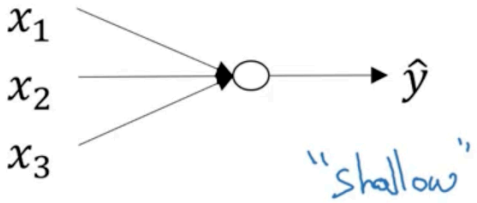
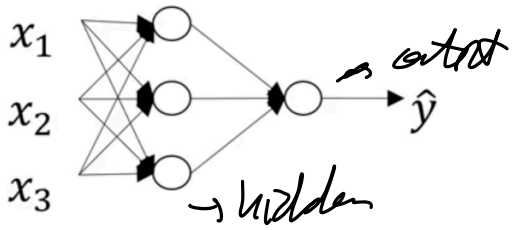
Deep L-layer Neural Network

What is a deep neural network?

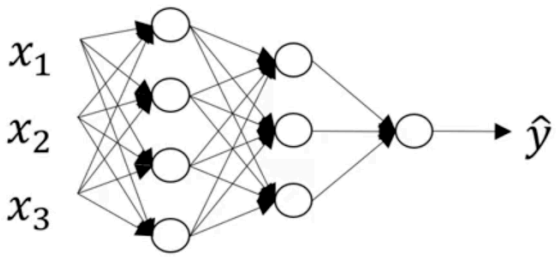
"2 layer NN"



