Date ___/___/___

Week 2

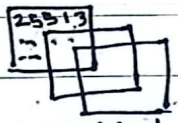
m training example ⇒ process without explicit for loop.

after forward propogation → backward propogation

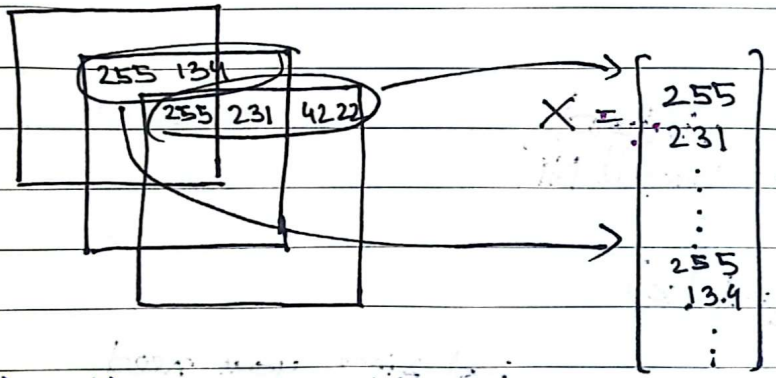
binary classification - cat or non-cat $x \rightarrow y$

64×64 image

rgb → all have separate 64×64 matrix



unrow pixel intensity value into feature vector



total dimension:

64×64×3 = 12288

$n_x = 12288$

Notation

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

↓

single training example

training set comprises of m training examples

$(x^{(i)}, y^{(i)})$ → input & output for first training example.

entire training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \ldots\}$

$m = M_{train}$ m_{test} = # test examples

↓

no. of training examples

MINDFUL NOTES

Date ___/___/___

compact notation of entire training set:

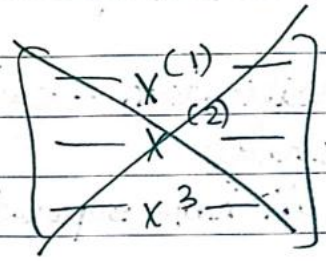
$$X = \begin{bmatrix} | & | & | & & | \\ x^{(1)} & x^{(2)} & x^{(3)} & \cdots & x^{(m)} \\ | & | & | & & | \end{bmatrix}$$

n_x rows

m cols

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

Stack in columns



won't use this notation

$X.shape = (n_x, m)$

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & y^{(3)} & \cdots & y^{(m)} \end{bmatrix}_{1 \times m}$$

$Y.shape = (1, m)$

Logistic Regression → Small NN (visualized)

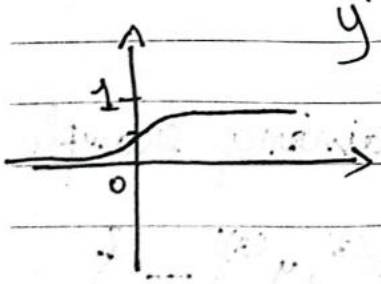
given x, want $\hat{y} = P(y=1|x)$

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

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

output $\hat{y} = w^T x + b$ X not good for classification (bad since you need output b/w 1 & 0)

$\hat{y} = \sigma(w^T x + b)$ so we use sigmoid function



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

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

Date ___/___/___

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

if z is very large: $\sigma(z) \approx \frac{1}{1+0} = 1$ (close to 1)

if z is very small / very large -ve number:

$$\sigma(z) = \frac{1}{1+e^{-z}} \approx \frac{1}{1+\text{big no.}} \approx 0 \text{ (close to 0)}$$

job in logistic regression is to learn parameters w & b, so that $\hat{y}$ becomes a good estimate of the chance of y being equal to 1.

Keep w & b parameters seperate.

different notation: → not going to be used in this course

- define extra feature $x_0 = 1$, so $x \in \mathbb{R}^{n+1}$
- $\hat{y} = \sigma(\theta^T x)$

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

=> for implementing neural networks, it is best to keep w & b seperate, so don't use the x_0 notation for this course

Date ___/___/___

Cost function - Logistic Regression

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

Given $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$,
want $\hat{y}^{(i)} \approx y^{(i)}$

m training example

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

$x^{(i)}$
$y^{(i)}$
$z^{(i)}$
} denotes data associated with ith training example.

