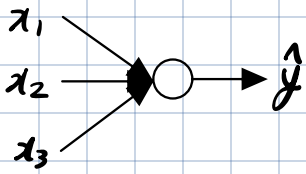
WEEK 4. Deep Neural Network

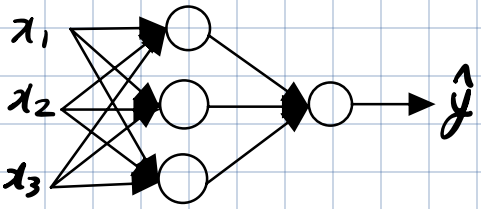
• Deep L-Neural Network

- What is a Deep Neural Network?



<Logistic Regression>

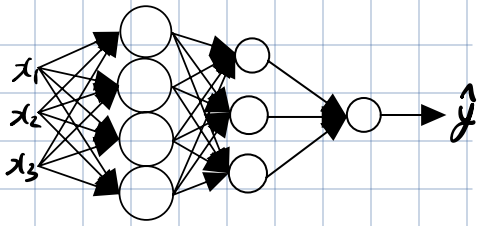
⇒ very shallow model (1 layer NN)



<1 hidden layer>

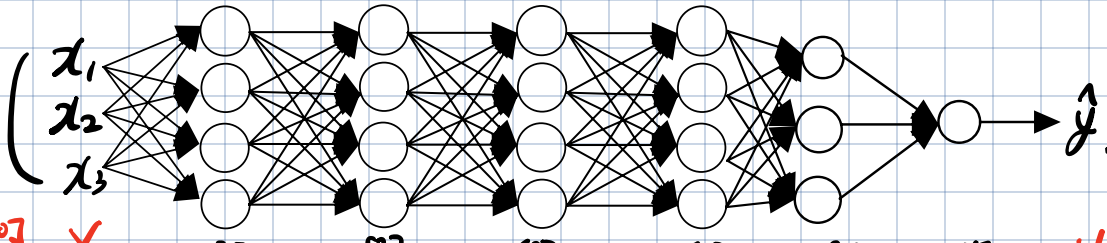
⇒ 2 layer NN

(don't count input layer)



<2 hidden layers>

⇒ 3 layer NN



<5 hidden layers>

⇒ deep model

can learn that shallow models can't learn

⇒ Increase the # of hidden layer to find best model (during validation)

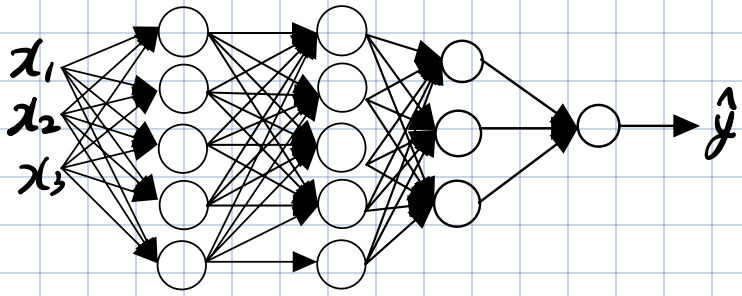
↳ hyperparameter

* Notation

- L: # of layer
- $n^{[l]}$: # of units in layer l
- $a^{[l]} = g^{[l]}(z^{[l]})$: activations in layer l
- $w^{[l]}$: weights for $z^{[l]}$, $b^{[l]}$
- X: input features = activations at layer 0 = $a^{[0]}$
- Y: Output = activations at the last layer L = $a^{[L]}$

• Forward Propagation in a Deep Network

— Single training sample x



General Rule of forward propagation

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

① Compute the activation of the first layer

$$z^{[1]} = w^{[1]} x + b^{[1]} \rightarrow a^{[0]}$$
$$a^{[1]} = g^{[1]}(z^{[1]})$$

