

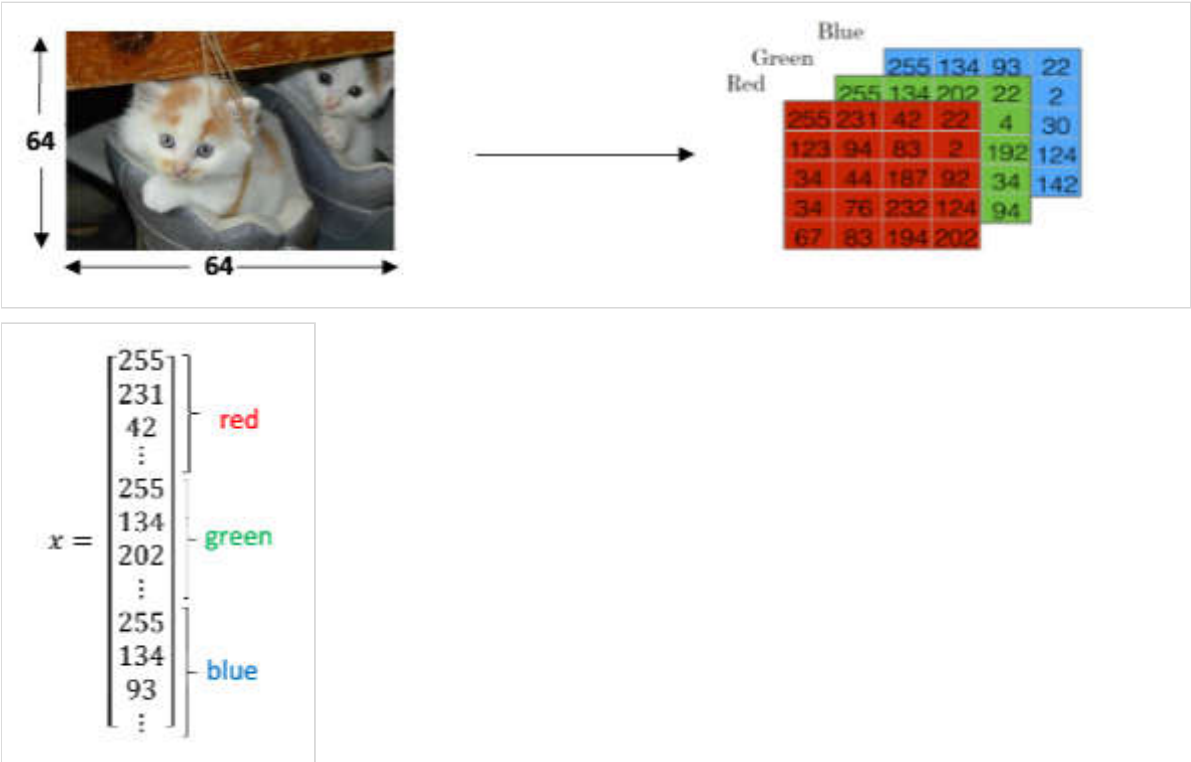
# 第二周 神经网络基础

Binary classification

forward propagation step / back propagation step

logistic regression is an algorithm for binary classification

in neural network  $X.shape = [n * m]$   $y.shape = 1 * m$

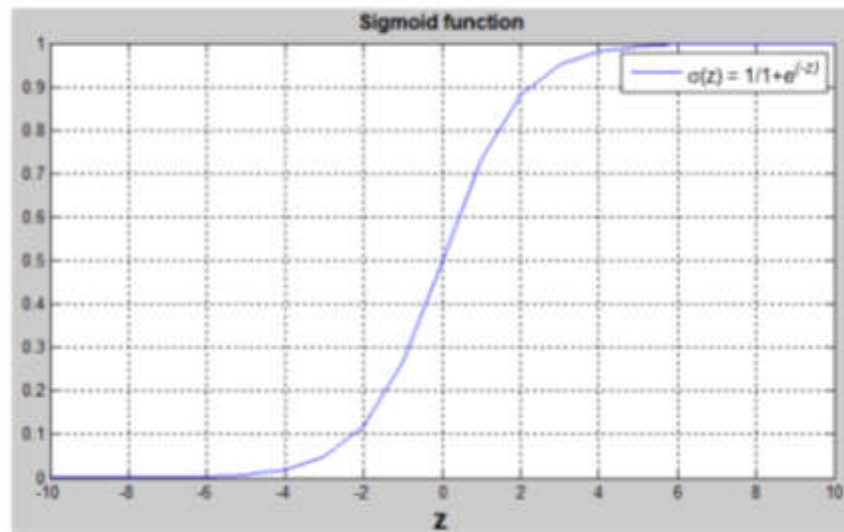


## LOGISTIC REGRESSION

in some problems ,we want that the output  $y$  is probability( $0 < y < 1$ )

