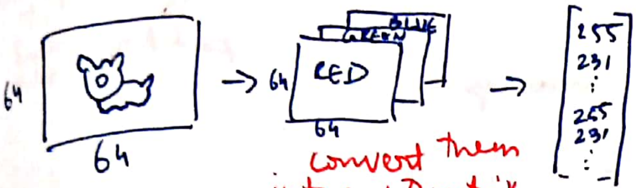
Week 2 : Basics of NN programming

Binary classification:

(Exa: Cat(1) or noncat(0))



convert them into a 1D matrix

$64 \times 64 \times 3 = 12288$

$n_x = 12288$

$y = \{1, 0\}$

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

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

Mtrain - no. of training examples

Mtest - no. of test examples

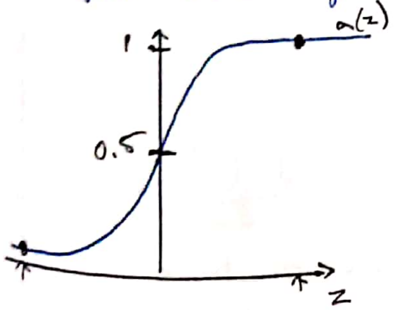
$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \end{bmatrix} \qquad X \in \mathbb{R}^{n_x \times m}$$

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

Logistic regression:

Given x, we want $\hat{y} = P(y=1/x)$ where $0 \leq \hat{y} \leq 1$

$\therefore$ We use sigmoid function $\hat{y} = \sigma(\overbrace{w^T x + b}^{z})$



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

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

If z large negative number, $\sigma(z) \approx \frac{1}{1 + \text{Big no.}} \approx 0$

Notation :

real number

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{n_x} \end{bmatrix} \begin{matrix} \} \; b \in \mathbb{R} \\ \left.\vphantom{\begin{matrix} a \\ b \\ c \\ d \end{matrix}}\right\} w \in \mathbb{R}^{n_x} \end{matrix}$$

n_x dimension vector

Logistic regression cost function:

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

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

prediction real value

∴ Loss (error) function: - for 1 example

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

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

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

∴ Cost function: - for m examples

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

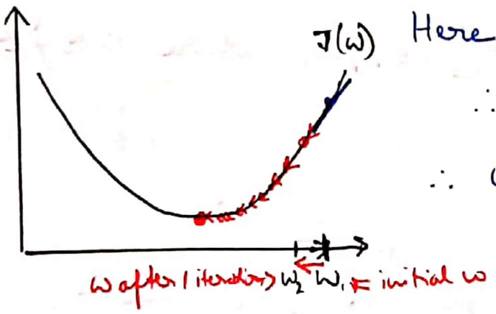
Gradient Descent:

Repeat (till convergence) {

$w := w - \alpha \frac{dJ(w)}{dw}$ ← This can be written as dw while coding and for b, db

}

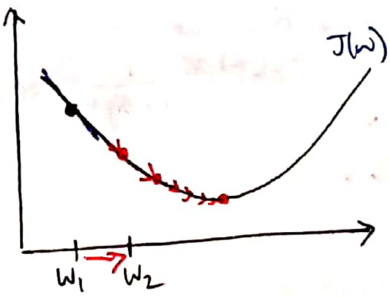
learning rate



Here slope is +ve ∴ $\frac{dJ(w)}{dw} > 0$

∴ $w := w - \alpha \cdot$ something

∴ w reduces



Here slope is -ve ∴ $\frac{dJ(w)}{dw} < 0$

∴ $w := w - \alpha \cdot -$something

$= w := w + \alpha \cdot$ something

∴ w increases

For $J(w,b)$,

Repeat {

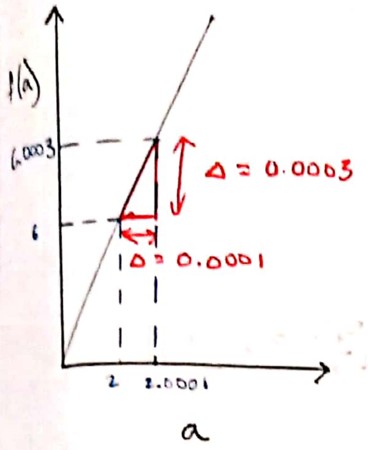
$w := w - \alpha \frac{\partial J(w,b)}{\partial w}$