↳ activation function g depends on layer

② Compute the activation of the second layer

$$z^{[2]} = w^{[2]} a^{[1]} + b^{[2]}$$
$$a^{[2]} = g^{[2]}(z^{[2]}$$
$$\vdots$$
$$\hat{y} = g^{[4]}(z^{[4]}) = a^{[4]}$$

— Vectorized forward Propagation for entire training samples

① Compute the activation of the first layer

$$Z^{[1]} = W^{[1]} X^{[1]} + b^{[1]} \rightarrow a^{[0]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$

② Compute the activation of the second layer

$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$$
$$A^{[2]} = g^{[2]}(Z^{[2]})$$
$$\vdots$$
$$\hat{Y} = g(Z^{[4]}) = A^{[4]}$$

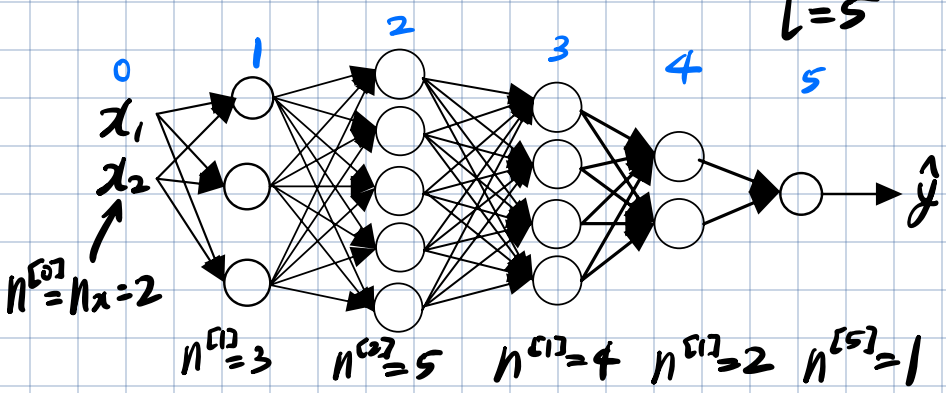
⇒ need for-loop

$l = 1 \dots 4$

Be careful of matrix dimensions!

- Getting your matrix dimensions right

— parameters $W^{[l]}$ and $b^{[l]}$



General Rule of dimensions

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

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

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

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

① Compute Forward propagation of 1st layer

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

① activations of first hidden layer ⇒ 3×1 vector $(n^{[1]}, 1)$

② 2 input features ⇒ (2,1) vector $(n^{[0]}, 1)$

③ Something makes $W(n^{[0]}, 1) = (n^{[1]}, 1)$

$\begin{bmatrix}\vdots\end{bmatrix}\begin{bmatrix}\vdots\end{bmatrix} = \begin{bmatrix}\vdots\end{bmatrix}$

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

④ to get output dimension of (3,1), it should be (3,1), too. $(n^{[1]}, 1)$

② Compute Forward propagation of 2nd layer

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

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

$(n^{[2]}, 1)\ (n^{[2]}, n^{[1]})\ (n^{[1]}, 1)$

— The dimension of derivatives should be same during back propagation.

