deeplearning.ai

Basics of Neural Network Programming

Binary Classification

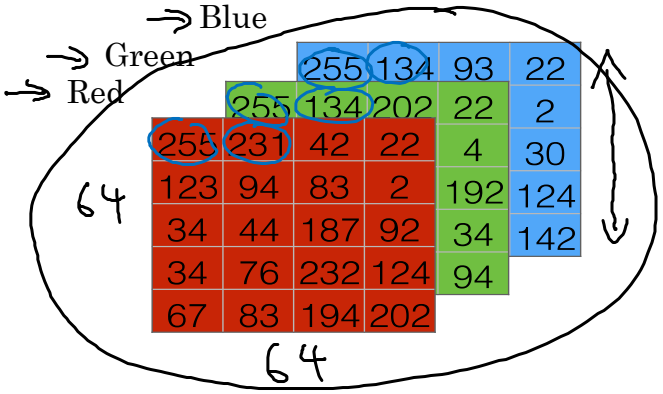
Binary Classification

64

64

1 (cat) vs 0 (non cat)

y



3 colour channels

$$x = \begin{bmatrix} 255 \\ 231 \\ \vdots \\ 255 \\ 134 \\ \vdots \end{bmatrix} \begin{matrix} \} \text{red} \\ \\ \\ \} \text{green} \\ \\ \} \text{blue} \end{matrix}$$

$64 \times 64 \times 3 = 12288$

$n = n_x = 12288$

$x \in \mathbb{R}^{n_x \times 1}$

$y = \{0, 1\}$

Notation

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

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

$m = m_{train}$ $\qquad m_{test}$ = #test examples.

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

n_x

training examples ↔ m

dim of each example

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

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & \cdots & y^{(m)} \end{bmatrix}$$

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

$Y.shape = (1, m)$

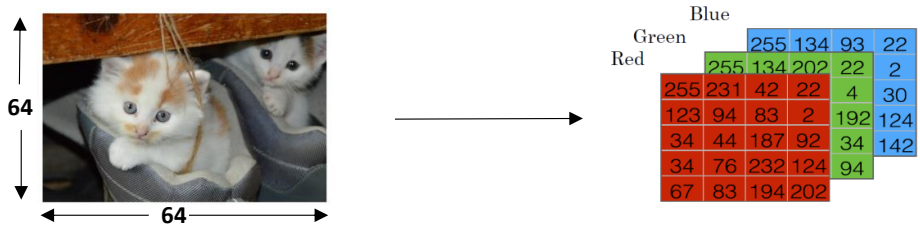
Binary Classification

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 store 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 $n_x = 64\ x\ 64\ x\ 3 = 12\ 288$.

n_x = dim of each example

m = number of examples

$$x = \begin{bmatrix} 255 \\ 231 \\ 42 \\ \vdots \\ 255 \\ 134 \\ 202 \\ \vdots \\ 255 \\ 134 \\ 93 \\ \vdots \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \rightarrow \text{Intensity} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ \\ \text{red} \\ \\ \\ \\ \text{green} \\ \\ \\ \\ \text{blue} \end{matrix}$$

$64 \times 64 \times 3 = \#\text{rows} = n_x$

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

$y \in \mathbb{R}$, $Y \in \mathbb{R}^{1 \times m}$ $\Rightarrow$ Y.shape = $(1, m)$

deeplearning.ai

Basics of Neural Network Programming

Logistic Regression

Logistic Regression

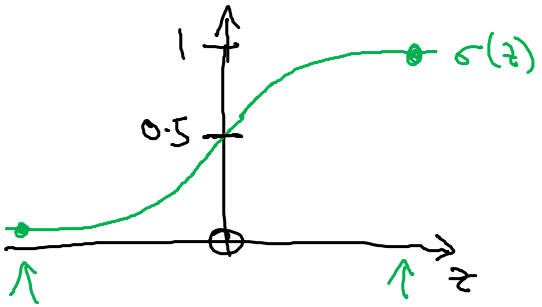
Given x, want $\hat{y} = P(y=1|x)$

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

$0 \le \hat{y} \le 1$

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

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



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

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

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

we don't use this notation.

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

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

If z large negative number

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

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.

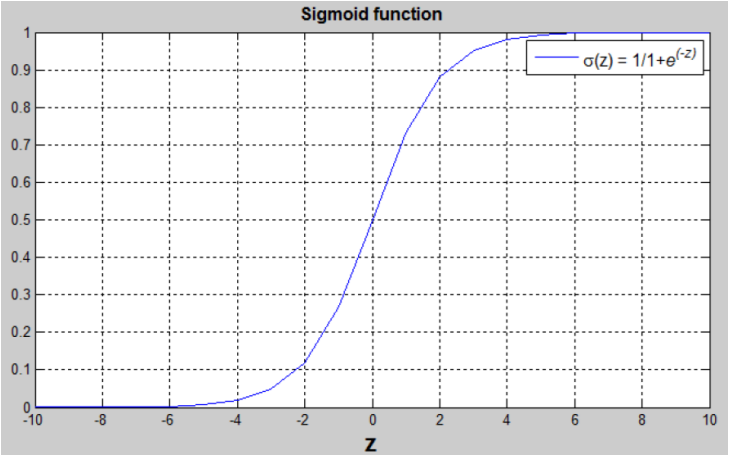
Example: Cat vs No - cat

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

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

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

$$\hat{y} = P(y=1|x) = \sigma(\underbrace{w^T x + b}_{z}) \Rightarrow \hat{y} = \begin{cases} 1 & z \to +\infty \\ 0 & z \to -\infty \end{cases}$$

