Result: $a^{[L]}$ → Lth layer Activation

$W^{[L]}$ → weights for the Lth layer

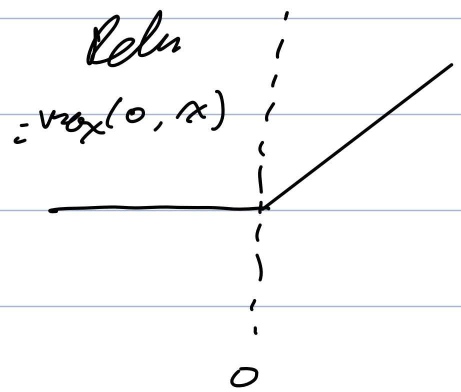
$b^{[L]}$ → bias for the Lth layer.

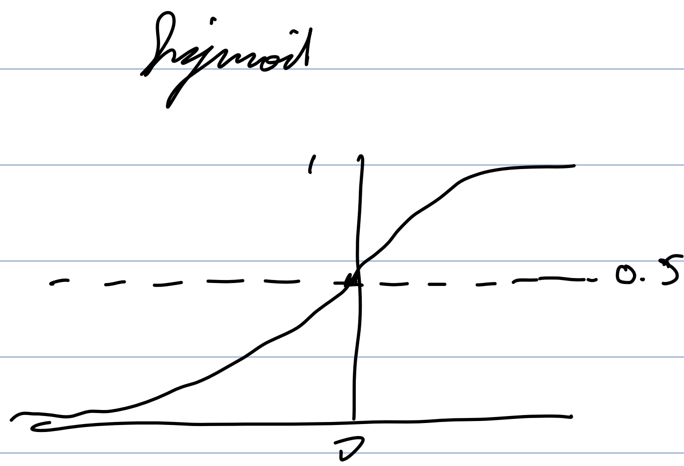
$x^{(i)}$ → ith training example.

$a_i^{[l]}$ → ith entry in the l layers activations.

