^{n_x}$ ，表示样本 $x$ 包含 $n_x$ 个特征；
- $y \in \{0,1\}$ ，目标值属于0、1分类；
- 训练数据： $\{(x(1),y(1)),(x(2),y(2)),\dots,(x(m),y(m))\}$

# Notation

$$(x, y) \quad x \in \mathbb{R}^{n_x}, y \in \{0, 1\}$$

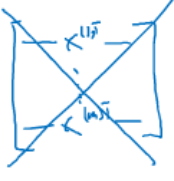
$$m \text{ training examples: } \{(\underline{x}^{(1)}, \underline{y}^{(1)}), (\underline{x}^{(2)}, \underline{y}^{(2)}), \dots, (\underline{x}^{(m)}, \underline{y}^{(m)})\}$$

$$M = M_{\text{train}}$$

$$M_{\text{test}} = \# \text{test examples.}$$

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \end{bmatrix} \begin{matrix} \updownarrow \\ n_x \end{matrix}$$

$\xleftarrow{\quad m \quad} \quad \xrightarrow{\quad}$

$$X \in \mathbb{R}^{n_x \times m} \quad X.\text{shape} = (n_x, m)$$


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

$$Y \in \mathbb{R}^{1 \times m}$$

$$Y.\text{shape} = (1, m)$$

Andrew Ng

## 2.2 逻辑回归

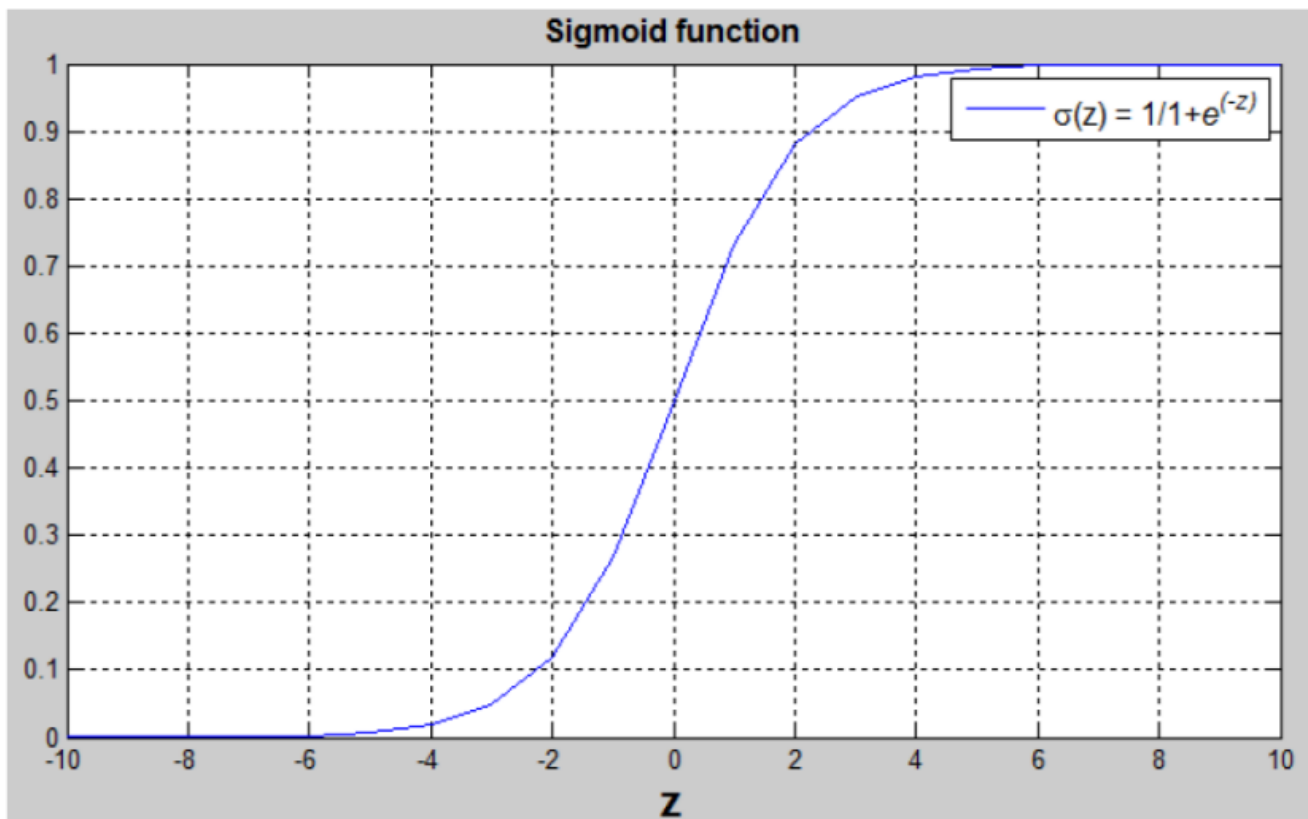
逻辑回归是一个用于监督学习问题的学习算法，输出值不是0就是1。

逻辑回归中用到的参数：

The parameters used in Logistic regression are:

- The input features vector:  $x \in \mathbb{R}^{n_x}$ , where  $n_x$  is the number of features
- The training label:  $y \in 0, 1$
- The weights:  $w \in \mathbb{R}^{n_x}$ , where  $n_x$  is the number of features
- The threshold:  $b \in \mathbb{R}$
- The output:  $\hat{y} = \sigma(w^T x + b)$
- Sigmoid function:  $s = \sigma(w^T x + b) = \sigma(z) = \frac{1}{1 + e^{-z}}$

sigmold函数的图像



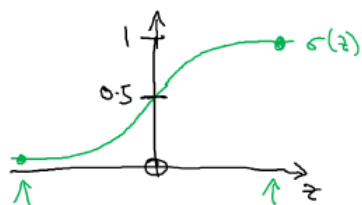
- 如果 $z$ 趋向于比较大的正数，那么sigmoid等于1。
- 如果 $z$ 趋向于比较大的负数，那么sigmoid等于0。
- 如果 $z$ 等于零，那么sigmoid等于0.5。

## Logistic Regression

Given  $x$ , want  $\hat{y} = \frac{P(y=1|x)}{P(y=0|x)}$   
 $x \in \mathbb{R}^{n_x}$   $0 \leq \hat{y} \leq 1$