$W \leftrightarrow dW,\ b \leftrightarrow db$

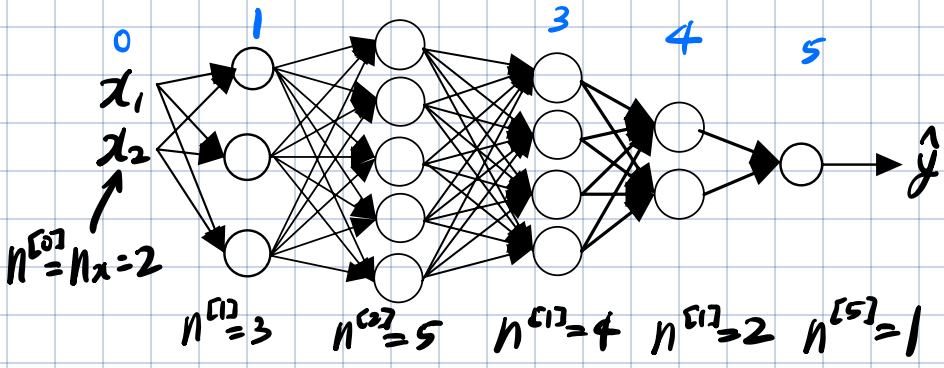
— The other important dimension checks

z

x

$a^{[l]} = g^{[l]}(z^{[l]})$ ⇒ z & a have same dimensions

- In vectorized implementation on entire training set
  ⇒ Z, a, x will be changed



General rule of dimensions in vectorized implementation

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

$l=0 : A^{[0]} = X = (n^{[0]}, m)$

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

① Compute Forward propogation of 1st layer

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

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

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

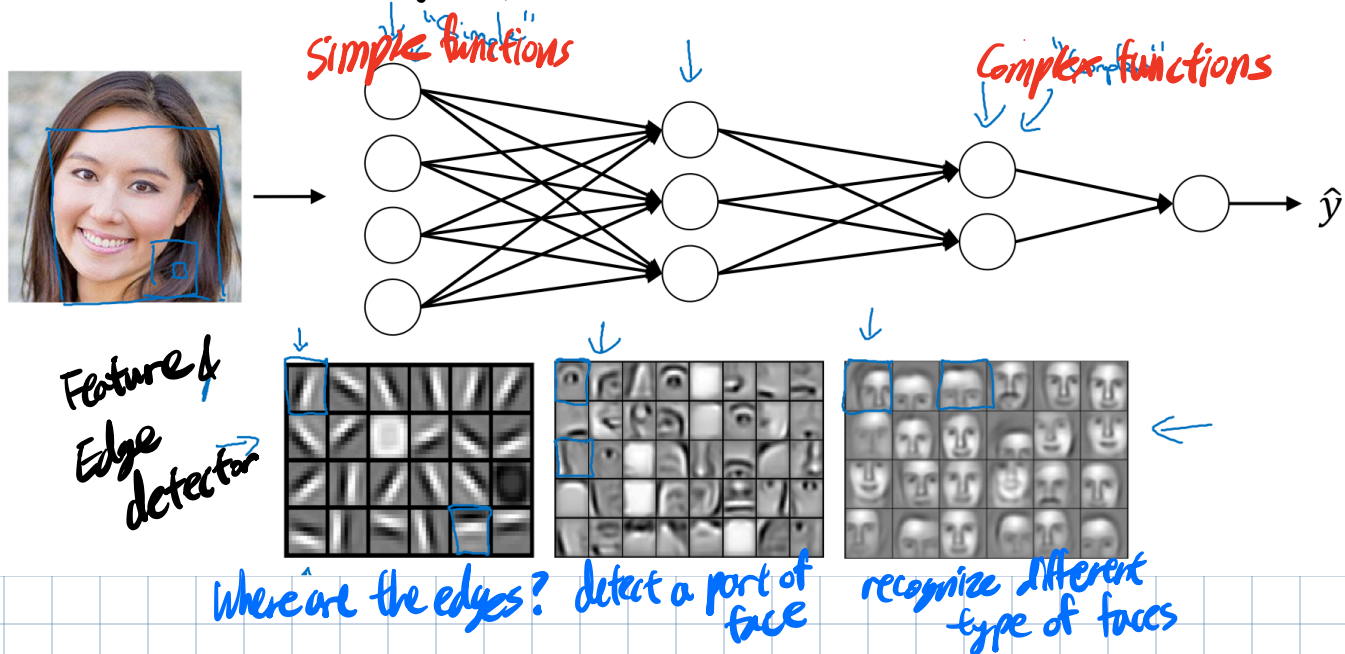
③ $(n^{[0]}, m)$

① $[z^{[1](1)}, z^{[1](2)}, \cdots z^{[1](m)}]$ → $(n^{[1]}, m)$

② stays the same $(n^{[1]}, n^{[0]})$

④ still $(n^{[1]}, 1)$ ⇒ broadcasting to $(n^{[1]}, m)$

- **Why deep representations?**
  - Intuition about deep representation



Speak recognition system: low level audio wave → words → sentence/phrase
(phonemes)

