Deep Neural Network:

→ We've seen fwd, bckwd propogation for a single layer, log reg & vectorization. We saw why its important to initiate vectors randomly. We'll use these together for a deeper NN.

Deep NN:

Log reg: Kinda shallow

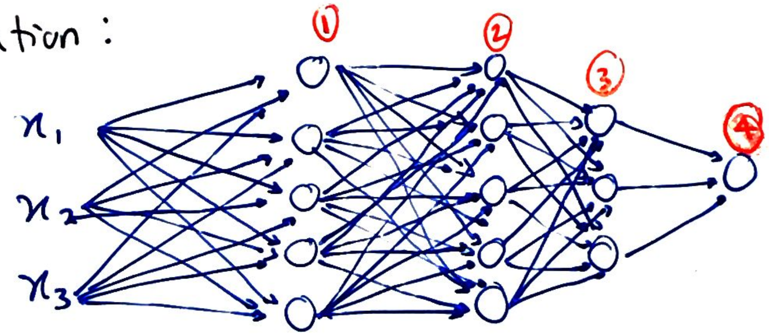
1 hidden layer: better but basic

Deeper NNs? functions that can be learned here, that shallower Networks are unable to recognize.

New Problems to think?

→ depth needed for a NN to work optimally : New hyper parameter

Notation:



→ 4 layed NN $L=4$

→ 3 hidden layers

→ 5, 5, 3

$n^{[l]}$ = # units in layer l

$n^{[1]} = 5$ $n^{[3]} = 3$

$n^{[2]} = 5$ $n^{[4]} = 1$

$n^{[0]} = n_x = 3$

$a^{[l]}$ = activations in layer l

$a^{[l]} = g^{[l]}(z^{[l]})$, $W^{[l]}$ = weights for $z^{[l]}$, $b^{[l]}$

Forward propagation:

for layer 1,

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

$a^{[1]} = g^{[1]}(z^{[1]})$ → $z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$

$a^{[2]} = g^{[2]}(z^{[2]})$ → ... →

$z^{[4]} = W^{[4]}a^{[3]} + b^{[4]}$

$a^{[4]} = g^{[4]}(z^{[4]}) = \boxed{\hat{y}}$

General form?

Going by the pattern $l: 1 \to 4$, we see,

$$\boxed{\begin{aligned} z^{[l]} &= W^{[l]}a^{[l-1]} + b^{[l]} \\ a^{[l]} &= g^{[l]}(z^{[l]}) \end{aligned}}$$

Vectorized form:

→ We'll basically need to stack together the computed z, a values.

$$Z = \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z_2^{[2](2)} & \cdots & z^{[n](n)} \\ | & | & & | \end{bmatrix} \qquad \hat{y} = g(Z^{[4]}) = A^{[4]}$$

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$$

$$A^{[l]} = g^{[l]}(Z^{[l]})$$

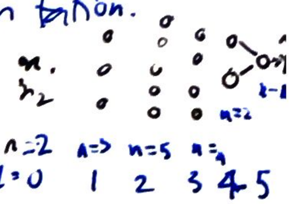
→ need for loop to compute activations for layer 1, 2, 3 ... n

for $l = 1 \dots 4$:

z, a ↕ Calculation needed.

Note on dimentions:

→ helpful to write the dims on paper before implementation.



Eg ①

$$Z^{[1]} = W^{[1]} \cdot x + b^{[1]}$$

$(3,1)$ ∴ $(3,2)$ $(2,1)$ $(3,1)$

$[\,\vdots\,] = [\,\vdots\;\vdots\,]\;[\,\vdots\,]$

$[n^{[1]}, 1]$ $[n^{[1]}, n^{[0]}]$ $(n^{[0]}, 1)$

②

$W^{[1]} : (n^{[1]}, n^{[0]})$

$W^{[2]} : (5, 3)$ $(n^{[2]}, n^{[1]})$

③

$$\rightarrow Z^{[2]} = W^{[2]} \cdot a^{[1]} + b^{[2]}$$

$(5,1)$ $(5,3)$ $(3,1)$ $(5,1)$

④ Similarly,

$W^{[3]} : (4, 5)$

$W^{[4]} : (2, 4)$

$W^{[5]} : (1, 2)$

Generic form:

$W^{[l]} : (n^{[l]}, n^{[l-1]})$ - dW has same dims

$b^{[l]} : (n^{[l]}, 1)$ db has same dims

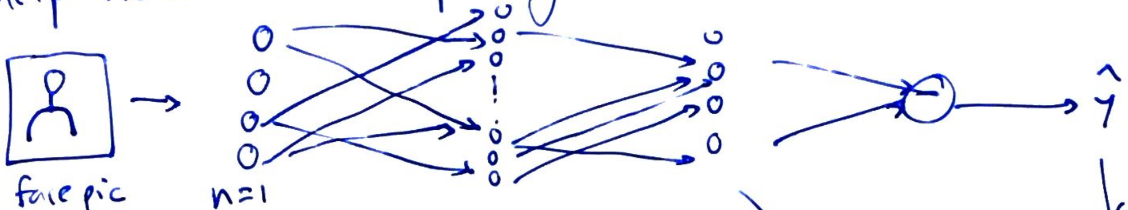
$z^{[l]}, a^{[l]} : (n^{[l]}, 1)$

$Z^{[l]}, A^{[l]} : (n^{[l]}, m)$ $dZ^{[l]}, dA^{[l]}$ have same dims

Why deep networks work:

deep network is computing

Eg



maybe feature or face detector

group edges to make features (part of faces

recognize or detect faces

Hey dat's our boi drake

→ audio recognition

low level audio wave forms → basic units of sound → phonems → words → Sentence Phrases

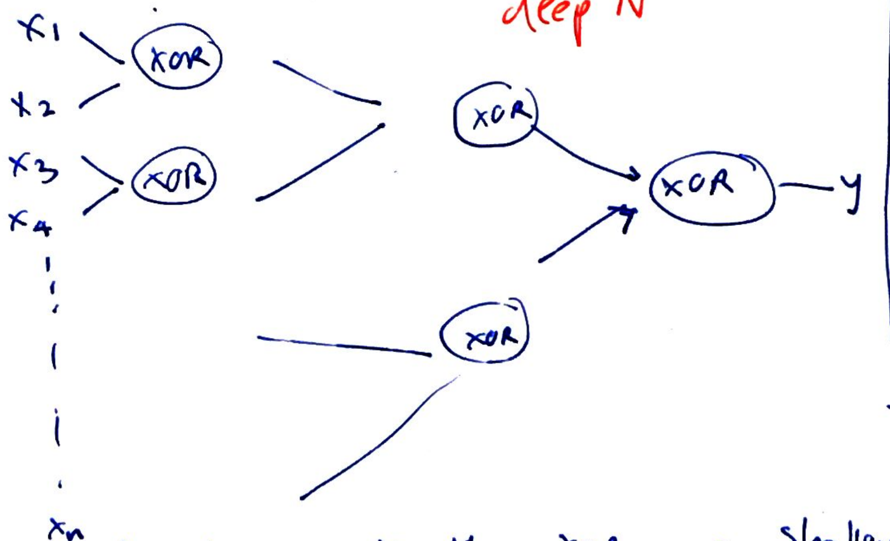
phonem C A T

Circuit thory:

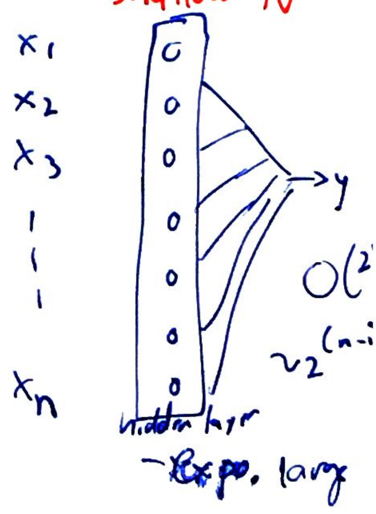
There are fns you can compute with a 'small' L-layer network that shallower networks require exponentially more hidden units to compute

x_1 XOR x_2 XOR x_3 XOR x_4 ..... x_n

deep N



shallow N



$O(2^{n}$

$\sim 2^{(n-1)}$

~~Expo~~ Expo. large

→ basically, to compute these XORs, a shallower networks hidden layer would to compute expo. larger computations.