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

deeplearning.ai

Basics of Neural Network Programming

Logistic Regression cost function

Logistic Regression cost function

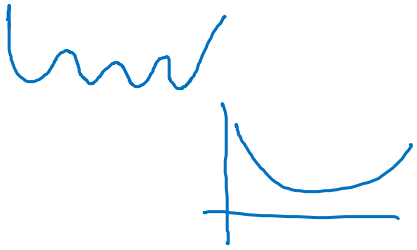
$\hat{y}^{(i)} = \sigma(w^T x^{(i)} + b)$, where $\sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}}$

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

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

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

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



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

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

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

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

Logistic Regression: Cost Function

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

Recap:

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

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

Goal

Given $\{(x^{(1)}, y^{(1)}), \cdots, (x^{(m)}, y^{(m)})\}$, 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.

$L(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{2}(\hat{y}^{(i)} - y^{(i)})^2$ → leads to non-convex problem ✗

$L(\hat{y}^{(i)}, y^{(i)}) = -(y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$ → convex ✓ (for logistic reg)

- 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.

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

(J: cost, L: loss)

Cost function = the average of loss functions over the traning set.

$$\begin{cases} \text{If } y=1 & p(y|x) = \hat{y} \\ \text{If } y=0 & p(y|x) = 1-\hat{y} \end{cases} \Rightarrow p(y|x) = \hat{y}^{y}(1-\hat{y})^{1-y} \Rightarrow \mathcal{L}(\hat{y}, y) = -\log p(y|x)$$

MLE $\prod_{i=1}^{m} p(y^{(i)}|x^{(i)}) = P(\text{all labels})$

$\Rightarrow \log P(\text{all label}) = \sum_{i=1}^{m} \log p(y^{i}|x^{i}) \Rightarrow J \equiv \frac{1}{m}\sum \mathcal{L}$

$-\mathcal{L}(\hat{y}^{i}, y^{i})$

deeplearning.ai

Basics of Neural Network Programming

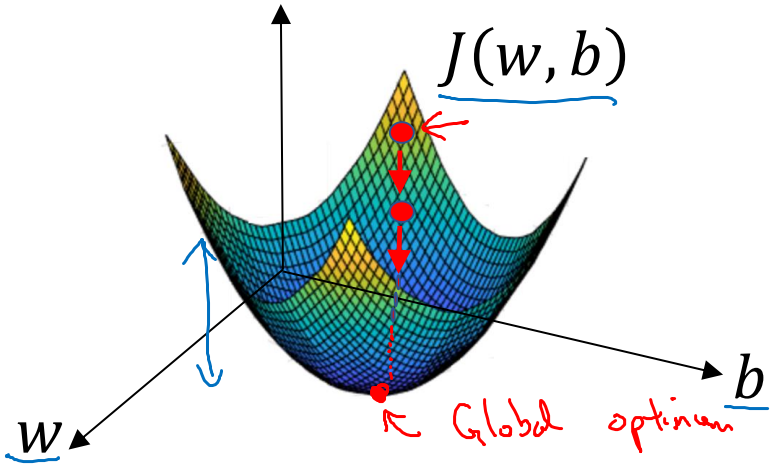
Gradient Descent

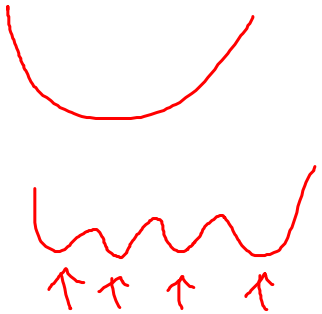
Gradient Descent

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

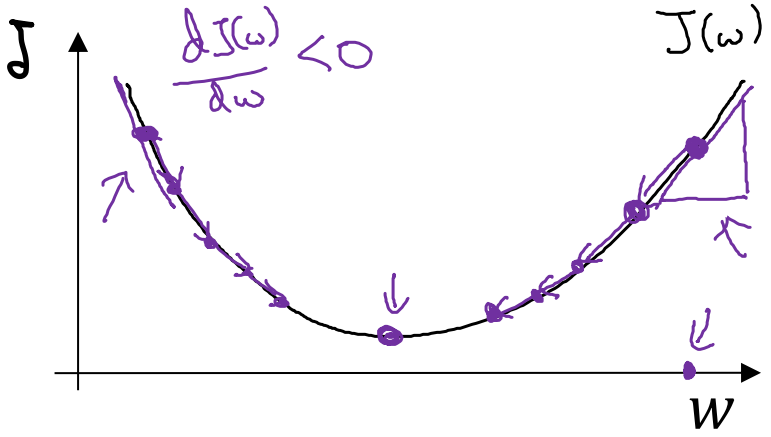
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

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





Gradient Descent



Repeat {

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

learning rate

"dw"

$w := w - \alpha dw$

}

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

$J(w,b)$

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

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

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

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

∂ "partial derivative"

