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
```

Slower / convoluted
code run → 474 ms

vectorized approach

$Z = \underbrace{np.dot(w, x)}_{w^T x} + b$ +b

faster / easier
→ 1.5 ms

written in C
numpy is faster coz it does SIMD parallel processing
towardsdatascience / How-fast-numpy-really-is
5 to 100 times faster approx

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 $A = \sigma(Z)$

Broadcasting in python it a real no.

$Z = [z^{(1)}\ z^{(2)}\ z^{(3)} \dots z^{(m)}] = w^T X + \underbrace{[b\ b\ b\ b \dots]}_{1 \times m} = \boxed{w^T x^{(1)} + b}\ \boxed{w^T x^{(2)} + b} \dots \boxed{w^T x^{(m)} + b}$

Vectorized Grad. Descent Computation:

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

repeats

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

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

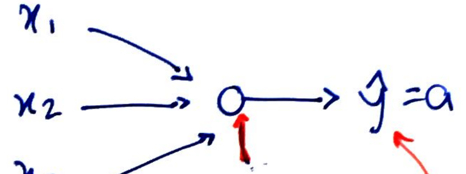
V $w := w - \alpha\, dw$
$b := b - \alpha\, db$

needs to be looped
I → V

Neural Networks:

- Neural Networks are a set of algorithms, modeled loosely after a human brain, that are designed to recognize patterns.
- helps us cluster & classify.

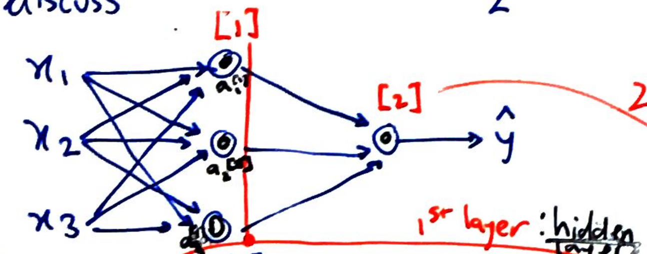
previosuly, we saw:



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

now, we'll discuss

2 layer NN



2^{nd} layer : output layer

1^{st} layer : hidden layer

0^{th} layer : Input layer

$a^{[0]} = X$

$W^{[1]}$

$b^{[1]}$

$z^{[1]} = W^{[1]} x + b^{[1]}$

$\downarrow$

$a^{[1]} = \sigma(z^{[1]})$

$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$

$\downarrow$

$a^{[2]} = \sigma(z^{[2]})$

$\downarrow$

$L(a^{[2]}, y)$

we do back propagation to calculate

$da^{[2]} \longrightarrow dz^{[2]} \longrightarrow dw^{[2]}, db^{[2]}$ & so on

R→L Back calculation

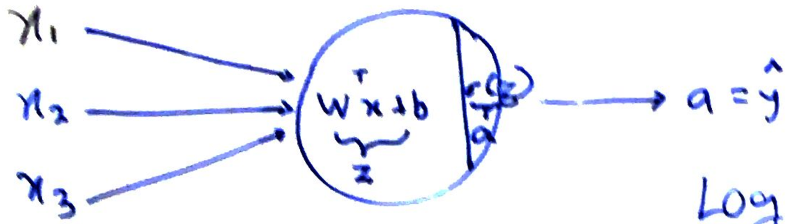
- NN → basically take a logistic regression & repeat it twice.

Notation

$a_1^{[1]} \rightarrow 1^{st}$ layer, 1^{st} weight

$a_2^{[1]} \rightarrow 1^{st}$ layer, 2^{nd} weight

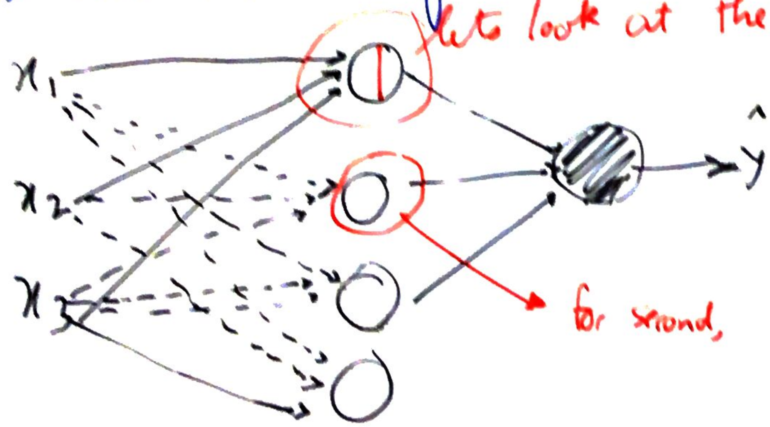
Computing a NN's output:



Log Reg.

$$\left.\begin{aligned} z &= w^T x + b \\ a &= \sigma(z) \end{aligned}\right\} \text{2 steps}$$

\# NN just does this a no. of times



lets look at the 1st Node:

$$\boxed{\begin{aligned} z_1^{[1]} &= w_1^{[1]T} x + b_1^{[1]} \\ a_1^{[1]} &= \sigma(z_1^{[1]}) \end{aligned}}$$

$a_i^{[l]}$ ← layer, i ← node in layer

for second,

$$\boxed{\begin{aligned} z_2^{[1]} &= w_2^{[1]T} x + b_2^{[1]} \\ a_2^{[1]} &= \sigma(z_2^{[1]}) \end{aligned}}$$

$$\underbrace{\begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix}}_{W^{[1]}} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}}_{b^{[1]}} \Rightarrow \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix} = z^{[1]}$$

transposed ∴ row

\# think as follows:
- → 4 logistic regression units
- → each unit has corresponding parameter w
- → by stacking together we get this

σ fn apply element wise

$$= \begin{bmatrix} a_1^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix} = a$$

$$\underset{(4\times1)}{z^{[1]}} = \underset{(4,3)}{W^{[1]}} \underset{(3,1)}{x} + \underset{(4,1)}{b^{[1]}}$$

$$\overset{(4,1)}{a^{[1]}} = \sigma(\overset{(4,1)}{z^{[1]}})$$

$$\underset{(1,1)}{z^{[2]}} = \underset{(1,4)}{W^{[2]}} \underset{(4,1)}{a^{[1]}} + \underset{(1,1)}{b^{[2]}}$$

$$\underset{(1,1)}{a^{[2]}} = \sigma(\underset{(1,1)}{z^{[2]}})$$