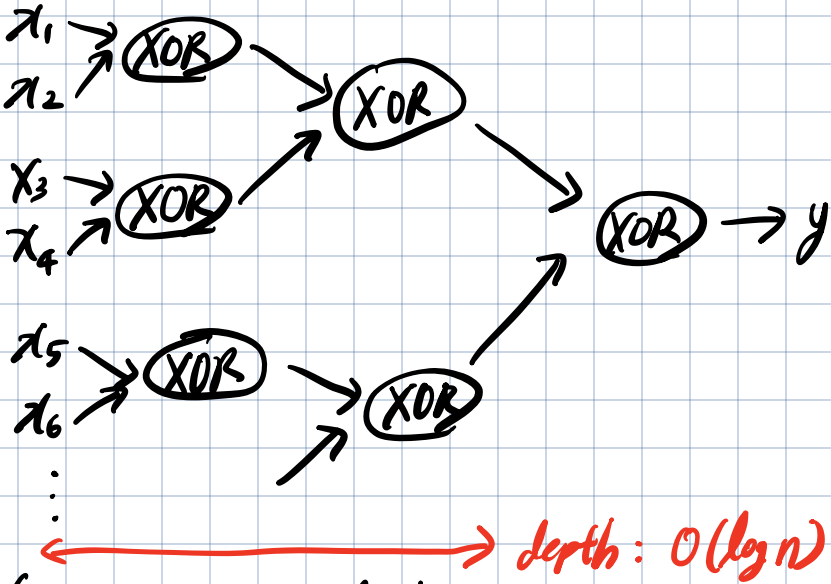
— Circuit theory & deep learning

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.

Low # of hidden layer

Let's say you compute 'XOR' of all the input features

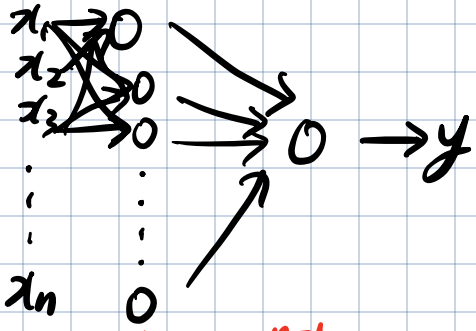
x_1 XOR x_2 XOR x_3 XOR ··· XOR x_n



You don't need many gates (node)

depth: $O(\log n)$

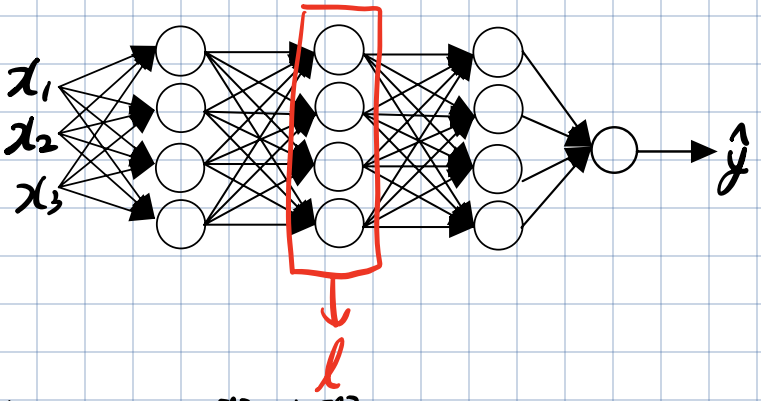
If you use 1 hidden layer...