Parameters:  $\underline{w} \in \mathbb{R}^{n_x}$ ,  $\underline{b} \in \mathbb{R}$ .

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



$$x_0 = 1, \quad x \in \mathbb{R}^{n_x+1}$$

$$\hat{y} = \sigma(\theta^T x)$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{n_x} \end{bmatrix} \begin{matrix} \leftarrow b \\ \leftarrow w \end{matrix}$$

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

$$\text{If } z \text{ large } \sigma(z) \approx \frac{1}{1+0} = 1$$

If  $z$  large negative number

$$\sigma(z) = \frac{1}{1 + e^{-z}} \approx \frac{1}{1 + \text{Big num}} \approx 0$$

Andrew Ng

# Logistic Regression cost function

$$\rightarrow \hat{y}^{(i)} = \sigma(w^T x^{(i)} + b), \text{ where } \sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}} \quad z^{(i)} = w^T x^{(i)} + b$$

Given  $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$ , want  $\hat{y}^{(i)} \approx y^{(i)}$ .

$x^{(i)}$   
 $y^{(i)}$   
 $z^{(i)}$  *i-th example.*

**Loss** (error) function:

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

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

If  $y=1$ :  $\mathcal{L}(\hat{y}, y) = -\log \hat{y} \leftarrow$  want  $\log \hat{y}$  large, want  $\hat{y}$  large.

If  $y=0$ :  $\mathcal{L}(\hat{y}, y) = -\log(1-\hat{y}) \leftarrow$  want  $\log(1-\hat{y})$  large ... want  $\hat{y}$  small

$$\text{Cost function: } J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)})]$$

Andrew Ng

## 2.3 逻辑回归损失函数

损失函数的公式

Loss (error) function:

The loss function measures the discrepancy between the prediction ( $\hat{y}^{(i)}$ ) and the desired output ( $y^{(i)}$ ). In other words, the loss function computes the error for a single training example.