J

$\begin{cases} w = w - \alpha \frac{\partial J}{\partial w} \\ b = b - \alpha \frac{\partial J}{\partial b} \end{cases}$ In Python, represent $\frac{\partial J}{\partial w} = dw$, $\frac{\partial J}{\partial b} = db$

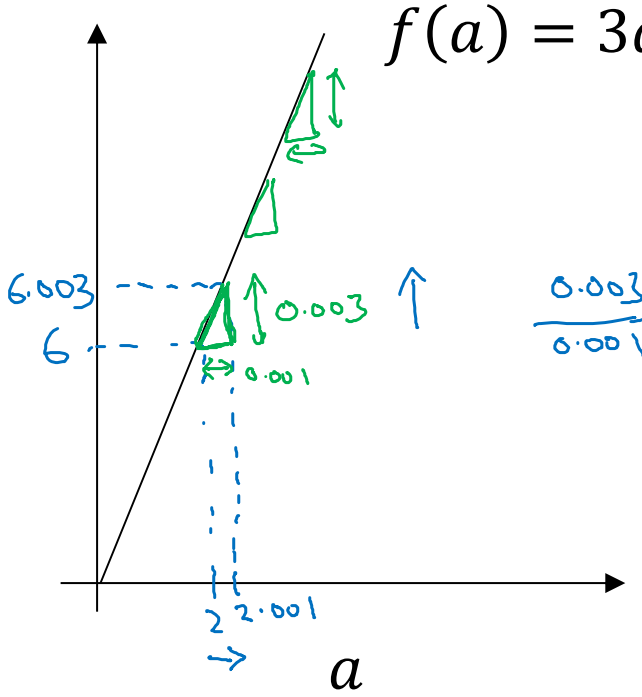
Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

Derivatives

Intuition about derivatives



$\frac{0.003}{0.001}$ $\frac{\text{height}}{\text{width}}$

→ $a = 2$ $f(a) = 6$

$a = 2.001$ $f(a) = 6.003$

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

→ $a = 5$ $f(a) = 15$

$a = 5.001$ $f(a) = 15.003$

slope at $a=5$ is also 3

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

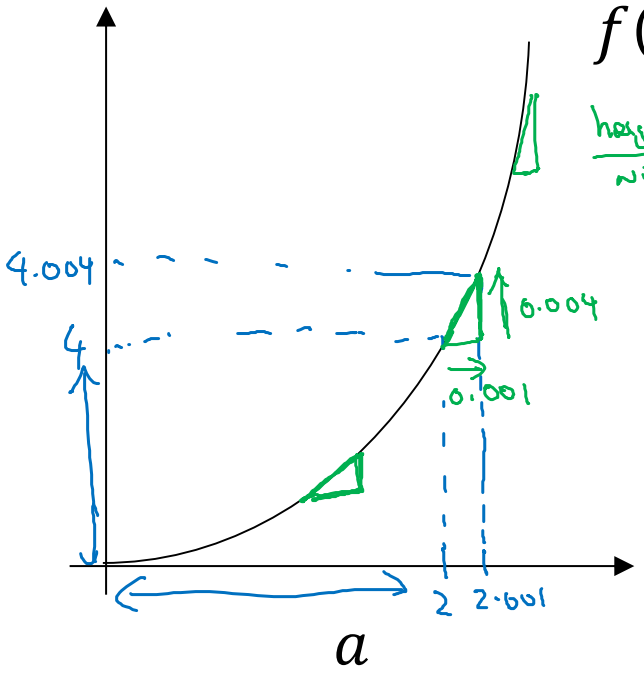
0.001 ←
0.00000001
0.000000001

deeplearning.ai

Basics of Neural Network Programming

More derivatives examples

Intuition about derivatives



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

0.001

$(2a) \times 0.001$

0.001 ←

0.00000....01 ←

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

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

(4.004001)

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

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

$a = 5$ $\quad f(a) = 25$

$a = 5.001$ $\quad f(a) \approx 25.010$

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

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

Andrew Ng

More derivative examples

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

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

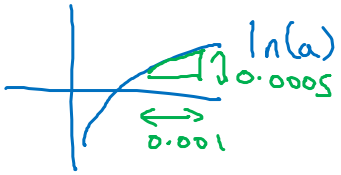
$a = 2.001$ $\quad f(a) \approx 4.004$

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

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

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

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



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

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

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

0.0005 $\quad 0.0005$

Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

Computation Graph

Computation Graph

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

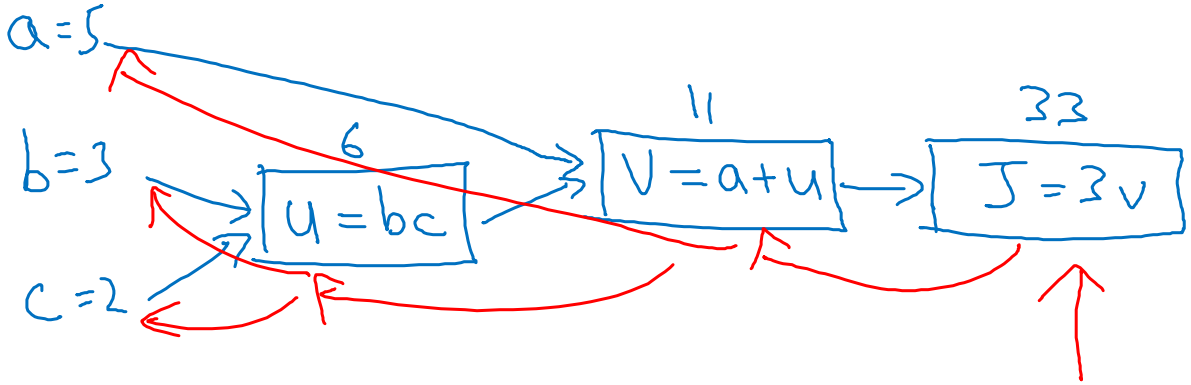
$J(a,b,c) = 3(a+\underbrace{bc}_{u}) = \underbrace{3(\underbrace{a+u}_{v})}_{J}$

