

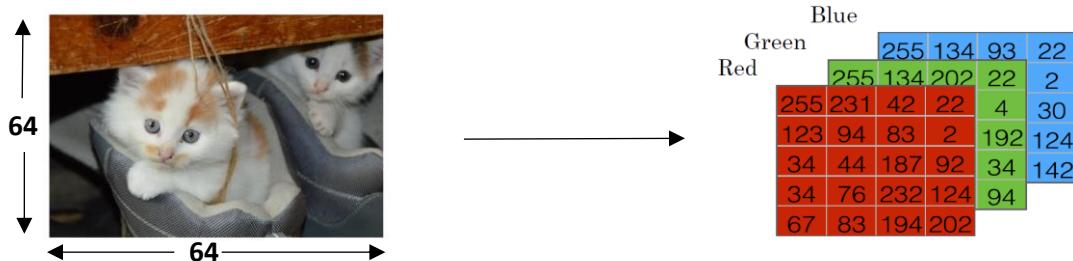
Binary Classification

In a binary classification problem, the result is a discrete value output.

- For example
- account hacked (1) or compromised (0)
 - a tumor malign (1) or benign (0)

Example: Cat vs Non-Cat

The goal is to train a classifier that the input is an image represented by a feature vector, x , and predicts whether the corresponding label y is 1 or 0. In this case, whether this is a cat image (1) or a non-cat image (0).



An image is stored in the computer in three separate matrices corresponding to the Red, Green, and Blue color channels of the image. The three matrices have the same size as the image, for example, the resolution of the cat image is 64 pixels X 64 pixels, the three matrices (RGB) are 64 X 64 each.

The value in a cell represents the pixel intensity which will be used to create a feature vector of n-dimension. In pattern recognition and machine learning, a feature vector represents an object, in this case, a cat or no cat.

To create a feature vector, x , the pixel intensity values will be “unroll” or “reshape” for each color. The dimension of the input feature vector x is $n_x = 64 \times 64 \times 3 = 12,288$.

$$x = \begin{bmatrix} 255 \\ 231 \\ 42 \\ \vdots \\ 255 \\ 134 \\ 202 \\ \vdots \\ 255 \\ 134 \\ 93 \\ \vdots \end{bmatrix} \left[\begin{array}{c} \text{red} \\ \text{green} \\ \text{blue} \end{array} \right]$$



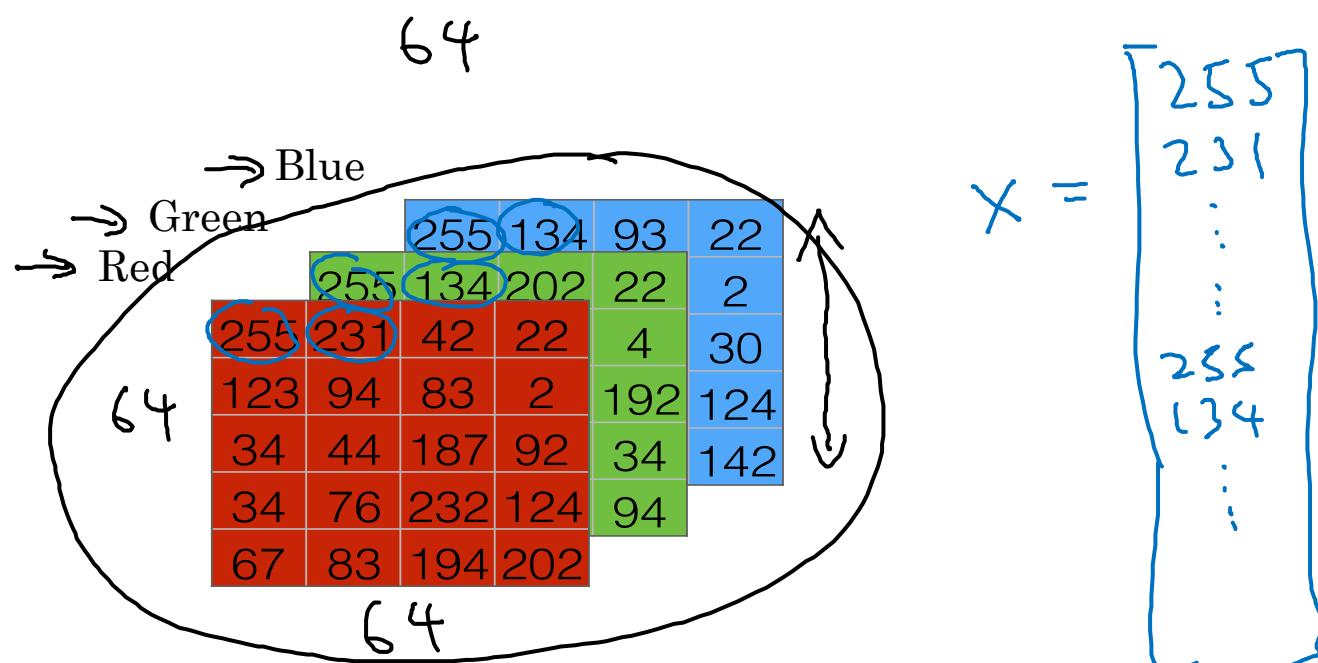
deeplearning.ai

Basics of Neural Network Programming

Binary Classification

Binary Classification

64 →  → 1 (cat) vs 0 (non cat)



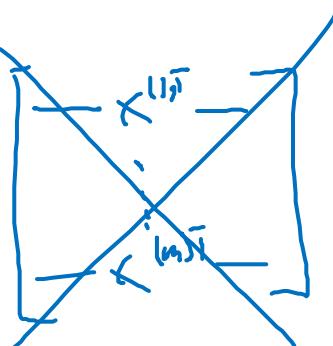
$$64 \times 64 \times 3 = 12288$$

$$n = n_x = 12288$$

$$X \rightarrow y$$

Notation

(x, y) $x \in \mathbb{R}^{n_x}$, $y \in \{0, 1\}$
 m training examples : $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$
 $M = M_{\text{train}}$ $M_{\text{test}} = \# \text{test examples.}$

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

$$Y = [y^{(1)} \ y^{(2)} \ \dots \ y^{(m)}]$$
$$Y \in \mathbb{R}^{1 \times m}$$
$$Y.\text{shape} = (1, m)$$

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

Logistic Regression

Logistic regression is a learning algorithm used in a supervised learning problem when the output y are all either zero or one. The goal of logistic regression is to minimize the error between its predictions and training data.

