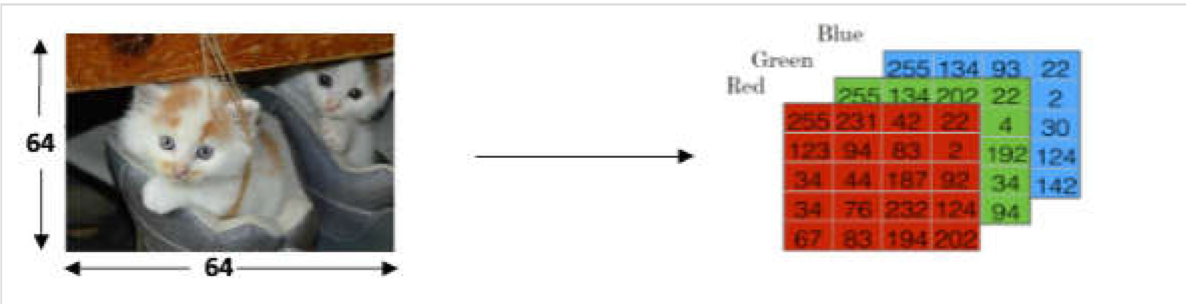
# 第二周 神经网络基础

Binary classification

forward propagation step / back propagation step

logistic regression is an algorithm for binary classification

in neural network X.shape = [n * m]  y.shape=1 * m



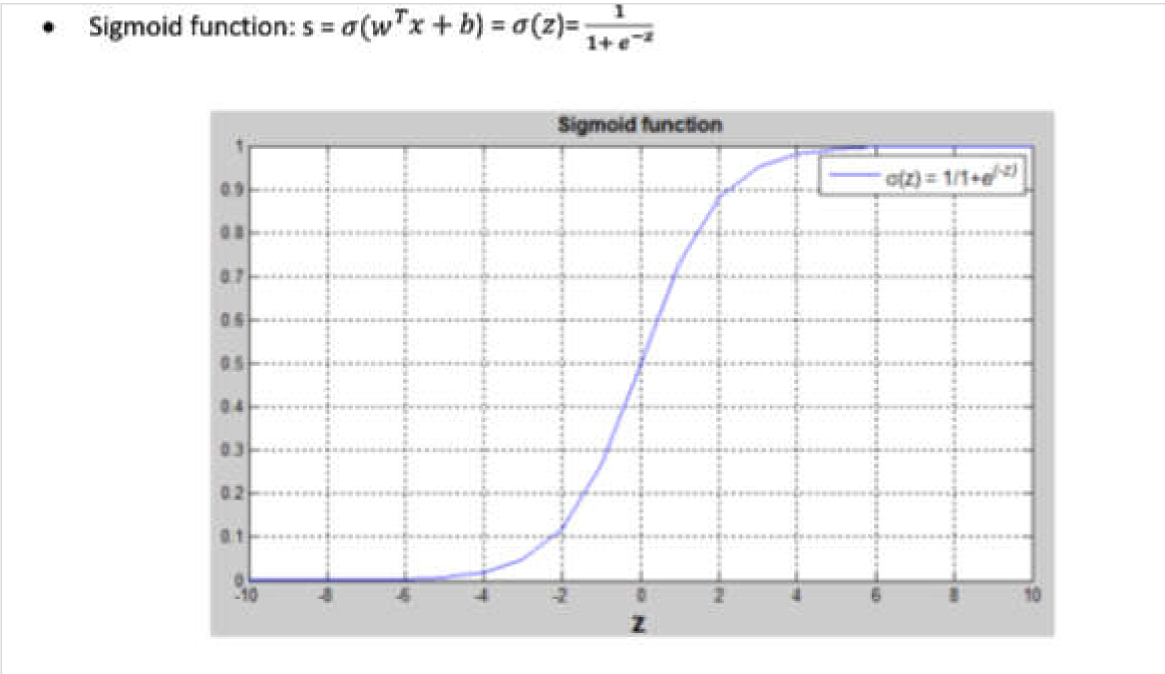

$$x = \begin{bmatrix} 255 \\ 231 \\ 42 \\ \vdots \\ 255 \\ 134 \\ 202 \\ \vdots \\ 255 \\ 134 \\ 93 \\ \vdots \end{bmatrix} \begin{array}{l} \left.\begin{matrix} \\ \\ \\ \\ \end{matrix}\right\} \text{red} \\ \left.\begin{matrix} \\ \\ \\ \\ \end{matrix}\right\} \text{green} \\ \left.\begin{matrix} \\ \\ \\ \\ \end{matrix}\right\} \text{blue} \end{array}$$

---

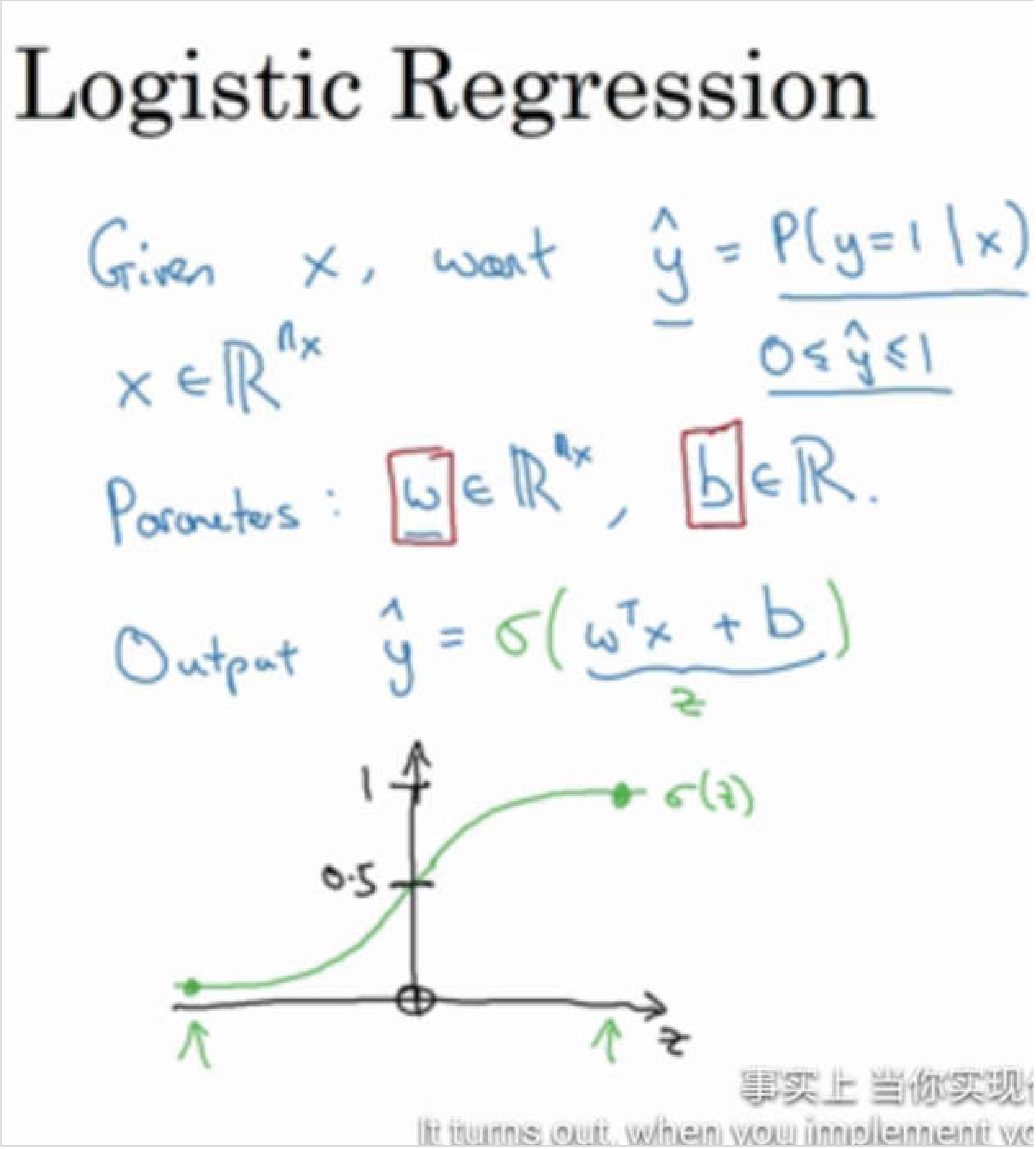
---

LOGISTIC REGRESSION

in some problems ,we want that the output y is probability(0<y<1) sigmoid function




the problem turns out to look for better w and b




---

---

logistic regression :cost function

loss function measures the discrepancy between the prediction and the desire output.