## sigmoid function

- Sigmoid function:  $s = \sigma(w^T x + b) = \sigma(z) = \frac{1}{1 + e^{-z}}$



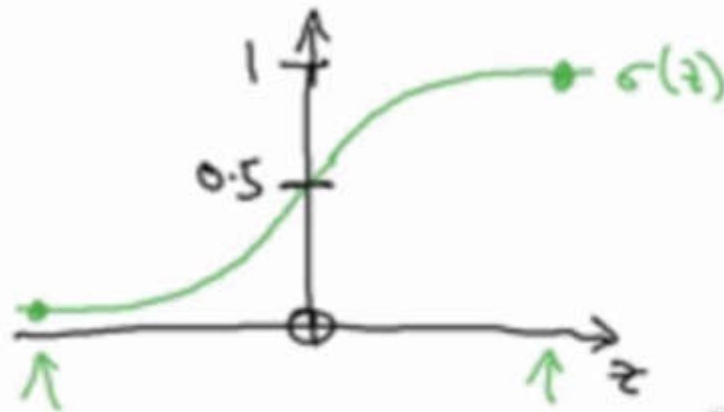
the problem turns out to look for better  $w$  and  $b$

# Logistic Regression

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

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

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



事实上 当你实现

it turns out, when you implement

---

logistic regression :cost function

loss function measures the discrepancy between the prediction and the desire output.

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

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

- 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

we do not use the first function, it will make the optimization problem to a non-convex function

use the second case-> it make the  $\hat{y}$  near the  $y$  (0 or 1)

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

cost function(measuring all training set) lost function(measuring single train example)