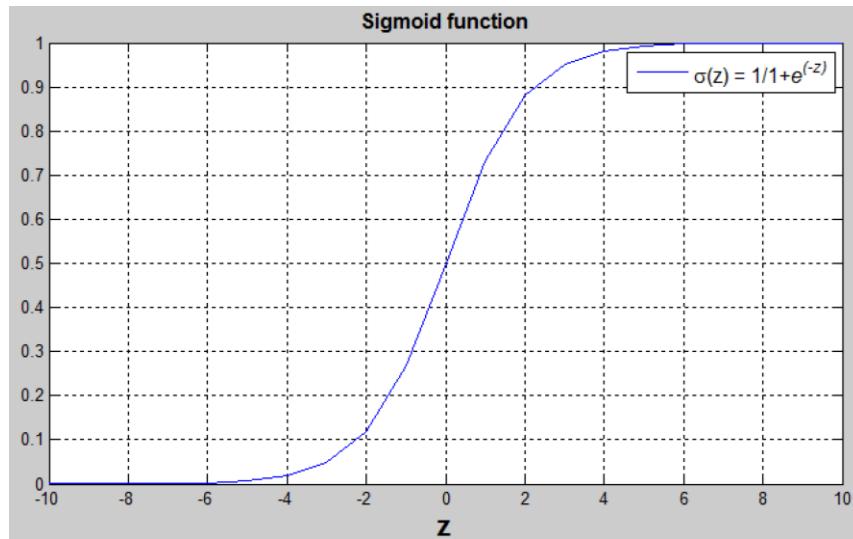
Example: Cat vs No - cat

Given an image represented by a feature vector x , the algorithm will evaluate the probability of a cat being in that image.

$$\text{Given } x, \hat{y} = P(y = 1|x), \text{ where } 0 \leq \hat{y} \leq 1$$

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



$(w^T x + b)$ is a linear function ($ax + b$), but since we are looking for a probability constraint between $[0,1]$, the sigmoid function is used. The function is bounded between $[0,1]$ as shown in the graph above.

Some observations from the graph:

- If z is a large positive number, then $\sigma(z) = 1$
- If z is small or large negative number, then $\sigma(z) = 0$
- If $z = 0$, then $\sigma(z) = 0.5$



deeplearning.ai

Basics of Neural Network Programming

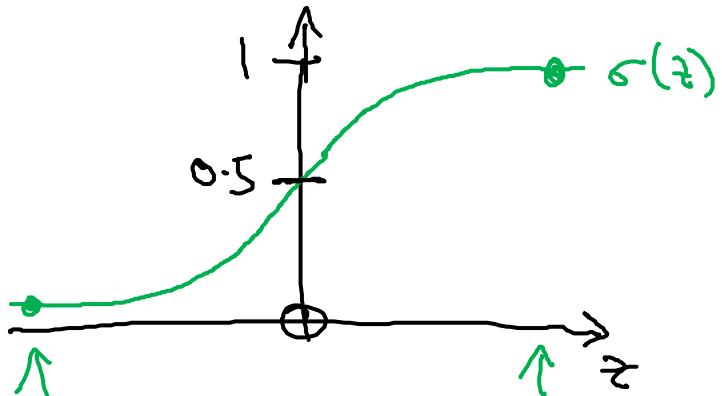
Logistic Regression

Logistic Regression

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

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

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



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

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

$$w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_{n_x} \end{bmatrix} \quad \left. \begin{array}{l} \{w_0\} \rightarrow b \\ \{w_1, w_2, \dots, w_{n_x}\} \rightarrow w \end{array} \right.$$

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

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

If z large negative number

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



deeplearning.ai

Basics of Neural Network Programming

Logistic Regression cost function

Logistic Regression cost function

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

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

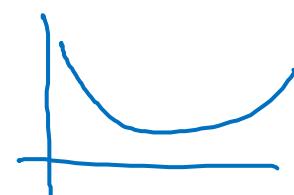
Loss (error) function:

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

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

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

i-th example.



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

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

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

Logistic Regression: Cost Function

To train the parameters w and b , we need to define a cost function.

Recap:

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

$x^{(i)}$ the i-th training example

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

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

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.