Loss (error) function:

$\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$ ✗ we don't do this in logistic regression, as optimization problem becomes non-convex. ~~~ local-optima problem

Loss function:

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

If $y = 1$: $\mathcal{L}(\hat{y}, y) = -\log \hat{y}$ want $\log \hat{y}$ large or = 1, as it can't be bigger than 1

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

Cost function:

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

Loss function → applied to single training example

cost function → cost of your parameter (entire training set)

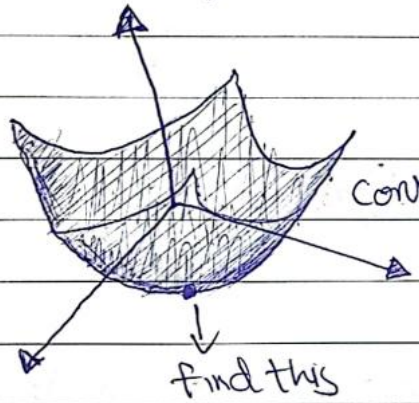
MINDFUL NOTES

Date ___/___/___

cost function → avg of the loss function of the entire training set

Gradient Descent

→ want to find w, b that minimizes J(w,b)



slope of a function at a point

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

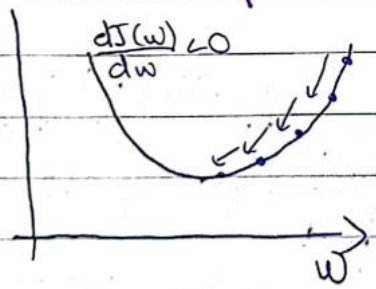
learning rate

"dw" variable name for derivative term in code

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

derivative is +ve	derivative is -ve
so you decrease	so you add
	$-(-dw)$
	$+dw$

GD takes a step downward in the steepest descent.



repeat {

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

dw → short form

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

}

Date ___/___/___

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

slope

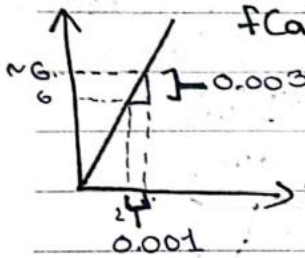
$J(w,b)$

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

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

actual — in code: — small d : ∂

Partial derivative: when J is a function of 2 or more variables, we use partial derivative.



$f(a) = 3a$

$a = 2 \quad f(a) = 6$

$a = 2.001 \quad a = 6.003$

Slope (derivative of f(a)) at a=2 is 3

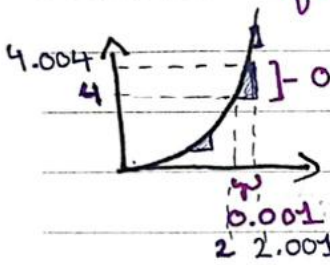
$\frac{\text{height}}{\text{width}} = \frac{0.003}{0.001}$

$a = 5 \quad f(a) = 15 \Rightarrow$ slope at a=5 is also 3

$a = 5.001 \quad f(a) = 15.003 \quad \frac{df(a)}{da} = 3 = \frac{d}{da} f(a)$

on a straight line:

the function's derivative doesn't change



$f(a) = a^2$

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

$a = 2.001 \quad f(a) \approx 4.004 \quad (4.004004)$

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

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

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

$a = 5 \quad f(a) = 25 \quad \frac{d}{da} f(a) = 10$ when $a = 5$

$a = 5.001 \quad f(a) = 25.010 \rightarrow$ goes up 10×

↳ slope is different, for different values of a

for any given value of a if you nudge it up by 0.001, f(a) goes up by $2a \times (0.001)$

derivative — nudge

nudge 0.001 → $f(a) \approx 4.004$ 4

nudge 0.000 ... 1 → $f(a) = 4.004$, the infitesmally small part goes away.

Date ___/___/___

why not exact? because derivatives are defined using these infinitesmally small nudges rather than 0.001

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

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

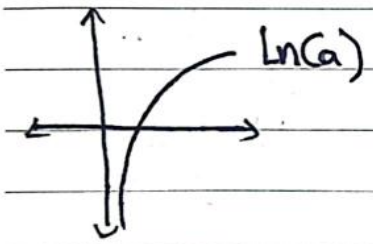
$f(a) = a^3$ $\frac{d}{da} f(a) = 3a^2$ $a=2$ $f(a)=8$

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

$f(a) = \log_e(a) = \ln(a)$ $\frac{d}{dv} f(a) = \frac{1}{a}$ $a=2$ $f(a)=0.69315$

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

$\frac{d}{dx} f(2) = \frac{1}{2}$



derivative is just a slope of line.

Computation Graph → organizes computation with blue arrow, left to right computation.

