§ Building your Deep Neural Network

1. 2-layer Neural Network

	Shape of W	**Shape of b**	**Activation**	**Shape of Activation**
Layer 1	$(n^{[1]}, 12288)$	$(n^{[1]}, 1)$	$Z^{[1]} = W^{[1]}X + b^{[1]}$	$(n^{[1]}, 209)$
Layer 2	$(n^{[2]}, n^{[1]})$	$(n^{[2]}, 1)$	$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$	$(n^{[2]}, 209)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Layer L-1	$(n^{[L-1]}, n^{[L-2]})$	$(n^{[L-1]}, 1)$	$Z^{[L-1]} = W^{[L-1]}A^{[L-2]} + b^{[L-1]}$	$(n^{[L-1]}, 209)$
Layer L	$(n^{[L]}, n^{[L-1]})$	$(n^{[L]}, 1)$	$Z^{[L]} = W^{[L]}A^{[L-1]} + b^{[L]}$	$(n^{[L]}, 209)$

Model structure: [LINEAR → RELU] × (L-1)

⇓

LINEAR

⇓

SIGMOID

Initialize model:

for l in range (1, L):

parameters['W' + str(l)] = np.random.randn(layer_dims[l], layer_dims[l-1]) * 0.01

parameters['b' + str(l)] = np.zeros((layer_dims[l], 1))

→ Wl: weight matrix of shape (layer_dims[l], layer_dims[l-1])

bl: bias vector of shape (layer_dims[l], 1)

2. Forward propagation module

(1) Linear forward

$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$ $\quad (A^{[0]} = X)$

Z = np.dot(W, A) + b

W: weight matrix, shape (layer_dims[l], layer_dims[l-1])

A: activations, shape (layer_dims[l-1], examples)

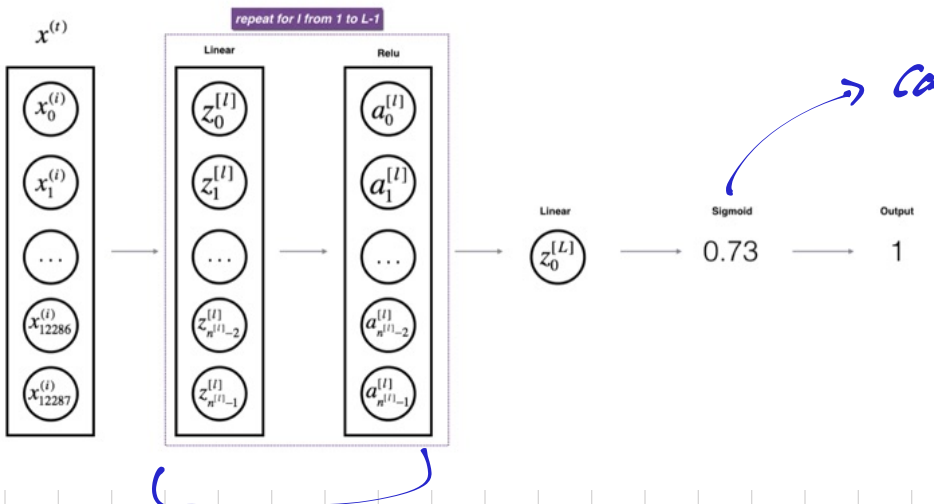
b: bias vectors, shape (layer_dims[l], 1)