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



deeplearning.ai

Basics of Neural Network Programming

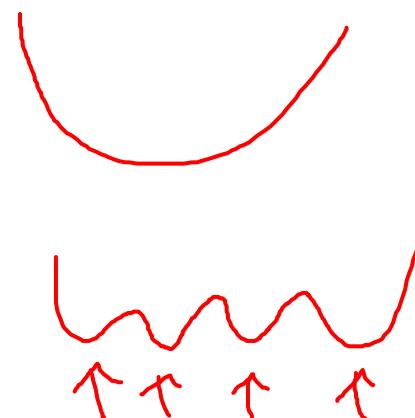
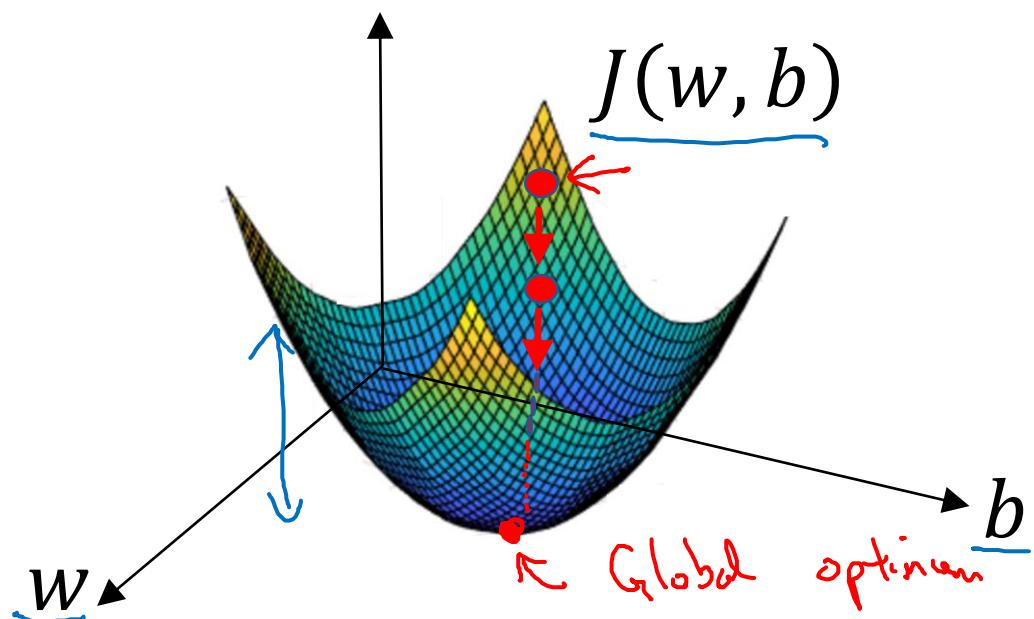
Gradient Descent

Gradient Descent

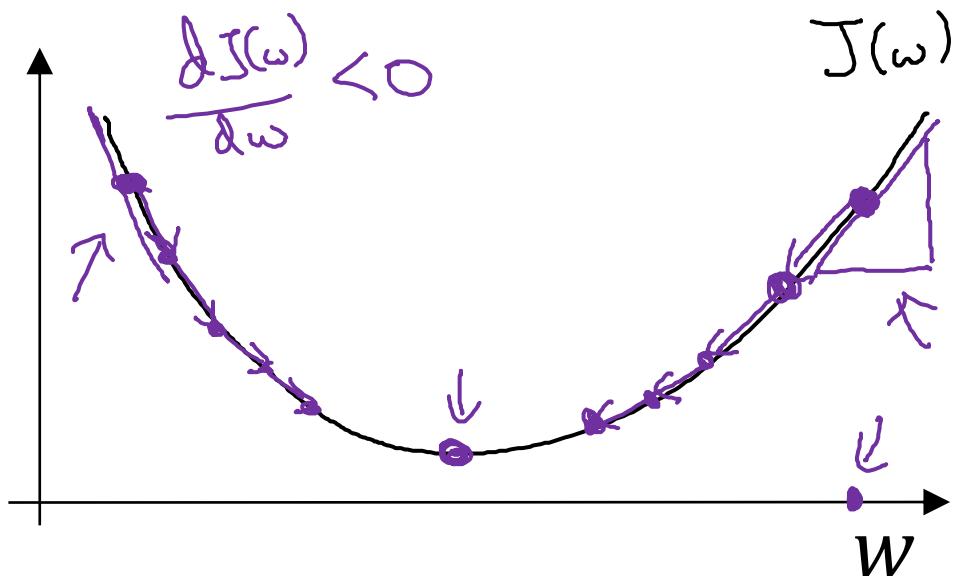
Recap: $\hat{y} = \sigma(w^T x + b)$, $\sigma(z) = \frac{1}{1+e^{-z}}$ 

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

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



Gradient Descent



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

"learning rate"

