

# Copyright Notice

These slides are distributed under the Creative Commons License.

[DeepLearning.AI](#) makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite [DeepLearning.AI](#) as the source of the slides.

For the rest of the details of the license, see <https://creativecommons.org/licenses/by-sa/2.0/legalcode>



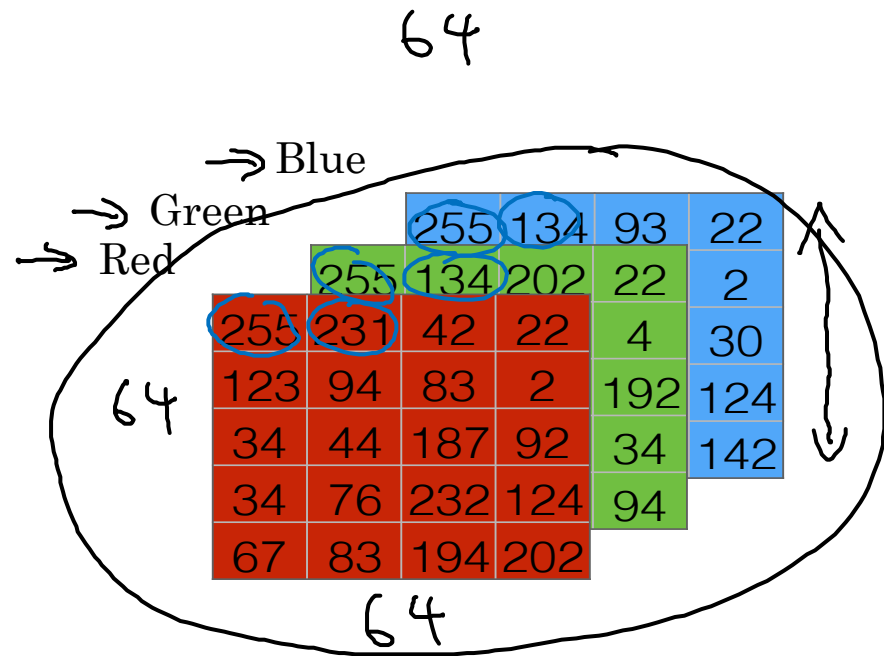
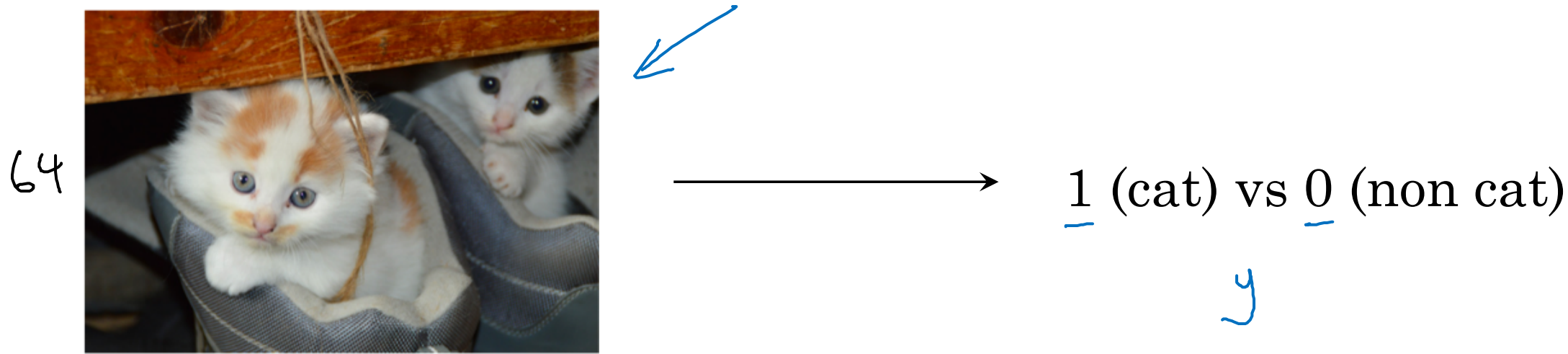
deeplearning.ai

# Basics of Neural Network Programming

---

## Binary Classification

# Binary Classification



$X = \begin{bmatrix} 255 \\ 231 \\ \vdots \\ 255 \\ 134 \\ \vdots \end{bmatrix}$

$64 \times 64 \times 3 = 12288$

$n = n_x = 12288$

$X \longrightarrow y$

# Notation

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

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

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

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

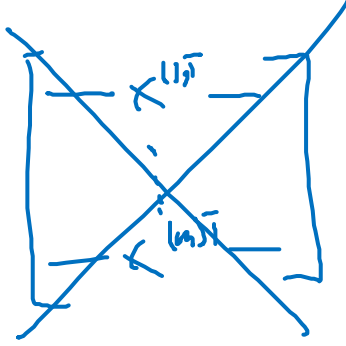
$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \end{bmatrix}$$


Diagram illustrating the matrix  $X$  (input features) with dimensions  $n_x$  (vertical) and  $m$  (horizontal). The matrix is shown as a collection of columns  $x^{(1)}, x^{(2)}, \dots, x^{(m)}$ . A small square box to the right of the matrix is crossed out with a large X, containing the labels  $x^{(1)}$  and  $x^{(m)}$ .