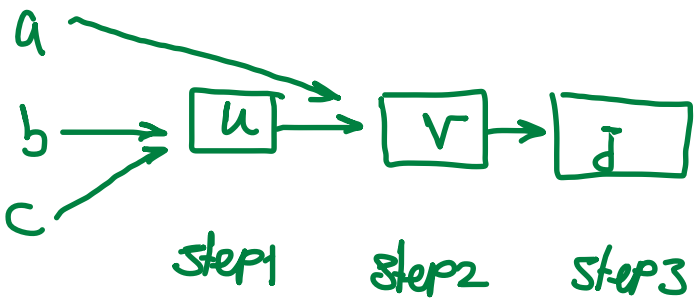
3 steps

$u = bc$

$v = a+u$

$J = 3v$





deeplearning.ai

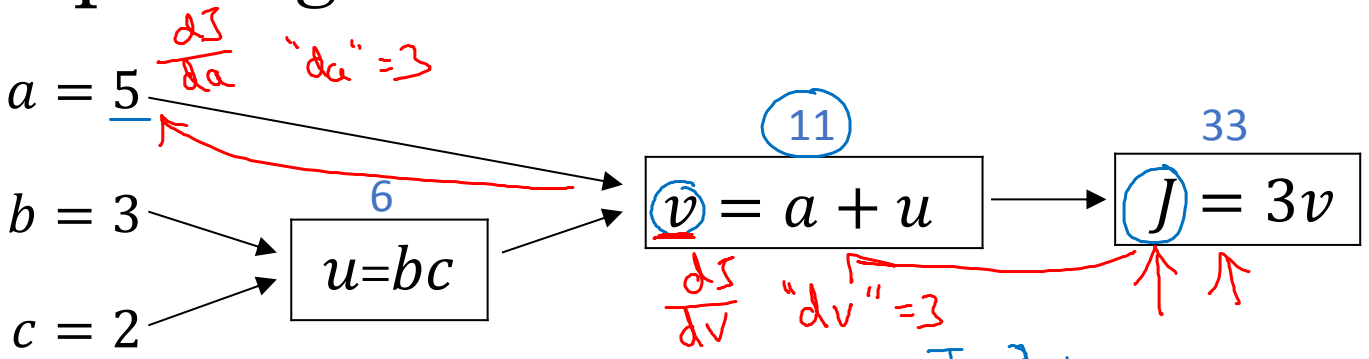
Basics of Neural Network Programming

Derivatives with a Computation Graph

Goal: $\frac{\partial(\text{Final output variable})}{\partial\, var} = \text{"dvar"} \equiv \frac{\partial J}{\partial var} = ?$ ↓ Python: $du = \frac{\partial J}{\partial u}$, $da = \frac{\partial J}{\partial a}$, ...

Back propagation $da = \frac{\partial J}{\partial a} = \frac{\partial J}{\partial v}\cdot\frac{\partial v}{\partial a} = dv\frac{\partial v}{\partial a}$

Computing derivatives



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

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

3×1

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

$a \to v \to J$

$J = 3v$

$v = 11 \to 11.001$

$J = 33 \to 33.003$

$a = 5 \to 5.001$

$\to v = 11 \to 11.001$

$J = 33 \to 33.003$

$\frac{d\,\text{FinalOutputVar}}{d\,var}$ → dJdvar → "dvar"