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

$J(w, b)$

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

$b := b - \alpha \frac{dJ(w, b)}{db}$

$\frac{dJ(w, b)}{dw}$ "partial derivative" J

$\frac{dJ(w, b)}{db}$

$\frac{dJ(w, b)}{dw}$

$\frac{dJ(w, b)}{db}$

Andrew Ng

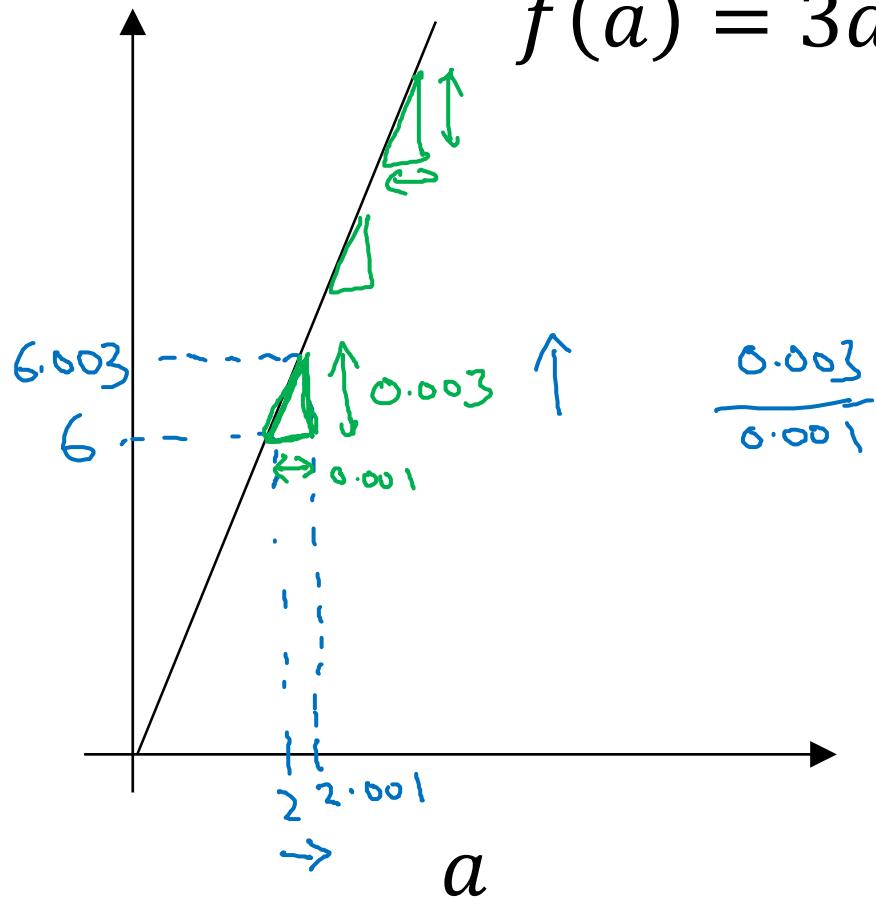


deeplearning.ai

Basics of Neural Network Programming

Derivatives

Intuition about derivatives



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

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

$$\rightarrow a = 5 \quad f(a) = 15$$
$$a = 5.001 \quad f(a) = 15.003$$

slope at $a=5$ is also 3

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

0.001
0.00000001
0.0000000001

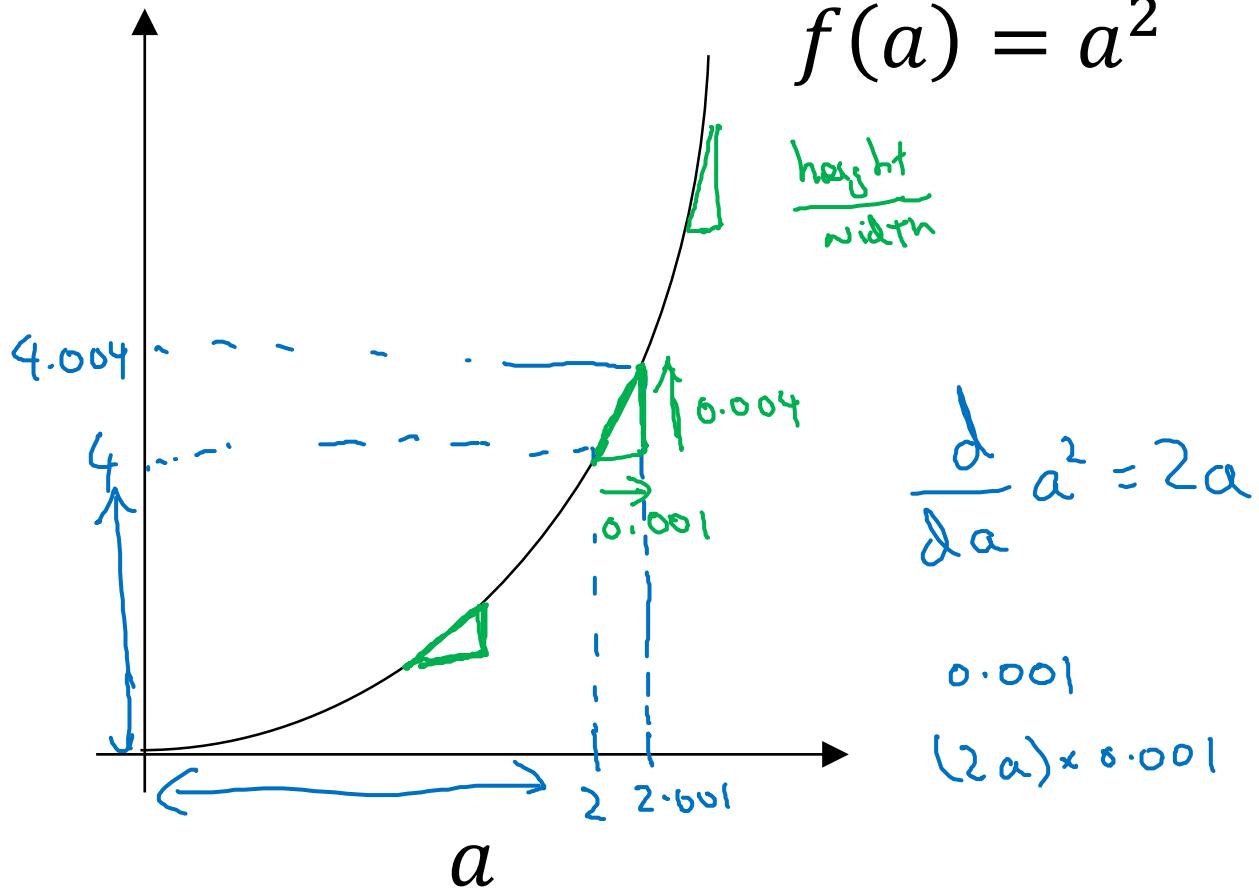


deeplearning.ai

Basics of Neural Network Programming

More derivatives
examples

Intuition about derivatives



$a = 2$ $f(a) = 4$
 $a = 2.001$ $f(a) \approx 4.004$

$\frac{(4.004 - 4)}{0.001}$

slope (derivative) of $f(a)$ at $a=2$ is 4.