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

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

- 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

we do not use the first function, it will make the optimization problem to a non-convex function

use the second case-> it make the y^ near the y(0 or 1)

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

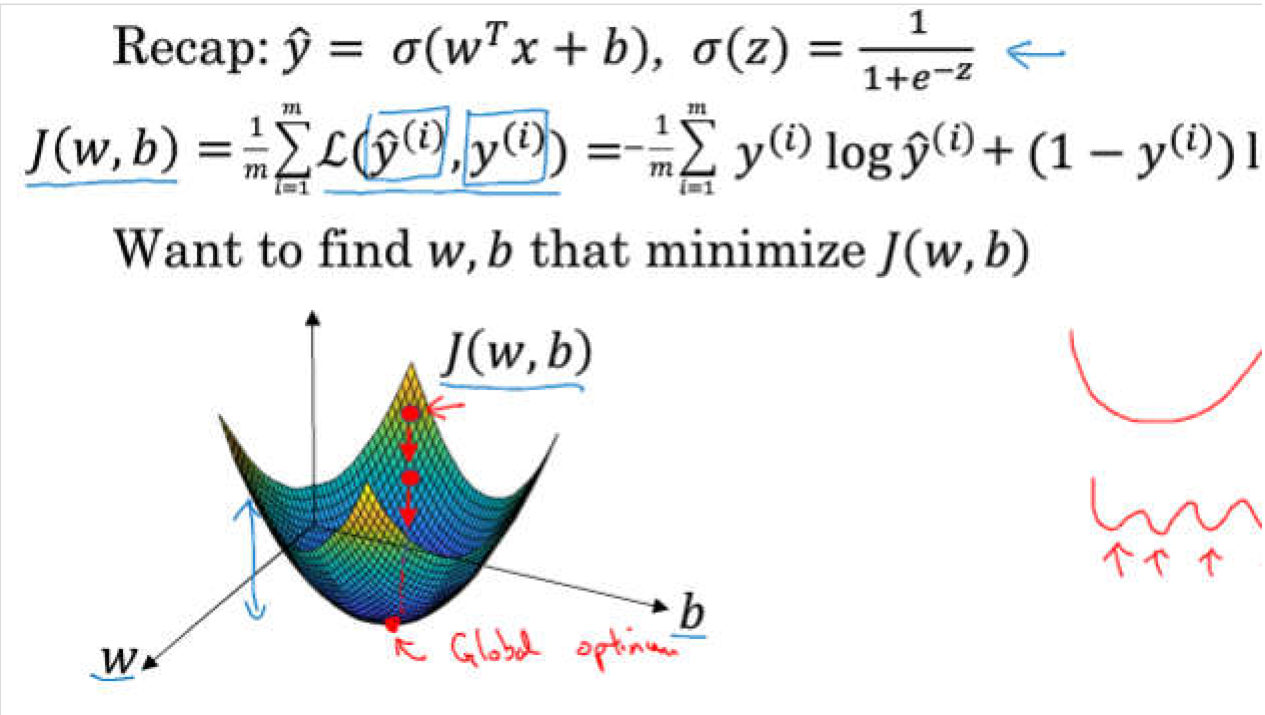
cost function(measuring all training set) lost function(measuring single train example)

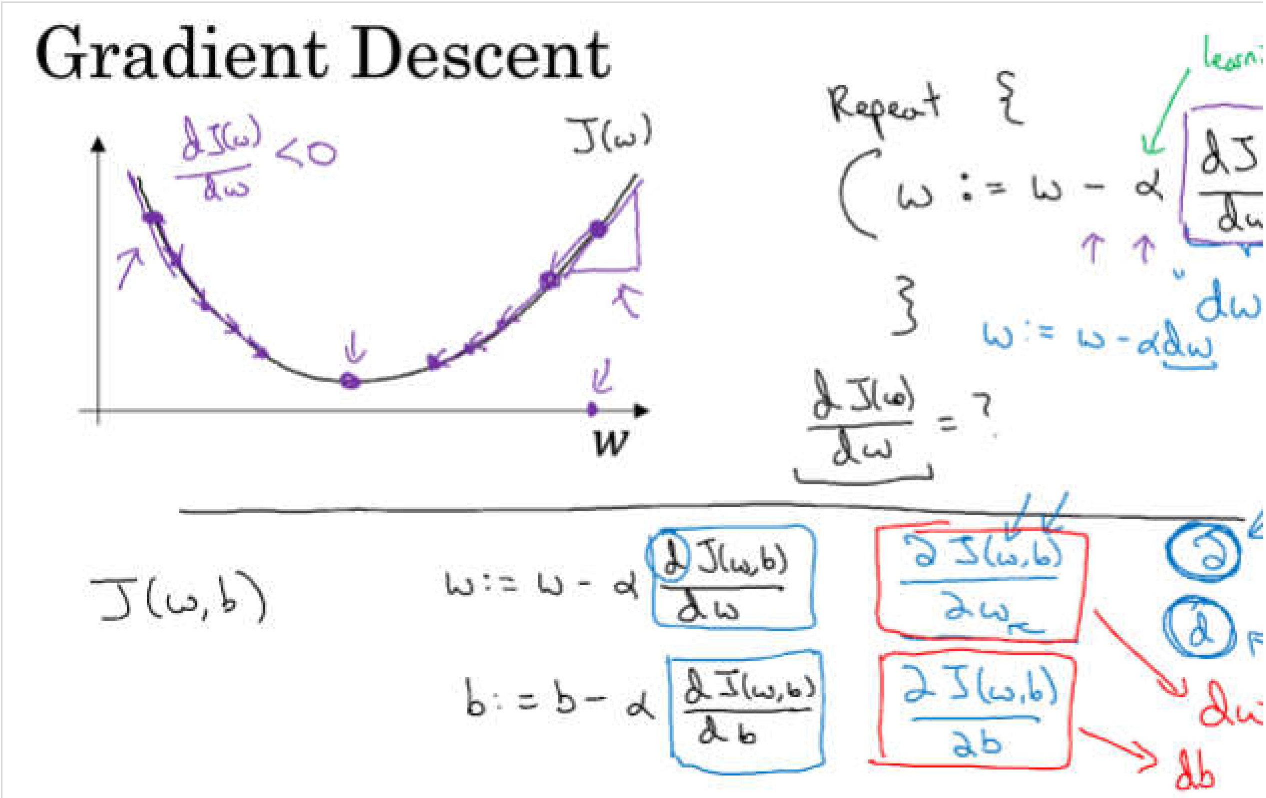
---

---

Gradient Descent

find w, b that minimize J(w,, b)





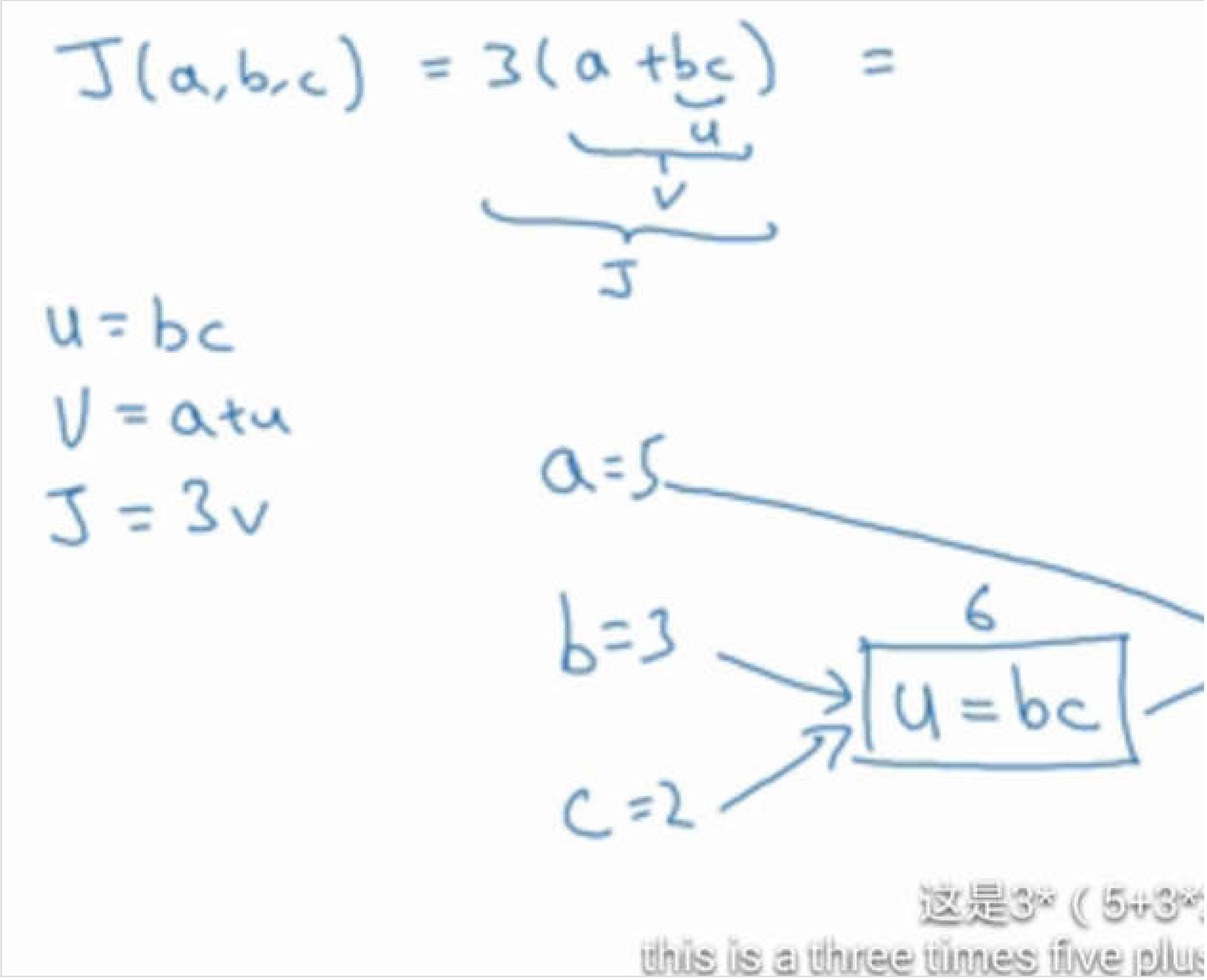

---

---

Computation Graph(计算图, forward)-》compute cost function

**the computations of neural network are all organized in terms of a forward path (compute the output)and a backward path( compute gradient)**