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

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

- If  $y^{(i)} = 1$ :  $L(\hat{y}^{(i)}, y^{(i)}) = -\log(\hat{y}^{(i)})$  where  $\log(\hat{y}^{(i)})$  and  $\hat{y}^{(i)}$  should be close to 1
- If  $y^{(i)} = 0$ :  $L(\hat{y}^{(i)}, y^{(i)}) = -\log(1 - \hat{y}^{(i)})$  where  $\log(1 - \hat{y}^{(i)})$  and  $\hat{y}^{(i)}$  should be close to 0

但是，对于logistic regression 来说，一般不适用平方错误来作为Loss Function，这是因为上面的平方错误损失函数一般是非凸函数 (non-convex)，其在使用低度下降算法的时候，容易得到局部最优解，而不是全局最优解。因此要选择凸函数。

- 我们的目标是最小化样本点的损失Loss Function，损失函数是针对单个样本点的。

代价函数

## Cost function

The cost function is the average of the loss function of the entire training set. We are going to find the parameters  $w$  and  $b$  that minimize the overall cost function.

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

代价函数是损失函数的平均值。

## 2.4 梯度下降法

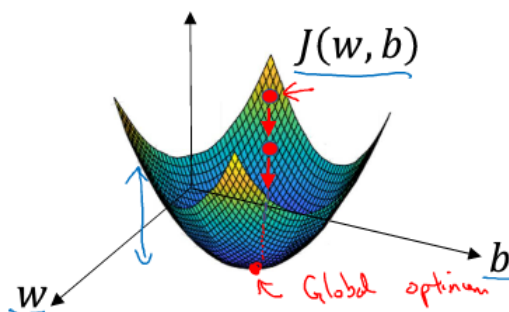
用梯度下降法 ( Gradient Descent ) 算法来最小化Cost function , 以计算出合适的 $w$ 和 $b$ 的值。

### Gradient Descent

Recap:  $\hat{y} = \sigma(w^T x + b)$ ,  $\sigma(z) = \frac{1}{1+e^{-z}}$  ←

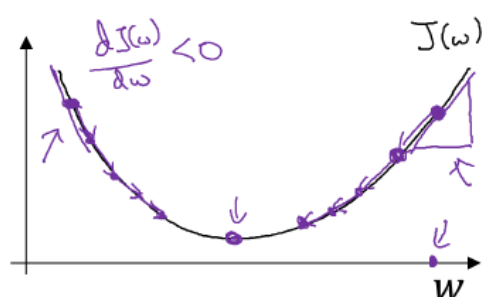
$$J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

Want to find  $w, b$  that minimize  $J(w, b)$



Andrew Ng

## Gradient Descent



Repeat {  
 $\omega := \omega - \alpha \frac{\partial J(\omega)}{\partial \omega}$   
 }  
 $\omega := \omega - \alpha \underline{dw}$

learning rate

"dw"

$\frac{\partial J(\omega)}{\partial \omega} = ?$

$J(w,b)$

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

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

$\frac{\partial J(w,b)}{\partial w}$

$\frac{\partial J(w,b)}{\partial b}$

"partial derivative"  
J

$d\omega$

$db$

Andrew Ng

每次迭代更新的修正表达式:

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

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

在程序代码中，我们通常使用

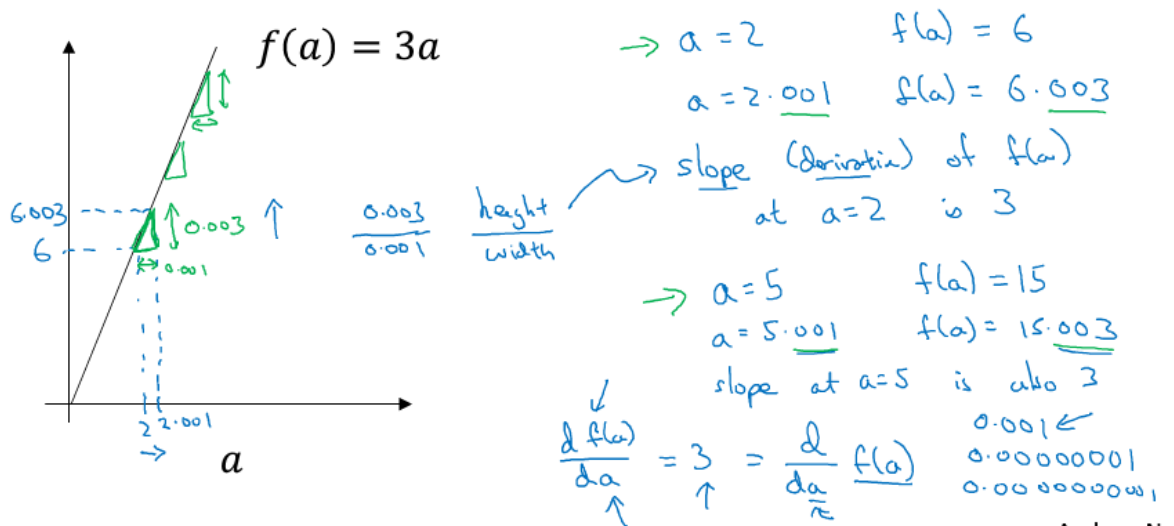
用dw来表示 $\frac{\partial J(w,b)}{\partial w}$ ，用db来表示 $\frac{\partial J(w,b)}{\partial b}$ 。

