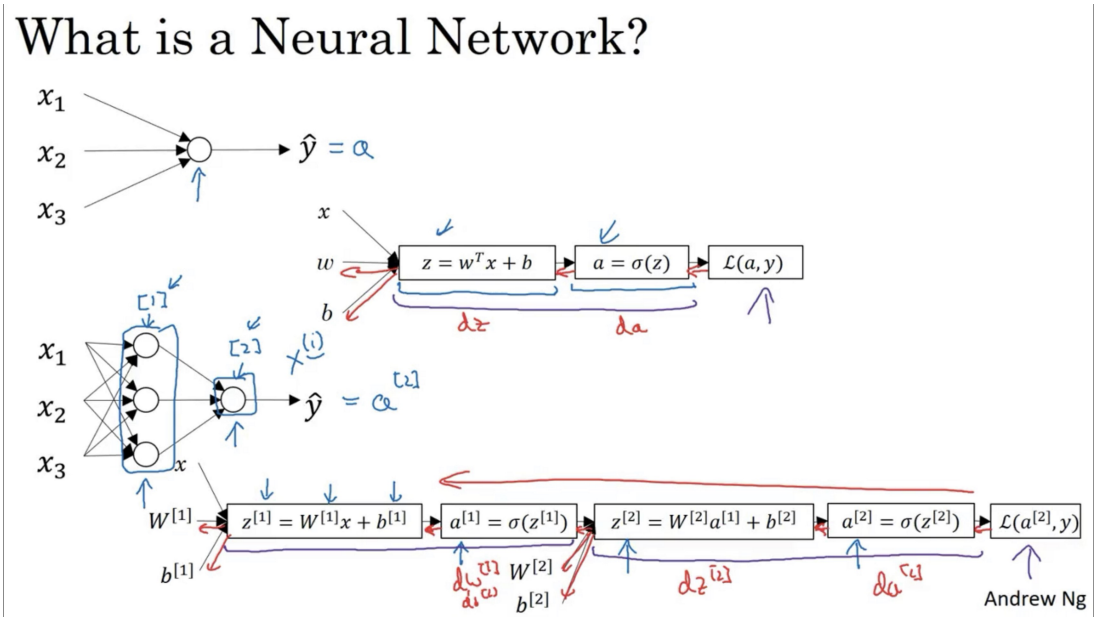
Week 3 - Shallow neural networks

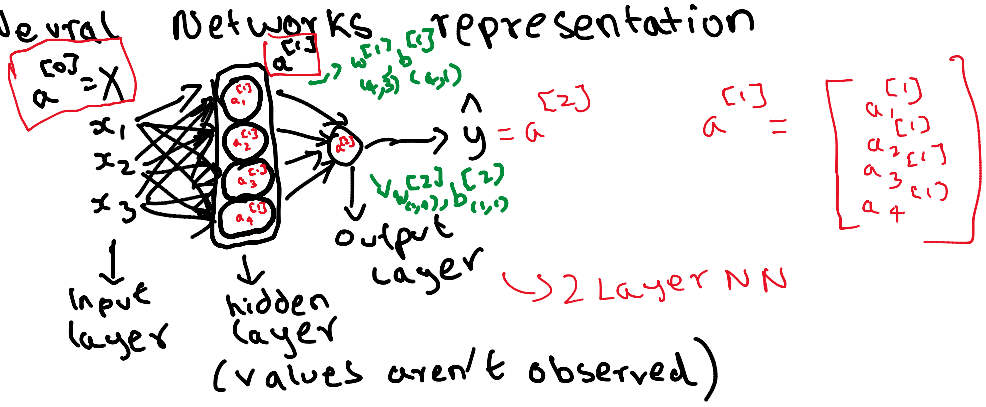
Wednesday, August 5, 2020 4:28 PM

① Neural Networks Overview

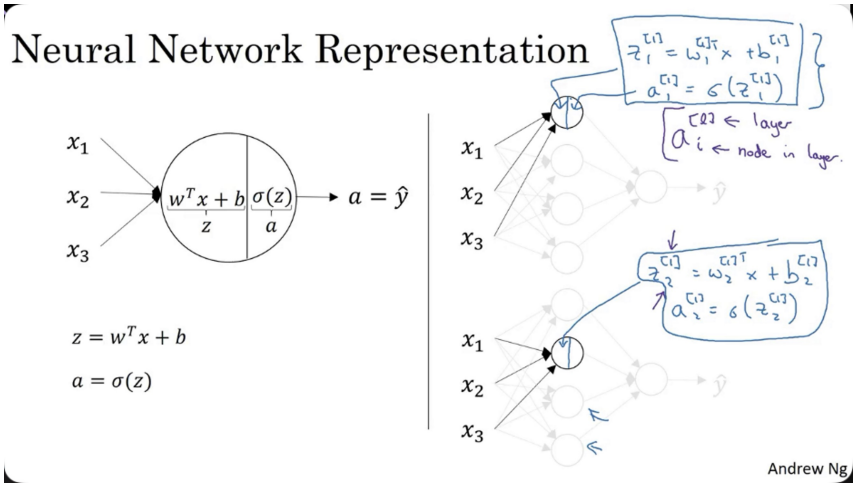
① Neural Network with hidden layer



② Neural Networks representation



③ Computing a Neural Network's output



$z^{[1]} \quad z^{[1]T} \quad a^{[1]} \quad a^{[1]} \quad a^{[2]}$

$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, \quad a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \quad a_4 = \sigma(z_4^{[1]})$$

