Vectorization :

→ Basically used to get rid of explicit folders in your code
→ crucial in deep learning since we're using large datasets.

Lets see an example

task $z = w^T x + b$

we can do this computation directly using vectors, or by looping through values :

Non vectorized approach

```
z = 0
for i in range(n, x)
    z += w[i] * x[i]      } w^T x
z += b                    + b
```

Vectorized approach

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

written in C
↳ numpy is faster coz it does SIMD parallel processing

4 towards data science / How-fast-is-numpy-really-is

5 to 100 times faster ap

code run →

Slower / convoluted
474 ms

faster / easier
→ 1.5 ms

Conclusion : whenever possible, avoid running for loops

functions discussed

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

Vectorized Logistic Regression :

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

(II) $A = \sigma(z)$

$Z = [z^{(1)}\ z^{(2)}\ z^{(3)} \dots z^{(m)}] = w^T X + [b\ b\ b\ b \dots]$ ← Broadcasting in python it's a real no.

$= [w^T x^{(1)} + b \quad w^T x^{(2)} + b \dots w^T x^{(m)} + b]$

Vectorized Grad. Descent Computation :

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

repeats

(IV) $db = \frac{1}{m}(np.sum(dZ))$; $dw = \frac{1}{m} X\, dZ^T$

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

(V) $w := w - \alpha\, dw$
$b := b - \alpha\, db$

needs to be looped (I) (V)