## 2.5 基本的神经网络编程--导数

## 理解导数

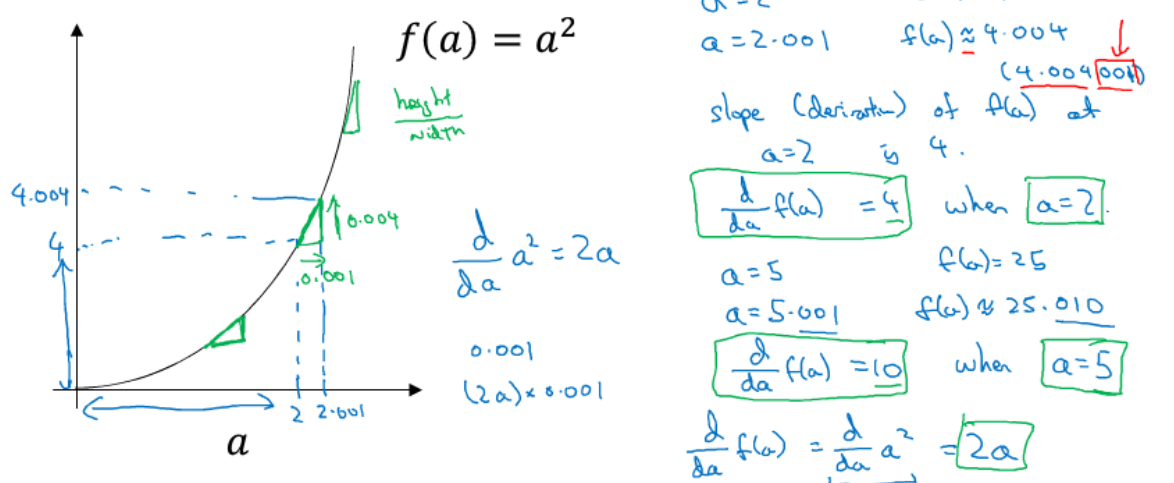
导数的直观理解就是经过某点的直线的斜率

## Intuition about derivatives



## 2.6 更过导数的例子

## Intuition about derivatives



更多倒数的例子

## More derivative examples

$$f(a) = a^2 \quad \frac{d}{da} f(a) = \frac{2a}{4}$$

$$a = 2 \quad f(a) = 4$$

$$a = 2.001 \quad f(a) \approx 4.004$$

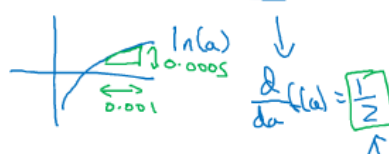
$$f(a) = a^3 \quad \frac{d}{da} f(a) = \frac{3a^2}{3 \times 2^2 = 12}$$

$$a = 2 \quad f(a) = 8$$

$$a = 2.001 \quad f(a) \approx 8.012$$

$$f(a) = \log_e(a)$$

$$\ln(a)$$

$$\frac{d}{da} f(a) = \frac{1}{a}$$


$$\frac{d}{da} f(a) = \frac{1}{2}$$

$$a = 2 \quad f(a) \approx 0.69315$$

$$a = 2.001 \quad f(a) \approx 0.69365$$

$$\Delta a = 0.001 \quad \Delta f(a) \approx 0.0005$$

Andrew Ng

## 2.7 计算图

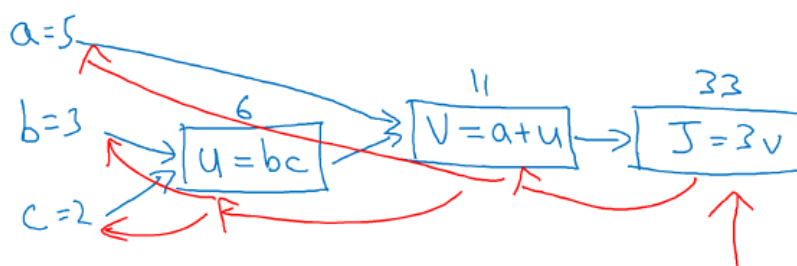
### Computation Graph

$$J(a, b, c) = 3(a + \underbrace{bc}_u) = 3(5 + \underbrace{3 \times 2}_v) = 33$$

