Vectorization

Vectorization is way faster ⇒ we rely on numpy. (For loop takes 300x more time)

GPU, CPU } has parallelism ⇒ SIMD calculations (single instruction, multiple-data)

Vectorization Examples

`np.exp(v)`	element-wise exponent
`np.log(v)`	element-wise log
`np.abs(v)`	
`np.maximum(v,0)`	takes element-wise max of elem and zero
`v**2`	element wise square
`1/v`	element wise multi inverse

`np.dot`, `@` } both act as matrix multiplication

LR Derivatives Vectorization

```
dw = np.zeros((n_x, 1))
dw += dz^(i) x^(i)      # scalar, vector (n_x, 1)
```

```
J = 0, dw1 = 0, dw2 = 0, db = 0        dw = np.zeros((n_x, 1))
for i = 1 to m:
    z^(i) = w^T x^(i) + b
    a^(i) = σ(z^(i))
    J += -[y^(i) log a^(i) + (1 - y^(i)) log(1 - a^(i))]
    dz^(i) = a^(i) - y^(i)
    dw1 += x_1^(i) dz^(i)
    dw2 += x_2^(i) dz^(i)
    db += dz^(i)
dw1 /= m, dw2 /= m, db /= m
```

$n_x = 2$

dw

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

$$dw_1 = \frac{1}{m} \sum_{i=1}^{m} x_1^{(i)} dz^{(i)}$$

Vectorizing Logistic Regression

w^T is a row vector

$$z^{(1)} = w^T x^{(1)} + b \qquad z^{(2)} = w^T x^{(2)} + b \qquad z^{(3)} = w^T x^{(3)} + b$$

$$a^{(1)} = \sigma(z^{(1)}) \qquad a^{(2)} = \sigma(z^{(2)}) \qquad a^{(3)} = \sigma(z^{(3)})$$

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

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

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

(capital) (1, m)

`Z = np.dot(w.T, x) + b` → "broadcasting"

$(1,1)$ $\mathbb{R}$ b will be broadcasted to all elem n vector

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

(capital)

vector valued sigmoid function

Vectorising LR's Gradient Computation

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

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

$A = \begin{bmatrix} a^{(1)} & a^{(2)} & \dots & a^{(m)} \end{bmatrix}$ $\quad Y = \begin{bmatrix} y^{(1)} & y^{(2)} & \dots & y^{(m)} \end{bmatrix}$

$dZ = A - Y = \begin{bmatrix} a^{(1)} - y^{(1)} & a^{(2)} - y^{(2)} & \dots & a^{(m)} - y^{(m)} \end{bmatrix}$

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

$dw = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} dz^{(i)} = \frac{1}{m} X\, dZ^T$

single value

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \end{bmatrix} \quad \begin{bmatrix} dz^{(1)} \\ dz^{(2)} \\ \vdots \\ dz^{(m)} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{m} x_1^{(i)} dz^{(i)} \\ \vdots \\ \sum_{i=1}^{m} x_{n_x}^{(i)} dz^{(i)} \end{bmatrix} / m = dw$$

As remember,

$dw_1 = \frac{1}{m} \sum_{i=1}^{m} x_1^{(i)} dz^{(i)}$ → and → this is exactly what we are doing above

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

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

Implementation

$Z = w^T X + b$
$= $ 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)

$\Rightarrow$

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

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

[this is for one iteration of GR, so if we want 1000 iterations, we can have an outer loop]

Broadcasting

axis 0 → vertical

axis 1 → horizontal

CASE 1) Constant

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

CASE 2) matrix

$$\underset{m,n}{\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}} + \underset{1,n}{\begin{bmatrix}100 & 200 & 300\end{bmatrix}} \Rightarrow$$

$$\Rightarrow \begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix} + \underset{m,n}{\begin{bmatrix}100 & 200 & 300\\100 & 200 & 300\end{bmatrix}}$$

General Rules

(m, n) matrix	+ − * /	(1, n)	copy m times to rows → (m, n)
		(m, 1)	→ (m, n) copy n times to colums
(m, 1)	+ − * /	R	→ copy value n times to (m, 1)

A Note on Python / Numpy Vectors

When we do np.array we get a rank 1 vector

a.shape = (5,) for example

this is not row vector, neither column vector.

⇒ Always give full dimension for the array you are creating.

DON'T USE { a = np.random.randn(5) → gives back rank 1 array (avoid)

a = np.random.randn(5,1) column vector

a = np.random.randn(1,5) row vector } USE THIS