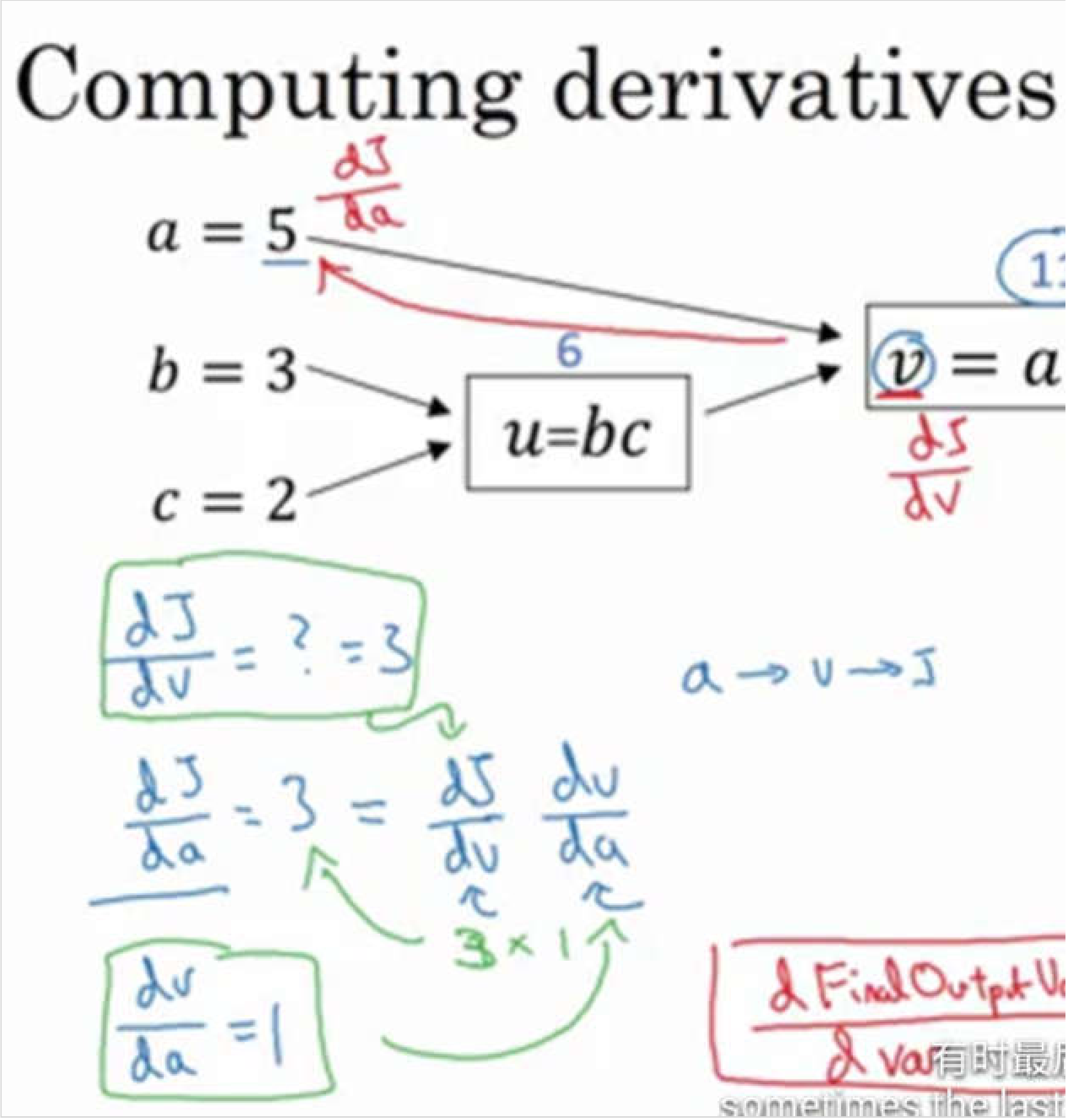
computation graph explain why should we do that

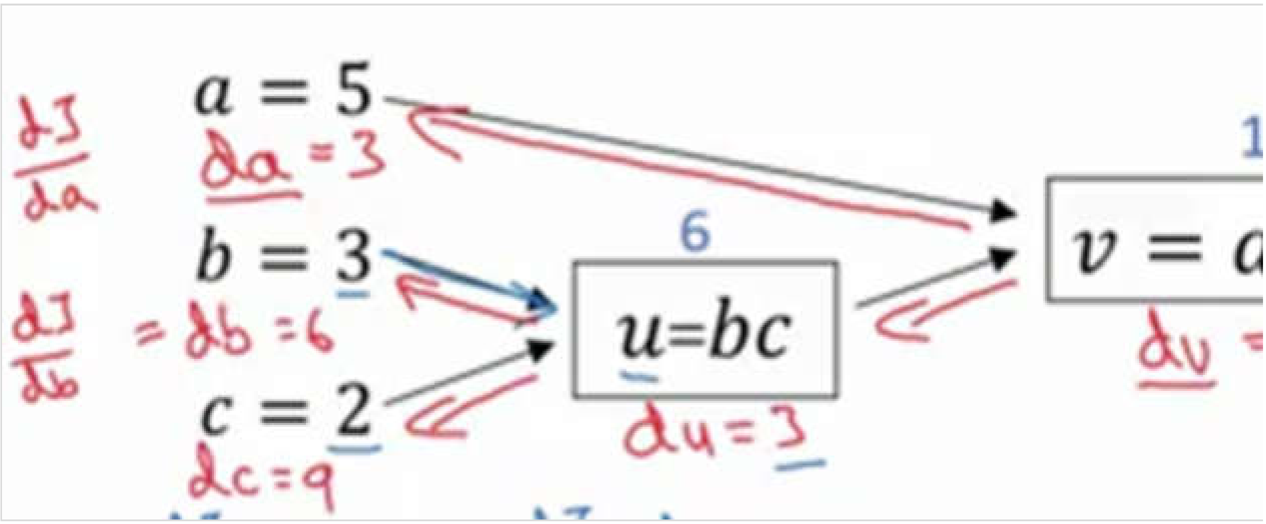



Derivatives with a Computation Graph(反向传递)

one step backwards

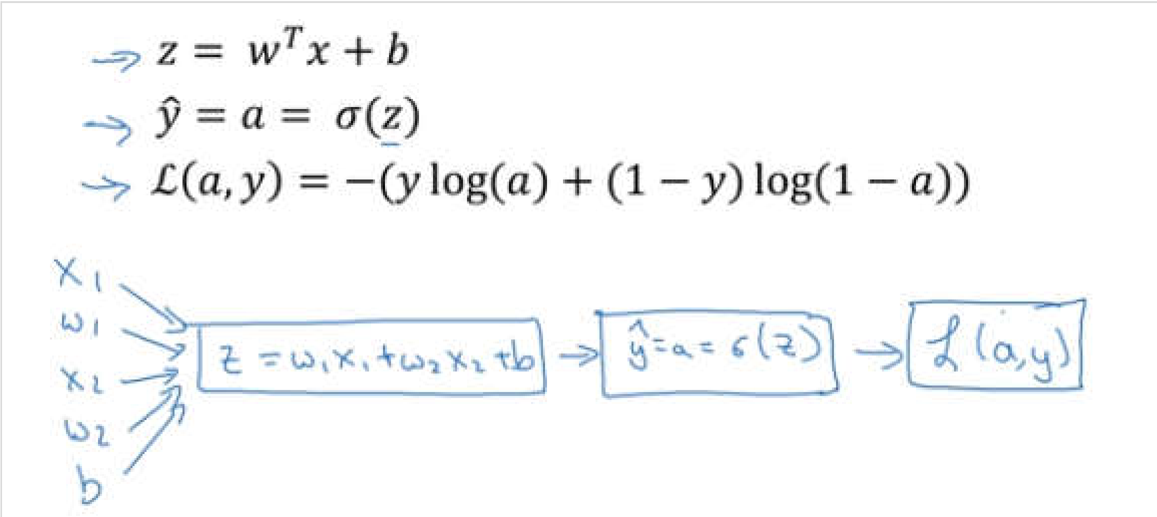
chain rule(链式法则)

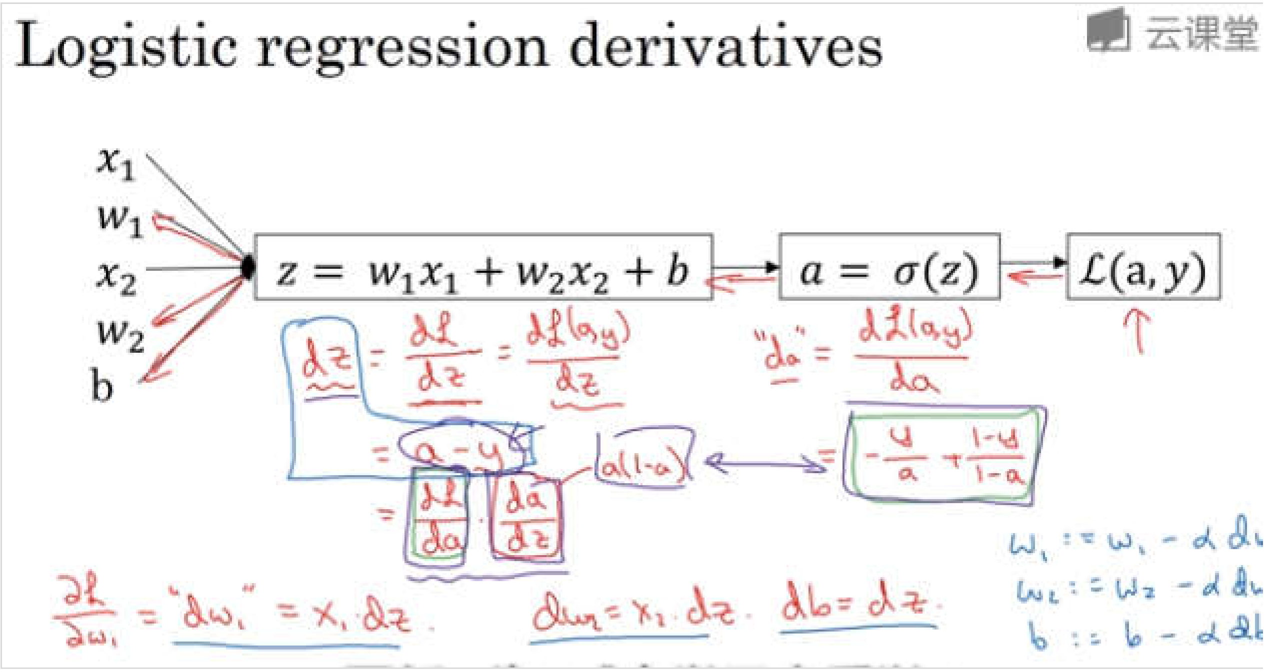

Computing derivatives
a = 5
b = 3
c = 2
u=bc
6
v = a
dJ/da
dJ/dv
dJ/dv = ? = 3
a → v → J
dJ/da = 3 = dJ/dv dv/da
3 × 1
dv/da = 1
d FinalOutputV
d var
有时最
sometimes the last