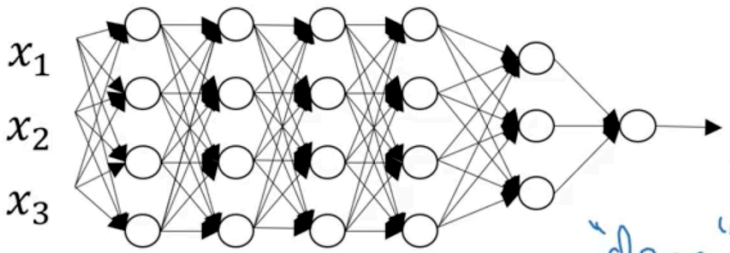
logistic regression



1 hidden layer



2 hidden layers



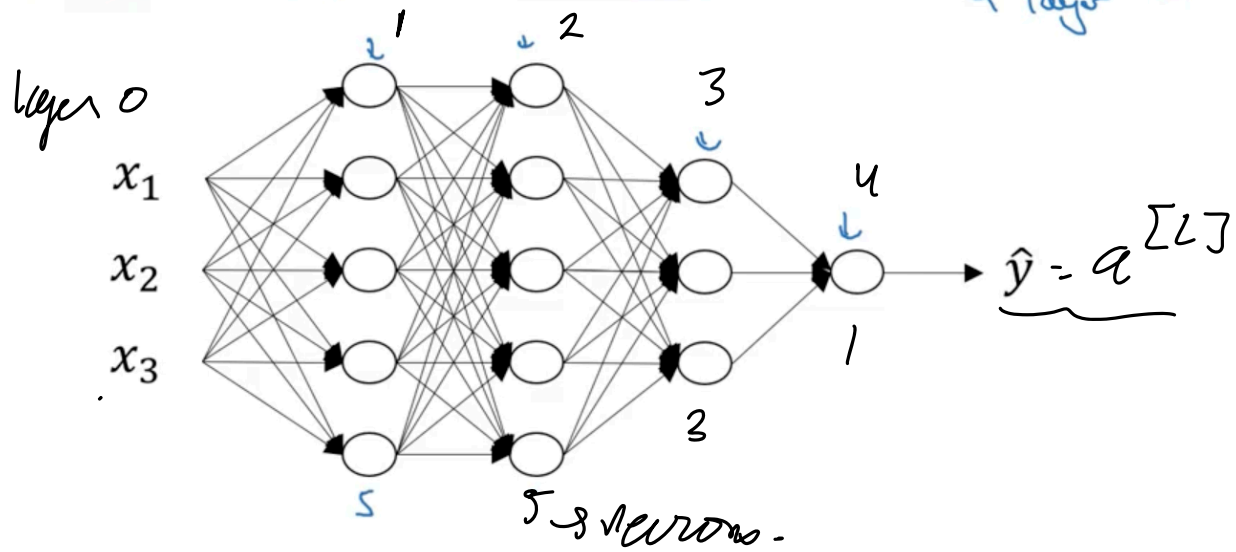
5 hidden layers

Andrew

deep layer NN can recognize functions that shallow ones can't

Deep neural network notation

4 layer NN



$x = a^{[0]}$

$L = 4$ (# of layers.

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

$n^{[1]} = 5$, $n^{[2]} = 5$, $n^{[3]} = 3$, $n^{[4]} = 1$

$n^{[0]} = n_x = 3$ → input

$n^{[L]} = 1$

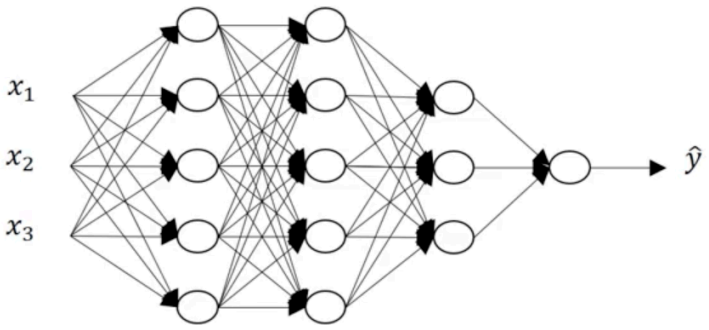
$a^{[l]}$ = Activations in layer.

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

$b^{[l]}$ = biases for $z^{[l]}$

FORWARD PROPAGATION IN A DEEP NETWORK

Forward propagation in a deep network



Given digit training example x: $a^{[0]}$

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

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

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

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

$\vdots$

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

$a^{[4]} = g^{[4]}(z^{[4]})$

$= \hat{y}$

$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$

$a^{[l]} = g^{[l]}(z^{[l]})$

VECTORIZED

$$Z^{[1]} = W^{[1]} X + b^{[1]}$$

$$A^{[1]} = g^{[1]}(Z^{[1]})$$

$$\vdots$$

$$\hat{Y} = g(Z^{[4]}) = A^{[4]}$$

$A^{[0]} = X$

$X = \begin{bmatrix} x_1 & & m \\ \downarrow & & \rightarrow \\ n_x & & \end{bmatrix}$

$A^{[1]} = \begin{bmatrix} & & m \\ \downarrow & & \rightarrow \\ n^{[1]} & & \end{bmatrix}$

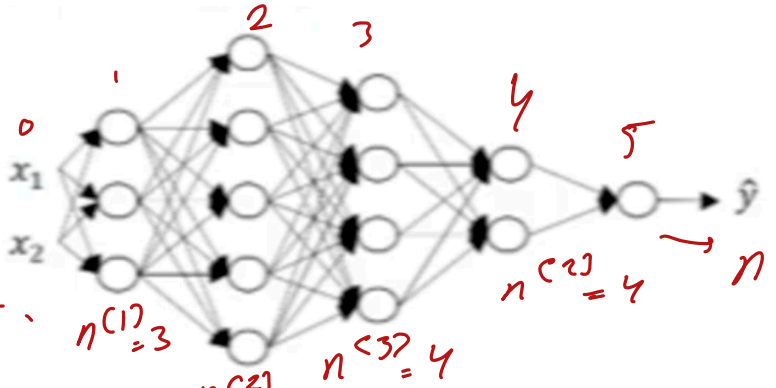
$n^{[1]}$ = # of neurons

for loop for the different layers → So one need to use a for loop here.