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

$a = 5$ $f(a) = 25$
 $a = 5.001$ $f(a) \approx 25.010$

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

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

More derivative examples

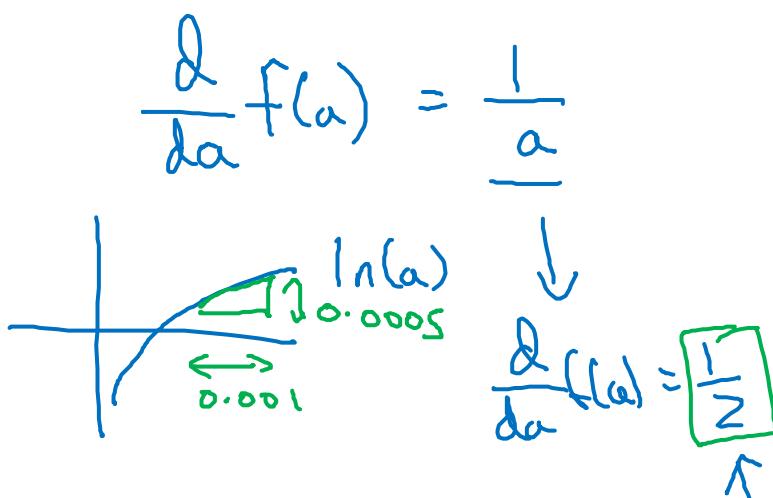
$$f(a) = a^2$$

$$\frac{\partial}{\partial a} f(a) = \underbrace{2a}_{4}$$

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

$$\frac{\partial}{\partial a} f(a) = \underbrace{3a^2}_{3 \times 2^2} = 12$$

$$f(a) = \begin{matrix} \log_e(a) \\ \ln(a) \end{matrix}$$



$$a = 2$$

$$a = 2.001$$

$$f(a) = 4$$

$$f(a) \approx 4.004$$

$$a = 2$$

$$a = \underline{2.001}$$

$$f(a) = 8$$

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

$$a = 2$$

$$a = \underline{2.001}$$

$$f(a) \approx 0.69315$$

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

$$\frac{0.0005}{0.0005}$$



deeplearning.ai

Basics of Neural Network Programming

Computation Graph

Computation Graph

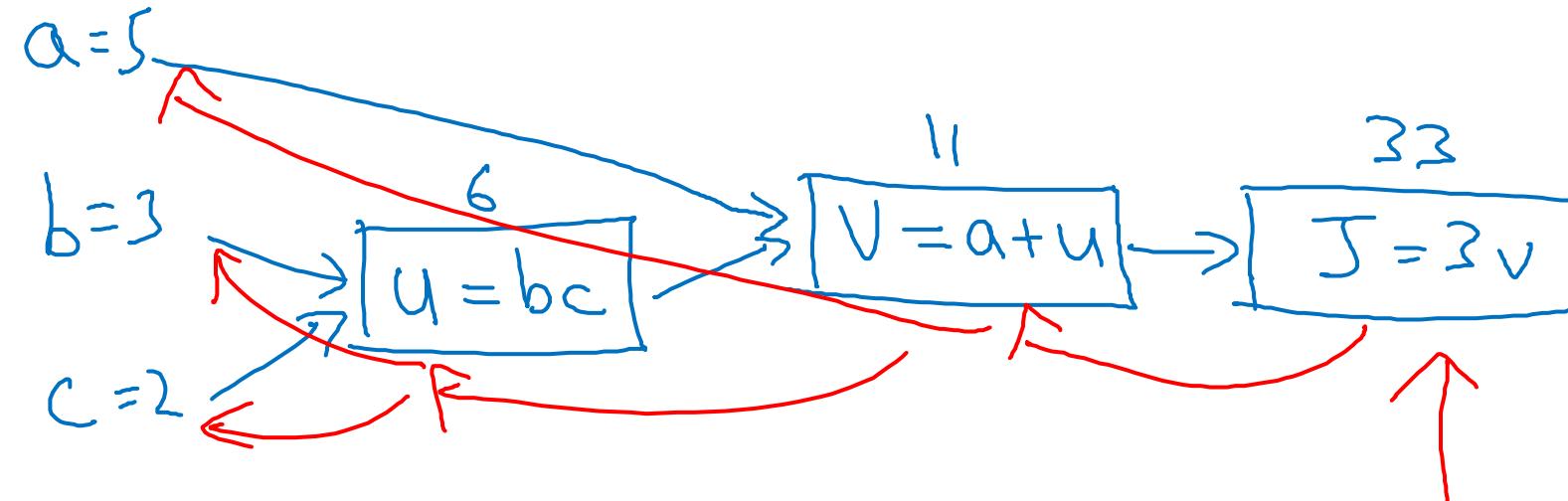
$$J(a, b, c) = 3(u + bc) = 3(5 + 3 \times 2) = 33$$

$\underbrace{u}_{\downarrow}$
 $\underbrace{v}_{\downarrow}$
 $\underbrace{J}_{\downarrow}$