a = 5
b = 3
c = 2
u=bc
v = a
6
dJ/da
da = 3
dJ/db = db = 6
dc = 9
du = 3
dv

2.9 logistic gradient descent

logistic compute graph



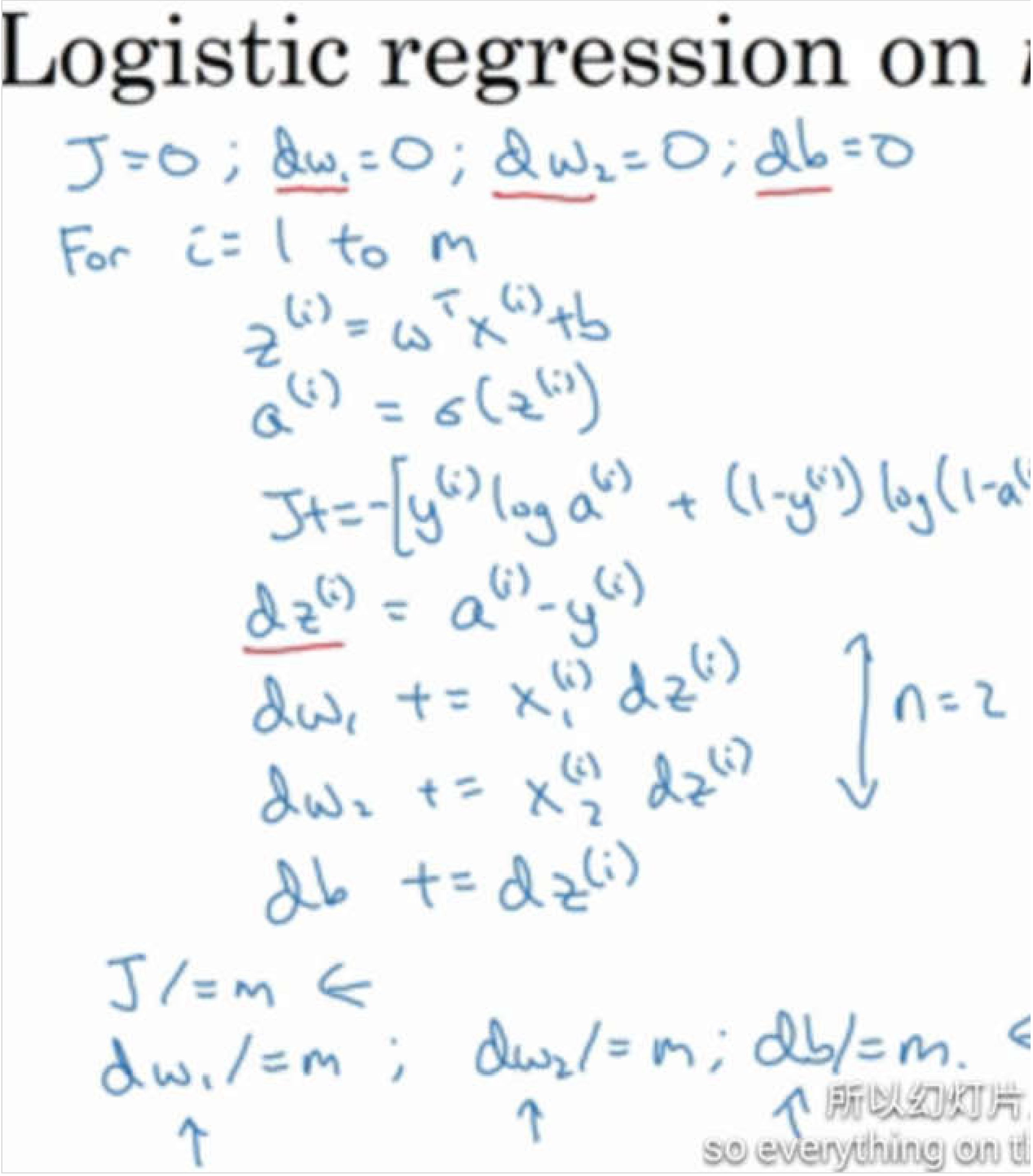

---

---

2.10 Gradient descent on m examples

one single step on gradient descent

two weakness: 1) two for loop -> vector



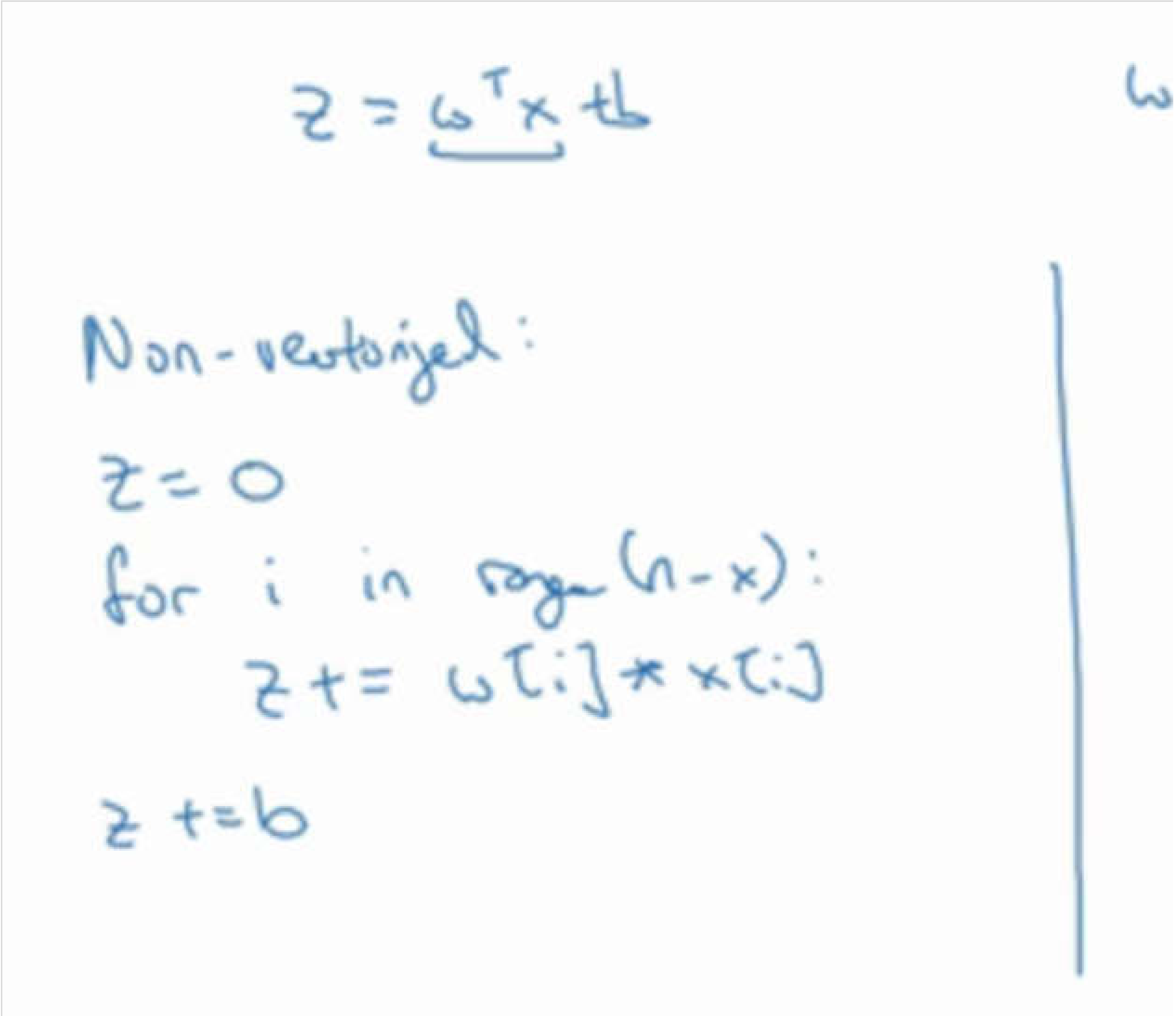

---

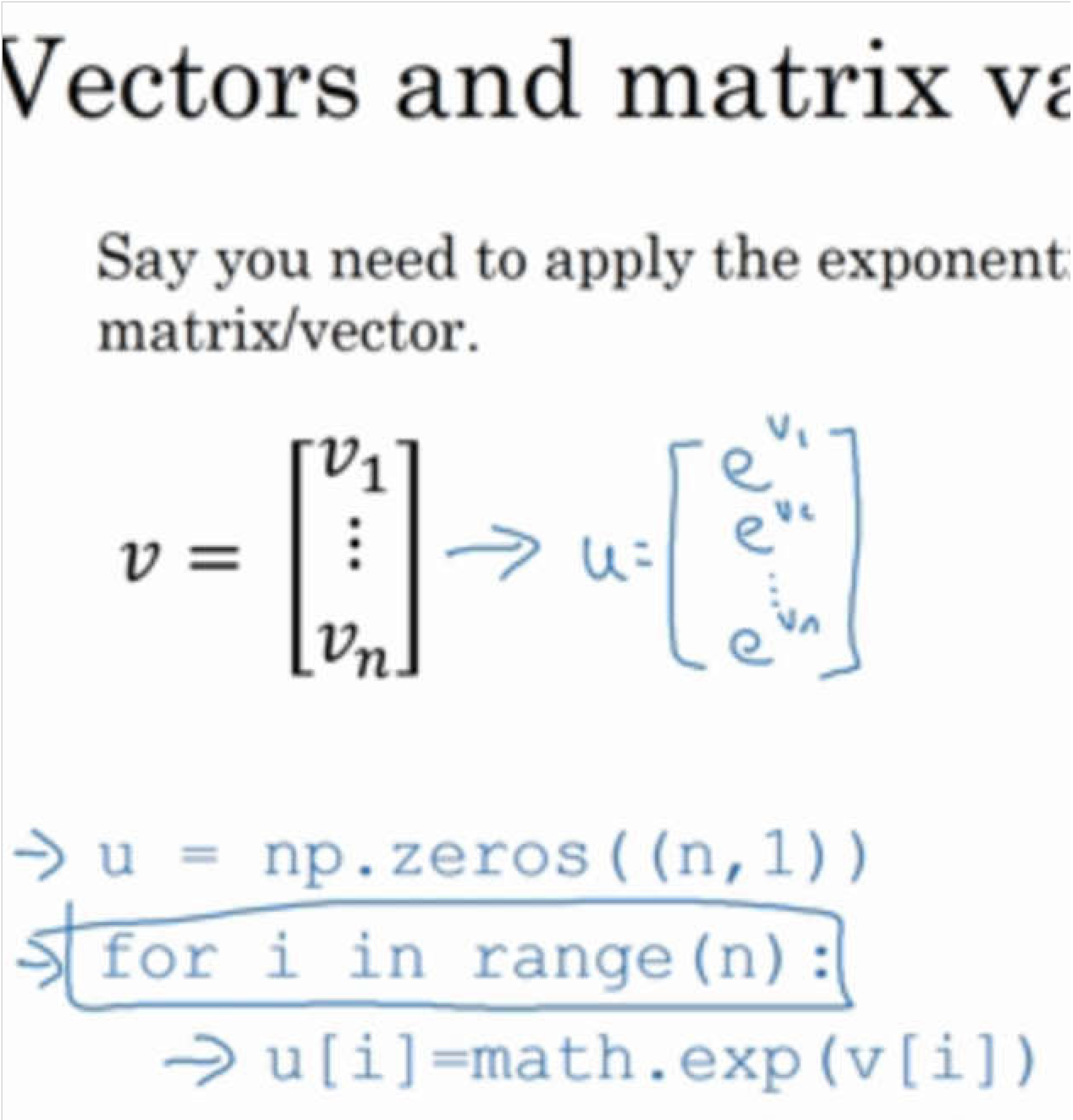
---

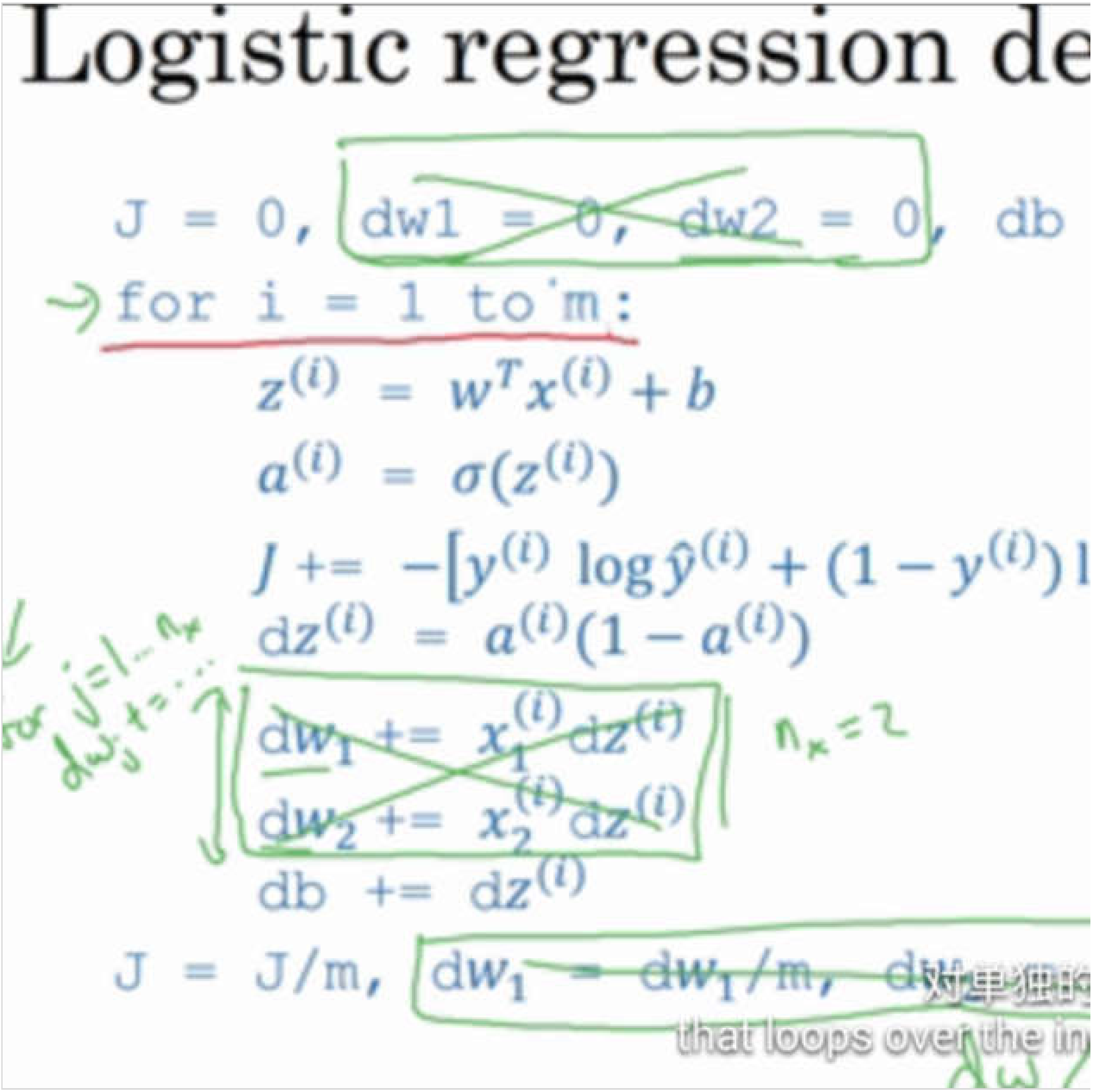