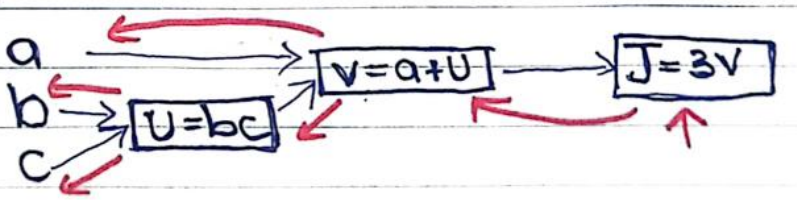
$J(a,b,c) = 3(a+bc)$

u, v, J

simpler graph:

3 distinct steps:

1) $u = bc$
2) $v = a+u$
3) $J = 3v$

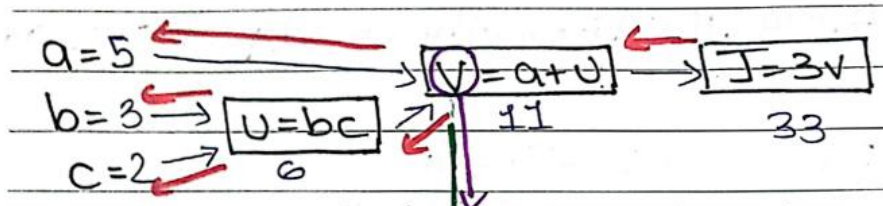


for computing derivative, go in opposite direction

MINDFUL NOTES

Date ___/___/___

Derivatives with Computation Graph



if we change value v, how does it change J?

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

$J = 3v$

$v = 11 \rightarrow 11.001$

$J = 33 \rightarrow 33.003$

3x

$f(a) = 3a$

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

$J = 3v$

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

→ to compute derivative of final output variable wrt. to v, then we have done one step of backward propagation.

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

$a = 5 \rightarrow 5.001$

$v = 11 \rightarrow 11.001$

$J = 33 \rightarrow 33.003$

change a → changes

chaine rule

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

$\frac{dv}{da} = 1$, $\frac{dJ}{dv} = 3$

$= 1 \times 3$

$= 3$

Date ___/___/___

final output variable (you really care about)
→ J in this case

will compute derivative with J of intermediate variables like a, b, c so name the final variable (J) like dFinalOutputVar, djVar in code.
'dVar'

$$\frac{dJ}{dv} = 3 = \frac{dJ}{dv} \cdot \frac{dv}{dv} \quad (3 \cdot 1)$$

v=6 → 6.001
v=11 → 11.001
J=33 → 33.003

$$\frac{dJ}{db} = \frac{dJ}{du} \cdot \frac{du}{db} \quad (3 \cdot 2) = 6$$

b=3 → 3.001
u=bc → 6 → 6.002 c=2

$$\frac{dJ}{dc} = \frac{dJ}{dv} \cdot \frac{dv}{dc} \quad (3 \cdot 3) = 9$$

★ when computing derivatives, the most efficient way to do so, is a right to left computation, following the direction of arrow.

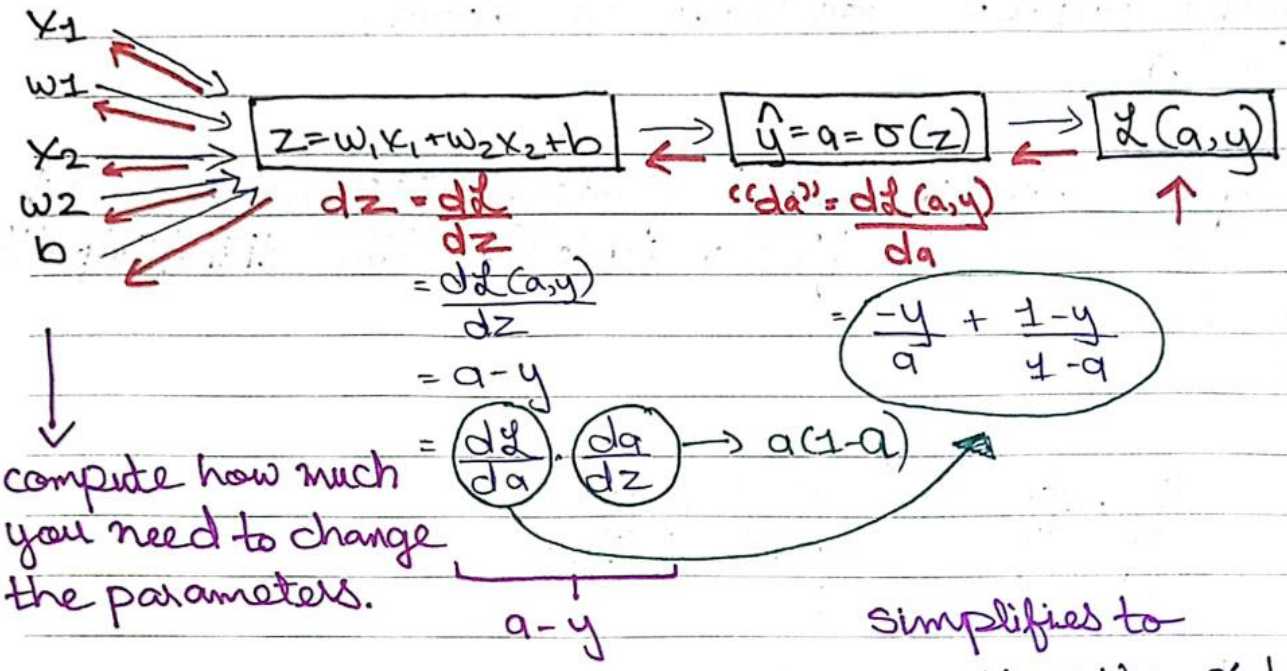
Gradient Descent for Logistic Regression