$$u = bc$$

$$v = a + u$$

$$J = 3v$$

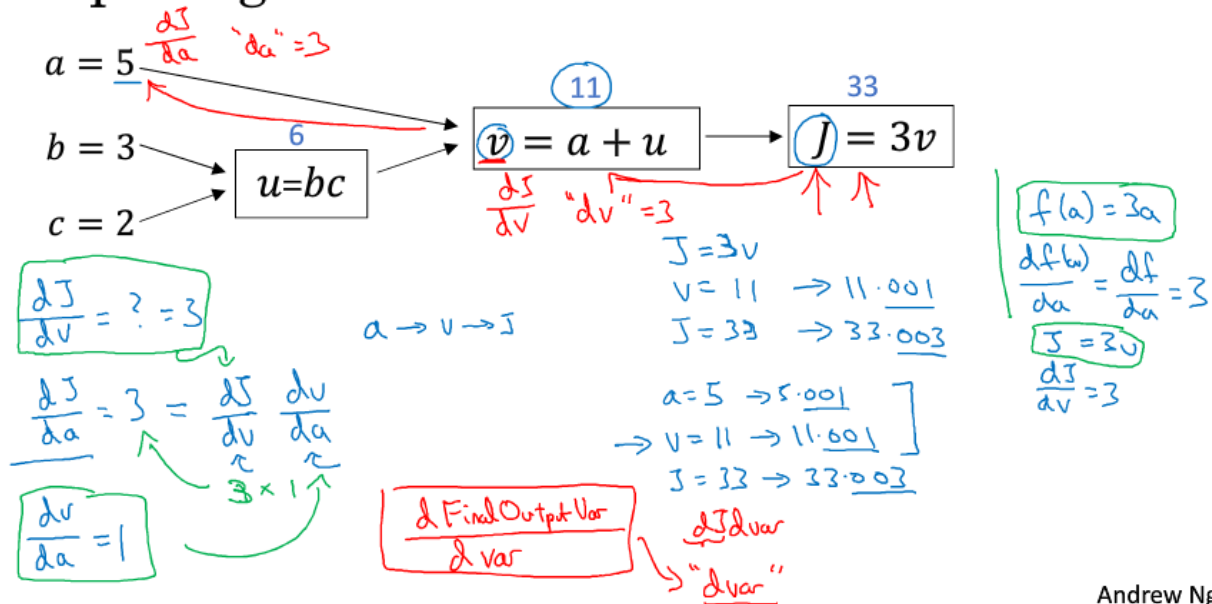


Andrew Ng

链式法则求导



## Computing derivatives



Andrew Ng

## 2.9 逻辑回归中的梯度下降法

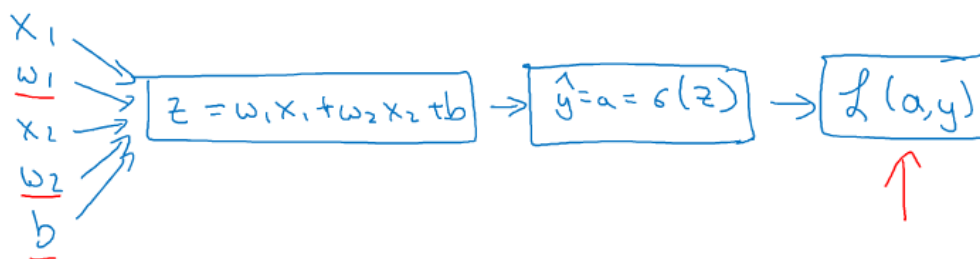
## 逻辑回归损失函数回顾

## Logistic regression recap

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

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

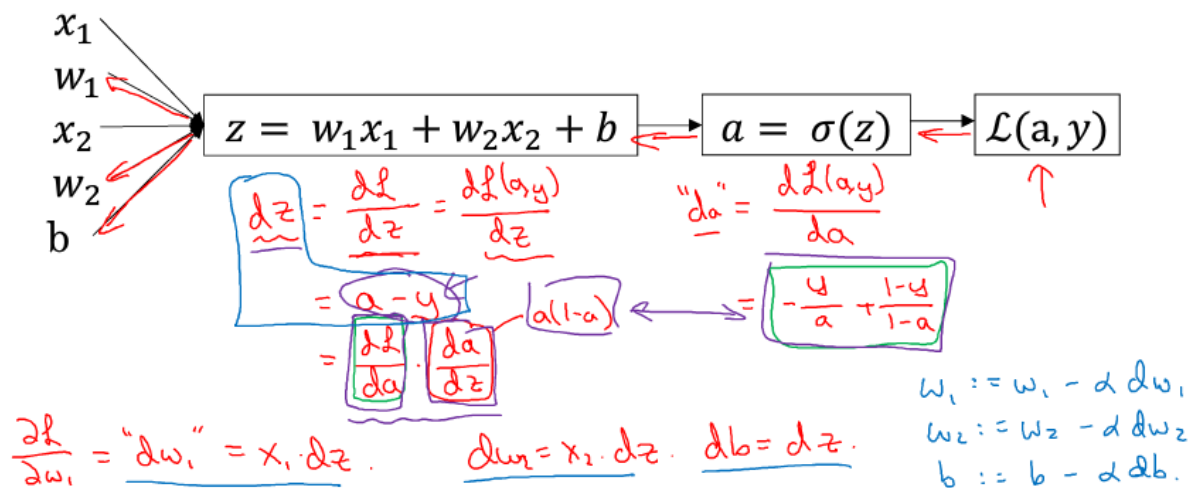
$$\rightarrow \mathcal{L}(a, y) = -(y \log(a) + (1 - y) \log(1 - a))$$



Andrew Ng

## 逻辑回归求导

# Logistic regression derivatives



Andrew Ng

## 2-10 m个样本的梯度下降法

### Logistic regression on $m$ examples

Initialization:  $J=0; \underline{dw_1}=0; \underline{dw_2}=0; \underline{db}=0$

For  $i=1$  to  $m$ :

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

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

$$J += -[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})]$$

$$dz^{(i)} = a^{(i)} - y^{(i)}$$

$$\begin{cases} dw_1 += x_1^{(i)} dz^{(i)} \\ dw_2 += x_2^{(i)} dz^{(i)} \\ db += dz^{(i)} \end{cases} \quad n=2$$

End For

Update parameters:

$$dw_1 /= m; \quad dw_2 /= m; \quad db /= m.$$

Vectorization:

$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial w_1}$$