2.11 vectorization

vectorization is the art of getting rid of explicit for loops

whenever possible , avoid using explicit for loops


z = w^T x + b
Non-vectorized:
z = 0
for i in range(n-x):
z += w[i] * x[i]
z += b

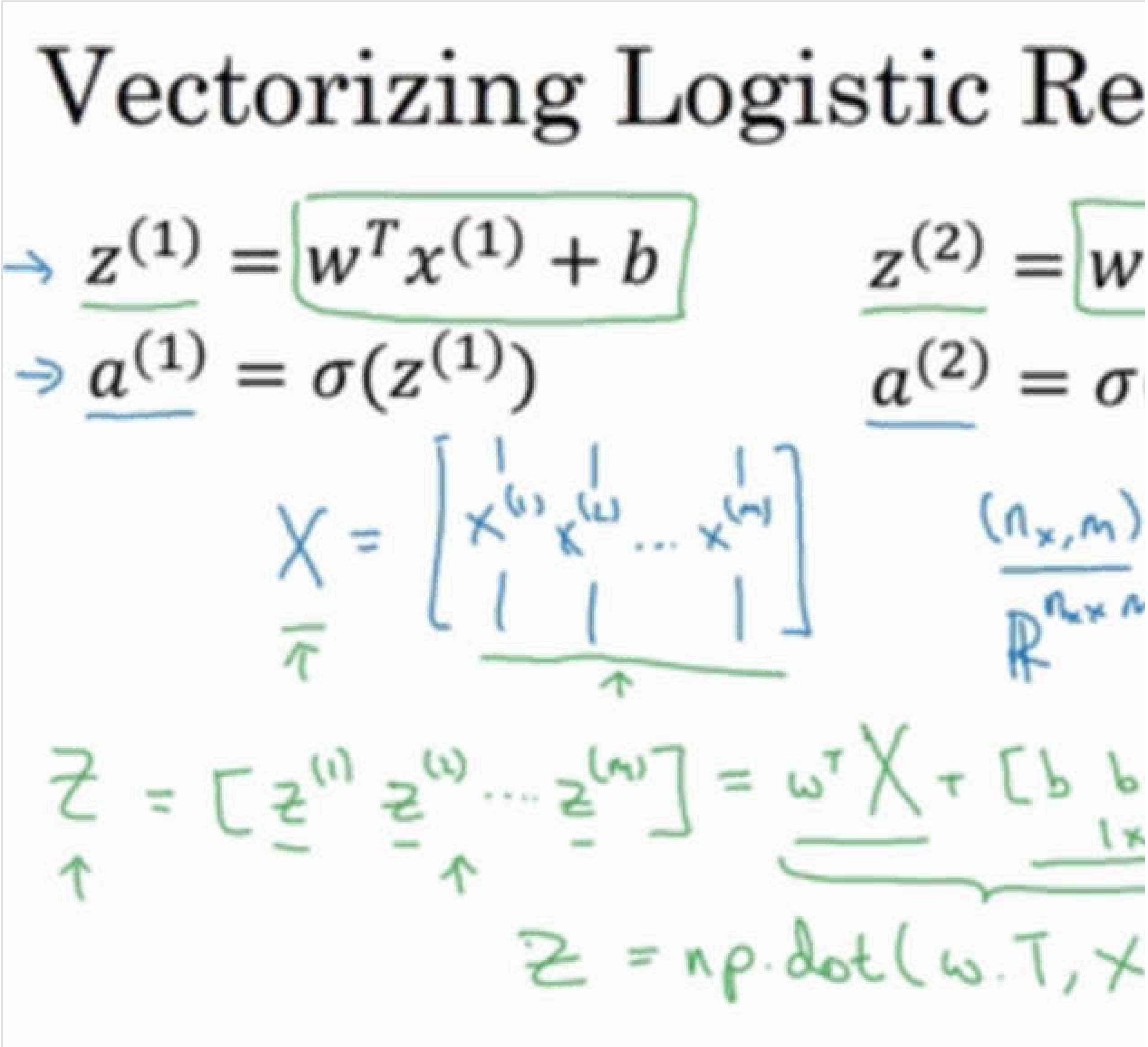

Vectors and matrix va
Say you need to apply the exponent
matrix/vector.
v = [v1 ⋮ vn]
u = [e^v1 e^v2 ⋮ e^vn]
u = np.zeros((n,1))
for i in range(n):
u[i]=math.exp(v[i])