vectorize

$$z^{[1]} = \begin{bmatrix} - w_1^{[1]T} - \\ - w_2^{[1]T} - \\ - w_3^{[1]T} - \\ - w_4^{[1]T} - \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}$$

$W^{[1]}$ $(4,3)$ $\quad b^{[1]}$ $(4,1)$

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix} = \sigma(z^{[1]})$$

- NN representation learning

Given input x:

$\rightarrow z^{[1]}_{(4,1)} = W^{[1]}_{(4,3)} a^{[0]}_{(3,1)} + b^{[1]}_{(4,1)}$ $\quad [a^{[0]} \rightarrow x]$

$\rightarrow a^{[1]} = \sigma(z^{[1]})$

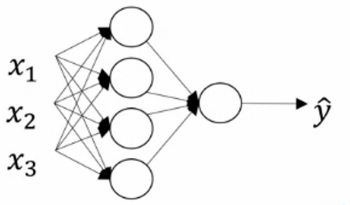
$\rightarrow z^{[2]}_{(1,1)} = W^{[2]}_{(1,4)} a^{[1]}_{(4,1)} + b^{[2]}_{(1,1)}$

$\rightarrow a^{[2]} = \sigma(z^{[2]})$

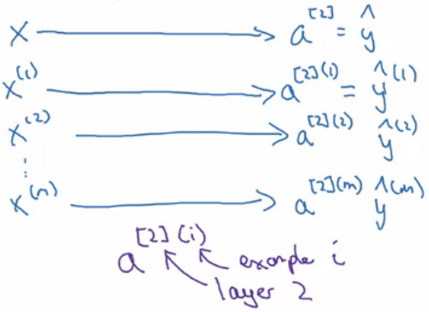
hidden layer

(4) Vectorizing across multiple examples

Vectorizing across multiple examples



$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \sigma(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma(z^{[2]})$$



for $i = 1$ to m,

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$
$$a^{[1](i)} = \sigma(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = \sigma(z^{[2](i)})$$

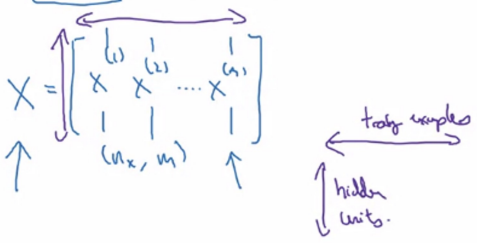
Andrew Ng

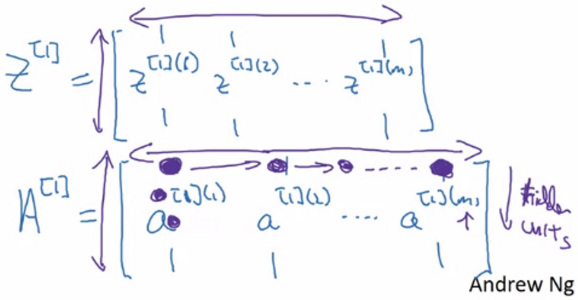
Vectorizing across multiple examples

```
for i = 1 to m:
```

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$
$$a^{[1](i)} = \sigma(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = \sigma(z^{[2](i)})$$

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$



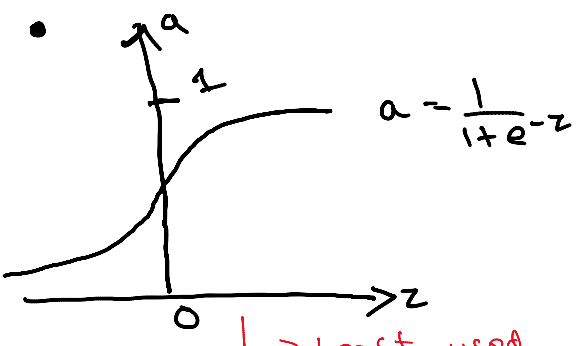


Andrew Ng

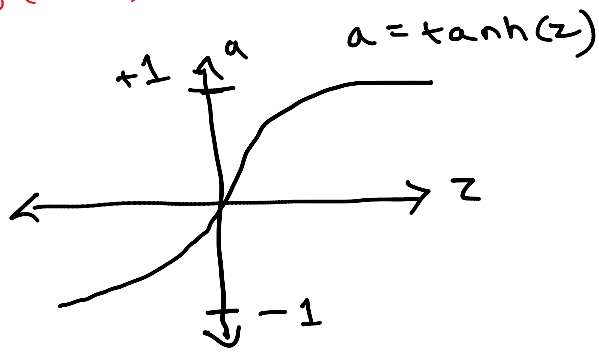
(5) Explanation for vectorised implementation