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

}

← we use ∂ (partial derivation) instead of d when there are more than 1 variables (Exa: here there is w and b)

Derivatives :



~~f(a) = 6.0006~~

when $a = 2$, $f(a) = 6$

$a = 2.0001$, $f(a) = 6.0003$

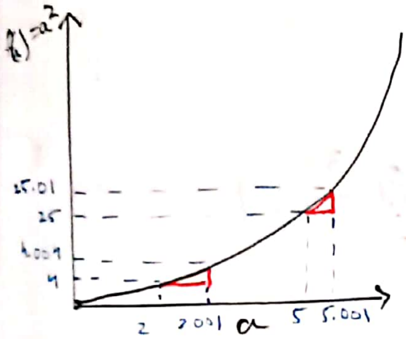
Slope/derivative of $f(a)$

$= \frac{\text{height}}{\text{width}} = \frac{0.003}{0.001} = 3$

$\therefore \frac{df(a)}{da}$ or $\boxed{\frac{d}{da} f(a) = 3 \text{ constant}}$ $\quad \therefore \frac{d}{da} f(a) = 3a$

Note : Here slope remains constant (always 3)

remember derivative of 3a was 3?



(1) when $a = 2$, $f(a) = 4$

$a = 2.001$, $f(a) = 4.004$

$\therefore$ slope/derivative of $f(a)$ at $a = 2$ is 4

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

(2) when $a = 5$, $f(a) = 25$

$a = 5.001$, $f(a) = 25.01$

$\therefore$ slope/derivative of $f(a)$ at $a = 5$ is 10

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

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

More examples:

$f(a) = a^3 \qquad \frac{d}{da} f(a) = 3a^2 \quad \Leftarrow \quad a = 2 \quad f(a) = 8$

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

$3 \times 2^2 = 12 \qquad \Delta = 12$

$f(a) = \log_e(a)$ or $\ln(a) \qquad \frac{d}{da} f(a) = \frac{1}{a} \quad \Leftarrow \quad a = 2 \quad f(a) = 0.69315$

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

$\frac{1}{2} = 0.0005 \qquad \Delta = 0.0005$

Scanned with CamScanner

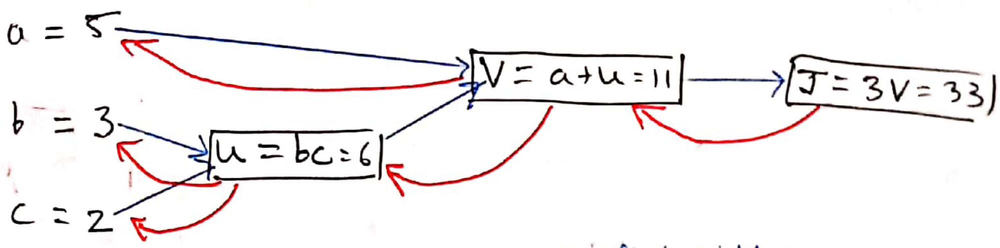
Computation Graph:

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

$\therefore\ u = bc$

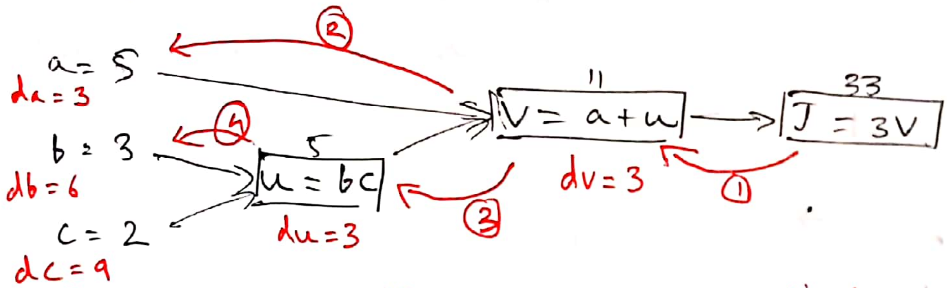
$v = a + u$

$J = 3v$



- Left-to-right pass
- Right-to-left pass ⇐ To find derivative

Finding derivatives:



① $\dfrac{dJ}{dv} = ?$

$J = 3V$

$V = 11 \rightarrow 11.001$

$J = 33 \rightarrow 33.003$

⇐ similar to $f(a) = 3a$

$\dfrac{d f(a)}{da} = 3$ from previous example

In coding notation this is dv

$\therefore\ \dfrac{dJ}{dV} = 3$

② $\dfrac{dJ}{da} = ?$

~~da~~ $a = 5 \rightarrow 5.001$

$V = 11 \rightarrow 11.001$

$J = 33 \rightarrow 33.003$

Chain rule.