$$w_1 := w_1 - \alpha \frac{\partial J}{\partial w_1}$$

$$w_2 := w_2 - \alpha \frac{\partial J}{\partial w_2}$$

$$b := b - \alpha \frac{\partial J}{\partial b}$$

Andrew Ng

对 $m$ 个样本来说，其Cost function表达式如下：

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

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

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

Cost function 关于w和b的偏导数可以写成所有样本点偏导数和的平均形式：

$$dw_1 = \frac{1}{m} \sum_{i=1}^m x_1^{(i)} (a^{(i)} - y^{(i)})$$

$$db = \frac{1}{m} \sum_{i=1}^m (a^{(i)} - y^{(i)})$$

## 2.14 向量化logistics回归的梯度输出

在深度学习的算法中，我们通常拥有大量的数据，在程序的编写过程中，应该尽最大可能的少使用loop循环语句，利用python可以实现矩阵运算，进而来提高程序的运行速度，避免for循环的使用。

什么是向量化？

### What is vectorization?

$$z = \underline{w^T x} + b$$

Non-vectorized:

$$z = 0$$

```
for i in range(n-x):  
    z += w[i] * x[i]
```

$$z += b$$

$$w = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix} \quad x = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix} \quad \begin{matrix} w \in \mathbb{R}^{n_x} \\ x \in \mathbb{R}^{n_x} \end{matrix}$$

Vectorized

$$z = \underbrace{\text{np.dot}(w, x)}_{w^T x} + b$$

→ GPU } SIMD - single instruction  
→ CPU } multiple data.