(2) Linear Activation forward

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

Sigmoid ReLu

(3) L-Layer Model



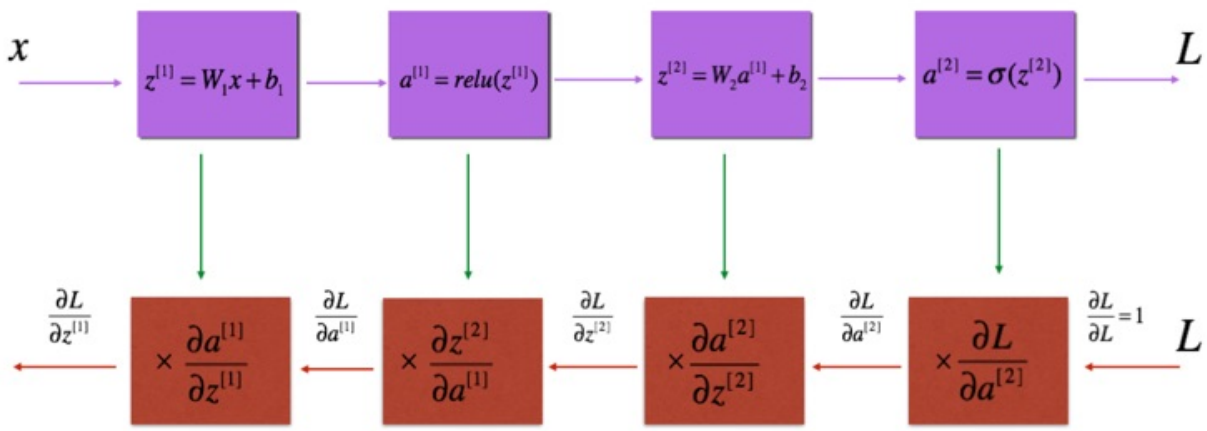
→ calculate Sigmoid at least

calculate [LINEAR → RELU] (L-1) times

3. Cost function

```
cost = (-1./m) * np.sum(np.multiply(Y, np.log(AL))
    + np.multiply((1-Y), np.log(1-AL)))
```

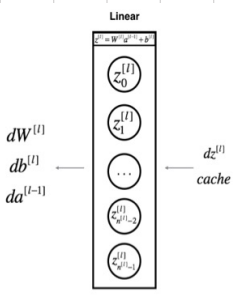
4. Backward Propagation module



Now, similar to forward propagation, you are going to build the backward propagation in three steps:

- LINEAR backward
- LINEAR -> ACTIVATION backward where ACTIVATION computes the derivative of either the ReLU or sigmoid activation
- [LINEAR -> RELU] × (L-1) -> LINEAR -> SIGMOID backward (whole model)

(1) Linear backward



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

$$db^{[l]} = \frac{1}{m} \sum_{i=1}^{m} dZ^{[l](i)}$$

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

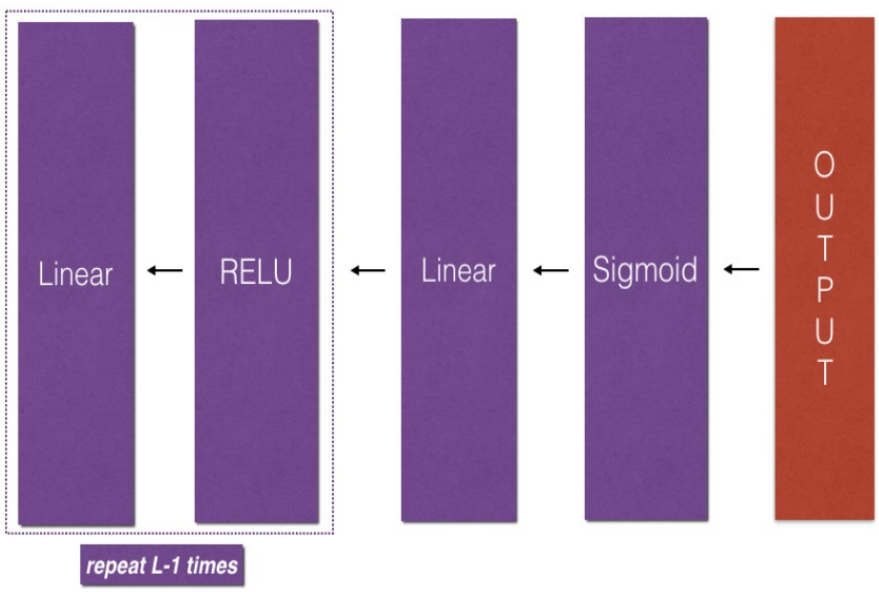
```
dW = (1./m) * np.dot(dZ, cache[0].T)
db = (1./m) * np.sum(dZ, axis=1, keepdims=True)
dA_prev = np.dot(cache[1].T, dZ)
```

(2) Linear_Activation backward

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

↳ Sigmoid / ReLU

(3) L-Model Backward



$$dAL = \frac{\partial \mathcal{L}}{\partial A^{[L]}}$$

```
dAL = -(np.divide(Y, AL) -
        np.divide(1-Y, 1-AL))
```

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