2.11 logistic vectorizition (without using for loops)

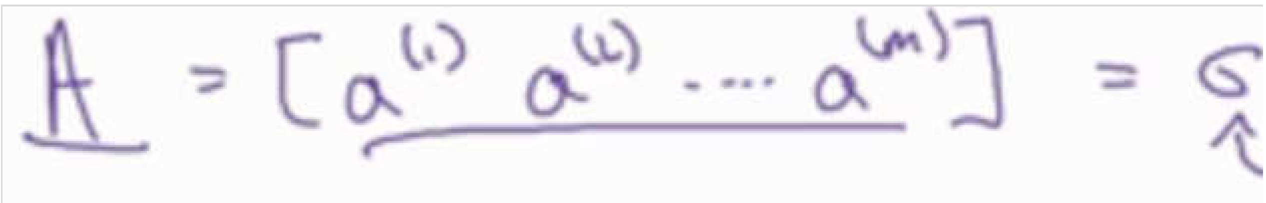
forward progressing(向量化正向传播)

no for loops




b will expand to [b,b,b,b,b..] -> "broodcasting"

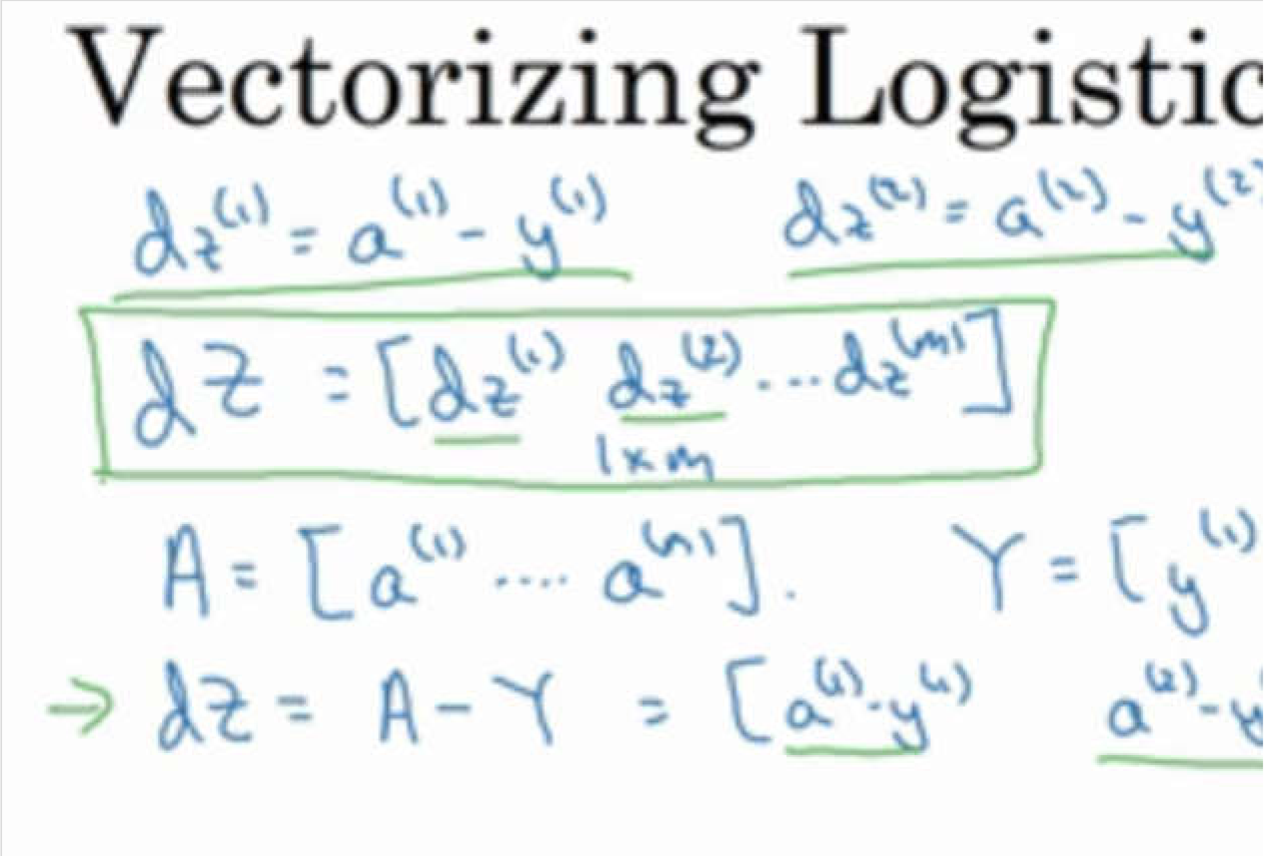
implement a vector valued sigmoid function(对向量进行 sigmoid操作)