$$u = bc$$

$$v = a + u$$

$$J = 3v$$



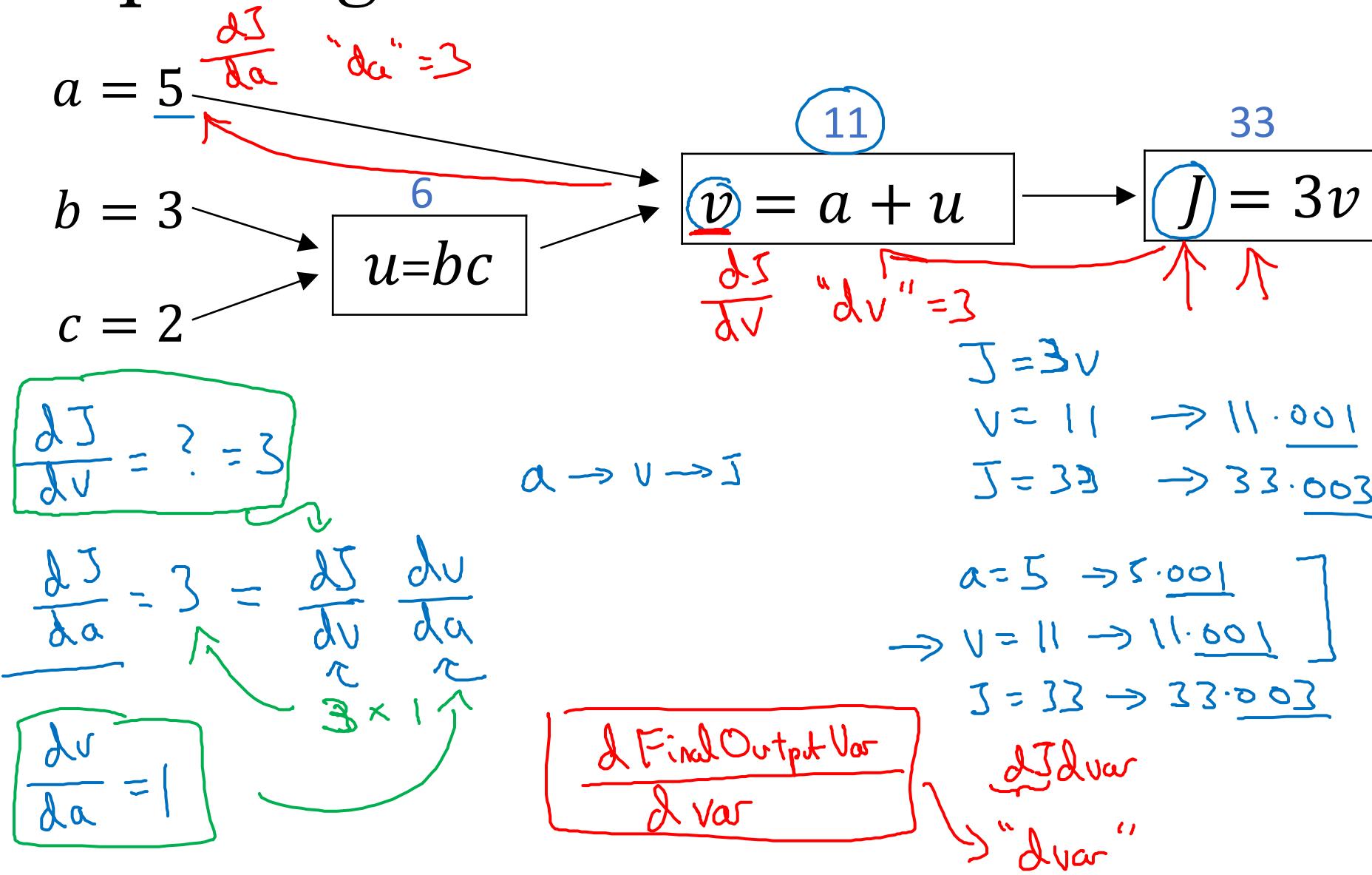


deeplearning.ai

Basics of Neural Network Programming

Derivatives with a Computation Graph

Computing derivatives



$f(a) = 3a$
 $\frac{df(b)}{da} = \frac{df}{da} = 3$
 $J = 3v$
 $\frac{dJ}{dv} = 3$

Computing derivatives

$\frac{\partial J}{\partial a} \rightarrow \underline{a = 5}$
 $\frac{\partial J}{\partial b} \rightarrow \underline{b = 3}$
 $\frac{\partial J}{\partial c} \rightarrow \underline{c = 2}$
 $\frac{\partial J}{\partial u} = 3$
 $\frac{\partial J}{\partial v} = 3$

$a = 5$ → $v = a + u$ → $J = 3v$
 $b = 3$ → $v = a + u$
 $c = 2$ → $u = bc$
 $u = 6$ → $v = a + u$
 $\frac{\partial u}{\partial v} = 3$
 $\frac{\partial u}{\partial a} = 1$
 $\frac{\partial u}{\partial b} = 6$
 $\frac{\partial u}{\partial c} = 2$
 $\frac{\partial J}{\partial u} = 3 = \frac{\partial J}{\partial v} \cdot \frac{\partial v}{\partial u}$
 $\frac{\partial J}{\partial b} = \frac{\partial J}{\partial u} \cdot \frac{\partial u}{\partial b} = 3 \cdot 6 = 18$
 $\frac{\partial J}{\partial a} = \frac{\partial J}{\partial u} \cdot \frac{\partial u}{\partial a} = 3 \cdot 1 = 3$
 $\frac{\partial J}{\partial c} = \frac{\partial J}{\partial u} \cdot \frac{\partial u}{\partial c} = 3 \cdot 2 = 6$

$v = 11$
 $J = 33$

$u = 6 \rightarrow 6.001$
 $v = 11 \rightarrow 11.001$
 $J = 33 \rightarrow 33.003$