$$X \in \mathbb{R}^{n_x \times m}$$

$$X.\text{shape} = (n_x, m)$$

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

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

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



deeplearning.ai

# Basics of Neural Network Programming

---

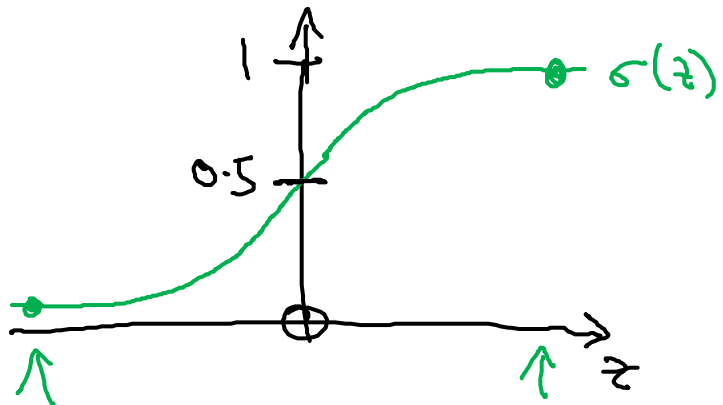
## Logistic Regression

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



~~$X_0 = 1, x \in \mathbb{R}^{n_x+1}$   
 $\hat{y} = \sigma(\theta^T x)$   
 $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{n_x} \end{bmatrix}$   $b \leftarrow$   
 $w \leftarrow$~~

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

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

If  $z$  large negative number

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



deeplearning.ai

# Basics of Neural Network Programming

---

## Logistic Regression cost function

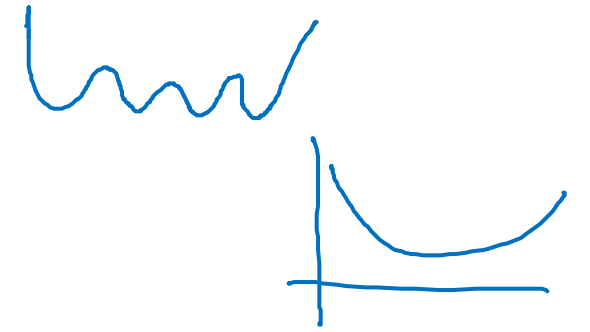
# Logistic Regression cost function

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

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

$x^{(i)}$   
 $y^{(i)}$   
 $z^{(i)}$   $i$ -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

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





deeplearning.ai

# Basics of Neural Network Programming

---

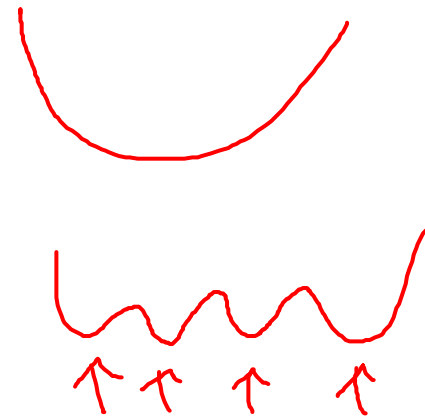
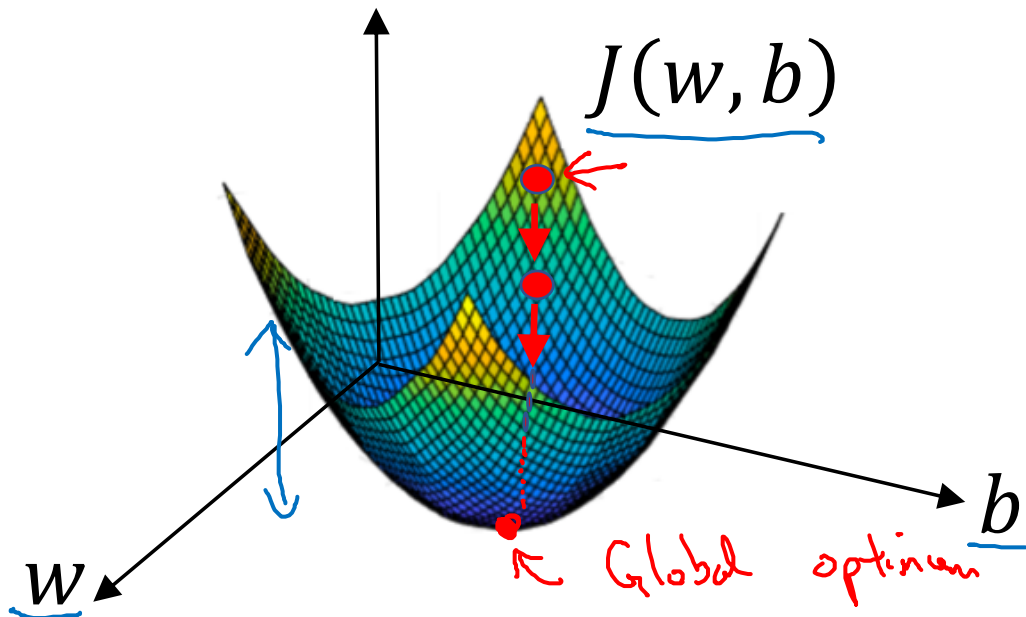
## Gradient Descent