Getting MATRIX DIMENSIONS RIGHT

Parameters $W^{[l]}, b^{[l]}$

$L = 5$



$n^{[0]} = n_x = 2$, $n^{[1]} = 3$, $n^{[2]} = 4$, $n^{[3]} = 4$, $n^{[4]} = 2$, $n^{[5]} = 1$

$n^{[1]} = 5$

$z^{[1]} = w^{[1]} x + b^{[1]}$ figure for now.

$(3,1)$ → $(n^{[1]}, 1)$

$(3,1)$

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

(m)

$(2,1)$

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

$(3,2)$

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

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

$\begin{bmatrix} \vdots \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots & \vdots \\ \vdots & \vdots \end{bmatrix} \begin{bmatrix} \vdots \\ \vdots \end{bmatrix}$ $w^{[l]} = (n^{[l]}, n^{[l-1]})$

$b^{[l]} = (n^{[l]}, 1)$

Stay the Same.

$w^{[l]} : (n^{[l]}, n^{[l-1]}) = dw^{[l]}$

$b^{[l]} : (n^{[l]}, 1) = db^{[l]}$

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

or $(n^{[l]}, m)$

der vectorized

$Z^{[l]} = A^{[l]} = (n^{[l]}, m)$ stacking examples.

Is just with dz is $(n^{[l]}, 1)$ and get broadcasted.

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

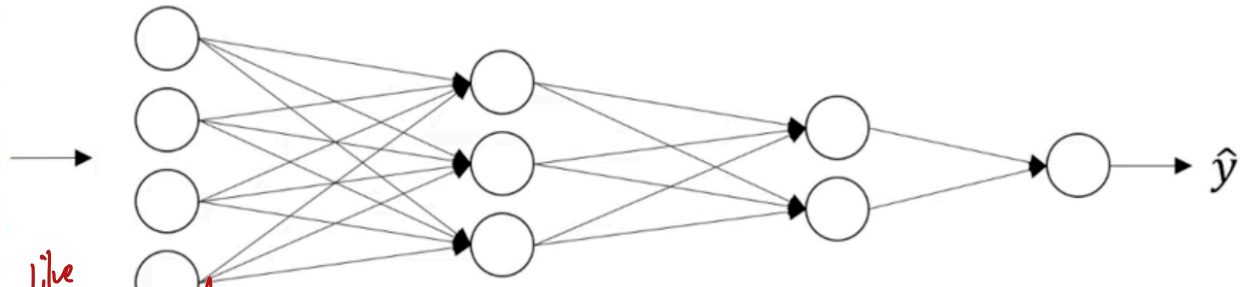
$Z^{[l]}, A^{[l]} : (n^{[l]}, m)$

$l=0 \Rightarrow A^{[0]} = X = (n_x, m)$

$dZ^{[l]}, dA^{[l]} : (n^{[l]}, m)$

With DEEP REPRESENTATIONS

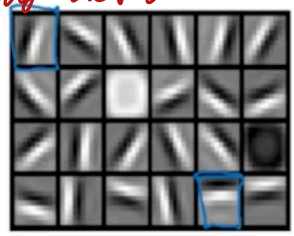
ntuition about deep representation

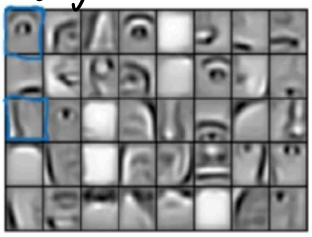


like edge detection.

looking at more

looking at small part





Kind of it like these early neurons trying to detect some edges,

Y this will try and find some body part (eye, nose)

Y then with these tries to recognize face.