In order to compute XOR, the hidden layer will need to be exponentially large

→ 2^{n-1} possible configurations ⇒ $O(2^n)$

- Building blocks of deep neural networks



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

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

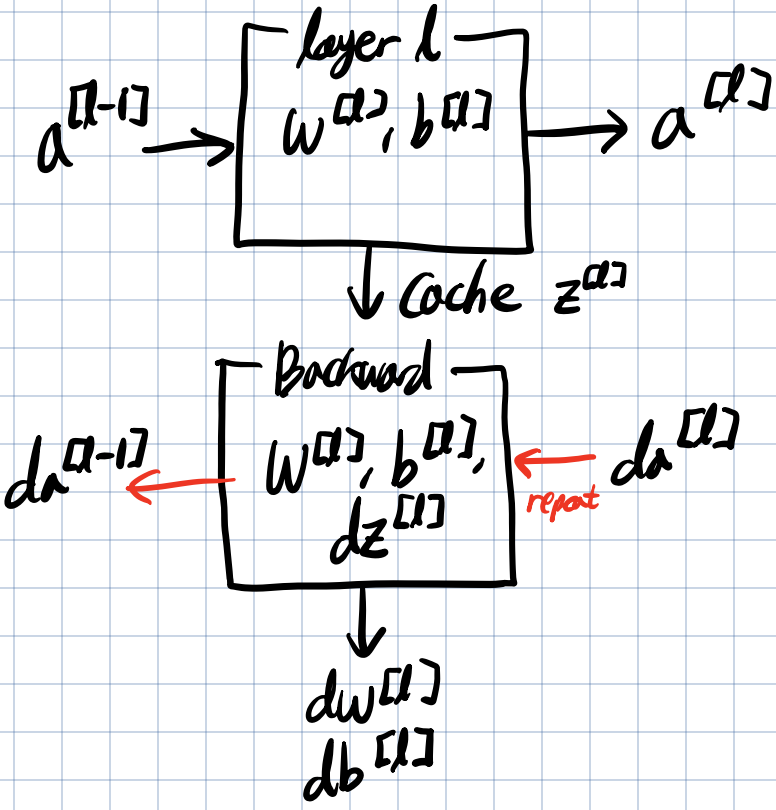
$z^{[l]}$: $W^{[l]}a^{[l-1]} + b^{[l]}$ → cache the value $z^{[l]}$ for later back propagation

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

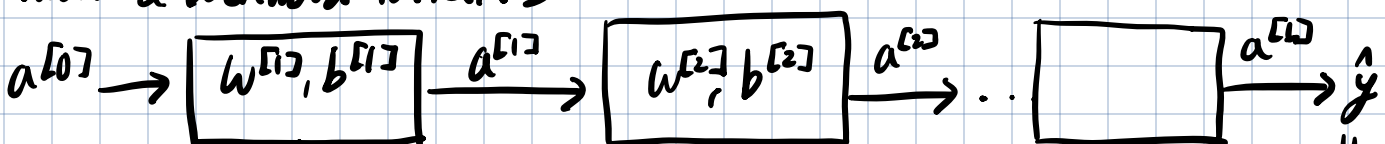
Backward: Input $da^{[l]}$, output $da^{[l-1]}$