# Gradient Descent

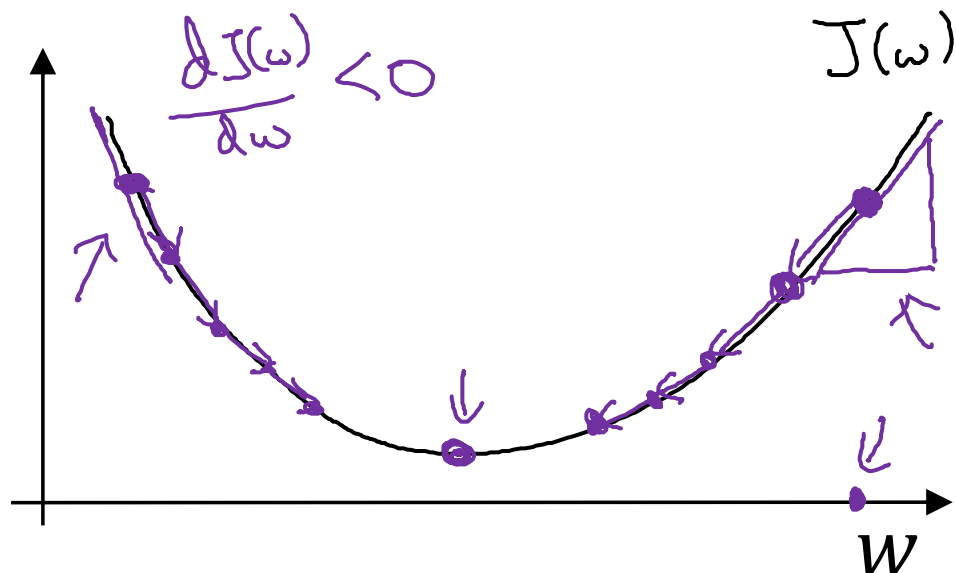
Recap:  $\hat{y} = \sigma(w^T x + b)$ ,  $\sigma(z) = \frac{1}{1+e^{-z}}$   $\leftarrow$