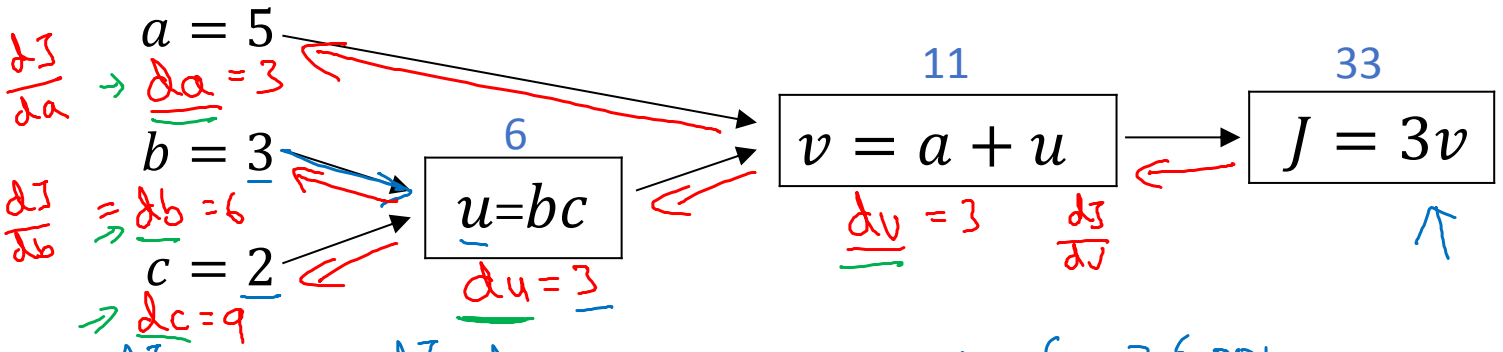
$f(a) = 3a$

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

$J = 3v$

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

Computing derivatives



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

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

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

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

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

$c = 2$
$.006$

$V = 11.002$
$J = 3V$

Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

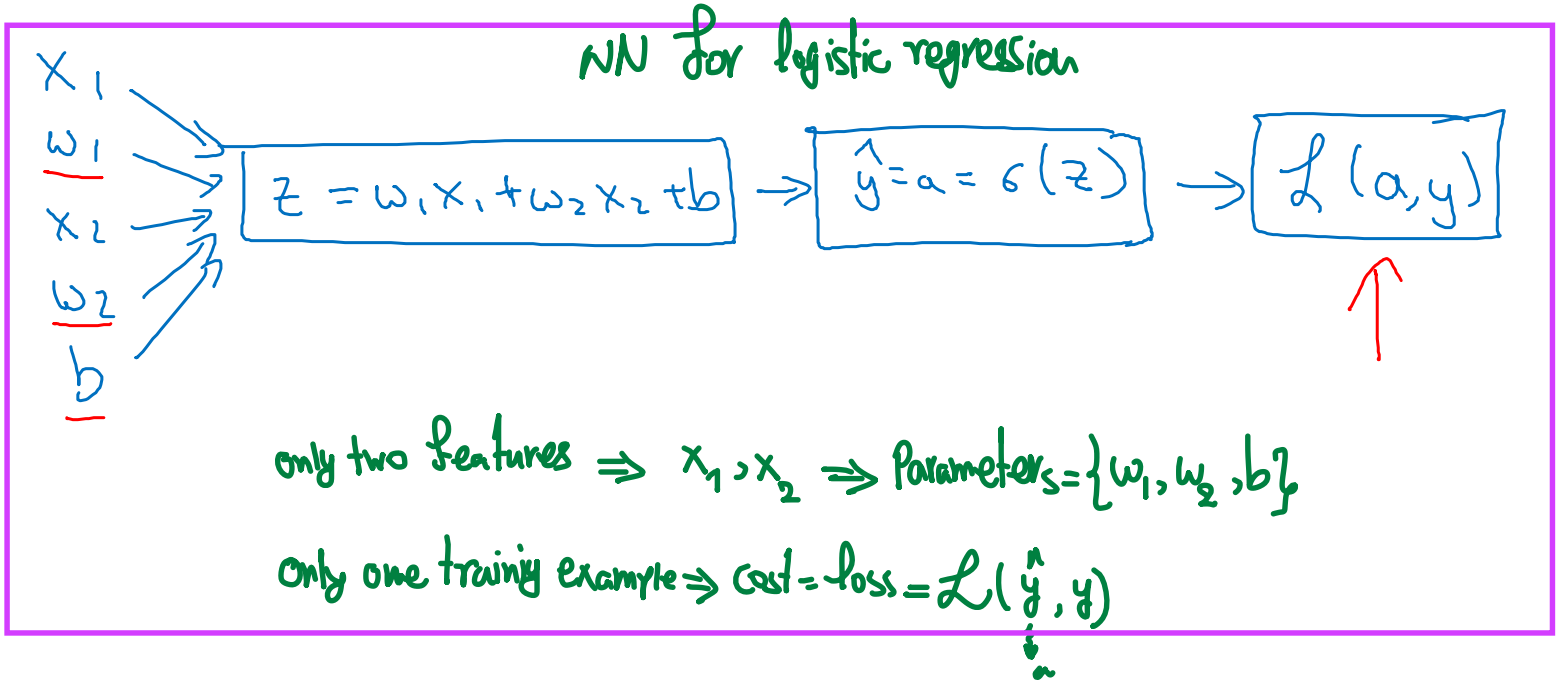
Logistic Regression Gradient descent

Logistic regression recap

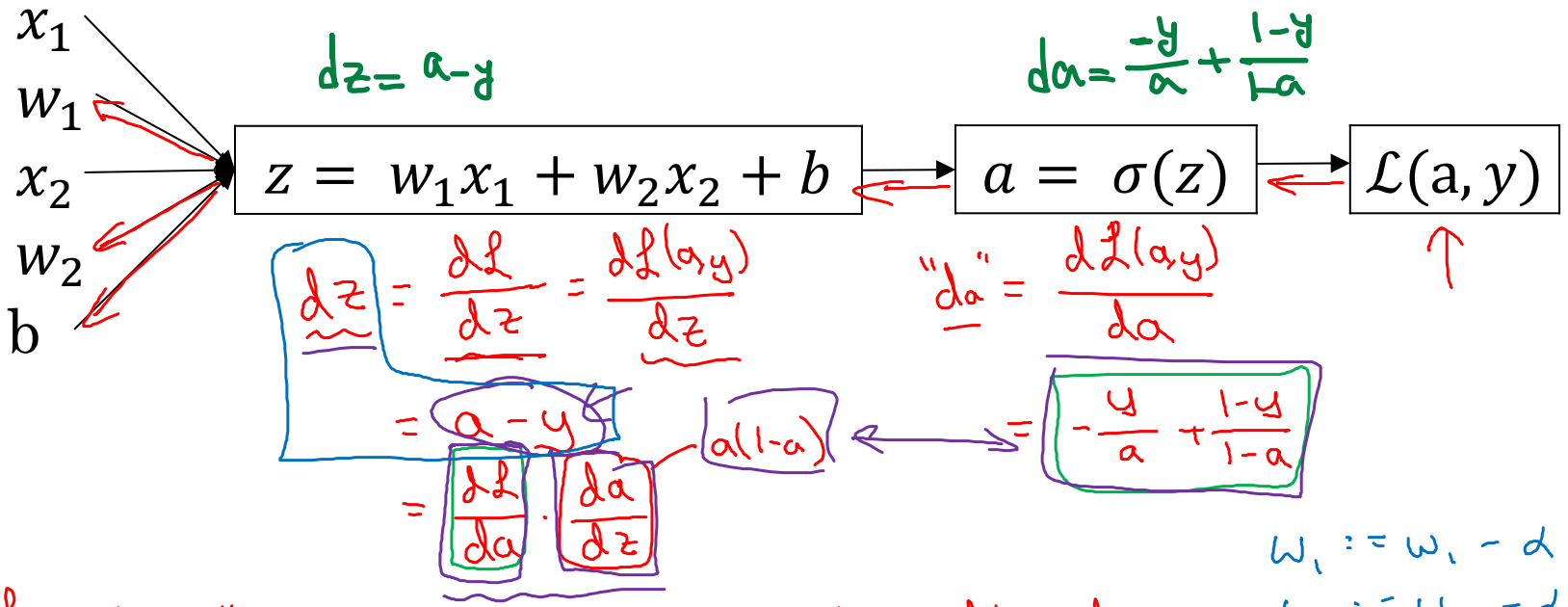
$$z = w^T x + b$$

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

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



Logistic regression derivatives



$\frac{\partial \mathcal{L}}{\partial w_1} = "dw_1" = x_1 \cdot dz$ $\quad dw_2 = x_2 \cdot dz$ $\quad db = dz$

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

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

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

$$\begin{cases} dw_1 = \frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial z}{\partial w_1} \cdot \frac{\partial \mathcal{L}}{\partial z} = x_1\, dz \\ dw_2 = x_2\, dz \\ db = dz \end{cases} \Rightarrow \begin{cases} w_1 = w_1 - \alpha\, dw_1 \\ w_2 = w_2 - \alpha\, dw_2 \\ b = b - \alpha\, db \end{cases}$$

deeplearning.ai

Basics of Neural Network Programming

Gradient descent on m examples

Logistic regression on m examples

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

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

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

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

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

$dw_1^{(i)}$ ← $(x^{(i)}, y^{(i)})$ for one observation

$$\Rightarrow \frac{\partial}{\partial w_1} J = \frac{1}{m}\sum_{i=1}^{m} dw_1^{(i)} \quad \text{where} \quad dw_1^{(i)} = \frac{\partial \mathcal{L}}{\partial w_1^{(i)}}$$

Logistic regression on m examples

$J=0;\ dw_1=0;\ dw_2=0;\ db=0$

loop1 (samples) → For $i=1$ to m

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

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