$b = 3 \rightarrow 3.001$
 $u = b \cdot c = 6 \rightarrow 6.002$
 $J = 33.006$

$v = 11.002$
 $J = 33.006$



deeplearning.ai

Basics of Neural Network Programming

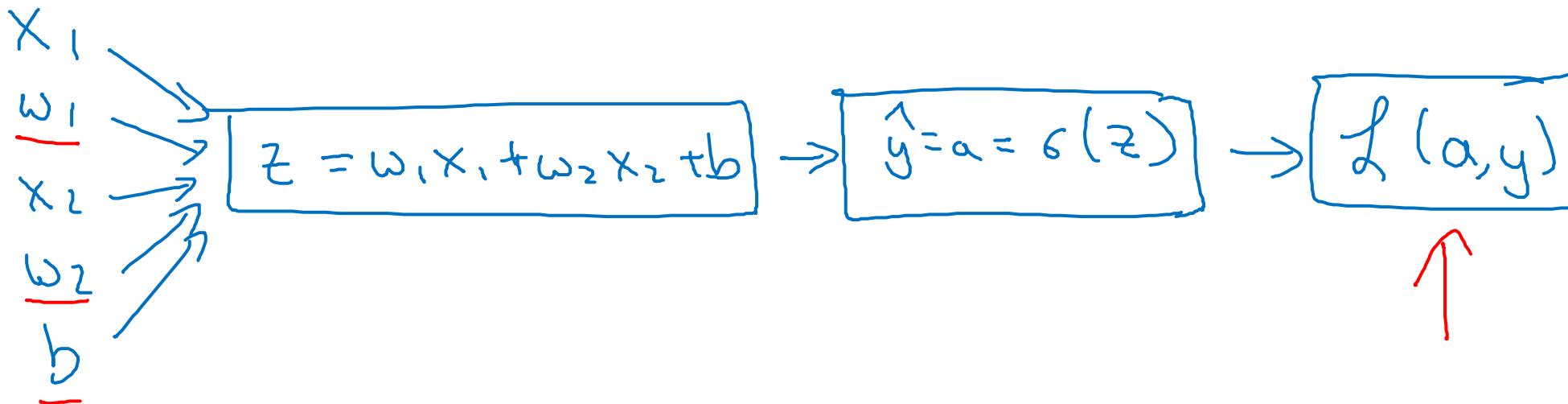
Logistic Regression Gradient descent

Logistic regression recap

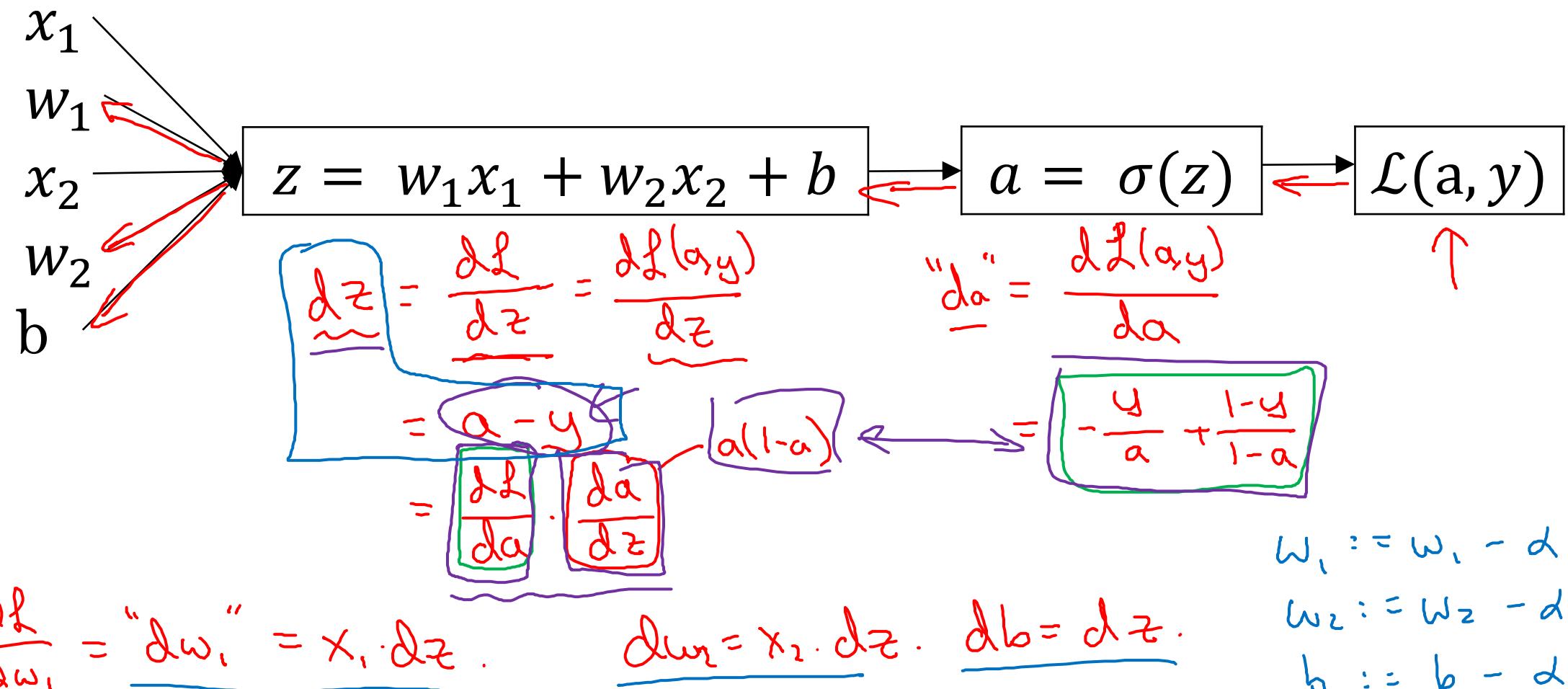
$$\rightarrow z = w^T x + b$$

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

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



Logistic regression derivatives



$$w_1 := w_1 - \alpha \frac{\partial \mathcal{L}}{\partial w_1}$$

$$w_2 := w_2 - \alpha \frac{\partial \mathcal{L}}{\partial w_2}$$

$$b := b - \alpha \frac{\partial \mathcal{L}}{\partial b}$$



deeplearning.ai

Basics of Neural Network Programming

Gradient descent on m examples

Logistic regression on m examples

$$\underline{J(w,b)} = \frac{1}{m} \sum_{i=1}^m l(a^{(i)}, y^{(i)}) \quad (x^{(i)}, y^{(i)})$$
$$\Rightarrow a^{(i)} = \hat{y}^{(i)} = g(z^{(i)}) = g(w^\top x^{(i)} + b) \quad \underline{dw_1^{(i)}}, \underline{dw_2^{(i)}}, \underline{db^{(i)}}$$