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

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



# Gradient Descent



Repeat {

$$w := w - \alpha \frac{dJ(w)}{dw}$$

learning rate

}

$$w := w - \alpha \underbrace{\frac{dJ(w)}{dw}}_{\text{"dw"}}$$

$$\frac{dJ(w)}{dw} = ?$$

$$J(w, b)$$

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

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

$$\partial$$

"partial derivative"  
J

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

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

$$\partial$$

dw

db



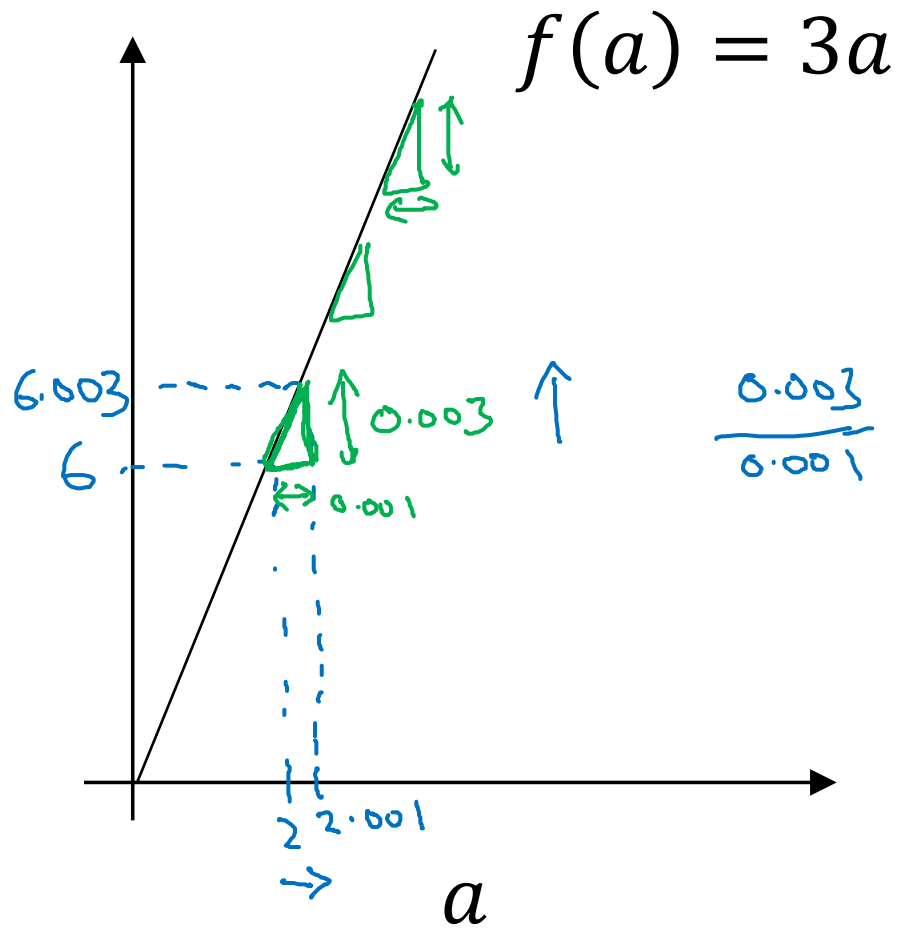
deeplearning.ai

# Basics of Neural Network Programming

---

## Derivatives

# Intuition about derivatives



$\rightarrow a = 2$        $f(a) = 6$   
 $a = 2.001$        $f(a) = 6.003$

$\rightarrow$  slope (derivative) of  $f(a)$   
 at  $a=2$  is 3

$\rightarrow a = 5$        $f(a) = 15$   
 $a = 5.001$        $f(a) = 15.003$   
 slope at  $a=5$  is also 3

$\downarrow$   
 $\frac{df(a)}{da} = 3 = \frac{d}{da} f(a)$   
 $\uparrow$

$0.001 \leftarrow$   
 $0.000000001$   
 $0.0000000001$



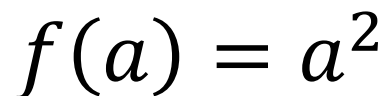
deeplearning.ai

# Basics of Neural Network Programming

---

More derivatives  
examples


0.001 ←  
0.000000...01 ←


$$\frac{\text{height}}{\text{width}}$$

$$\frac{d}{da} a^2 = 2a$$

$$0.001$$

$$(2a) \times 0.001$$

$a = 2$                        $f(a) = 4$   
 $a = 2.001$                  $f(a) \approx 4.004$   
     $(4.004 \boxed{004})$    
 slope (derivative) of  $f(a)$  at  
 $a = 2$  is 4.

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

$$\begin{array}{ll} a=5 & f(a)=25 \\ a=5.001 & f(a) \approx 25.010 \end{array}$$

$$\frac{d}{da} f(a) = 10 \quad \text{when} \quad a = 5$$

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

# More derivative examples

$$f(a) = a^2$$

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

$$a = 2$$

$$f(a) = 4$$

$$a = 2.001$$

$$f(a) \approx 4.004$$

$$f(a) = a^3$$

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

$$a = 2$$

$$f(a) = 8$$

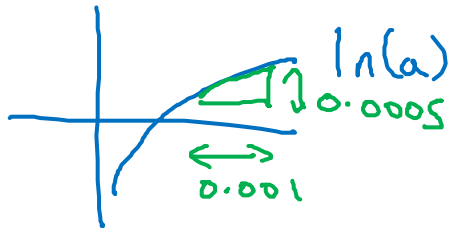
$$a = \underline{2.001}$$

$$f(a) \approx \underline{8.012}$$

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

$$\ln(a)$$

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



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

$$a = 2$$

$$f(a) \approx 0.69315$$

$$\downarrow$$
  

$$a = \underline{2.001}$$

$$\downarrow$$
  

$$\underline{f(a) \approx 0.69365}$$

$$\downarrow$$
  

$$0.0005$$





deeplearning.ai

# Basics of Neural Network Programming

---

## Computation Graph

# Computation Graph

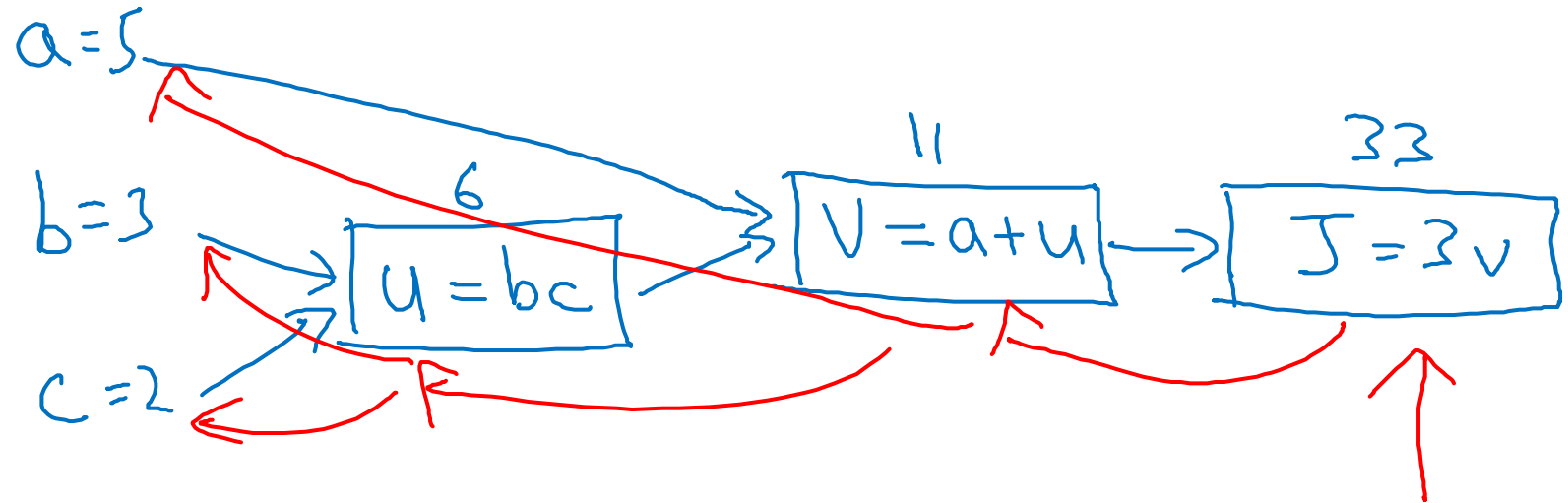
$$J(a,b,c) = 3(a + \underbrace{bc}_u) = 3(5 + 3 \times 2) = 33$$