Helper Functions

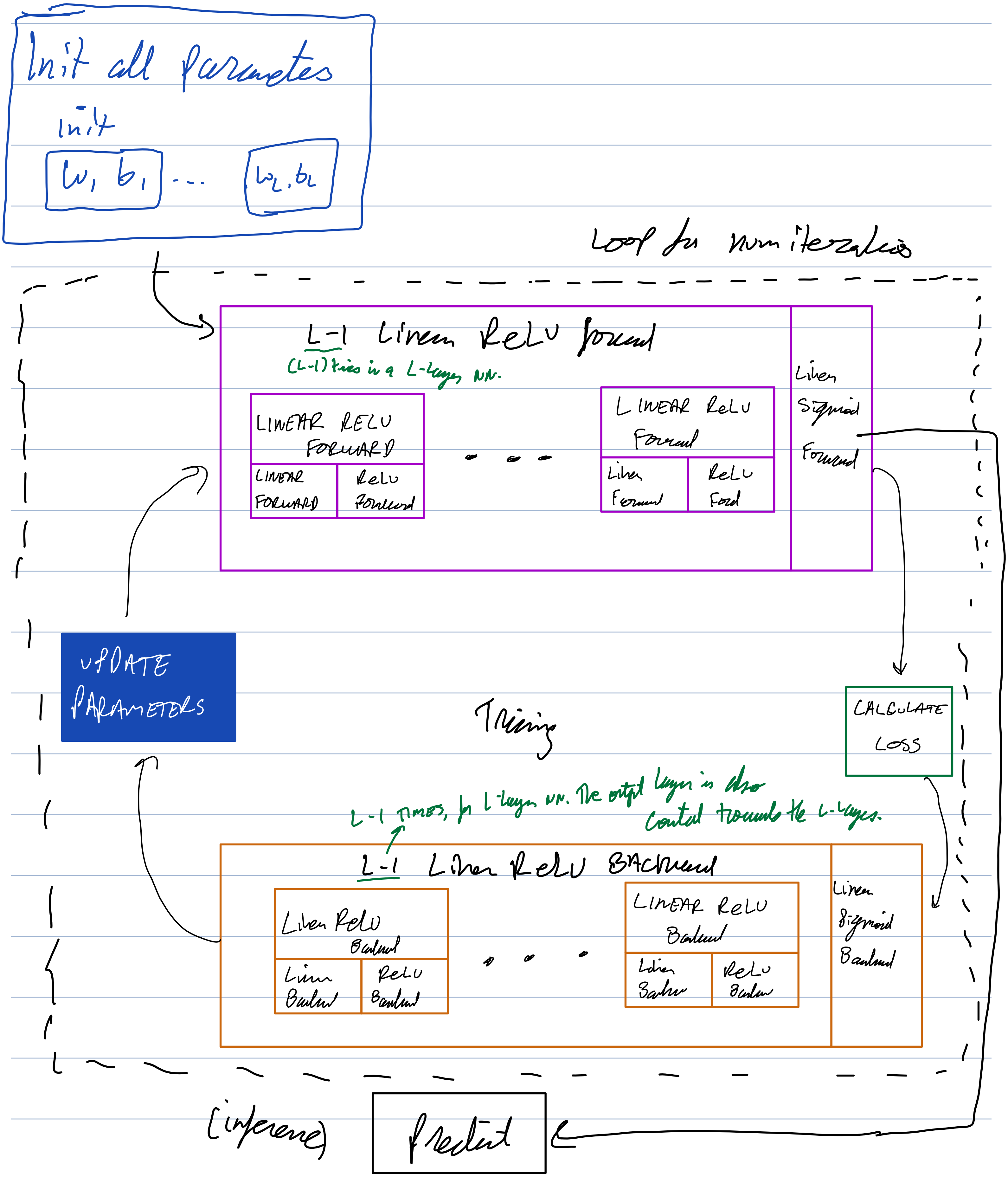
Non-linear units like ReLU →





outline:

Helper fns to build 2 layer NN and L-layer NN.



1) Linear part of layer's forward prop → $Z^{[l]}$
2) The activation function (ReLU / Sigmoid)

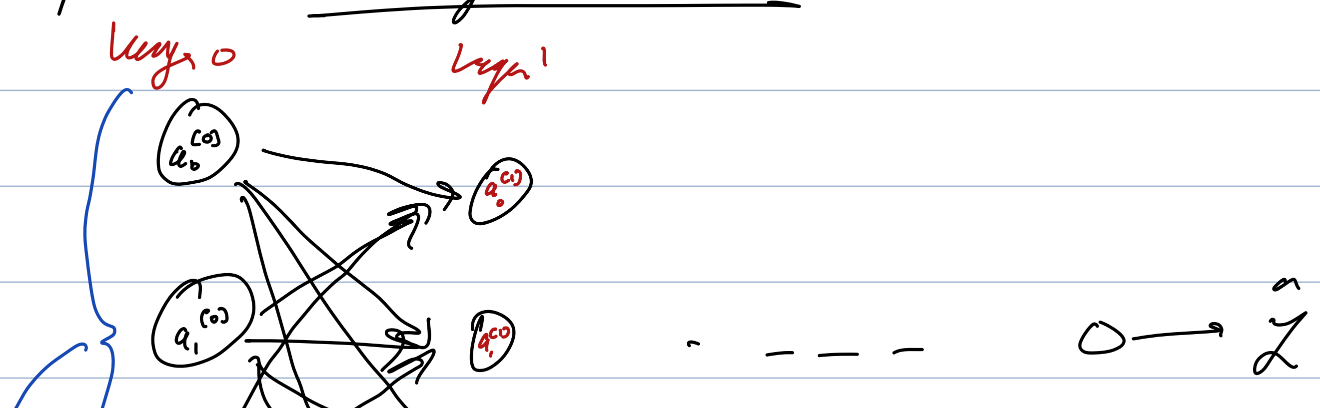
Combine these 2 into a [LINEAR → ACTIVATION] forward fxn.
Stack these L-1 times and add a final Sigmoid for the final layer L.
This gives you a L_model_forward