### 逻辑回归向量化

输入矩阵X：(  $n_x, m$  )

权重矩阵w：(  $n_x, 1$  )

偏置b：为一个常数

输出矩阵Y：(  $1, m$  )

所有m个样本的线性输出Z可以用矩阵表示：

$$Z = w^T X + b$$

python 代码

```
Z = np.dot(w,T,X) + b
A = sigold(Z)
```

## 2.12 更多的向量化的例子--神经网络

### Neural network programming guideline

Whenever possible, avoid explicit for-loops.

$$u = Av$$

$$u_i = \sum_j A_{ij} v_j$$

$u = \text{np.zeros}(n, 1)$   
 for i ...  
   for j ...  
      $u[i] += A[i][j] * v[j]$

$$u = \text{np.dot}(A, v)$$

### Vectors and matrix valued functions

Say you need to apply the exponential operation on every element of a matrix/vector.

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \rightarrow u = \begin{bmatrix} e^{v_1} \\ e^{v_2} \\ \vdots \\ e^{v_n} \end{bmatrix}$$

$\rightarrow u = \text{np.zeros}(n, 1)$   
 $\rightarrow \text{for } i \text{ in range}(n):$   
    $\rightarrow u[i] = \text{math.exp}(v[i])$

import numpy as np  
 $u = \text{np.exp}(v)$   
 $\text{np.log}(v)$   
 $\text{np.abs}(v)$   
 $\text{np.maximum}(v, 0)$   
 $v ** 2$        $1/v$

# Logistic regression derivatives

$J = 0, \text{ ~~dw1 = 0, dw2 = 0~~, db = 0}$        $dw = \text{np.zeros}((n-x, 1))$   
 $\rightarrow$  for i = 1 to n:  
 $z^{(i)} = w^T x^{(i)} + b$   
 $a^{(i)} = \sigma(z^{(i)})$   
 $J += -[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$   
 $dz^{(i)} = a^{(i)}(1 - a^{(i)})$   
 $\downarrow$  for  $j=1 \dots n_x$   
 $\left[ \begin{array}{l} \text{~~dw1 += x}_1^{(i)} dz^{(i)}~~ \\ \text{~~dw2 += x}_2^{(i)} dz^{(i)}~~ \end{array} \right] \quad n_x = 2 \quad dw += x^{(i)} dz^{(i)}$   
 $db += dz^{(i)}$   
 $J = J/m, \text{ ~~dw1 = dw1/m, dw2 = dw2/m~~, db = db/m}$   
 $dw /= m.$

## 2.13 向量化逻辑回归的梯度计算

### Vectorizing Logistic Regression

$$dz^{(1)} = a^{(1)} - y^{(1)} \quad dz^{(2)} = a^{(2)} - y^{(2)} \quad \dots$$

$$\boxed{dz} = \begin{bmatrix} dz^{(1)} & dz^{(2)} & \dots & dz^{(m)} \end{bmatrix}_{1 \times m} \quad \leftarrow$$

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

$$\rightarrow dz = A - Y = \begin{bmatrix} a^{(1)} - y^{(1)} & a^{(2)} - y^{(2)} & \dots \end{bmatrix}$$

$$\rightarrow dw = 0$$

$$\begin{bmatrix} dw += x^{(1)} dz^{(1)} \\ dw += x^{(2)} dz^{(2)} \\ \vdots \end{bmatrix}$$

$$dw /= m$$

$$db = 0$$

$$\begin{bmatrix} db += dz^{(1)} \\ db += dz^{(2)} \\ \vdots \\ db += dz^{(m)} \end{bmatrix}$$

$$db /= m$$

$$db = \frac{1}{m} \sum_{i=1}^m dz^{(i)}$$

$$= \frac{1}{m} \text{np.sum}(dz)$$

$$dw = \frac{1}{m} X dz^T$$

$$= \frac{1}{m} \begin{bmatrix} x^{(1)} & \dots & x^{(m)} \\ 1 & & 1 \end{bmatrix} \begin{bmatrix} dz^{(1)} \\ \vdots \\ dz^{(m)} \end{bmatrix}$$

$$= \frac{1}{m} [x^{(1)} dz^{(1)} + \dots + x^{(m)} dz^{(m)}]$$

$n \times 1$

Andrew Ng

# Implementing Logistic Regression

$J = 0, dw_1 = 0, dw_2 = 0, db = 0$

for  $i = 1$  to  $m$ :

