deep Learning_c1_week4

Created: 19/01/29 22:08 **Updated:** 19/02/01 10:54

Deep neural network

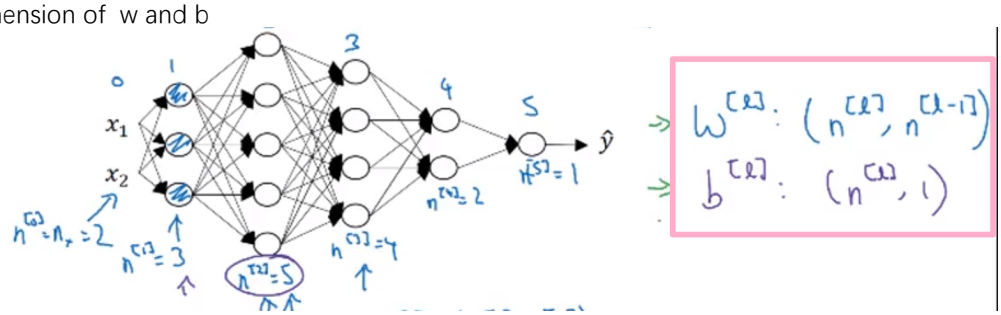
- notation
- forward propagation
  - use for loop go through each layer
  - each layer compute

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

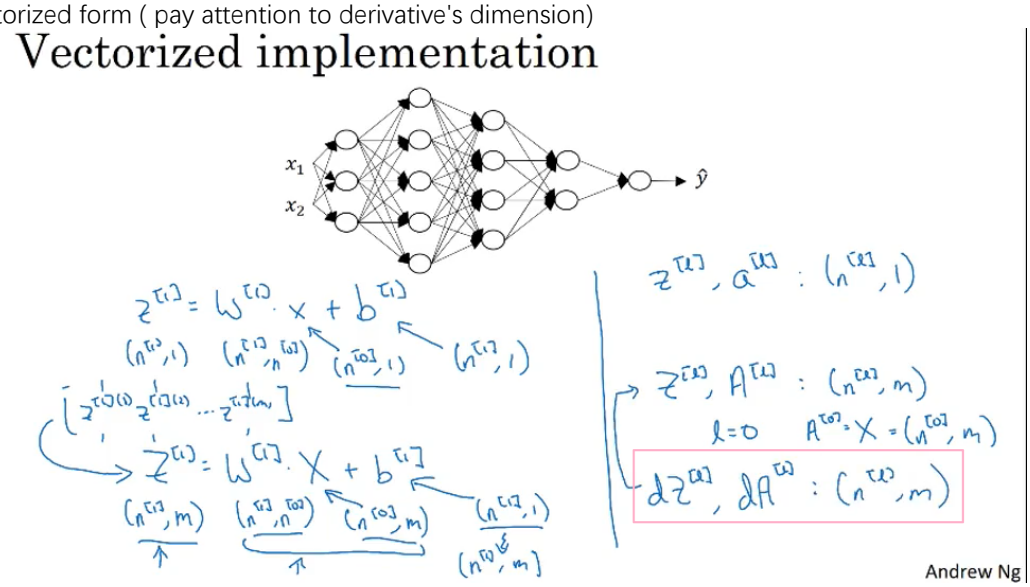
$$A^{[n]} = g(Z^{[n]})$$

, while g() is the active function.

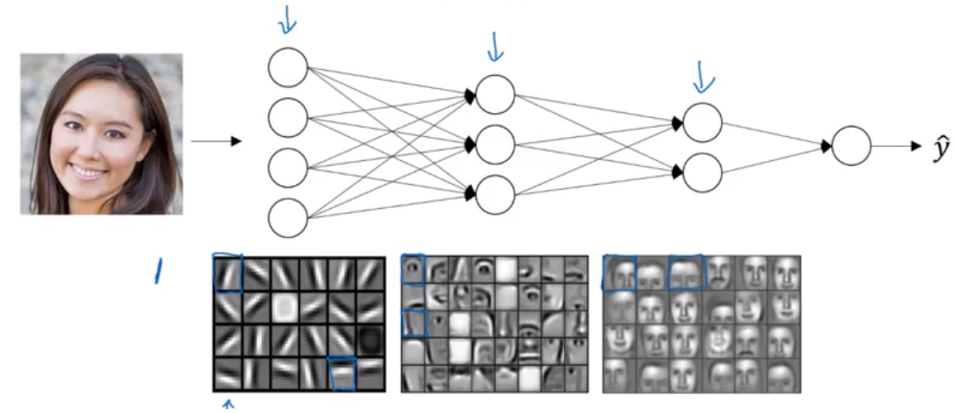
- check your matrix dimensions!
  - FP: dimension of w and b