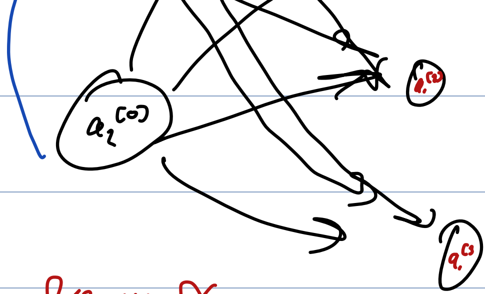
- Compute Loss

- Compute gradients to be able to update parameters, and keep iterating.

↳ For every forward fxn, there's a backward fxn that needs those values, so we store them in a CACHE for every forward module.

Example → 2 Layer NN.





Freeing X.

$x_0 = a_0^{[0]}$

$n_x = 4$,

4 nodes of activations.

These are like each pixel.

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

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

$a^{[l]} \to$ Activation in layer l.

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

$w^{[l]}$ = weight for $z^{[l]}$

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

X: is a training example.

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

$a^{[1]} = \underset{\text{fxn}}{g}^{[1]}(z^{[1]})$

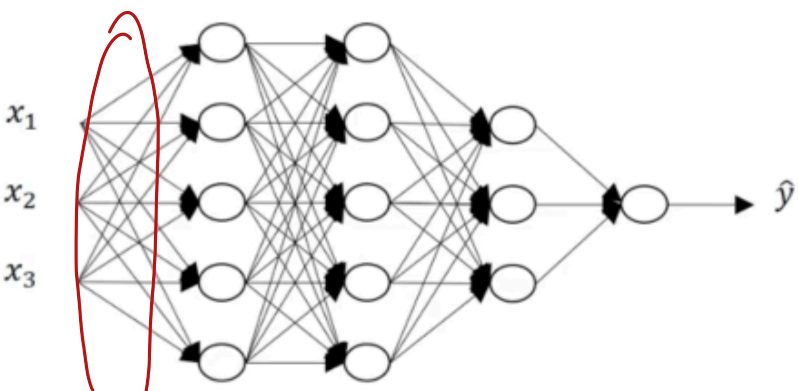
$$z^{[2]} = w^{[2]} a^{[1]} + b^{[1]}$$

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

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

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

Forward propagation in a deep network



Apply weights to prev activation

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

apply func (linear or ReLU etc.)

$$g(w^{[l]} a^{[l-1]} + b^{[l]})$$

VECTORIZED TO DO WHOLE LAYER IN PARALLEL

$$Z^{[l]} = W^{[l]} A^{[l-1]} + B^{[l]}$$ → VECTOR → BROADCAST

$(n^{[l]}, 1)$ $\quad$ $(n^{[l-1]}, 1)$ → m $\quad$ $(n^{[l]}, 1)$ to m Training examples