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

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

$$J += -[y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log(1 - a^{(i)})]$$

$$dz^{(i)} = a^{(i)} - y^{(i)} \leftarrow$$

$$\left[ \begin{array}{l} dw_1 += x_1^{(i)} dz^{(i)} \\ dw_2 += x_2^{(i)} dz^{(i)} \end{array} \right] \leftarrow dw += x^{(i)} * dz^{(i)}$$

$$db += dz^{(i)}$$

$$J = J/m, dw_1 = dw_1/m, dw_2 = dw_2/m$$

$$db = db/m$$

for  $iter$  in  $range(1000)$ :  $\leftarrow$

$$Z = w^T X + b$$

$$= np.dot(w.T, X) + b$$

$$A = \sigma(Z)$$

$$dZ = A - Y$$

$$dw = \frac{1}{m} X dZ^T$$

$$db = \frac{1}{m} np.sum(dZ)$$

$$w := w - \alpha dw$$

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

Andrew Ng

```
Z = np.dot(w.T, X) + b
A = sigmoid(Z)
dZ = A - Y
dw = 1/m * np.dot(X, dZ.T)
db = 1/m * np.sum(dZ)
```

```
w = w - alpha * dw
b = b - alpha * db
```

## 2.15 python 中的广播

# Broadcasting example

Calories from Carbs, Proteins, Fats in 100g of different foods:

	Apples	Beef	Eggs	Potatoes
Carb	56.0	0.0	4.4	68.0
Protein	1.2	104.0	52.0	8.0
Fat	1.8	135.0	99.0	0.9

$= A$   
(3,4)

59 cal  
 $\frac{56}{59} \approx 94.9\%$

↓ 0  
→ 1

Calculate % of calories from Carb, Protein, Fat. Can you do this without explicit for-loop?

`cal = A.sum(axis = 0)`  
`percentage = 100 * A / (cal.reshape(1,4))`  
 ↑(3,4) / (1,4)

# Broadcasting example

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + \begin{bmatrix} 100 \\ 100 \\ 100 \\ 100 \end{bmatrix} = 100$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 100 & 200 & 300 \\ 100 & 200 & 300 \end{bmatrix}$$

(m,n) (2,3) (1,n) → (m,n) (2,3)

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 100 & 100 & 100 \\ 200 & 200 & 200 \end{bmatrix} =$$

(m,n) (m,1) ↓ (m,n)

↓ ↓ ↓

←  
←

# General Principle

$$\begin{array}{ccc} (m, n) & + & (1, n) \rightsquigarrow (m, n) \\ \text{matrix} & * & \\ & / & (n, 1) \rightsquigarrow (m, n) \end{array}$$

$$\begin{array}{ccc} (m, 1) & + & \mathbb{R} \\ \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} & + & 100 = \begin{bmatrix} 101 \\ 102 \\ 103 \end{bmatrix} \\ [1 \ 2 \ 3] & + & 100 = [101 \ 102 \ 103] \end{array}$$

Matlab/Octave: bsxfun

## 2.16 python numpy 向量的说明

- 虽然在Python有广播的机制，但是在Python程序中，为了保证矩阵运算的正确性，可以使用reshape()函数来对矩阵设定所需要计算进行计算的维度，这是个好的习惯；
- 如果用下列语句来定义一个向量，则这条语句生成的a的维度为(5, )，既不是行向量也不是列向量，称为秩(rank)为1的array，如果对a进行转置，则会得到a本身，这在计算中会给我们带来一些问题。

## Python / numpy vectors

```
import numpy as np

a = np.random.randn(5)

a = np.random.randn((5, 1))

a = np.random.randn((1, 5))

assert(a.shape == (5, 1))
```



- 如果需要定义  $(5, 1)$  或者  $(1, 5)$  向量，要使用下面标准的语句：

```
a = np.random.randn(5,1)
b = np.random.randn(1,5)
```

- 可以使用assert语句对向量或数组的维度进行判断。assert会对内嵌语句进行判断，即判断a的维度是不是  $(5, 1)$ ，如果不是，则程序在此处停止。使用assert语句也是一种很好的习惯，能够帮助我们及时检查、发现语句是否正确。

```
assert(a.shape == (5,1))
```

- 可以使用reshape函数对数组设定所需的维度

```
a.reshape((5,1))
```