$$\underbrace{\underbrace{a + u}_v}_J$$

$$u = bc$$

$$V = a + u$$

$$J = 3V$$





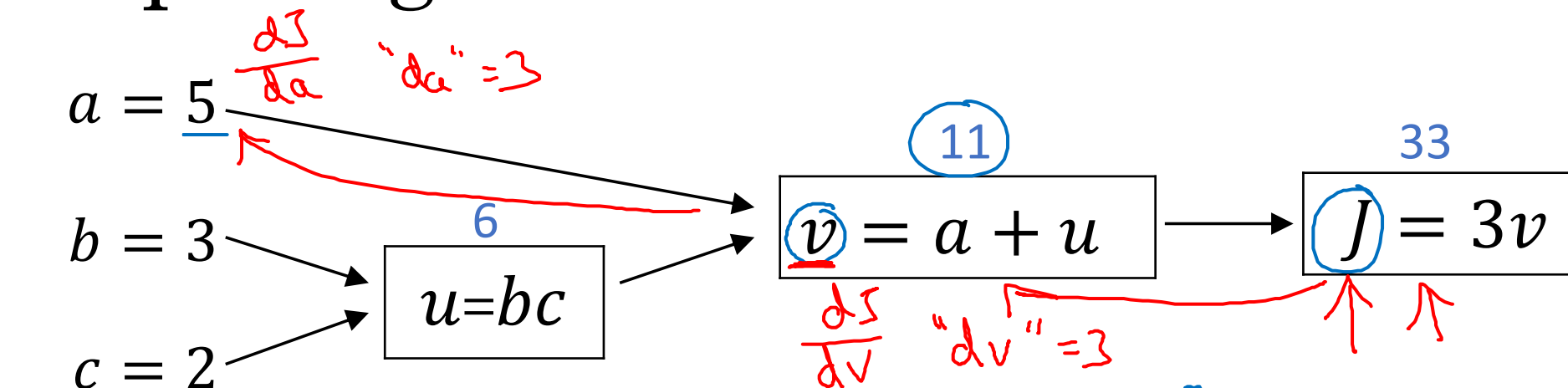
deeplearning.ai

# Basics of Neural Network Programming

---

## Derivatives with a Computation Graph

# Computing derivatives



$$\frac{dJ}{dv} = ? = 3$$

$$\frac{dJ}{da} = 3 = \frac{dJ}{dv} \frac{dv}{da}$$

$$\frac{dv}{da} = 1$$

$$a \rightarrow v \rightarrow J$$

$$\frac{d \text{ Final Output Var}}{d \text{ var}}$$

$$J = 3v$$

$$v = 11 \rightarrow 11.001$$

$$J = 33 \rightarrow 33.003$$

$$a = 5 \rightarrow 5.001$$

$$\rightarrow v = 11 \rightarrow 11.001$$

$$J = 33 \rightarrow 33.003$$

$$\frac{dJ}{d \text{ var}}$$

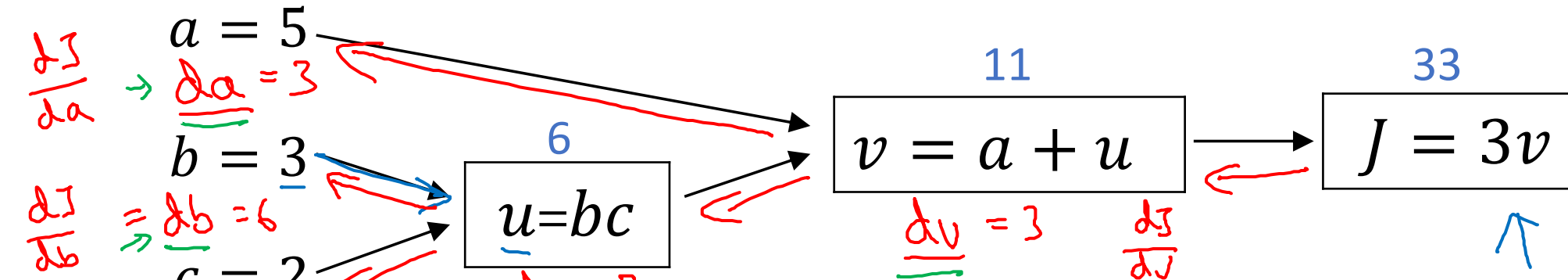
$$f(a) = 3a$$

$$\frac{df(a)}{da} = \frac{df}{da} = 3$$

$$J = 3v$$

$$\frac{dJ}{dv} = 3$$

# Computing derivatives



$$\frac{dJ}{du} = 3 = \frac{dJ}{dv} \cdot \frac{dv}{du}$$

$\underbrace{\quad}_{3} \quad \underbrace{\quad}_{1}$

$$\frac{dJ}{db} = \frac{dJ}{du} \cdot \frac{du}{db} = 6$$

$\underbrace{\quad}_{\rightarrow 3} \quad \underbrace{\quad}_{=2}$

$$\frac{dJ}{da} = \frac{dJ}{du} \cdot \frac{du}{da} = 9$$

$\underbrace{\quad}_{\rightarrow 3} \quad \underbrace{\quad}_{=3}$

$$\begin{aligned} u &= 6 \rightarrow 6.001 \\ v &= 11 \rightarrow 11.001 \\ J &= 33 \rightarrow 33.003 \end{aligned}$$

$$b = 3 \rightarrow 3.001$$

$$\begin{aligned} u &= b \cdot c = 6 \rightarrow 6.002 \\ J &= 33.006 \end{aligned}$$