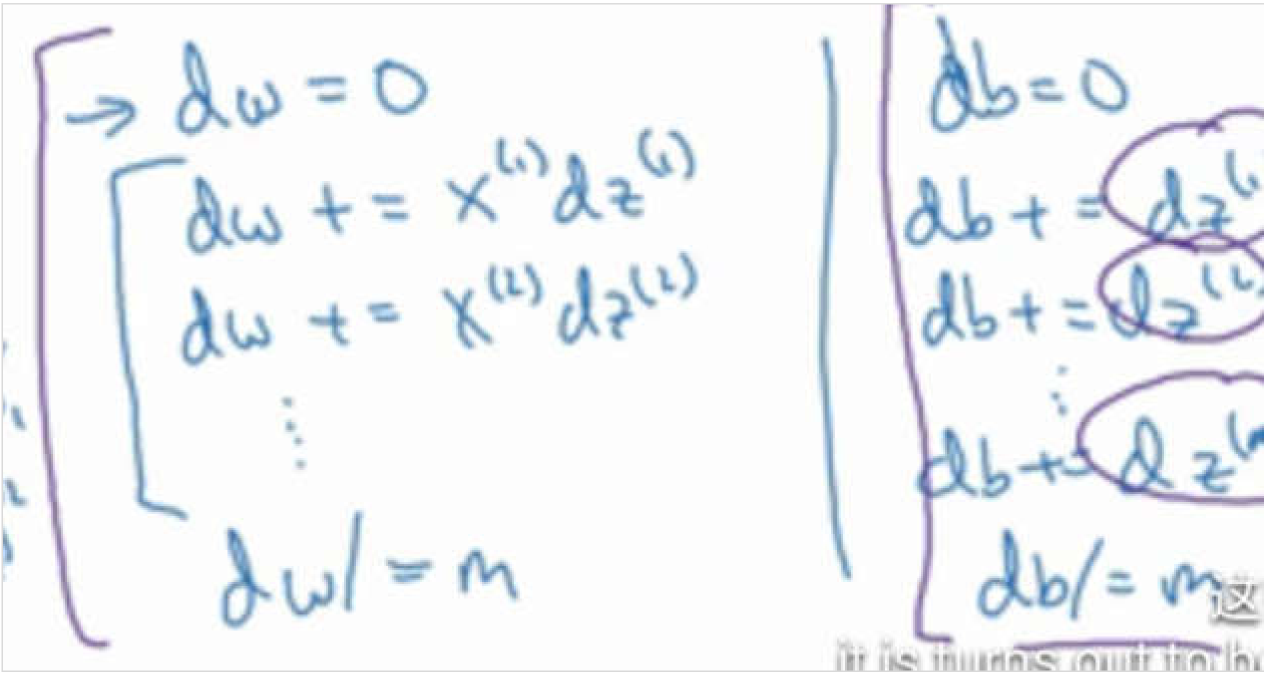


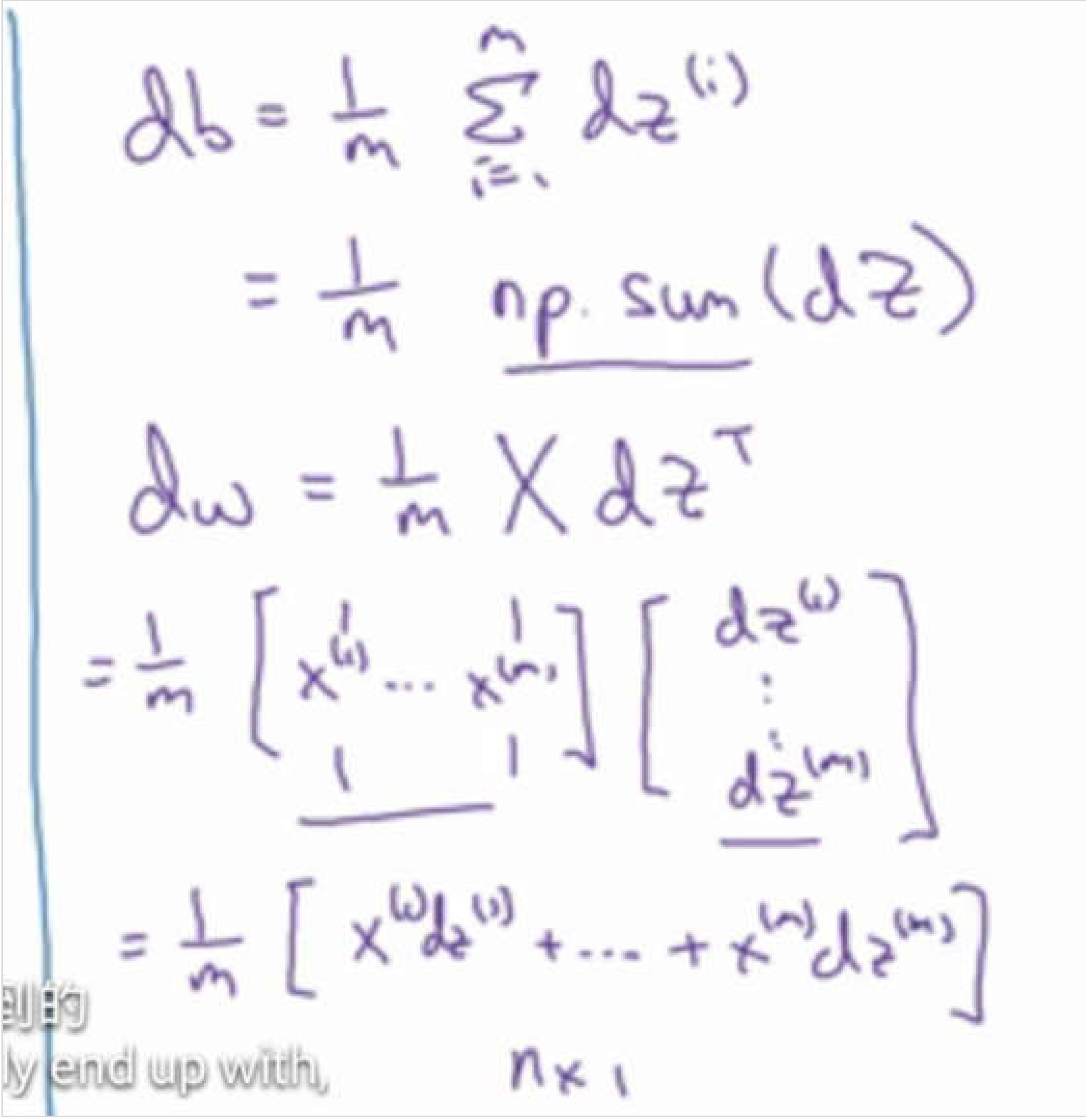

---

---

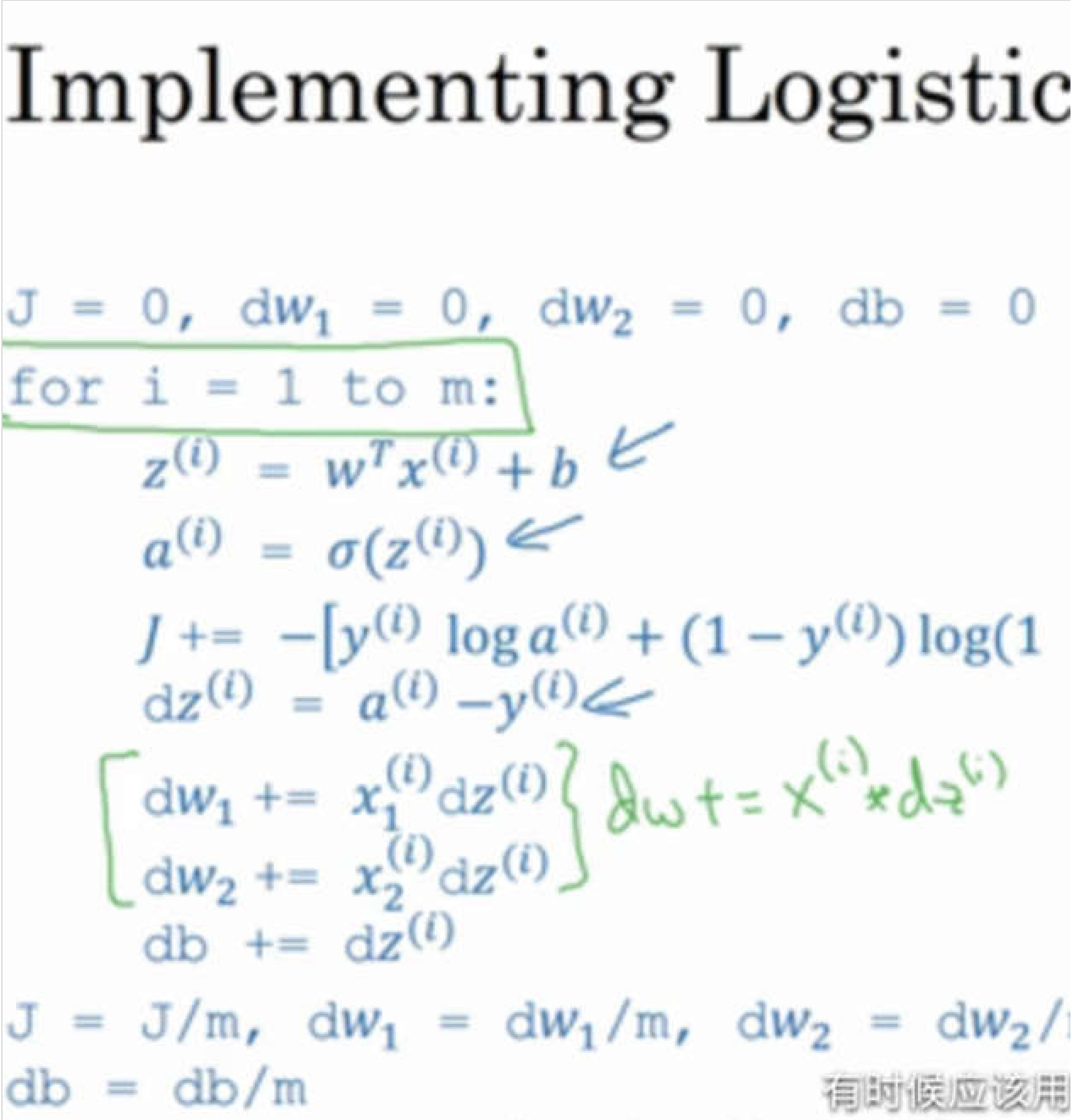
2.12 vectorizing logistic regression gradient computation(向量化反向传播)


Vectorizing Logistic
dz(1) = a(1) - y(1)
dz(2) = a(2) - y(2)
dZ = [dz(1) dz(2) ... dz(m)]
1×m
A = [a(1) .... a(m)]
Y = [y(1)
→ dZ = A - Y = [a(1) - y(1) a(2) - y


→ dw = 0
dw += x(1)dz(1)
dw += x(2)dz(2)
dw /= m
db = 0
db += dz(1)
db += dz(2)
db += dz(m)
db /= m




a highly vectorize logistic

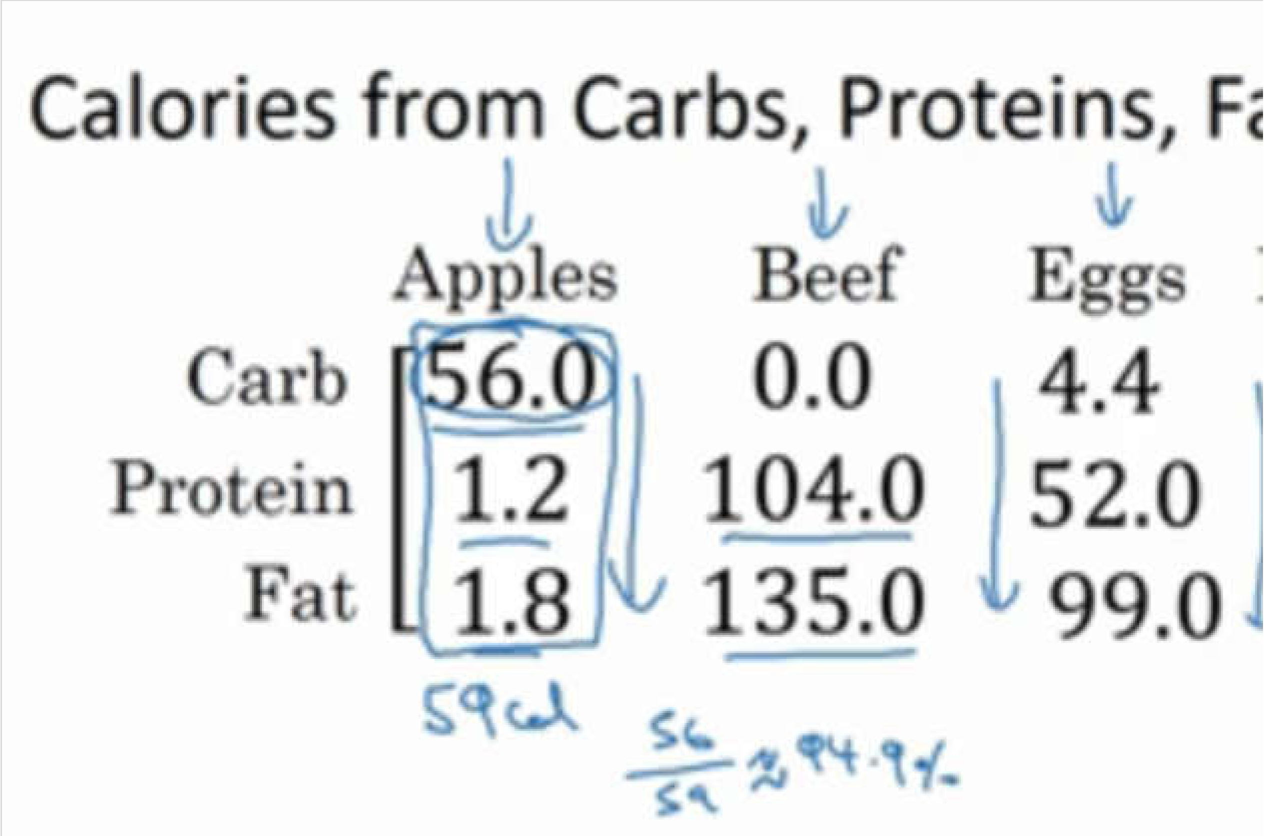



---

---

2.13 broadcasting(广播)

a tech make the python and numpy together calculate




In [6]:

```python
import numpy as np

A = np.array([[56.0, 0.0, 4.4, 68.0],
              [1.2,104.0,52.0,8.0],
              [1.8,135.0,99.0,0.9]])

print(A)
```

```
[[  56.     0.     4.4   68. ]
 [   1.2  104.    52.     8. ]
 [   1.8  135.    99.     0.9]]
```