---

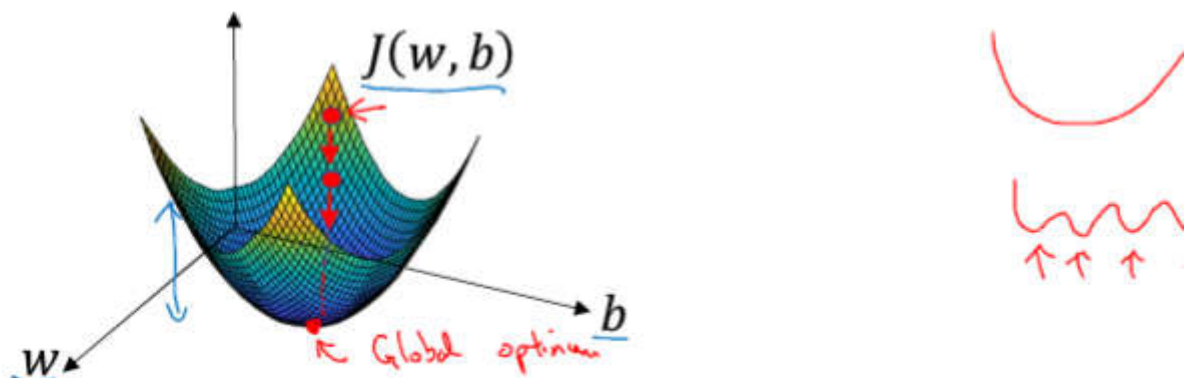
## Gradient Descent

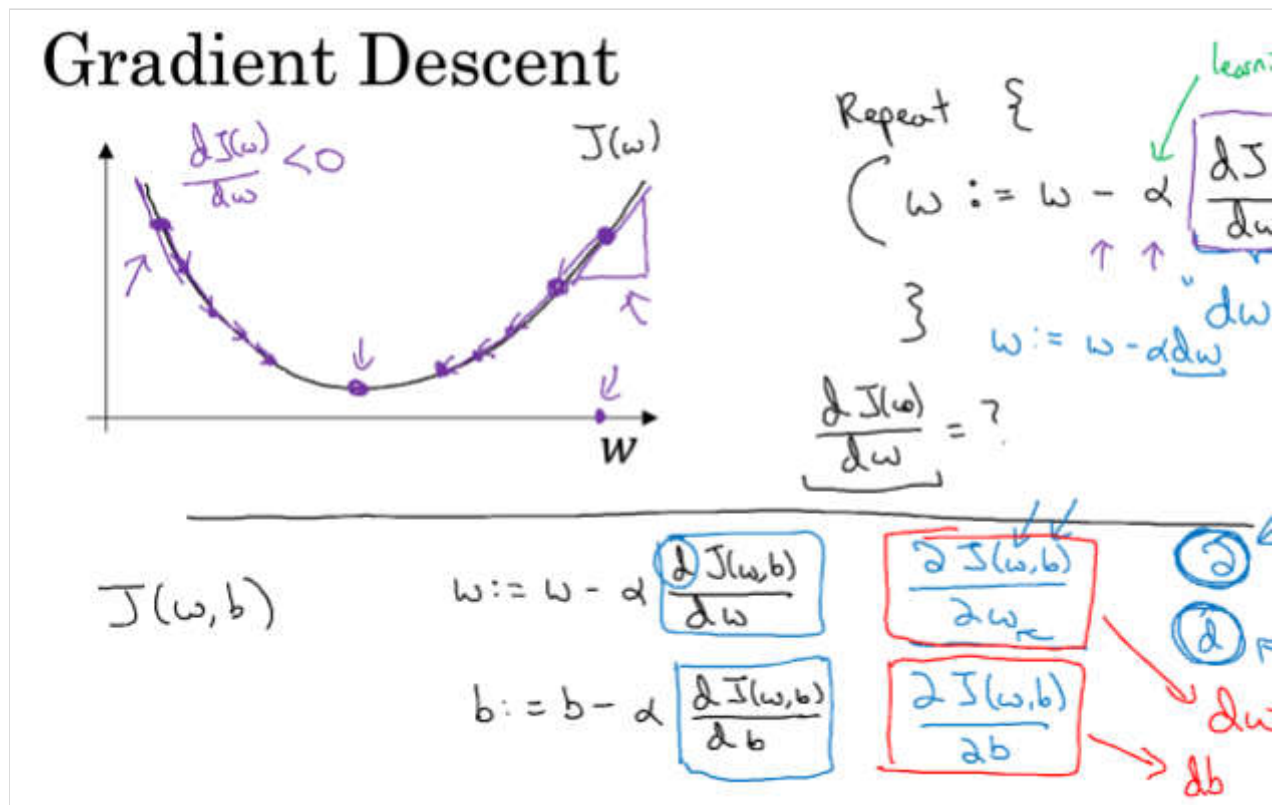
find  $w, b$  that minimize  $J(w, b)$

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  $J(w, b)$





Computation Graph(计算图, forward)-> compute cost function

**the computations of neural network are all organized in terms of a forward path (compute the output) and a backward path (compute gradient)**

computation graph explain why should we do that

Handwritten notes illustrating the forward pass of a computation graph for the function  $J(a, b, c) = 3(a + bc)$ .

The function is written as  $J(a, b, c) = 3(a + \underbrace{bc}_u) =$ . Below this, the intermediate variables are defined:  $u = bc$ ,  $v = a + u$ , and  $J = 3v$ .

The forward pass is shown with inputs  $a = 5$ ,  $b = 3$ , and  $c = 2$ . The computation for  $u = bc$  is shown in a box, with arrows pointing to it from  $b$  and  $c$ . The result of  $u$  is 6, which is then used in the final computation  $J = 3(a + u) = 3(5 + 6) = 33$ .