m

To turn $(n^{[l-1]}, 1) \to (n^{[l]}, 1)$

$W^{[l]} \Rightarrow (n^{[l]}, n^{[l-1]})$

$$(n^{[l]}, \underbrace{n^{[l-1]}}) \times (\underbrace{n^{[l-1]}}, 1)$$

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

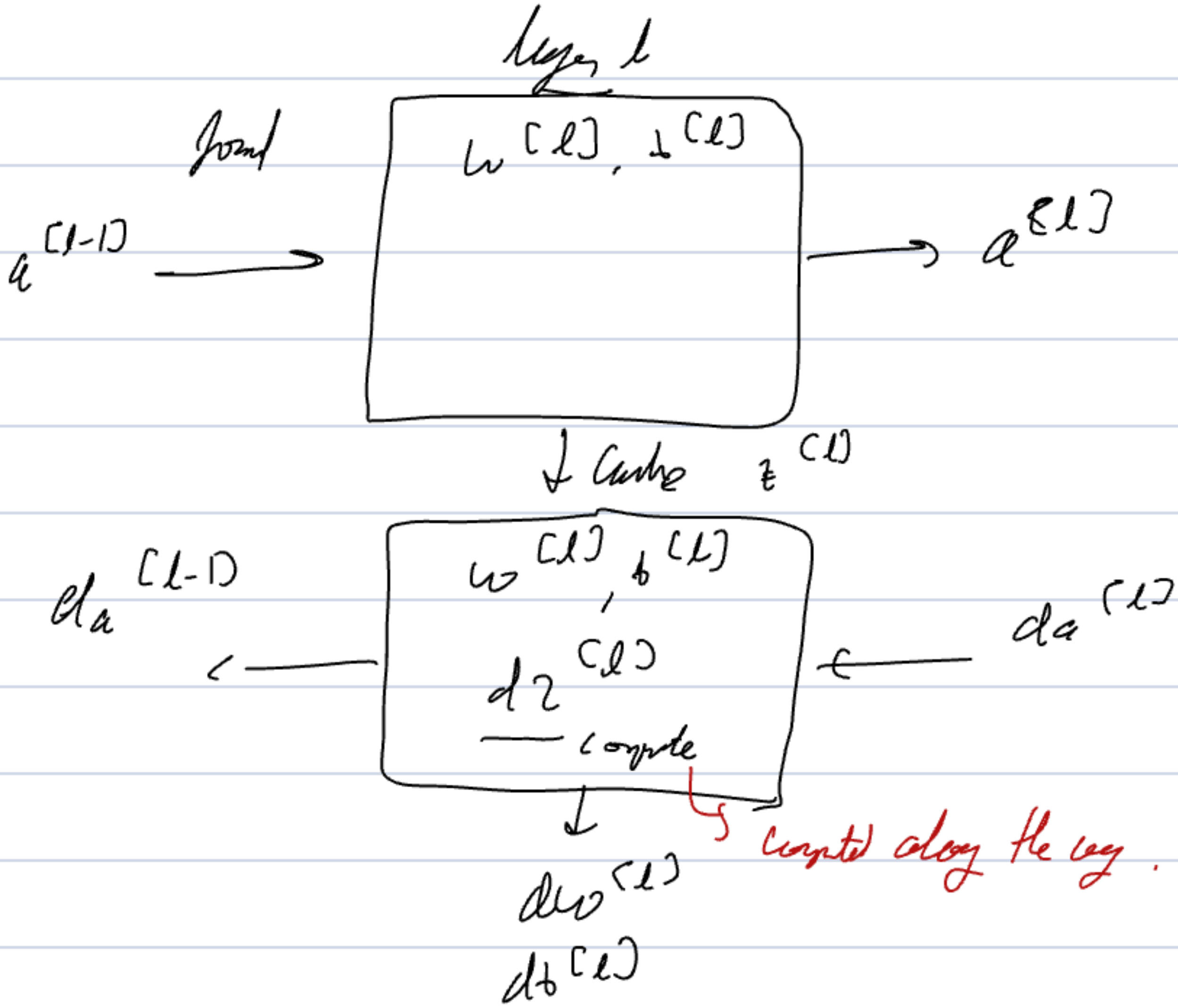
Range 1 to M → for m Training examples

→ And if the depth is $O(\log n)$ & $O(2^n)$ hidden units

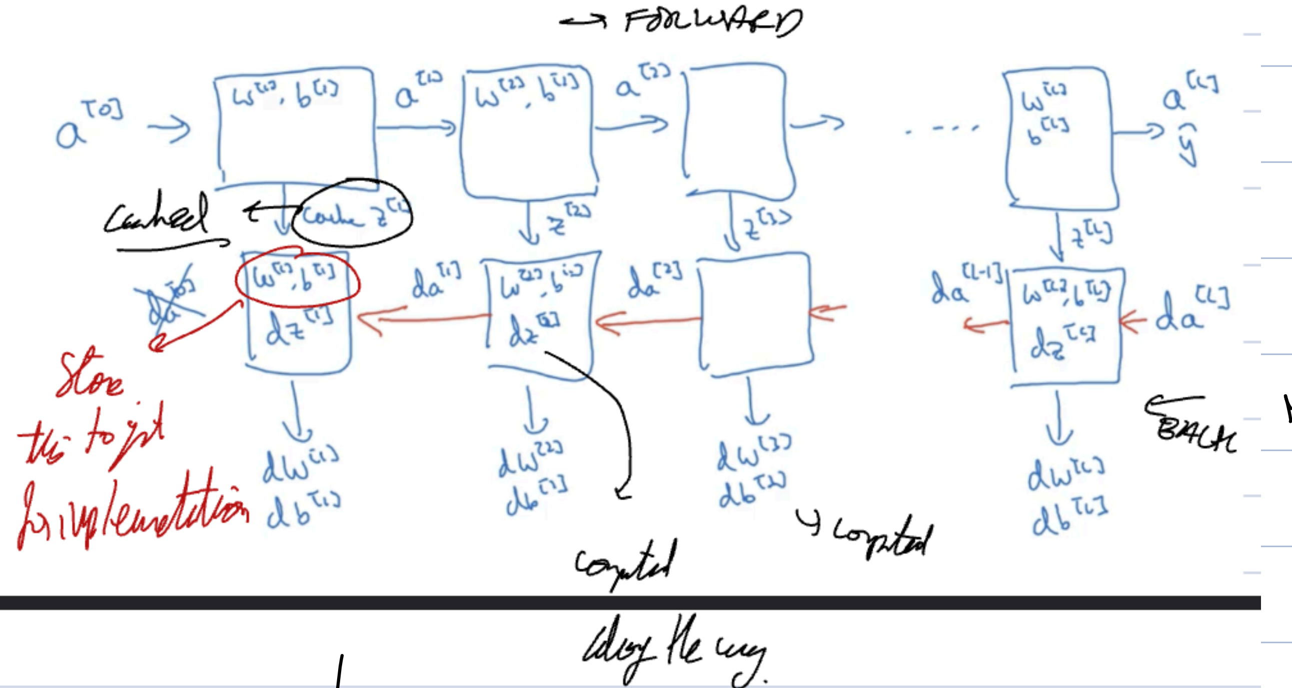
Forward → $z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$ → Cache $z^{[l]}$

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

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



Forward and backward functions

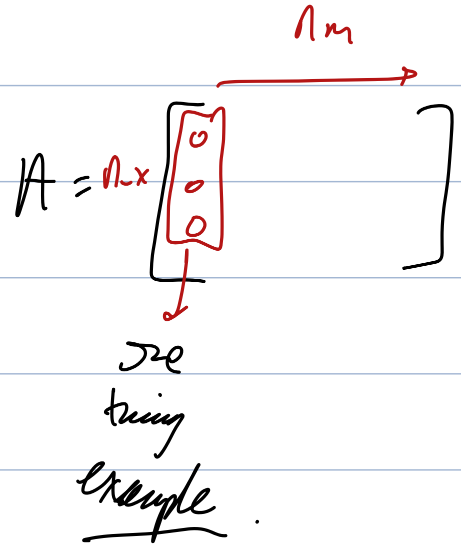


↓

Then use these to update the weights for next forward iteration.

$a^{[0]}$ → single training example.

$A^{[0]}$ → all training examples.



BACKWARD

input

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

element wise

cached earlier

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

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

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

↓

VECTORIZED (m examples)

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

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

Element wise ↑

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

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

2) To next step