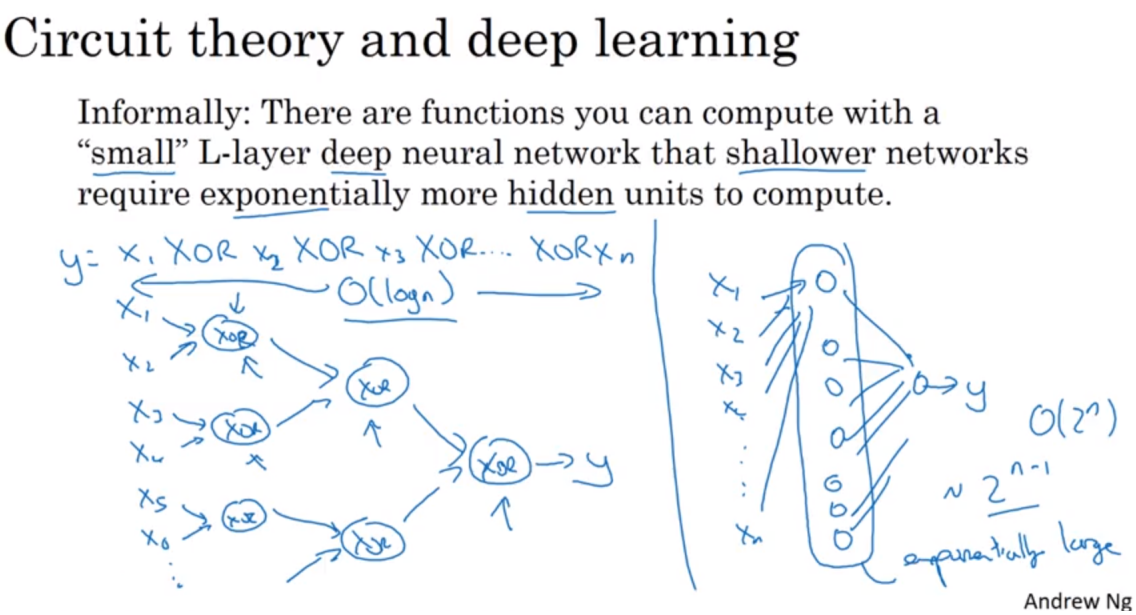
  - in vectorized form (pay attention to derivative's dimension)



- intuition about why deep NN has such amazing power
  - summary: **each units(low level) 👉(composing)👉each layer(high level) small to large**
    - e.g. CNN



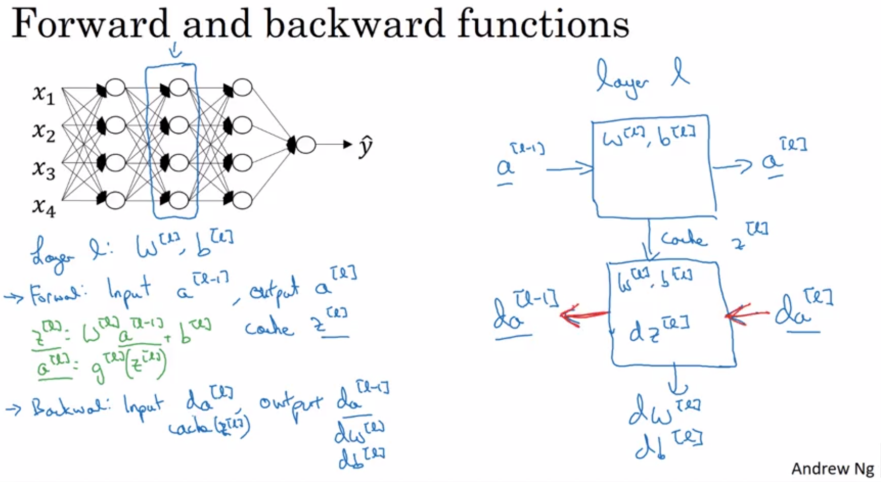
- e.g. take Circuit as analogy
  - more layer👉need less "gates" to complete same task