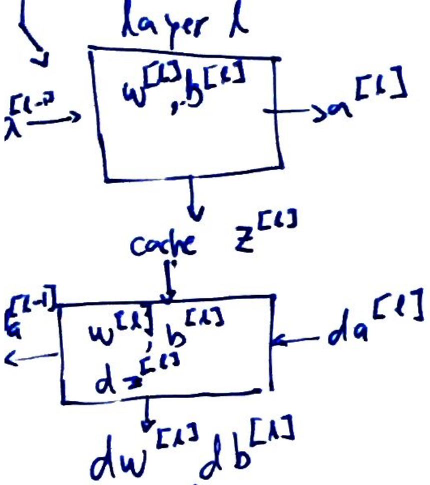
Building blocks of a deep NN

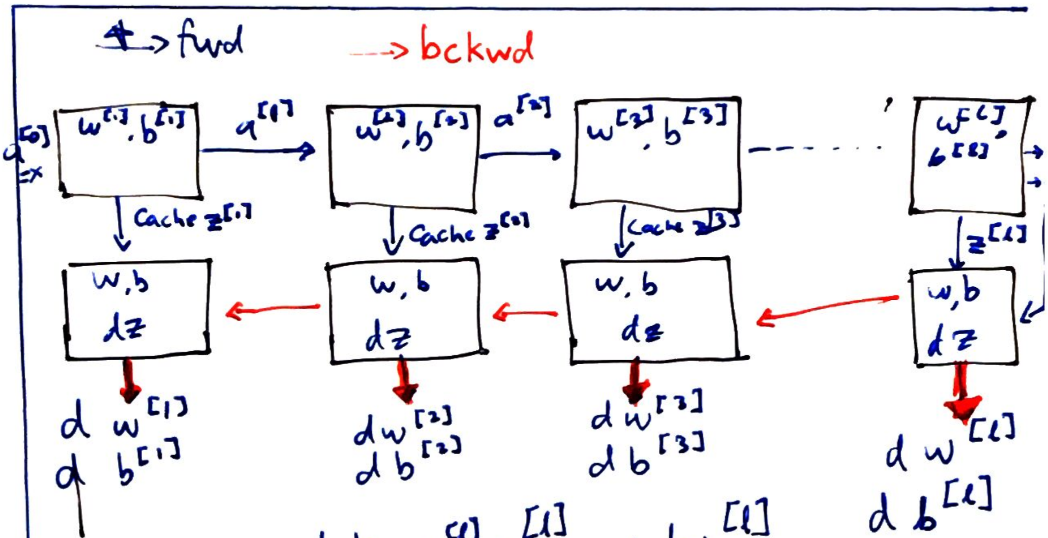
→ visualizing ideas behind fwd, bckwd propagation

for layer l, $W^{[l]}, b^{[l]}$

forward prgⁿ → $z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$, $a^{[l]} = g^{[l]}(z^{[l]})$
& cache $z^{[l]}$ for bckwd p.

Backward propagation: Input $da^{[l]}$, Cache $z^{[l]}$, [output $da^{[l-1]}$, $dW^{[l]}$, $db^{[l]}$]





Updates $W^{[l]} := W^{[l]} - \alpha dW^{[l]}$
$b^{[l]} := b^{[l]} - \alpha db^{[l]}$

we also use $W^{[l]}, b^{[l]}$ in bckwd propagation

Propagations:

- forward
inputs $a^{[l-1]}$, outputs $a^{[l]}$ & caches $z^{[l]}$

$$\left.\begin{aligned} Z^{[l]} &= W^{[l]} A^{[l-1]} + b^{[l]} \\ A^{[l]} &= g^{[l]}(Z^{[l]}) \end{aligned}\right\} \text{vectorized version}$$

- backward
inputs $da^{[l]}$, outputs $da^{[l-1]}$, $dW^{[l]}$, $db^{[l]}$

$*$ = pairwise mult.