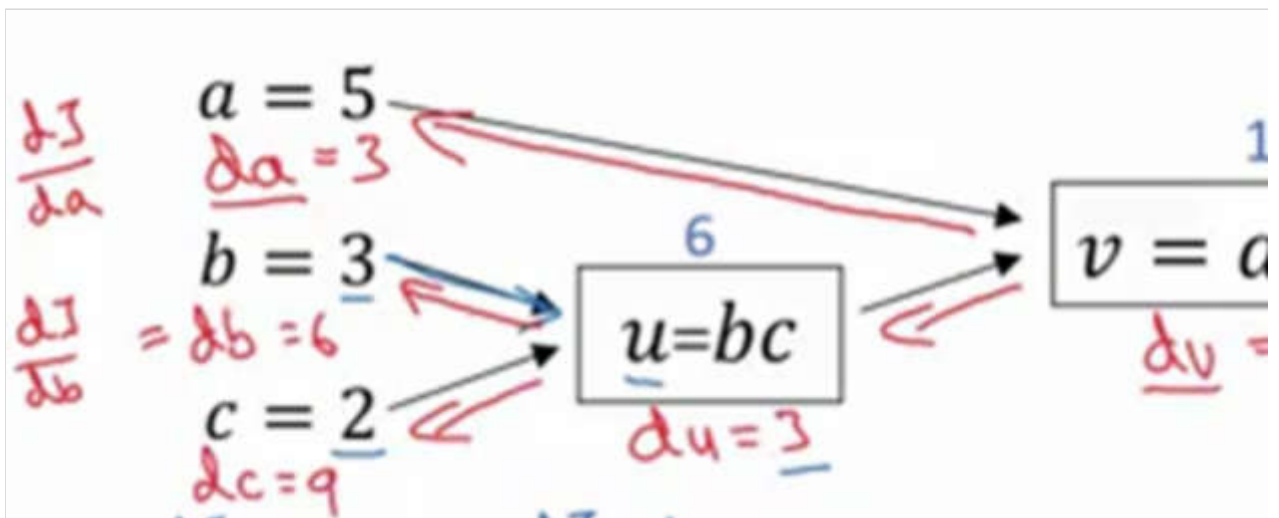
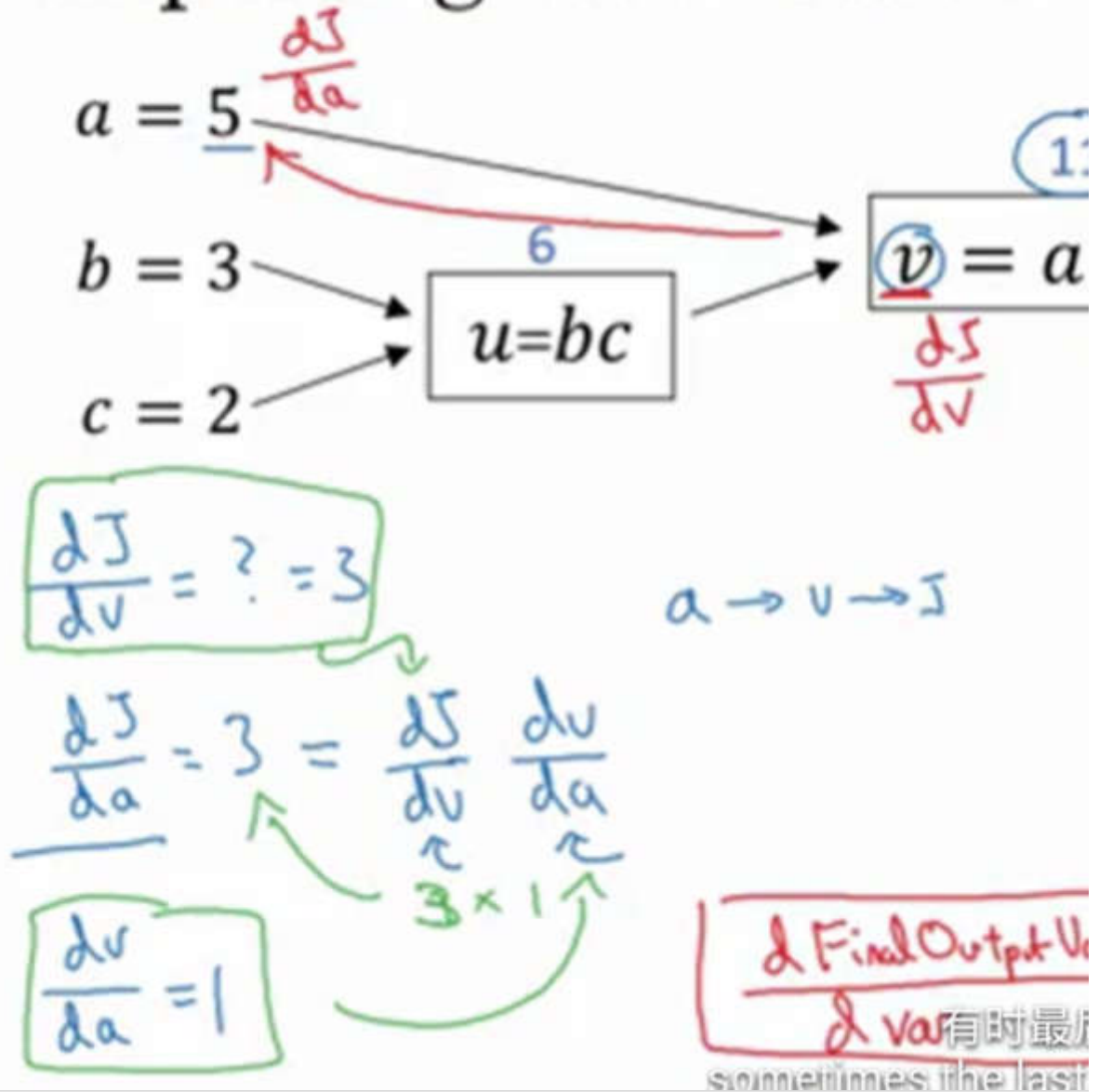
Chinese text at the bottom right: 这是3\* (5+3\*  
English text at the bottom right: this is a three times five plus