$$\begin{aligned} c &= 2 \\ &1.006 \end{aligned}$$

$$\begin{aligned} v &= 11.002 \\ J &= 3v \end{aligned}$$



deeplearning.ai

# Basics of Neural Network Programming

---

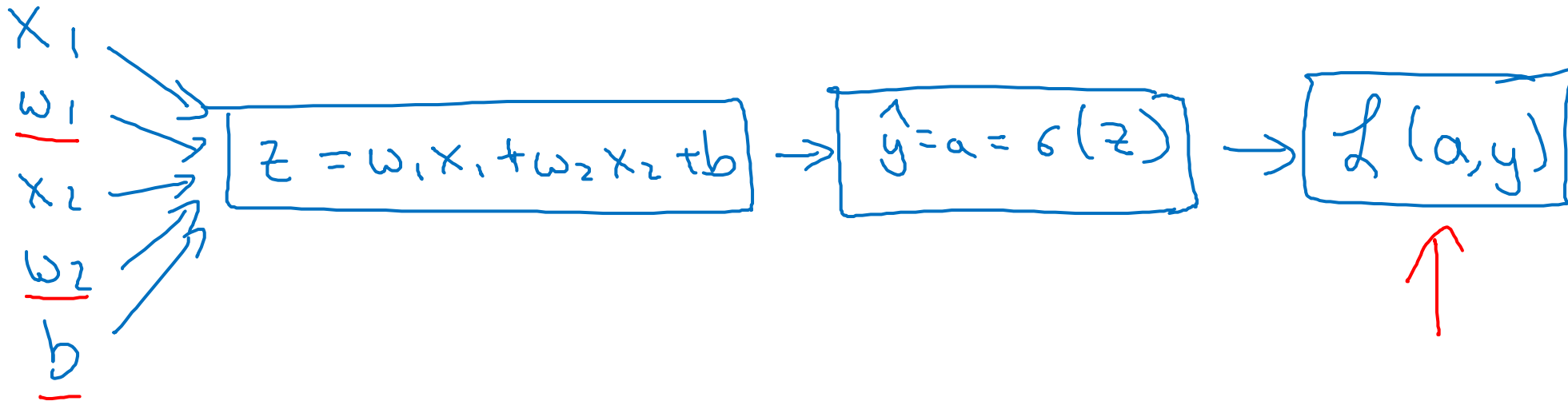
Logistic Regression  
Gradient descent

# Logistic regression recap

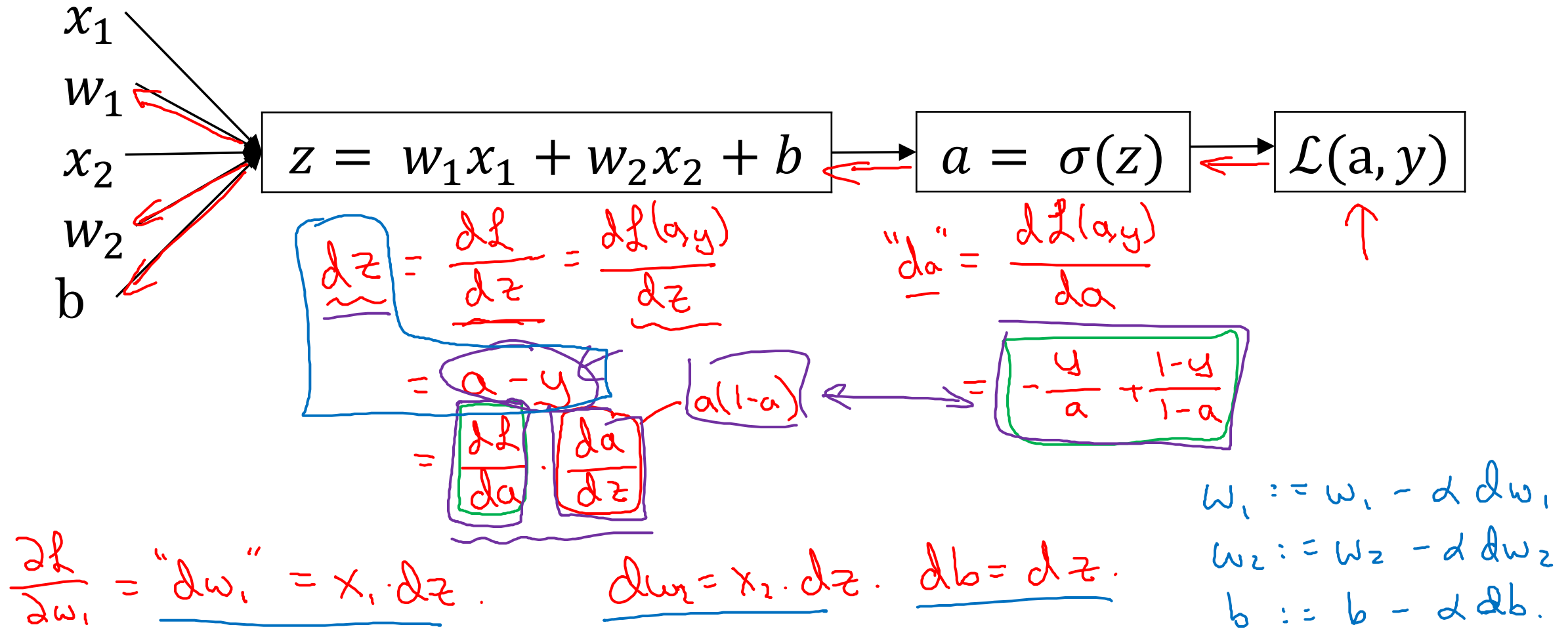
→  $z = w^T x + b$

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

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



# Logistic regression derivatives







deeplearning.ai

# Basics of Neural Network Programming

---

Gradient descent  
on *m* examples

# Logistic regression on $m$ examples

$$\underline{J(w, b)} = \frac{1}{m} \sum_{i=1}^m \ell(a^{(i)}, y^{(i)})$$

$$\rightarrow a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b)$$

$$(x^{(i)}, y^{(i)})$$

$$\underline{dw_1^{(i)}}, \underline{dw_2^{(i)}}, \underline{db^{(i)}}$$

$$\underline{\frac{\partial}{\partial w_1} J(w, b)} = \frac{1}{m} \sum_{i=1}^m \underbrace{\frac{\partial}{\partial w_1} \ell(a^{(i)}, y^{(i)})}_{\underline{dw_1^{(i)}} - (x^{(i)}, y^{(i)})}$$

# Logistic regression on $m$ examples

$$J=0; \text{ dw}_1=0; \text{ dw}_2=0; \text{ db}=0$$

→ For  $i=1$  to  $m$

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

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

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

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

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

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

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

$n=2$

$dw_3$   
 $\vdots$   
 $dw_n$

$J /= m \leftarrow$

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

$$dw_1 = \frac{\partial J}{\partial w_1}$$

$$w_1 := w_1 - \alpha \text{ dw}_1$$

$$w_2 := w_2 - \alpha \text{ dw}_2$$

$$b := b - \alpha \text{ db.$$

Vectorization



deeplearning.ai

# Basics of Neural Network Programming

---