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

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

loop2 (features)

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

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

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

$n = 2$ #features

dw_3 … dw_n

$J /= m$

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

↑ average over m samples

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

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

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

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

Vectorization → To prevent the for loops

deeplearning.ai

Basics of Neural Network Programming

Vectorization

What is vectorization?

$z = w^T x + b$

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

Non-vectorized:

```
z = 0
for i in range(n_x):
    z += w[i] * x[i]
z += b
```

Vectorized

```
z = np.dot(w, x) + b
```

($w^T x$)

→ GPU
→ CPU

} SIMD - single instruction multiple data.

jupyter Vectorization demo Last Checkpoint: 3 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3

Code

```
a = np.random.rand(1000000)
b = np.random.rand(1000000)

tic = time.time()
c = np.dot(a,b)
toc = time.time()

print(c)
print("Vectorized version:" + str(1000*(toc-tic)) +"ms")

c = 0
tic = time.time()
for i in range(1000000):
    c += a[i]*b[i]
toc = time.time()

print(c)
print("For loop:" + str(1000*(toc-tic)) + "ms")
```

```
250286.989866
Vectorized version:1.5027523040771484ms
250286.989866
For loop:474.29513931274414ms
```

vectorized → 1,5 ms

non-vectorized → 474,2 ms

In [ ]:

deeplearning.ai

Basics of Neural Network Programming

More vectorization examples

Neural network programming guideline

Whenever possible, avoid explicit for-loops.

non-vectorized

$u = Av$

$u_i = \sum_j A_{ij} v_j$

```
u = np.zeros((n,1))
for i ...
    for j ....
        u[i] += A[i][j] * v[j]
```

vectorized

```
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])
```

non-vectorized

```
import numpy as np
u = np.exp(v)
```

```
np.log(v)
np.abs(v)
np.maximum(v,0)
v**2
1/v
```

vectorized operations in numpy

exp, log, abs, maximum, **, /,

Samples مربوط به ... یکی از loop‌ها را توانست حذف کند ← جلوتر حتی همان یکی را هم حذف می‌کند.

