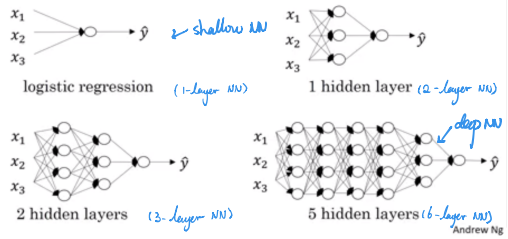
We have seen the structure and mathematical computation of shallow NN. A deep NN is quite similar to a shallow NN but just a few more hidden layers stacking up.

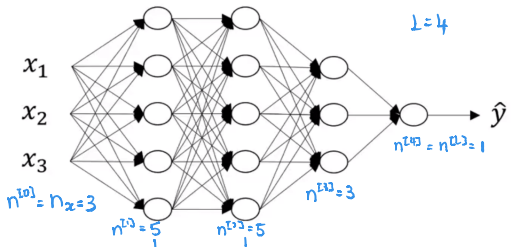
Deep neural network structure



Deep neural network can learn functions that shallow neural network can't. It can first detect simpler things in early layer (e.g. edges of images), then it can build those up to detect more complex thing in later layers (e.g. eyes, noses etc.) and eventually probe output (e.g. face). By Circuit Theory, "small" deep NN can compute functions that shallower NN requires exponentially more hidden units to compute.

We can then treat the number of layers as another hyperparameter that we can try different values of, i.e. L.

Notations:



L: number of layers

$n^{[l]}$: number of units/nodes in layer l

$a^{[l]}$: activations in layer l, $a^{[l]} = g^{[l]}(z^{[l]})$

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

$a^{[L]} = \hat{y}$

using for-loop:

$z^{[1]} = w^{[1]}x + b^{[1]}$ $z^{[2]} = w^{[2]}A^{[1]} + b^{[2]}$ ... $z^{[l]} = w^{[l]}A^{[l-1]} + b^{[l]}$

$A^{[1]} = g^{[1]}(z^{[1]})$ $A^{[2]} = g^{[2]}(z^{[2]})$ ... $A^{[l]} = g^{[l]}(z^{[l]})$ for $l = 0, 1, \dots L$ and $\hat{Y} = A^{[L]}$

matrix size:

X: $n_x \times m$	$A^{[1]}$: $n^{[1]} \times m$	$A^{[l-1]}$: $n^{[l-1]} \times m$	$A^{[L-1]}$: $n^{[L-1]} \times m$
$dw^{[1]}, w^{[1]}$: $n^{[1]} \times n_x$	$dw^{[2]}, w^{[2]}$: $n^{[2]} \times n^{[1]}$	$dw^{[l]}, w^{[l]}$: $n^{[l]} \times n^{[l-1]}$	$dw^{[L]}, w^{[L]}$: $n^{[L]} \times n^{[L-1]}$
$db^{[1]}, b^{[1]}$: $n^{[1]} \times 1$	$db^{[2]}, b^{[2]}$: $n^{[2]} \times 1$	$db^{[l]}, b^{[l]}$: $n^{[l]} \times 1$	$db^{[L]}, b^{[L]}$: $n^{[L]} \times 1$
$dz^{[1]}, z^{[1]}$: $n^{[1]} \times m$	$dz^{[2]}, z^{[2]}$: $n^{[2]} \times m$	$dz^{[l]}, z^{[l]}$: $n^{[l]} \times m$	$dz^{[L]}, z^{[L]}$: $n^{[L]} \times m$ $\Rightarrow$ $Y, A^{[L]}: n^{[L]} \times m$

Computation

initialising: parameter["w" + str(i)] = 0.01 * np.random.randn(layer[i], layer[i-1])

parameter["b" + str(i)] = 0.01 * np.random.randn(layer[i], 1)

for layer l, given $w^{[l]}$, $b^{[l]}$, $A^{[l-1]}$, for $l = 1, 2, \dots L$

forward propagation: $z^{[l]} = w^{[l]} A^{[l-1]} + b^{[l]}$

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

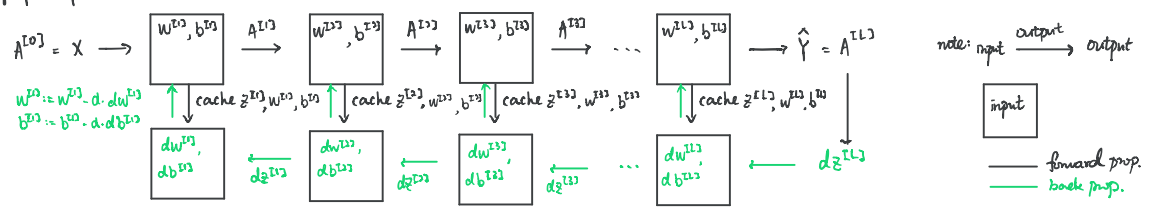
back propagation: $dz^{[l]} = dA^{[l]} * g^{[l]\prime}(z^{[l]})$ $\rightarrow$ so $dz^{[l-1]} = w^{[l]T} dz^{[l]} * g^{[l-1]\prime}(z^{[l-1]})$ and $dz^{[1]} = A^{[1]} - Y$

where $dA^{[l-1]} = w^{[l]T} dz^{[l]}$ and $dA^{[L]} = \left[-\frac{y^{(1)}}{a^{(1)}} + \frac{1-y^{(1)}}{1-a^{(1)}} \quad \dots \quad -\frac{y^{(m)}}{a^{(m)}} + \frac{1-y^{(m)}}{1-a^{(m)}} \right] = -\frac{Y}{A^{[L]}} + \frac{1-Y}{1-A^{[L]}}$

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

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

graph representation:



Hyperparameter and parameter.

- parameters: $W^{[l]}$, $b^{[l]}$ → control the value of output.
- hyperparameters: learning rate α
  # iteration
  # layer L
  # hidden units $n^{[1]}$, $n^{[2]}$, ...
  Choice of activation functions
  momentum, mini batch size, regularization variables ...

→ control the values of parameters, need tuning to find the best set.