## Vectorization

# What is vectorization?

$$z = \underbrace{w^T x}_{\text{dot product}} + b$$

Non-vectorized:

$$z = 0$$

for  $i$  in  $\text{range}(n-x)$ :

$$z += w[i] * x[i]$$

$$z += b$$

$$w = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix} \quad x = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix}$$

$$w \in \mathbb{R}^{n_x}$$

$$x \in \mathbb{R}^{n_x}$$

Vectorized

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

$\Rightarrow$  GPU } SIMD - single instruction  
 $\Rightarrow$  CPU } multiple data.



deeplearning.ai

# Basics of Neural Network Programming

---

More vectorization  
examples

# 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.

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

```
→ u = np.zeros((n,1))  
→ for i in range(n):  
    → u[i]=math.exp(v[i])
```

```
import numpy as np  
u = np.exp(v)  
  
np.log(v)  
np.abs(v)  
np.maximum(v, 0)  
v**2  
1/v
```



# Logistic regression derivatives

$$J = 0, \quad \boxed{\cancel{dw_1 = 0, dw_2 = 0}}, \quad db = 0$$

$$dw = np.zeros((n-x, 1))$$

→ for i = 1 to n:

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

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

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

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

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

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

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

$$n_x = 2$$

$$dw += x^{(i)} dz^{(i)}$$

$$J = J/m, \quad \boxed{\cancel{dw_1 = dw_1/m, dw_2 = dw_2/m}}, \quad db = db/m$$

$$dw /= m.$$



deeplearning.ai

# Basics of Neural Network Programming

---

## Vectorizing Logistic Regression

# Vectorizing Logistic Regression

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

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

$$\begin{aligned} z^{(3)} &= w^T x^{(3)} + b \\ a^{(3)} &= \sigma(z^{(3)}) \end{aligned}$$