## Derivatives with a Computation Graph(反向传递)

one step backwards

chain rule(链式法则)

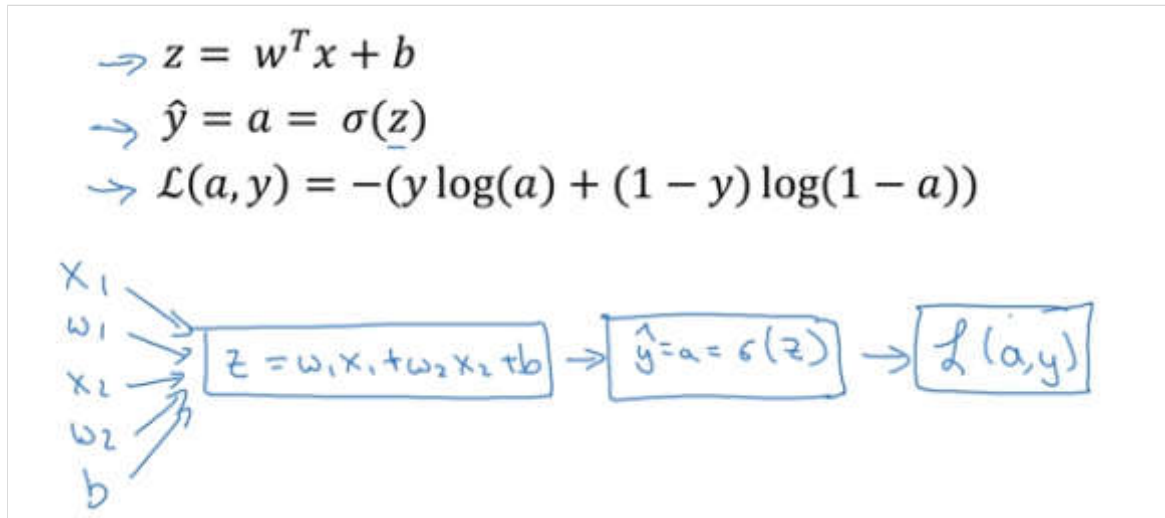
# Computing derivatives





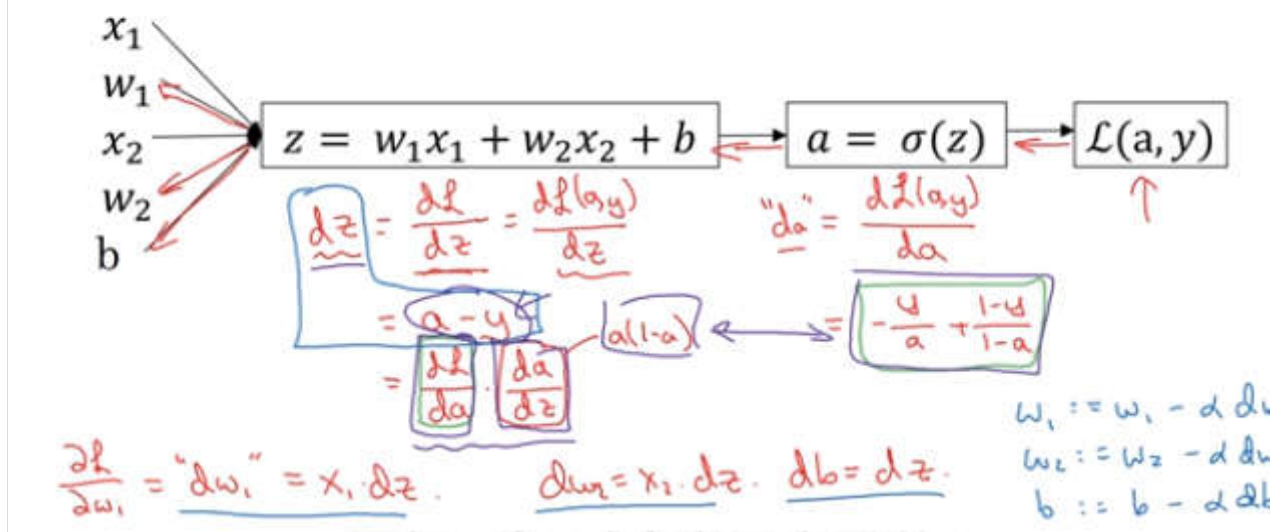
## 2.9 logistic gradient descent

logistic compute graph



## Logistic regression derivatives

云课堂



## 2.10 Gradient descent on m examples

one single step on gradient descent

two weakness: 1) two for loop  $\rightarrow$  vector



# Logistic regression on

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

For  $i=1$  to  $m$

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

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

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

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

$$dw_1 += x_1^{(i)} dz^{(i)}$$

$$dw_2 += x_2^{(i)} dz^{(i)}$$

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

$\updownarrow n=2$

$J/=m \leftarrow$

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

$\uparrow$

$\uparrow$

$\uparrow$

所以幻灯片  
so everything on the

## 2.11 vectorization

vectorization is the art of getting rid of explicit for loops

whenever possible, avoid using explicit for loops

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

 $\omega$ 

Non-vectorized:

$$z = 0$$

for i in range(n - x):

$$z += \omega[i] * x[i]$$

$$z += b$$

# Vectors and matrix va

Say you need to apply the exponent  
matrix/vector.

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

`→ u = np.zeros((n,1))`

`→ for i in range(n):`

`→ u[i]=math.exp(v[i])`

# Logistic regression de

$J = 0, \quad \boxed{dw_1 = 0, \quad dw_2 = 0, \quad db = 0}$   