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

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

MINDFUL NOTES

Date ___/___/___



Gradient Descent on m Examples

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

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

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

$dw_1^{(i)} \rightarrow (x^{(i)}, y^{(i)})$

Simplifies to

$w_1 = w_1 - \alpha\, dw_1$

$w_2 = w_2 - \alpha\, dw_2$

$b = b - \alpha\, db$

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

$dw_1^{(i)}, dw_2^{(i)}, db^{(i)}$ — notation for one training example

implements 1 step of gradient descent, Date ___/___/___
so you have to do multiple times:

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

for i = 1 to m → 2nd for loop

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

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

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

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

$dw_1 += x_1^{(i)} dz^{(i)}$ ↕ // assuming n=2] 1 for loop over the features

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

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

$J /= m$

$dw_1 /= m;\ dw_2 /= m;\ db /= m$] used as accumulators, so they do not have superscript

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

$w_1 := w_1 - \alpha dw_1$

$w_2 = w_2 - \alpha dw_2$

$b = b - \alpha db$] // to implement 1 step of GD, you do

2 weaknesses here:

1) you need to write 2 for loops, 1 loop over the features
2) having explicit for loops in your code for deep learning algorithms, makes your algorithm run less efficiently.

↓

Solution? → ★ **Vectorization Techniques** - allows you to get rid of explicit for loops

$z = w^T x + b \qquad w = \begin{bmatrix} \\ \\ \end{bmatrix};\ x = \begin{bmatrix} \\ \\ \end{bmatrix} \quad w \in R^{n_x} \quad x \in R^{n_x}$

non vectorized:

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

vectorized:

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

$w^T x$

MINDFUL NOTES

Date ___/___/___

say you can/need to apply the exponential operation on every element of a matrix

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

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

vectorized short code for applying exp. op on every element

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

```
np.log(v)
np.abs(v)
np.maximum(v,0)   → element wise max
v**2              → square
1/v               → inv
```

other useful functions

Vectorizing Logistic Regression's Gradient output

Vectorizing Logistic Regression

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

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

$\dots$

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

how to do it without explicit for loop?

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

$w^T X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$ → row vector

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

$$= \begin{bmatrix} w^T x^{(1)} + b & w^T x^{(2)} + b & \cdots & w^T x^{(m)} + b \end{bmatrix}_{1 \times m}$$

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

← (1,1) real no. here, python automatically converts it to (1,m) row vector (Broadcasting)

$$A = \begin{bmatrix} a^{(1)} & a^{(2)} & \cdots & a^{(m)} \end{bmatrix} = \sigma(Z)$$

MINDFUL NOTES

Date ___/___/___

Vectorizing Logistic Regression's Gradient Output

$dz^{(1)} = a^{(1)} - y^{(1)}$ $\quad dz^{(2)} = a^{(2)} - y^{(2)}$ ...