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

↑  
 $\mathbb{R}^{n_x \times m}$

$$\begin{aligned} & \text{---} \rightarrow \\ & \text{---} \rightarrow \end{aligned}$$

↑  
 $\mathbb{R}^{1 \times m}$

$$w^T \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

$$\underline{z} = \begin{bmatrix} z^{(1)} & z^{(2)} & \dots & z^{(m)} \end{bmatrix} = \underbrace{w^T X}_{1 \times m} + \underbrace{[b \ b \dots b]}_{1 \times m} = \begin{bmatrix} w^T x^{(1)} + b & w^T x^{(2)} + b & \dots & w^T x^{(m)} + b \end{bmatrix}$$

↑  
 $\mathbb{R}^{1 \times m}$

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

↑  
 $\mathbb{R}^{(1,1)}$

"Broadcasting"

$$\underline{A} = \begin{bmatrix} a^{(1)} & a^{(2)} & \dots & a^{(m)} \end{bmatrix} = \sigma(\underline{z})$$

↑  
 $\mathbb{R}^{(1,1)}$



deeplearning.ai

# Basics of Neural Network Programming

---

## Vectorizing Logistic Regression's Gradient Computation

# Vectorizing Logistic Regression

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

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

.....

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

$1 \times m$

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

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

$$\rightarrow dw = 0$$

$$dw += \frac{x^{(1)} dz^{(1)}}{m}$$

$$dw += \frac{x^{(2)} dz^{(2)}}{m}$$

$\vdots$

$$dw /= m$$

$$db = 0$$

$$db += dz^{(1)}$$

$$db += dz^{(2)}$$

$$\vdots$$

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

$$db /= m$$

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

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

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

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

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

# Implementing Logistic Regression

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

for  $i = 1$  to  $m$ :

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

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

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

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

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

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

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

$$db = db/m$$

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

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

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

$$A = \sigma(Z)$$

$$dZ = A - Y$$

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

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

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

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



deeplearning.ai

# Basics of Neural Network Programming

---

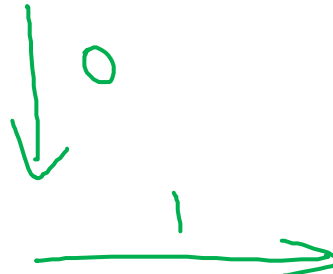
## Broadcasting in Python

# 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	= A (3,4)
Protein	1.2	104.0	52.0	8.0	
Fat	1.8	135.0	99.0	0.9	

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



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

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

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

$(m,n) \quad (2,3) \qquad (1,n) \rightsquigarrow (m,n) \quad (2,3)$

↓      ↓      ↓

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

$(m,n) \qquad (m,1) \rightsquigarrow (m,n)$

←  
←

# General Principle

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

$$(m, 1) \rightsquigarrow (m, n)$$

$$(m, 1)$$

+

$$\mathbb{R}$$

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

+

$$100$$

$$= \begin{bmatrix} 101 \\ 102 \\ 103 \end{bmatrix}$$

$$[1 \ 2 \ 3]$$

+

$$100$$

$$= [101 \quad 102 \quad 103]$$

Matlab/Octave: bsxfun



deeplearning.ai

# Basics of Neural Network Programming

---

Explanation of logistic  
regression cost function  
(Optional)

# Logistic regression cost function

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

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

If  $y=1$  :  $p(y|x) = \hat{y}$

If  $y=0$  :  $p(y|x) = \underline{1 - \hat{y}}$

# Logistic regression cost function

$$\begin{aligned} \rightarrow & \text{If } y = 1: p(y|x) = \hat{y} \\ \rightarrow & \text{If } y = 0: p(y|x) = 1 - \hat{y} \end{aligned} \quad \left. \vphantom{\begin{aligned} \rightarrow & \text{If } y = 1: p(y|x) = \hat{y} \\ \rightarrow & \text{If } y = 0: p(y|x) = 1 - \hat{y} \end{aligned}} \right\} p(y|x)$$

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

$$\text{If } y=1: p(y|x) = \hat{y} \underbrace{(1-\hat{y})^0}_{=1}$$

$$\text{If } y=0: p(y|x) = \hat{y}^0 \underbrace{(1-\hat{y})^{(1-0)}}_{=1} = 1 \times (1-\hat{y}) = \underline{1-\hat{y}}$$

$$\begin{aligned} \uparrow \log p(y|x) &= \log \hat{y}^y (1-\hat{y})^{(1-y)} = y \log \hat{y} + (1-y) \log (1-\hat{y}) \\ &= - \underbrace{\ell(\hat{y}, y)}_{\downarrow} \end{aligned}$$

# Cost on $m$ examples

$$\log p(\text{labels in training set}) = \log \prod_{i=1}^m p(y^{(i)} | x^{(i)}) \quad \leftarrow$$

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

$$= - \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

Maximum likelihood  
estimator  $\nwarrow$

$$\text{Cost:} \quad \underbrace{J(w, b)}_{\uparrow \text{(minimize)}} = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$