Deep representation

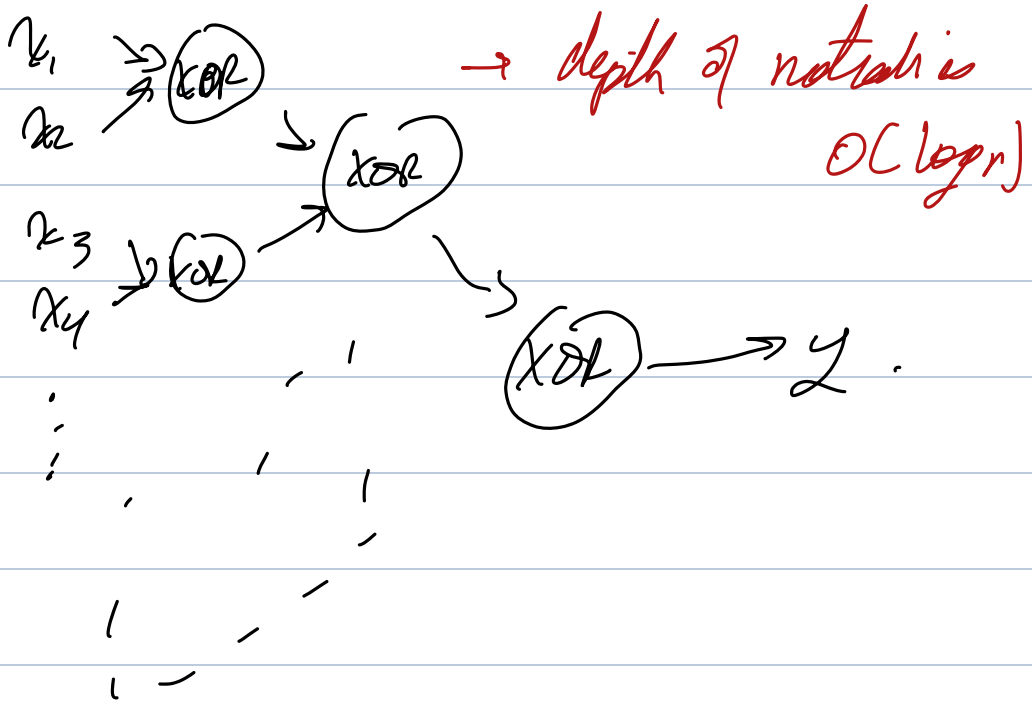
Audio → low level audio wave form → Phonemes C A T → words → Sentences

Circuit theory and deep learning

Informally: There are functions you can compute with a "small" L-layer deep neural network that shallower networks require exponentially more hidden units to compute.

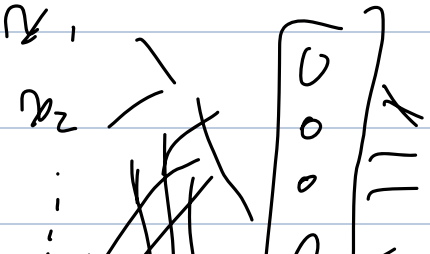
#of hidden units/neurons is small.

Eg. x_1 XOR x_2 XOR x_3 XOR ... XOR x_n.



→ depth of network is $O(\log n)$

Depth of network allowed can only use $\log n$ layers.



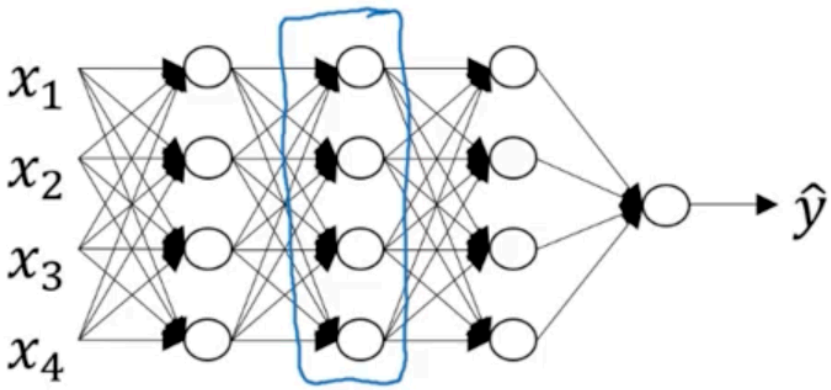
$O(2^n)$ hidden

units

n_x exponentially large

BUILDING BLOCKS OF NEURAL NETWORKS

Forward and backward functions



Layer l: $W^{[l]}, b^{[l]}$

Forward: Input $a^{[l-1]}$, output $a^{[l]}$

Forward:

$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$ ⇒ Cache $z^{[l]}$ for backward prop.