  - **the number of layers could be a hyper parameters**

implement deep neural network

- basic block
  - forward propagation
  - use a **cache** to save z
  - backward propagation



- details

Forward propagation for layer l

→ Input $a^{[l-1]}$

→ Output $a^{[l]}$, cache $(z^{[l]})$

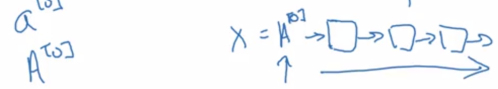
$z^{[l]} = W^{[l]} \cdot a^{[l-1]} + b^{[l]}$

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

Vectorized:

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

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



Backward propagation for layer l

→ Input $da^{[l]}$

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

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

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

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

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

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

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

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

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

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

- some hyper parameters

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

Hyperparameters: learning rate α

#iterations

#hidden layers L

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

choice of activation function

Later: Momentum, mini-batch size, regularizations, ...

- just a empirical process

Forward and backward propagation

$$
\begin{aligned}
Z^{[1]} &= W^{[1]}X + b^{[1]} \\
A^{[1]} &= g^{[1]}(Z^{[1]}) \\
Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\
A^{[2]} &= g^{[2]}(Z^{[2]}) \\
&\vdots \\
A^{[L]} &= g^{[L]}(Z^{[L]}) = \hat{Y}
\end{aligned}
$$

$$
\begin{aligned}
dZ^{[L]} &= A^{[L]} - Y \\
dW^{[L]} &= \frac{1}{m} dZ^{[L]} A^{[L]T} \\
db^{[L]} &= \frac{1}{m} np.\mathrm{sum}(\mathrm{d}Z^{[L]}, axis = 1, keepdims = True) \\
dZ^{[L-1]} &= dW^{[L]T} dZ^{[L]} g'^{[L]}(Z^{[L-1]}) \\
&\vdots \\
dZ^{[1]} &= dW^{[L]T} dZ^{[2]} g'^{[1]}(Z^{[1]}) \\
dW^{[1]} &= \frac{1}{m} dZ^{[1]} A^{[1]T} \\
db^{[1]} &= \frac{1}{m} np.\mathrm{sum}(\mathrm{d}Z^{[1]}, axis = 1, keepdims = True)
\end{aligned}
$$

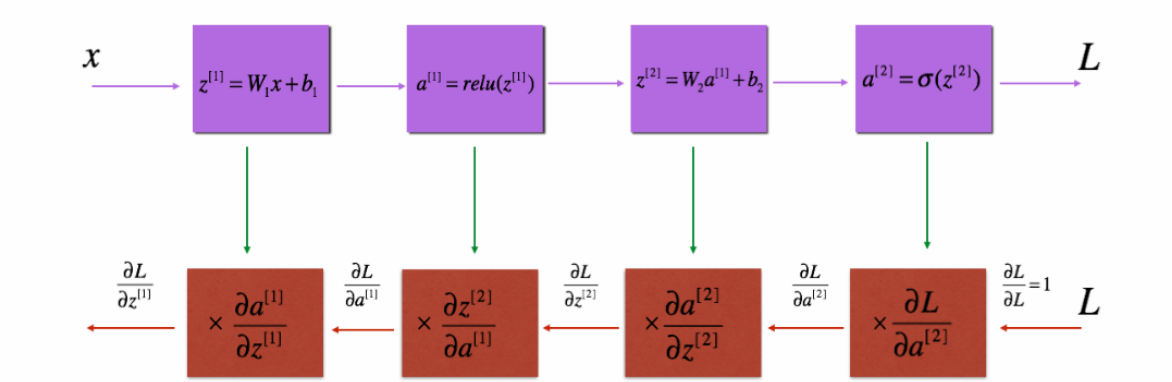


Figure 3 : Forward and Backward propagation for *LINEAR->RELU->LINEAR->SIGMOID*
The purple blocks represent the forward propagation, and the red blocks represent the backward propagation.

---

coding!!!

- use caches(datatype:dictionary)to save and pass parameters
- elegant python (programming skill)

```
# Implement [LINEAR -> RELU]*(L-1). Add "cache" to the "caches" list.
for l in range(1, L):
    A_prev = A
    ### START CODE HERE ### (≈ 2 lines of code)
    A, cache = linear_activation_forward(A_prev parameters['W'+str(l)],parameters['b'+str(l)],activation="relu")
    caches.append(cache)
    ### END CODE HERE ###
```