$dz^{[l]} = da^{[l]} * g^{[l]\prime}(z^{[l]})$

$dW^{[l]} = dz^{[l]} * a^{[l-1]T}$

$db^{[l]} = dz^{[l]}$; $da = W^{[l]T} \cdot dz^{[l]}$

$dz^{[l]} = W^{[l+1]T} dz^{[l+1]} * g^{[l]\prime}(z^{[l]})$

Vectorized version:

$$dZ^{[l]} = dA^{[l]} * g^{[l]\prime}(Z^{[l]})$$

cached

$$dW^{[l]} = \frac{1}{m} dZ^{[l]} \cdot A^{[l-1]T}$$

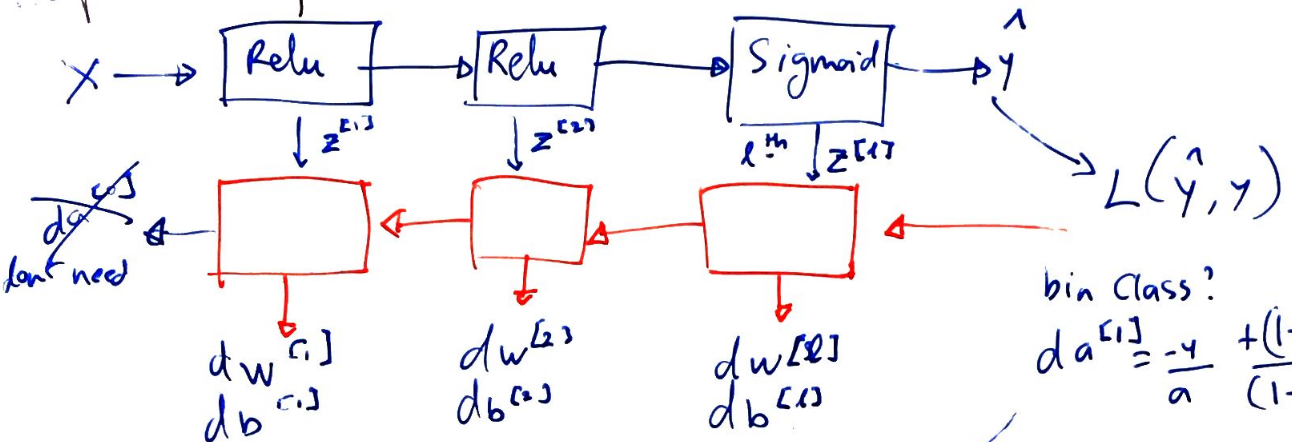
need

$$db^{[l]} = \frac{1}{m} \text{np.sum}(dZ^{[l]}, \text{axis}=1, \text{keepdims}=\text{True})$$

$$dA^{[l-1]} = W^{[l]T} \cdot dZ^{[l]}$$

output

Update Cycle:



bin Class?

$$da^{[l]} = \frac{-y}{a} + \frac{(1-y)}{(1-a)}$$

vectorized vn =

$$dA^{[l]} = \left(\frac{-y^{[1]}}{a^{[1]}} + \frac{1-y^{[1]}}{1-a^{[1]}} \dots \dots \frac{y^{[m]}}{a^{[m]}} + \frac{1-y^{[m]}}{1-a^{[m]}} \right.$$

Hyperparameters:

Hyper params values impact learning algo's params

Params: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]} \dots$

Hyper params: Learning rate α

controls

iterations

hidden layer L

hidden units $n^{[1]}, n^{[2]} \dots$

choice of activation f^n

later we'll also see Momentum, minibatch size, regularizations, ...

→ Applied deep learning is a very emperical process

gotta try a lot of values