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

$1 \times m$ or m dimensional row vector

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

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

We now have to work to remove the second for loop

```
dw = 0                          db = 0
dw += x(1) dz(1)                db += dz(1)
dw += x(2) dz(2)      for loop  :                 for loop
dw /= m                         db += dz(m)
                                db /= m
```

vectorized sol:

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

$dw = \frac{1}{m} X \cdot dZ^{T}$ // $= \frac{1}{m} \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \end{bmatrix} \begin{bmatrix} dz^{(1)} \\ \vdots \\ dz^{(m)} \end{bmatrix}$

single iteration of gradient descent for logistic regression // $\frac{1}{m} [x^{(1)} dz^{(1)} + \dots + x^{(m)} dz^{(m)}]$ $n \times 1$

: ($z = w^T x + b$) // not code, just explanation

```
Z = np.dot(w.T, x) + b
A = σ(Z)
dZ = A - Y
dw = 1/m X dZ^T
db = 1/m np.sum(dZ)
w = w - α dw
b = b - α db
```

// for multiple iterations, you still need a for loop over the no. of iterations.

MINDFUL NOTES

Date ___/___/___

Broadcasting in Python

eg1: Calories from Carbs, Protens, 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
Fats	1.8	135.0	99.0	0.9

$= A_{3\times4}$

↓ Sum down the column

59 cal

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

Q. Calculate % of calories from carbs, protens, fats.

Sum each of the columns, then divide throughout the matrix to get % of calories from carb, protens, fats
Can you do this without an explicit for loop?

Code:

```
import numpy as np
A = np.array([56.0, 0.0, 4.4, 68.0], ....)
print(A)
cal = A.sum(axis=0)
print(cal)
percentage = 100 * A / cal.reshape(1,4)
print(percentage)
```

broadcasting 1x4
(redundant here although but takes O(1).)

Date ___/___/___

eg2: $\begin{bmatrix}1\\2\\3\\4\end{bmatrix} + 100$ → add in python, it will auto-expand to $\begin{bmatrix}100\\100\\100\\100\end{bmatrix}$ = $\begin{bmatrix}101\\102\\103\\104\end{bmatrix}$

↓ works for parameters, eg: b (constant)

⇒ this type of broadcasting works for both row vector & column vector

ex2: $\underset{m\times n}{\begin{bmatrix}1&2&3\\4&5&6\end{bmatrix}} + \begin{bmatrix}100&200&300\end{bmatrix}$ (1,n) ⇝ (m,n) python turns to $\begin{bmatrix}100&200&300\\100&200&300\end{bmatrix}$ = $\begin{bmatrix}101&202&303\\104&205&306\end{bmatrix}$

ex3: $\underset{m\times n}{\begin{bmatrix}1&2&3\\4&5&6\end{bmatrix}} + \underset{(m\times 1)}{\begin{bmatrix}100\\200\end{bmatrix}}$ in python ⇝ (m,n) $\begin{bmatrix}100&100&100\\200&200&200\end{bmatrix}$ = $\begin{bmatrix}101&102&103\\204&205&206\end{bmatrix}$

General Principle

operand	ops	operand	
1) (m,n) matrix	+ − * /	(1,n) matrix ⇝ (m,n)	= (m,n)
2) (m,m) matrix		(m,1) matrix ⇝ (m,n)	= (m,n)

3) (m,1) + R

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

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

* read documentation of numpy for fully general view of broadcasting & python, more of what i'll

Date ___/___/___

Tips & Tricks

The ability of python to allow you to use broadcast -ing operations & more generally, the greater flexibility of python numpy programming language, is both a weakness & a strength

↓ introduces a lot of bugs with this amount of flexibility

↓ can get a lot of work done with a single line of code

eg: column vector + row vector, can throw a dimension mismatch or type error something, or you may get a matrix of a sum of a row & column vector (there is an internal logic to these strange effects of python)

Couple of Tips & Tricks to eliminate & simplify all of the strange looking bugs:

```
import numpy as np                    # fanny data structure
a=np.random.randn(5)                  # five random gaussian variables
                                        stored in Array A.
print(a.shape)  #(5,)
print(a.T)
# a & a transpose look the same here
```

(rank 1 array, nor row vector, nor column vector)

```
print(np.dot(a,a.T)) #  a x a^T = get a number; when you shouldn't (4.0657..)
```

key takeaway:

don't use rank 1 array as they cause bugs.

MINDFUL NOTES

Date ___/___/___

Sol:

```
a = np.random.randn(5,1) // returns column vector
print(a.T)               // now it looks diff. from a
print(np.dot(a,a.T)      // [[...]] gives a 1x5 matrix now
```

dont use rank 1 array,
so make it a column or row vector

```
a = np.random.randn(5,1)  // a.shape = 5,1  col vector
a = np.random.randn(1,5)  // a.shape = 1,5  row vector
(So behaviour of vectors are understandable)
assert(a.shape == (5,1))  // serves as documentation
                          as well
```