If a affects v affects J,

Then $\dfrac{dJ}{da}$ ⇐ effect of a on J

$= \dfrac{dJ}{dv} \cdot \dfrac{dv}{da}$ ← effect of a on v

effect of v on J

In coding notation this is da

$\therefore\ \dfrac{dJ}{da} = 3 = \dfrac{dJ}{dV} \cdot \dfrac{dv}{da}$

$3 \cdot 1 = 3$

we found this in step ①

③ $\dfrac{dJ}{du} = \dfrac{dJ}{dV} \cdot \dfrac{dV}{du} = 3 \cdot 1 = 3$

du

$u = 6 \rightarrow 6.001$

$V = 11 \rightarrow 11.001$

$J = 33 \rightarrow 33.003$

④ $\dfrac{dJ}{db} = \dfrac{dJ}{du} \cdot \dfrac{du}{db} = 3 \cdot 2 = 6$

db

$b = 3 \rightarrow 3.001$

$u = b \cdot c \Rightarrow 6 \rightarrow 6.002$

3.001×2

c = 2 here only b is ↑

⑤ $\dfrac{dJ}{dc} = \dfrac{dJ}{du} \cdot \dfrac{du}{dc} = 3 \cdot 3 = 9$

dc

$c = 2 \rightarrow 2.001$

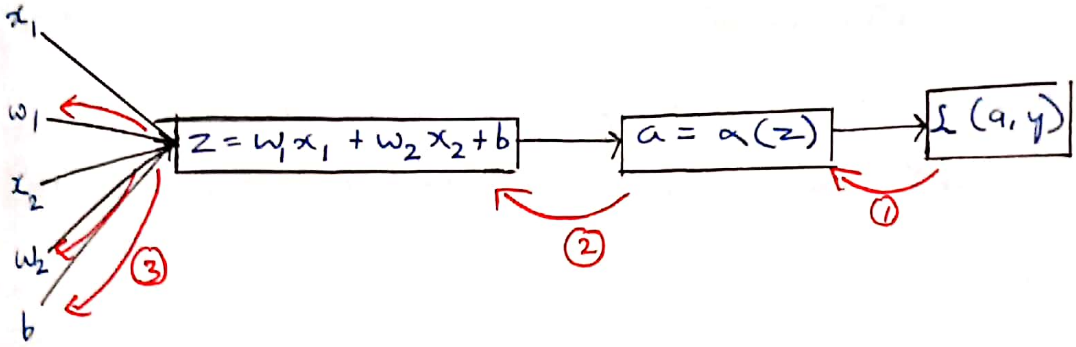
$u = b \cdot c = 6 \rightarrow 6.003$ ⇐ 2.001×3

Logistic Regression Gradient Descent:

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

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

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



① $da = \dfrac{d\mathcal{L}(a,y)}{da} = -\dfrac{y}{a} + \dfrac{1-y}{1-a}$ //

② $dz = \dfrac{dL}{da} \cdot \dfrac{da}{dz} = a - y$ //

$a(1-a)$

$\uparrow$ $\dfrac{dL}{dz}$

③ $dw_1 = \dfrac{dL}{dw_1} = x_1 \, dz$

$dw_2 = \dfrac{dL}{dw_2} = x_2 \, dz$

$db = \dfrac{dL}{db} = dz$

$\therefore$ Gradient descent (one step):

~~Repeat~~

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

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

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

Gradient Descent for m examples:

The cost function for logistic regression is,

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

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

In the previous example we find $dw_1^{(i)}$, $dw_2^{(i)}$, $db^{(i)}$ for a single training example $(x^{(i)}, y^{(i)})$

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

Algorithm:

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

For $i = 1$ to M

Forward propagation
$$\left\{\begin{aligned} z^{(i)} &= w^T x^{(i)} + b \\ a^{(i)} &= \sigma(z^{(i)}) \\ J &\mathrel{+}= -[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})] \end{aligned}\right.$$

Backward propagation
$$\left\{\begin{aligned} dz^{(i)} &= a^{(i)} - y^{(i)} \\ dw_1 &\mathrel{+}= x_1^{(i)} dz^{(i)} \\ dw_2 &\mathrel{+}= x_2^{(i)} dz^{(i)} \\ db &\mathrel{+}= dz^{(i)} \end{aligned}\right.$$

For $n = 2$, otherwise we would find $dw_1, dw_2, dw_3, dw_4 \ldots dw_n$

$J \mathrel{/}= m$

$dw_1 \mathrel{/}= m$

$dw_2 \mathrel{/}= m$

$db \mathrel{/}= m$

Using for loops is inefficient, so we use vectorization.

Vectorization:

We need to compute $z = w^T x + b$

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

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

Non-vectorized	Vectorized
z = 0 for i in range(n-x): z + = w[i] * x[i] z += b	$z = \underbrace{np.dot(w, x)}_{w^T x} + b$

Code

```
import numpy as np

a = np.array([1, 2, 3, 4])
```
– initialise array

```
u = np.exp(v)
```
– you can use np functions that'll take effect on all the vector values

np.log(v) np.abs(v) np.maximum(v, 0) ....

→ Here we will need a loop, instead we can use vectorization

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

However even here, we use a for loop to iterate through all the training examples. Instead we can vectorize

$$[z^{(1)}\ z^{(2)} \dots z^{(m)}] \rightarrow Z = np.dot(w^T, X) + b$$

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

↑ with broadcasting it'll become $[b\ b \dots b]$ $\mathbb{R}^{1 \times m}$

$$A = \sigma(Z)$$

$[a^{(1)}\ a^{(2)} \dots a^{(m)}]$

$$dz = A - Y$$

$[dz^{(1)}\ dz^{(2)} \dots dz^{(m)}]$ $[a^{(1)}\ a^{(2)} \dots a^{(m)}]$ $[y^{(1)} \dots y^{(m)}]$

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

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

After that we update w and b:

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

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

Full algorithm:

for iter in range(1000): ↙ running gradient descent 1000 times (we can't vectorise this)

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

$$A = \sigma(z)$$

$$dz = A - Y$$

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

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

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

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

Broadcasting in Python

Python automatically rescales variables to fit the array/vector calculation

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

② $\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix} + \begin{bmatrix}100 & 200 & 300\end{bmatrix} \Rightarrow \begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix} + \begin{bmatrix}100 & 200 & 300\\100 & 200 & 300\end{bmatrix} = \begin{bmatrix}101 & 202 & 303\\104 & 205 & 306\end{bmatrix}$

③ $\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix} + \begin{bmatrix}100\\200\end{bmatrix} \Rightarrow \begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix} + \begin{bmatrix}100 & 100 & 100\\200 & 200 & 200\end{bmatrix} = \begin{bmatrix}101 & 102 & 103\\204 & 205 & 206\end{bmatrix}$

Common Bugs

① While initialising vectors, declare both the no. of rows & columns.

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

column vector — row vector

If you do a=np.random.randn(5), then a' won't work and np.dot(a, aT) won't work as expected

② You can use a = a.reshape((5,1))

Derivation for Logistic Regression Cost Function (Optional)

If $y = 1, \quad P(y|x) = \hat{y}$ - ①

If $y = 0, \quad P(y|x) = 1-\hat{y}$ - ②

$$P(y|x) = \hat{y}^y (1-\hat{y})^{(1-y)}$$

This equation is verified

If $y=1, \; P(y|x) = \hat{y}^1(1-\hat{y})^0 = \hat{y}$ - ①

If $y=0, \; P(y|x) = \hat{y}^0(1-\hat{y})^1 = 1-\hat{y}$ - ②

maximise this

$\uparrow \log P(y|x) = \log \hat{y}^y (1-\hat{y})^{(1-y)}$

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

$= -\mathcal{L}(\hat{y}, y) \downarrow$ To minimise loss (↓ since we added -)

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

For m training examples