Features مربوط به ...

Logistic regression derivatives

J = 0, dw1 = 0, dw2 = 0, db = 0 ⇒ $dw = np.zeros((n_x, 1))$

for i = 1 to n:

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

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

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

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

for j = 1 ... n_x:

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

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

$n_x = 2$

⇒ $dw \mathrel{+}= x^{(i)} dz^{(i)}$

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

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

⇒ $dw /= m$

deeplearning.ai

Basics of Neural Network Programming

Vectorizing Logistic Regression

Vectorizing Logistic Regression

$$z^{(1)} = w^T x^{(1)} + b$$
$$a^{(1)} = \sigma(z^{(1)})$$

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

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

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

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

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

$1 \times m$

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

$b \in \mathbb{R}$ $(1,1)$ "Broadcasting"

$$A = [a^{(1)} \; a^{(2)} \cdots a^{(m)}] = \sigma(Z)$$

Vectorizing Logistic Regression

$$z^{(1)} = w^T x^{(1)} + b$$
$$a^{(1)} = \sigma(z^{(1)})$$

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

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

$$X = \begin{bmatrix} \vdots & & \vdots \\ x^{(1)} & \cdots & x^{(m)} \\ \vdots & & \vdots \end{bmatrix}_{n_x \times m} \Rightarrow Z = \begin{bmatrix} z^{(1)} & z^{(2)} & \cdots & z^{(m)} \end{bmatrix}_{1 \times m}, \quad b = [b \; b \; \cdots \; b]_{1 \times m}$$

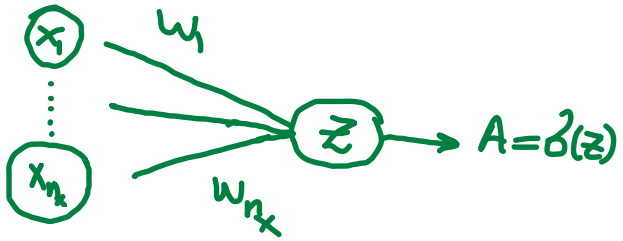
$$A = \begin{bmatrix} a^{(1)} & \cdots & a^{(m)} \end{bmatrix}$$

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

transpose

$+$ = broadcasting operator

$$A = \sigma(Z)$$



deeplearning.ai

Basics of Neural Network Programming

Vectorizing Logistic Regression's Gradient Computation

Vectorizing Logistic Regression

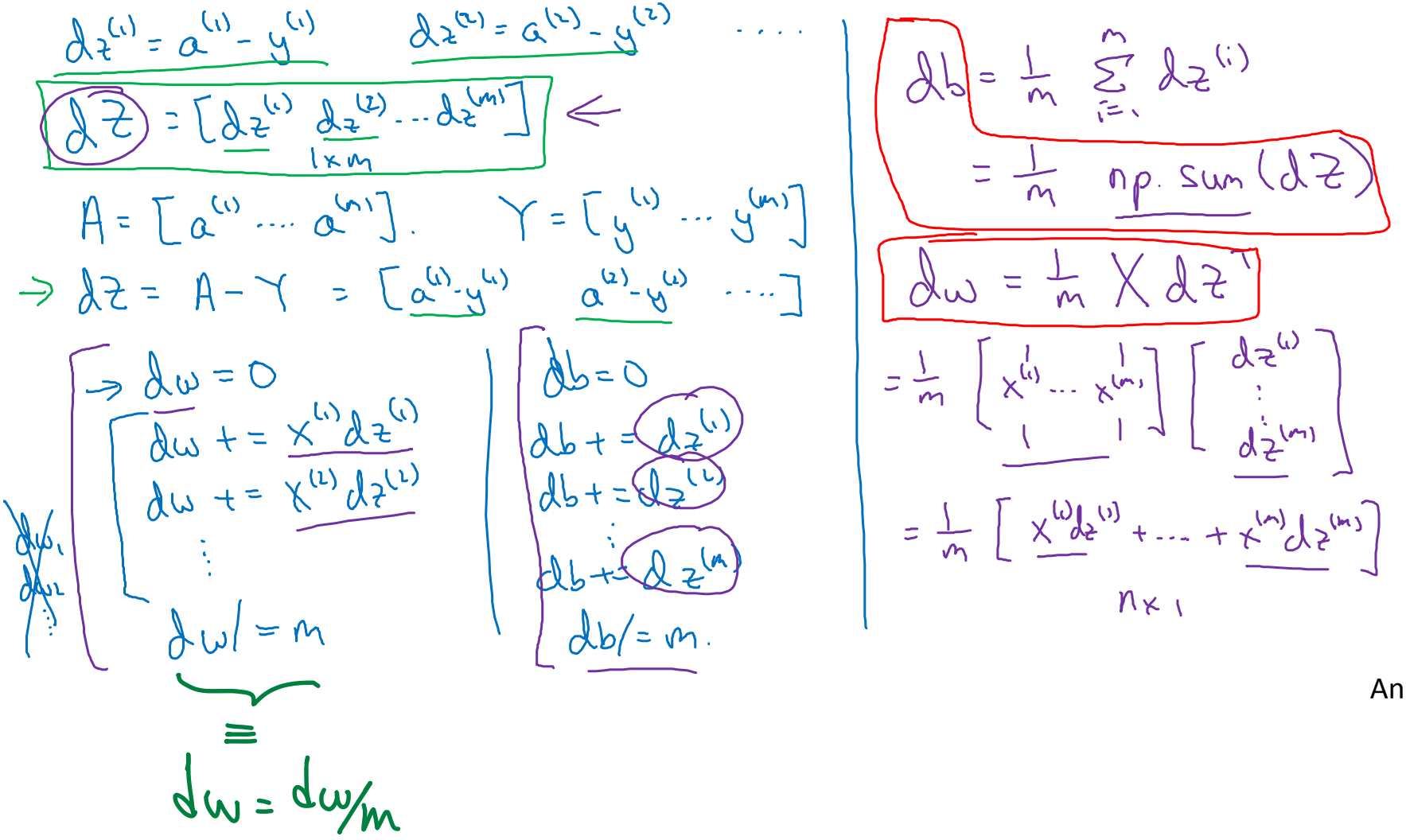
$$dZ = [dz^{(1)} \cdots dz^{(m)}] \;,\; A = [a^{(1)} \cdots a^{(m)}] \;,\; Y = [y^{(1)} \cdots y^{(m)}]$$