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

Logistic regression on m examples

$$J=0; \underline{\Delta w_1}=0; \underline{\Delta w_2}=0; \underline{\Delta b}=0$$

→ For $i = 1$ to m

$$z^{(i)} = \omega^\top x^{(i)} + b$$

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

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

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

$$\begin{aligned} \Delta w_1 &+= x_1^{(i)} \Delta z^{(i)} \\ \Delta w_2 &+= x_2^{(i)} \Delta z^{(i)} \end{aligned}$$

$$\begin{aligned} \Delta w_3 & \\ \vdots & \\ \Delta w_n & \Delta b += \Delta z^{(i)} \end{aligned}$$

$$J / m \leftarrow$$

$$\Delta w_1 / m; \Delta w_2 / m; \Delta b / m. \leftarrow$$

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

$$w_1 := w_1 - \alpha \underline{\Delta w_1}$$

$$w_2 := w_2 - \alpha \underline{\Delta w_2}$$

$$b := b - \alpha \underline{\Delta b}.$$

Vectorization



deeplearning.ai

Basics of Neural Network Programming

Vectorization

What is vectorization?

$$z = \underbrace{\omega^T x}_{\text{Non-vectorized}} + b$$

Non-vectorized:

$$z = 0$$

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

$$z += b$$

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

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

Vectorized

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

\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.

Neural network programming guideline

Whenever possible, avoid explicit for-loops.

$$u = Av$$

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

$u = np.zeros((n,))$

for i ...

 for j ...

$u[i] += A[:, i] * v[j]$

$$u = np.dot(A, v)$$

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  
v/v
```

Logistic regression derivatives

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

$$d\omega = np.zeros((n_x, 1))$$

for i = 1 to n:

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

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

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

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

for j=1..n^x
 $dw_j += \frac{\partial J}{\partial w_j}$

$$\left. \begin{aligned} dw_1 &+= x_1^{(i)} dz^{(i)} \\ dw_2 &+= x_2^{(i)} dz^{(i)} \end{aligned} \right| \quad n_x = 2$$
$$db += dz^{(i)}$$

$$d\omega += x^{(i)} dz^{(i)}$$

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

$$d\omega /= m.$$



deeplearning.ai

Basics of Neural Network Programming

Vectorizing Logistic Regression

Vectorizing Logistic Regression

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

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

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

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

$\xrightarrow{\text{---}}$

$$\frac{(n_x, m)}{\mathbb{R}^{n_x \times m}}$$

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

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

$$\dots \underline{\underline{\omega^T x^{(m)}}} + b$$

$$\rightarrow \overline{z} = \text{np.dot}(\underline{\underline{\omega^T}}, \underline{\underline{X}}) + \underline{\underline{b}} \quad R^{(1,1)}$$

"Broadcasting"

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



deeplearning.ai

Basics of Neural Network Programming

Vectorizing Logistic Regression's Gradient Computation

Vectorizing Logistic Regression

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

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

$$dz = \begin{bmatrix} dz^{(1)} & dz^{(2)} & \dots & dz^{(m)} \end{bmatrix}^T$$

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

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

$$\begin{aligned} \rightarrow dw &= 0 \\ dw + &= \frac{x^{(1)} dz^{(1)}}{} \\ dw + &= \frac{x^{(2)} dz^{(2)}}{} \\ &\vdots \\ dw &= m \end{aligned}$$

$$\begin{aligned} db &= 0 \\ db + &= dz^{(1)} \\ db + &= dz^{(2)} \\ &\vdots \\ db + &= dz^{(m)} \\ db &= m \end{aligned}$$

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

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

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

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

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

Implementing Logistic Regression

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

for i = 1 to m:

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

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

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

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

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

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

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

$$db = db/m$$

```
for iter in range(1000):  
    z = w^T X + b  
    = np.dot(w.T, X) + b  
    A = g(z)  
    dZ = A - Y  
    dw = 1/m * X * dZ^T  
    db = 1/m * 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	
Protein	1.2	104.0	52.0	8.0	
Fat	1.8	135.0	99.0	0.9	
	59 cal	$\frac{56}{59} \approx 94.9\%$			

$$= A_{(3,4)}$$



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

cal = A.sum(axis = 0)

percentage = $100 * A / (\text{cal} / \text{A.reshape}(1, 4))$

$\uparrow (3,4) / (1,4)$

Broadcasting example

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

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 100 & 200 & 300 \\ 100 & 200 & 300 \end{bmatrix} \xleftarrow{(m,n) \quad (2,3)} \xrightarrow{(1,n) \rightsquigarrow (m,n) \quad (2,3)}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 100 & 100 & 100 \\ 200 & 200 & 200 \end{bmatrix} = \xleftarrow{(m,n)} \xleftarrow{(m,1)}$$

General Principle

$$\begin{array}{c} (m, n) \\ \text{matrix} \\ \hline \end{array} \quad \begin{array}{c} + \\ - \\ \times \\ \diagup \end{array} \quad \begin{array}{c} (1, n) \\ (m, 1) \end{array} \quad \rightsquigarrow (m, n)$$

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

Matlab/Octave: bsxfun



deeplearning.ai

Basics of Neural Network Programming

A note on python/
numpy vectors

Python Demo

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))
```