 $\rightarrow$  for  $i = 1$  to  $m$ :  
 $z^{(i)} = w^T x^{(i)} + b$   
 $a^{(i)} = \sigma(z^{(i)})$   
 $J += -[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$   
 $dz^{(i)} = a^{(i)}(1 - a^{(i)})$   
 $\swarrow$   $\text{for } j=1 \dots n_x$   
 $\quad \boxed{\begin{aligned} dw_1 &+= x_1^{(i)} dz^{(i)} \\ dw_2 &+= x_2^{(i)} dz^{(i)} \end{aligned}}$   $n_x = 2$   
 $db += dz^{(i)}$   
 $J = J/m, \quad \boxed{dw_1 = dw_1/m, \quad dw_2 = dw_2/m, \quad db = db/m}$   
 that loops over the in

## 2.11 logistic vectorization (without using for loops)

forward progressing(向量化正向传播)

no for loops

# Vectorizing Logistic Re

$$\begin{aligned}
 \rightarrow \underline{z^{(1)}} &= \underline{w^T x^{(1)} + b} & \underline{z^{(2)}} &= \underline{w^T x^{(2)} + b} \\
 \rightarrow \underline{a^{(1)}} &= \sigma(z^{(1)}) & \underline{a^{(2)}} &= \sigma(z^{(2)})
 \end{aligned}$$

$$\underline{X} = \begin{bmatrix} x^{(1)} & x^{(2)} & \dots & x^{(n)} \\ | & | & & | \\ | & | & & | \\ | & | & & | \end{bmatrix} \quad \begin{matrix} (n_x, m) \\ \mathbb{R}^{n_x \times n} \end{matrix}$$

$$\underline{z} = [\underline{z^{(1)}} \quad \underline{z^{(2)}} \quad \dots \quad \underline{z^{(n)}}] = \underline{w^T X} + \underbrace{[b \quad b \quad \dots \quad b]}_{1 \times n}$$

$$\underline{z} = \text{np.dot}(w.T, X) + b$$

b will expand to [b,b,b,b,b..] -> "broadcasting"

implement a vector valued sigmoid function(对向量进行 sigmoid操作)

$$\underline{A} = [\underline{a^{(1)}} \quad \underline{a^{(2)}} \quad \dots \quad \underline{a^{(n)}}] = \sigma(\underline{z})$$