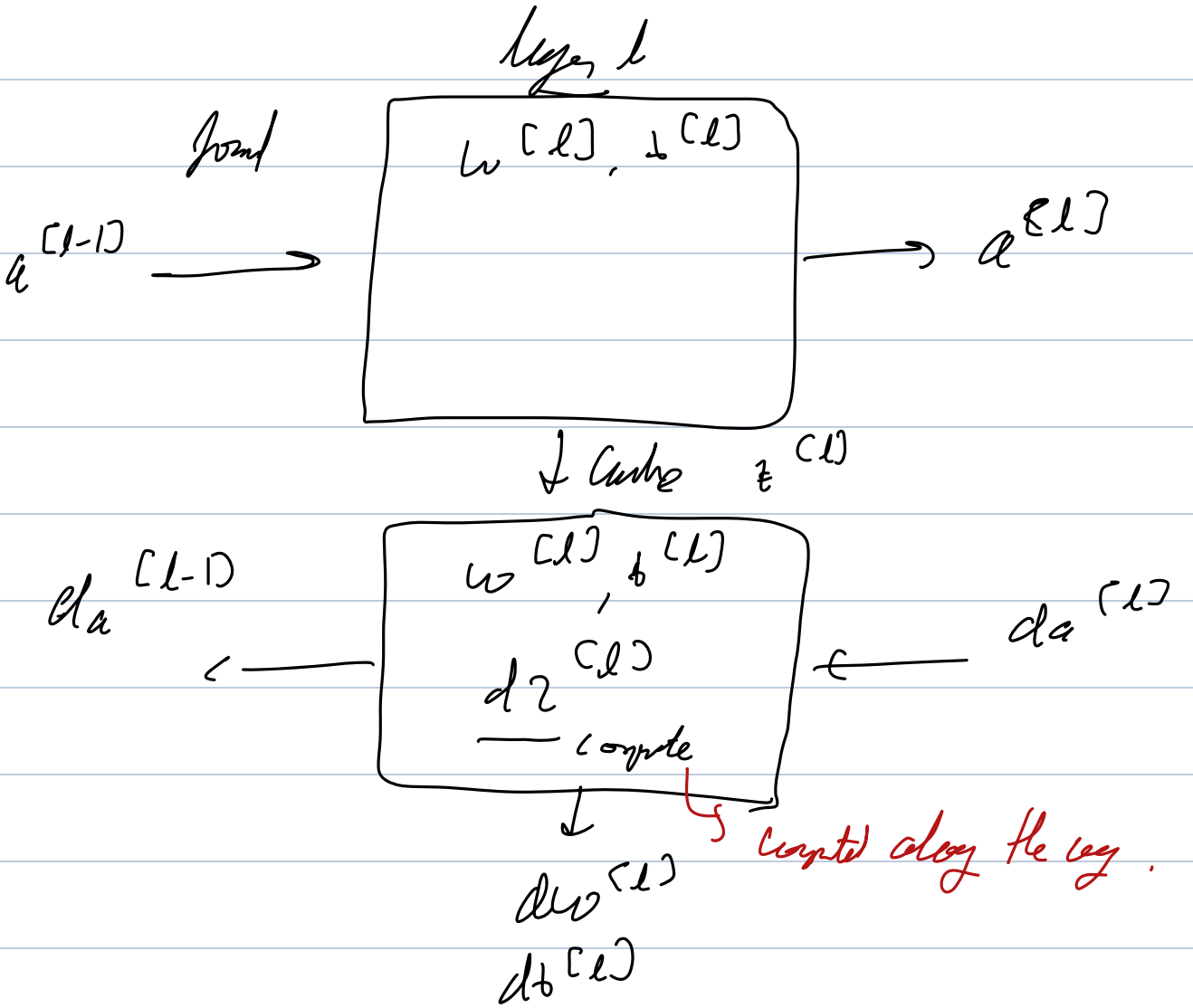
$a^{[l]} = g^{[l]}(z^{[l]})$

Backward: Input $da^{[l]}$, output $da^{[l-1]}$