$$\Rightarrow dZ = A - Y$$

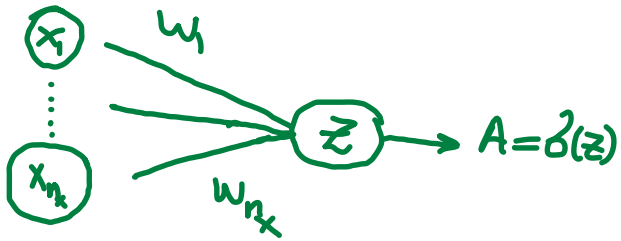
$$db = \frac{1}{m} \sum_{i=1}^{m} dz^{(i)} = \frac{1}{m} np.sum(dZ)$$

$$dw = \frac{1}{m} X dz^{T} = \frac{1}{m} np.dot(X, dz.T)$$

Vectorizing Logistic Regression



Andrew Ng



Implementing Logistic Regression

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

for i = 1 to m:

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

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

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

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

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

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

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

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

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

$db = db/m$

for iter in range(1000):

$Z = w^T X + b$

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

$A = \sigma(Z)$

$dZ = A - Y$

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

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

$w := w - \alpha dw$

$b := b - \alpha db$

for propagation

back propagation

Full Algorithm Gradient desent for logistic regression

deeplearning.ai

Basics of Neural Network Programming

Broadcasting in Python

می‌توان ابعاد ماتریس را به دلخواه ما تغییر کند reshape : x.reshape(1,4) ابعاد 1×4 می‌گیرد.

A.sum(axis=0) جمع روی "ستون"های A را می‌دهد که $A_{3\times4}$ است $A_{1\times4}$ تا شد می‌شود.

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

$= A_{(3,4)}$

59cal

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

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

Broadcasting example

$$\begin{bmatrix}1\\2\\3\\4\end{bmatrix} + \begin{bmatrix}100\\100\\100\\100\end{bmatrix} \cancel{100} = \begin{bmatrix}101\\102\\103\\104\end{bmatrix}$$

$$\underset{(m,n)}{\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}} + \underset{(1,n)\rightsquigarrow(m,n)}{\begin{bmatrix}100 & 200 & 300\\100 & 200 & 300\end{bmatrix}} = \begin{bmatrix}101 & 202 & 303\\104 & 205 & 306\end{bmatrix}$$

$$\underset{(m,n)}{\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}} + \underset{(m,1)\rightsquigarrow(m,n)}{\begin{bmatrix}100\\200\end{bmatrix}} \begin{bmatrix}100 & 100\\200 & 200\end{bmatrix} = \begin{bmatrix}101 & 102 & 103\\204 & 205 & 206\end{bmatrix}$$

General Principle of BroadCasting

(m, n) matrix $\quad \pm \quad * \quad / \quad$ $(1, n) \rightsquigarrow (m, n)$

(m, n) matrix $\quad \pm \quad * \quad / \quad$ $(m, 1) \rightsquigarrow (m, n)$

$(m, 1) \quad + \quad \mathbb{R}$

$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \quad + \quad 100 \quad = \quad \begin{bmatrix} 101 \\ 102 \\ 103 \end{bmatrix}$

$[1 \;\; 2 \;\; 3] \quad + \quad 100 \quad = \quad [101 \;\; 102 \;\; 103]$

Matlab/Octave: bsxfun $\rightsquigarrow$ Broadcasting in MATLAB and octave.

deeplearning.ai

Basics of Neural Network Programming

A note on python/ numpy vectors

Python / numpy vectors

```
import numpy as np
```

(5,)	`a = np.random.randn(5)`	"rank 1 array"	✗
(5,1)	`a = np.random.randn((5,1))`	column vector	✓
(1,5)	`a = np.random.randn((1,1))`	row vector	✓
T/F	`assert(a.shape == (5,1))`	checking dimensions	
(5,1)	`a=a.reshape((5,1))`	Transform dimensions	