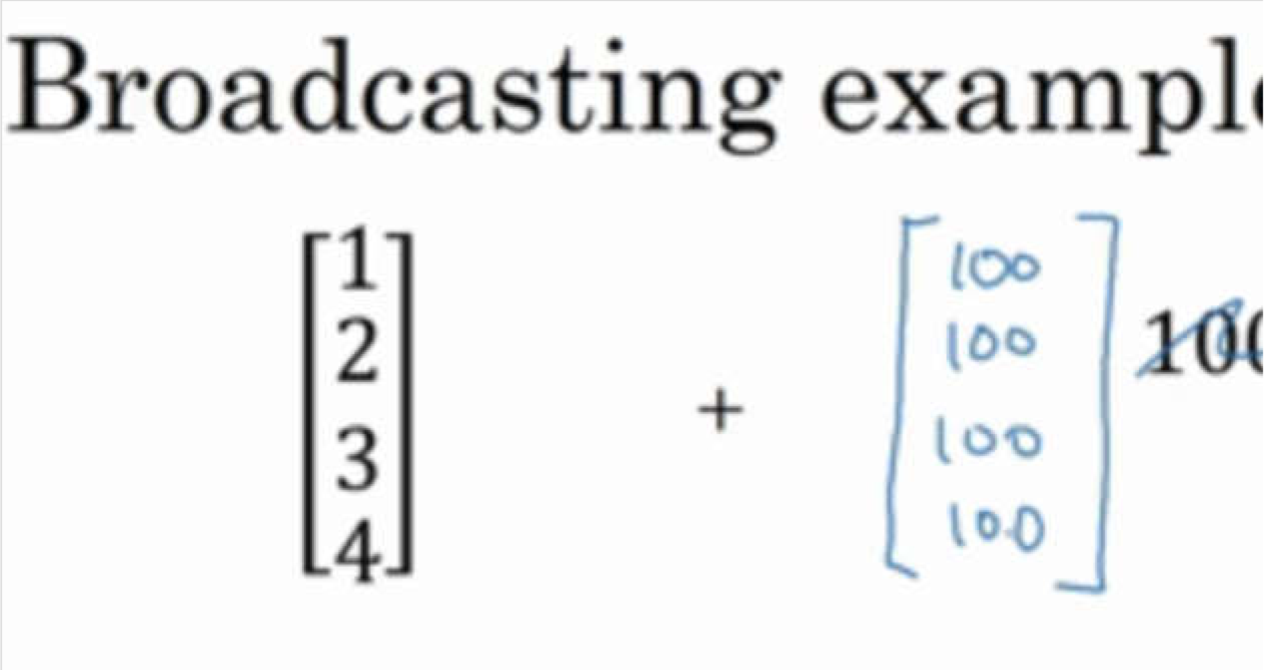
In [7]:

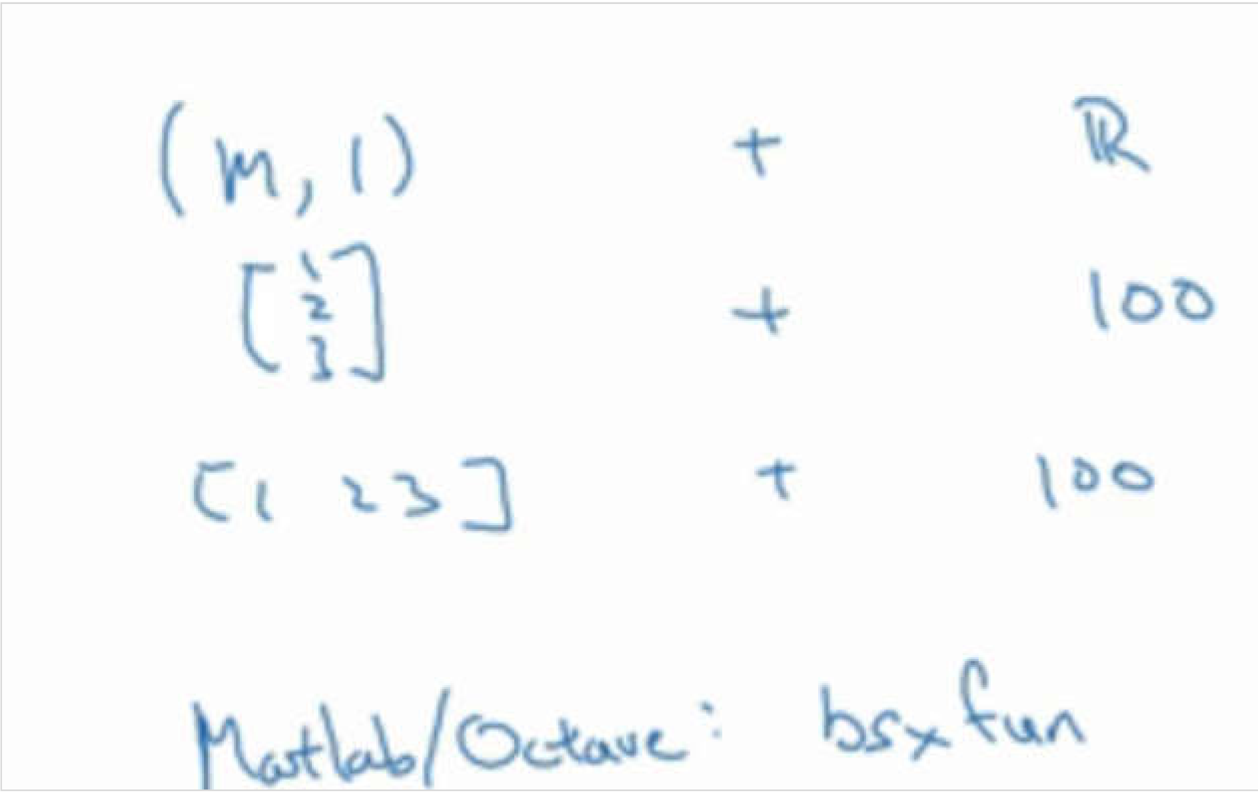
```python
cal = A.sum(axis=0)
print(cal)
```

```
[  59.   239.   155.4   76.9]
```

In [ ]:

```python
percentage = 100*A/cal.reshape(1,4)
```

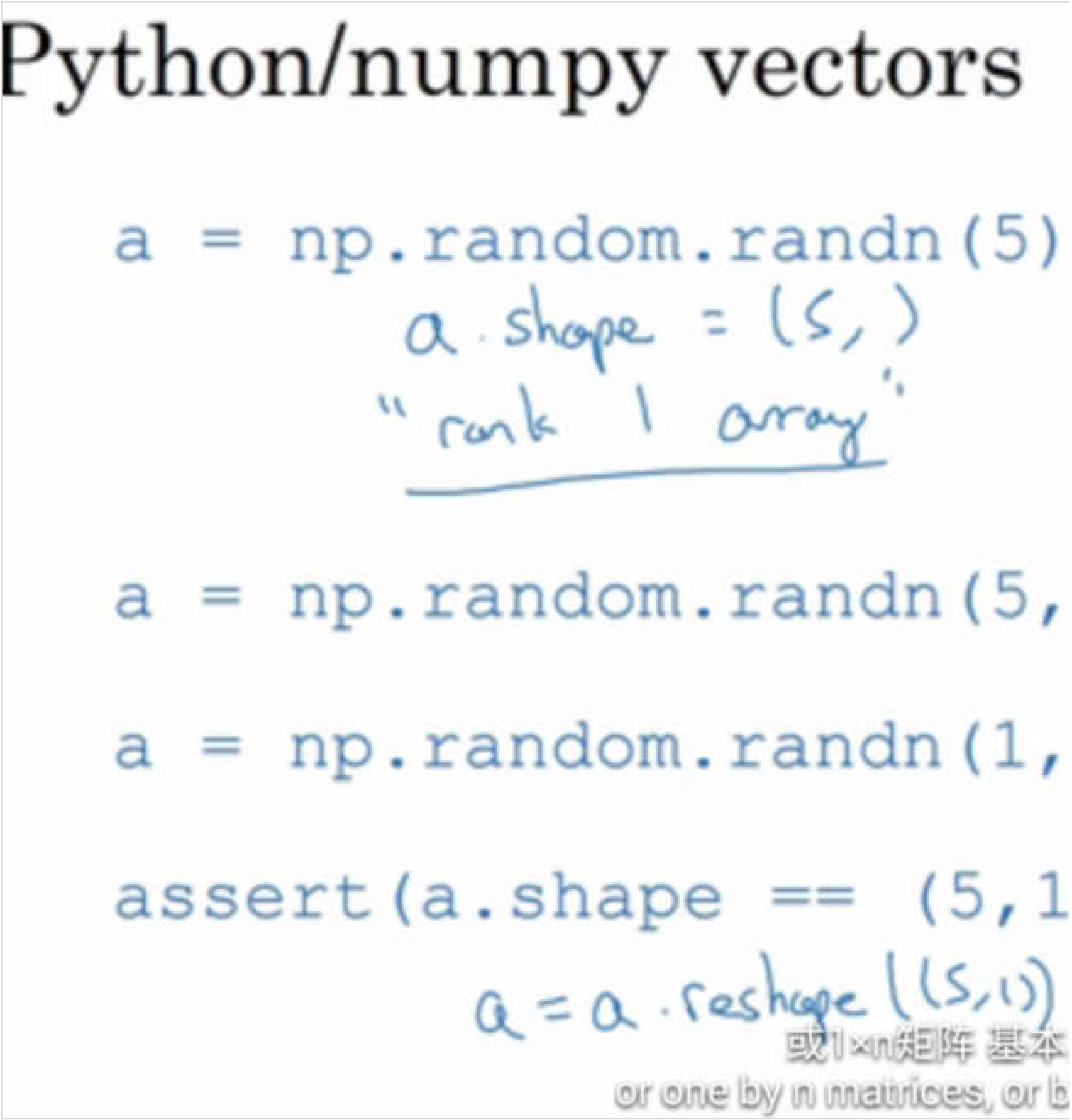



2.12 notations

never use np.random.randn(5) it will raise errors

alway use (1, 5)


Python/numpy vectors
a = np.random.randn(5)
a.shape = (5, )
"rank 1 array"
a = np.random.randn(5,
a = np.random.randn(1,
assert(a.shape == (5,1
a = a.reshape((5,1))
或1×n矩阵 基本
or one by n matrices, or b