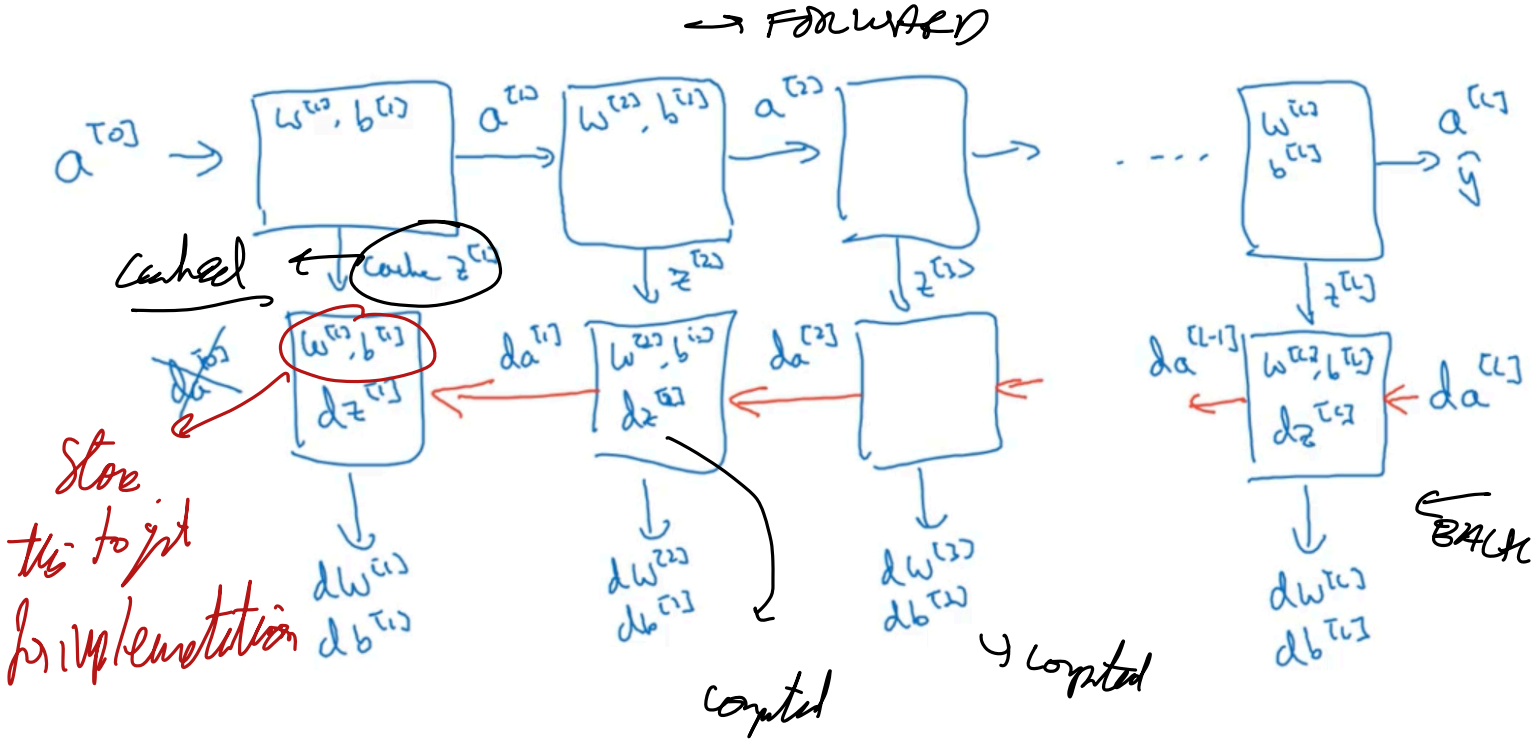
Cache ($z^{[l]}$),

$dW^{[l]}$

$db^{[l]}$



Forward and backward functions



along the way.