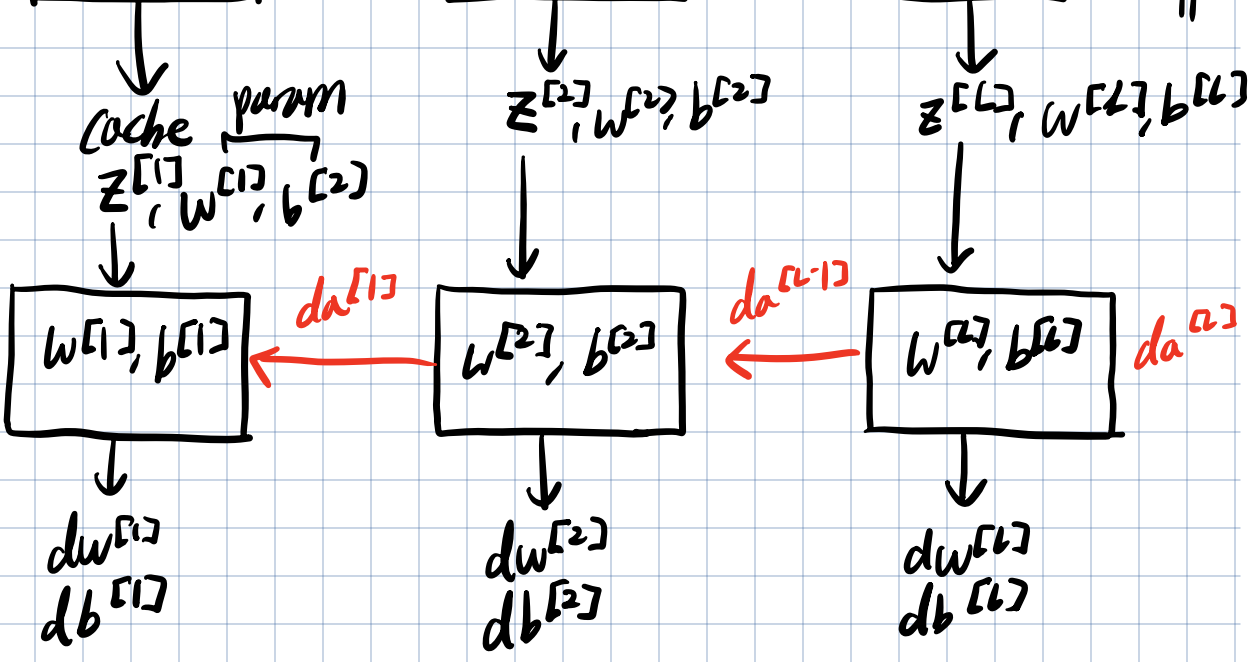
use cache($z^{[l]}$)

$dw^{[l]}$
$db^{[l]}$



- Forward & Backward functions





$$w^{[l]} = w^{[l-1]} - \alpha dw^{[l]}$$
$$b^{[l]} = b^{[l-1]} - \alpha db^{[l]}$$

* Cache : storing value of z for backpropagation

- **Forward and backward propagation**

  - Forward propagation for layer l

    Input $a^{[l-1]}$

    Output $a^{[l]}$, cache($z^{[l]}$) ← $w^{[l]}, b^{[l]}$

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

<Vectorized>

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

  - Backward propagation

    Input: $da^{[l]}$

    Output: $da^{[l-1]}, dW^{[l]}, db^{[l]}$

$$dz^{[l]} = da^{[l]} * g^{[l]\prime}(z^{[l]})$$
$$dw^{[l]} = dz^{[l]} \cdot a^{[l-1]}$$
$$db^{[l]} = dz^{[l]}$$

→ cache

$$dz^{[l]} = dA^{[l]} * g^{[l]\prime}(z^{[l]})$$
$$dW^{[l]} = \frac{1}{m} dz^{[l]} * A^{[l-1]T}$$
$$db^{[l]} = \frac{1}{m} np.sum(dz^{[l]}, axis=1,$$