2.12 vectorizing logistic regression gradient computation(向量化反向传播)

# Vectorizing Logistic

$$dz^{(1)} = a^{(1)} - y^{(1)}$$

$$dz^{(2)} = a^{(2)} - y^{(2)}$$

$$dz = [dz^{(1)} \quad dz^{(2)} \dots dz^{(m)}]$$

$1 \times m$

$$A = [a^{(1)} \dots a^{(m)}]$$

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

$$\rightarrow dz = A - Y = [a^{(1)} - y^{(1)} \quad a^{(2)} - y^{(2)} \dots a^{(m)} - y^{(m)}]$$

$$\rightarrow dw = 0$$

$$\left[ \begin{array}{l} dw + = x^{(1)} dz^{(1)} \\ dw + = x^{(2)} dz^{(2)} \\ \vdots \\ dw + = x^{(m)} dz^{(m)} \end{array} \right]$$

$$dw / = m$$

$$db = 0$$

$$\left[ \begin{array}{l} db + = dz^{(1)} \\ db + = dz^{(2)} \\ \vdots \\ db + = dz^{(m)} \end{array} \right]$$

$$db / = m$$



$$\begin{aligned}
 db &= \frac{1}{m} \sum_{i=1}^m dz^{(i)} \\
 &= \frac{1}{m} \text{np.sum}(dz) \\
 dw &= \frac{1}{m} X dz^T \\
 &= \frac{1}{m} \begin{bmatrix} x^{(1)} & \dots & x^{(m)} \\ \hline 1 & & 1 \end{bmatrix} \begin{bmatrix} dz^{(1)} \\ \vdots \\ \underline{dz^{(m)}} \end{bmatrix} \\
 &= \frac{1}{m} \left[ x^{(1)} dz^{(1)} + \dots + x^{(m)} dz^{(m)} \right]
 \end{aligned}$$

的  
 ly end up with,  $n \times 1$

a highly vectorize logistic



# Implementing Logistic

```

J = 0, dw1 = 0, dw2 = 0, db = 0
for i = 1 to m:
    z(i) = wTx(i) + b ←
    a(i) = σ(z(i)) ←
    J += -[y(i) log a(i) + (1 - y(i)) log(1
    dz(i) = a(i) - y(i) ←
    [ dw1 += x1(i) dz(i) } dw += x(i) * dz(i)
      dw2 += x2(i) dz(i) }
      db += dz(i)
J = J/m, dw1 = dw1/m, dw2 = dw2/m
db = db/m

```

有时候应该用

## 2.13 broadcasting(广播)

a tech make the python and numpy together calculate

# Calories from Carbs, Proteins, Fat

	Apples	Beef	Eggs
Carb	56.0	0.0	4.4
Protein	1.2	104.0	52.0
Fat	1.8	135.0	99.0

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

```
In [6]: import numpy as np

A = np.array([[56.0, 0.0, 4.4, 68.0],
              [1.2, 104.0, 52.0, 8.0],
              [1.8, 135.0, 99.0, 0.9]])

print(A)
```

```
[[ 56.   0.   4.4  68. ]
 [  1.2 104.  52.   8. ]
 [  1.8 135.  99.   0.9]]
```

```
In [7]: cal = A.sum(axis=0)
print(cal)

[ 59.  239.  155.4  76.9]
```

```
In [ ]: percentage = 100*A/cal.reshape(1,4)
```

# Broadcasting example

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

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

Matlab/Octave: `bsxfun`

## 2.12 notations

never use `np.random.randn(5)` it will raise errors

always use `(1, 5)`

# Python/numpy vectors

```
a = np.random.randn(5)
```

*a.shape = (5,)*

*"rank 1 array"*

```
a = np.random.randn(5,
```

```
a = np.random.randn(1,
```

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

*a = a.reshape((5,1))*

或 1×n 矩阵 基本

or one by n matrices, or b