1 iteration through forward and back, and then:

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

Forward and Backward Propagation

Forward for layer l.

→ input $a^{[l-1]}$ output: $a^{[l]}$ (activations), cache ($z^{[l]}$) + w/ $w^{[l]}$, $b^{[l]}$

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

Vectorized

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

$a^{[0]}$ ← single training example

$A^{[0]}$

$X = A^{[0]}$ → [ ] → [ ] → [ ] → [ ] →

Backward at layer l

input $da^{[l]}$ output $da^{[l-1]}, dW^{[l]}, db^{[l]}$

element wise product

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

$$dw^{[l]} = dz^{[l]} \cdot a^{[l-1]T}$$

$$db^{[l]} = dz^{[l]}$$

$$da^{[l-1]} = w^{[l]T} \cdot dz^{[l]}$$

↓ Vectorized is the same idea

input

$$dz^{[l]} = dA^{[l]} * g^{[l]'}(Z^{[l]})$$

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

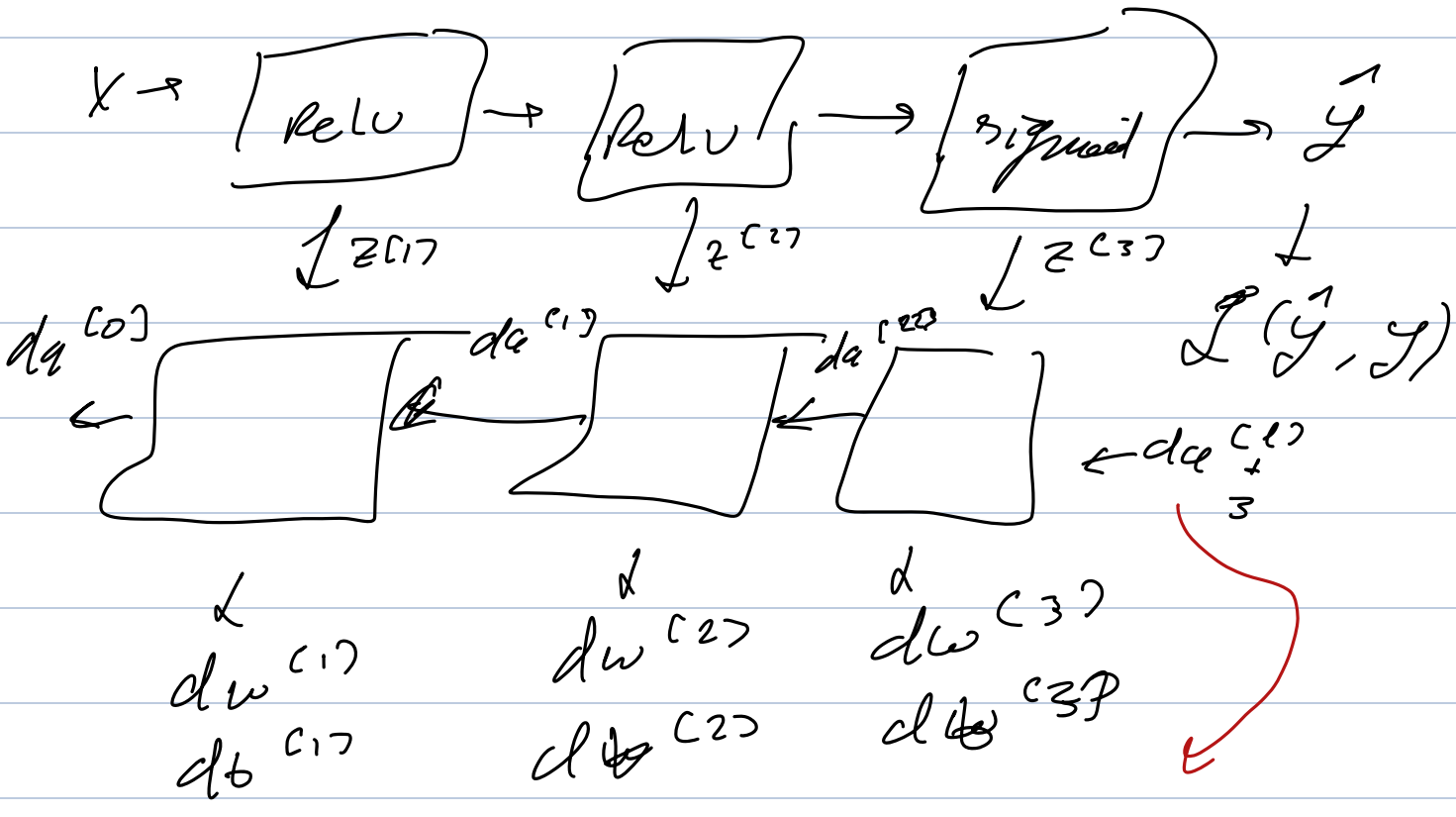
$$db^{[l]} = \frac{1}{m} np.sum(dZ^{[l]}, axis=1, keepdims=True)$$