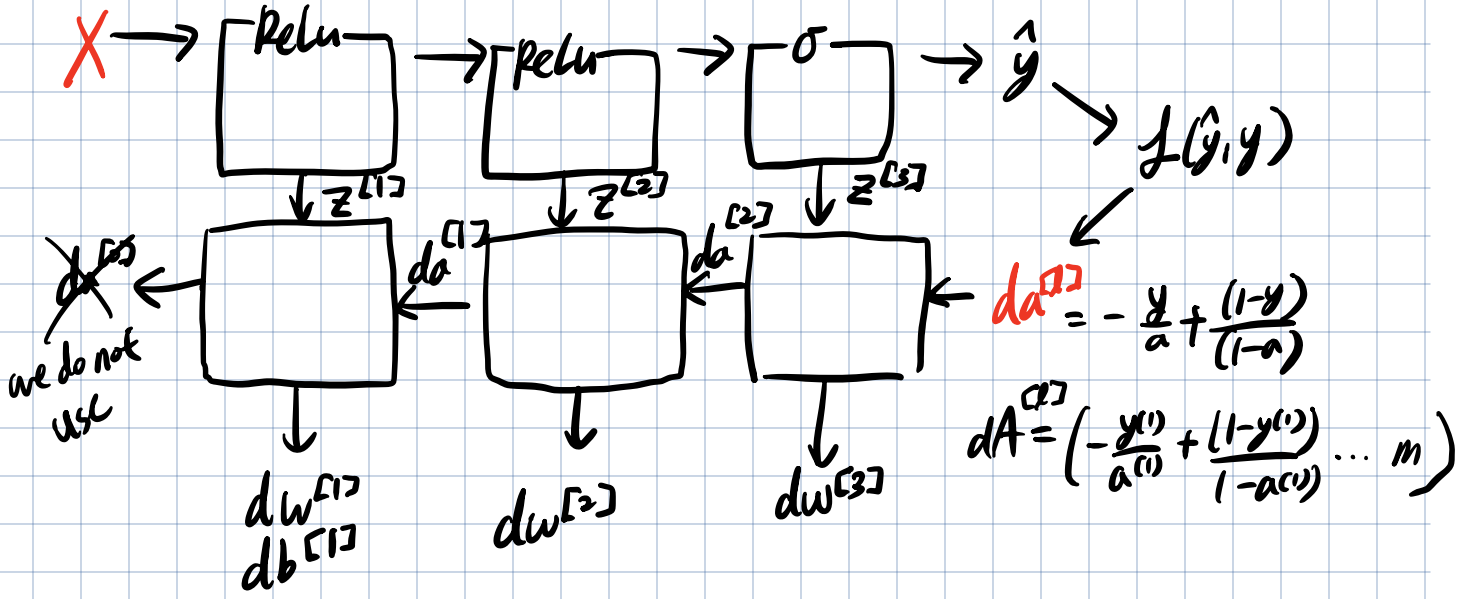
$da^{[l-1]} = W^{[l]T} dz^{[l]}$

↳ $dz^{[l]} = W^{[l+1]T} dz^{[l+1]} * g^{[l]'}(z^{[l]})$

keepdims = True)

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

- Summary



- Parameters vs Hyperparameters

- Hyperparameters?

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

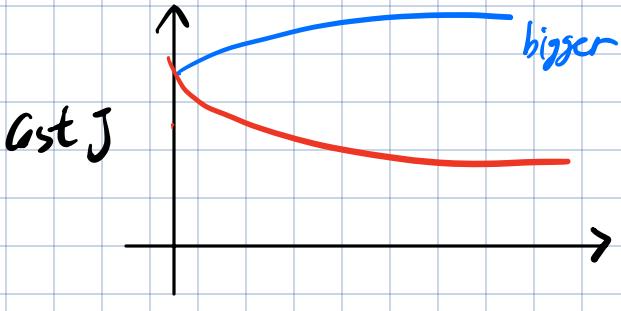
Hyperparameters: learning rate α, # iterations, # hidden layers, Choice of activation functions, momentum, minibatch size, ...

control W & b

⇒ determine final values of Parameters

- Applied deep learning is a very empirical process

⇒ If you are not sure what's the best value?



try a single value of α and check if J decreases and try larger value for α

⇒ find α for fast learning & minimal J

- Try a range of values and see if it works
- There is no always-best hyperparameters (∵ CPU, GPU change)