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

→ to next step.

→ what you need for gradient descent

Summary.



$$dA^{[2]} = \left(-\frac{y^{(1)}}{a^{(1)}} + \frac{(1-y^{(1)})}{(1-a^{(1)})} \right) \quad da^{[2]} = \frac{-y}{a} + \frac{(1-y)}{(1-a)}$$

$$\cdots \frac{y^{(m)}}{a^{(m)}} + \frac{(1-y^{(m)})}{(1-a^{(m)})}$$

really the derivative of $\frac{d\hat{y}}{dz}$ w.r.t. a

we've done this before.

if this is Sigmoid at the end

PARAMETERS VS HYPERPARAMETERS

=> Go through the optional reading on FFNN in depth in WK 4.

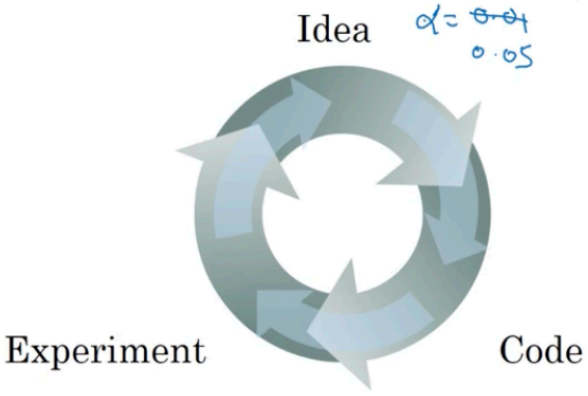
PARAMETERS VS HYPERPARAMETERS

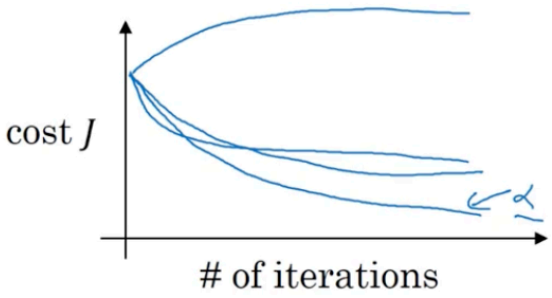
Parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]}$ ...

Hyperparameters:
- learning rate α.
- # of iterations
- # hidden layers L
- # hidden units $n^{[1]}, n^{[2]}$ => Neurons per layer.
- choice of activation fcn.

Later: Momentum, mini batch size etc.

Applied deep learning is a very empirical process → try and see what works.





0 1 2 3 4

$[n_x, 4, 3, 2, 1]$

$w^{[1]} = < n^{[1]}, n^{[1-1]} >$

n_1 i

$i = 0 \rightarrow [0, 5)$

$w_1 = \text{layer}(1.$