(6) Activation functions

$a = \sigma(z^{[1]}) \rightarrow g(z^{[1]})$

- 

↳ Least used



$a = \tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$

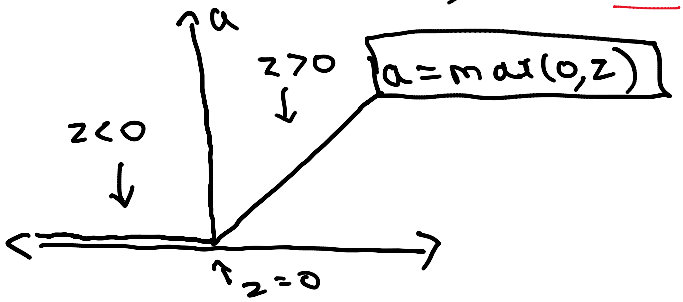
↳ works better for every layer except output layer/binary

→ Downside of sigmoid & tanh is if z is very small or very large, it slows down

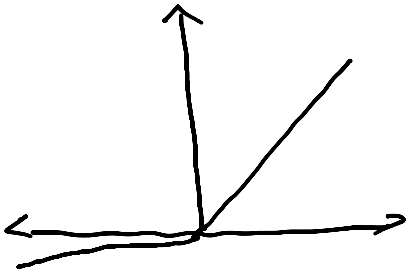
tanh is if z is very small or very large, it slows down gradient descent

for every layer except output layer/binary classifi.

- ReLU (Rectified Linear Unit)
  $a = \max(0, z)$ → Default/most used



- Leaky ReLU



(7) Need For activation functions

- if $g(z) = z$ → "Linear activation function" (or) "identity function"

$$\Rightarrow a^{[1]} = z^{[1]} = w^{[1]}x + b^{[1]}$$

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

$$\Rightarrow a^{[2]} = w^{[2]}(w^{[1]}x + b^{[1]}) + b^{[2]}$$

$$\Rightarrow \underbrace{(w^{[2]}w^{[1]})}_{w'}x + \underbrace{(w^{[2]}b^{[1]} + b^{[2]})}_{b'}$$

$$\Rightarrow w'x + b' \rightarrow \text{Linear function}$$

output isn't interesting or useful

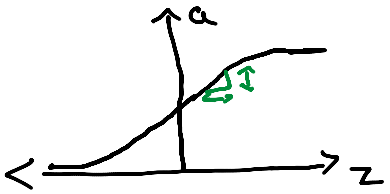
- Linear activation functions are useful only in special cases like house price prediction

(8) Derivatives of activation functions

- Sigmoid

↑a

- Sigmoid



$$g(z) = \frac{1}{1+e^{-z}}$$

$$\frac{d}{dz} g(z) = \text{slope of } g(z) \text{ at } z$$

$$= \left(\frac{1}{1+e^{-z}}\right)\left(1 - \frac{1}{1+e^{-z}}\right)$$

$$= g(z)\,(1-g(z))$$

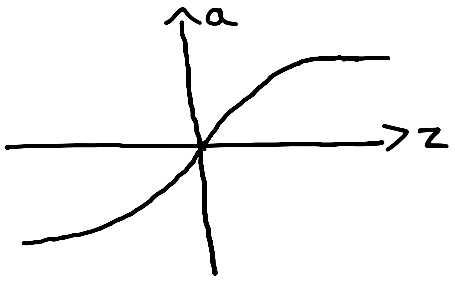
$$\boxed{g'(z) = a\,(1-a)}$$

if z >> 0: $g(z) \approx 1$, $g'(z) \approx 0$

if $z << 0$: $g(z) \approx 0$, $g'(z) \approx 0$

if $z = 0$: $g(z) = 1/2$, $g'(z) = 1/4$

- tanh



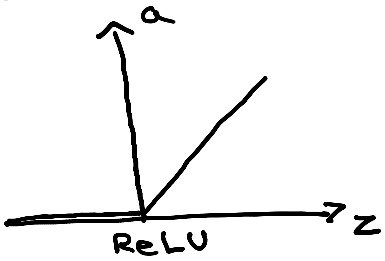
$$g(z) = \tanh(z)$$

$$= \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = \frac{d}{dz} g(z) = 1 - (\tanh z)^2$$

$$\boxed{g'(z) = 1 - a^2}$$

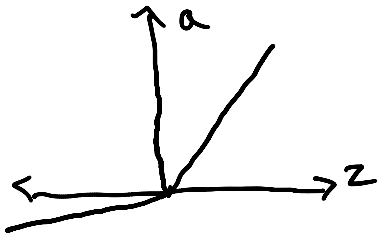
- ReLU



$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 0 & z < 0 \\ 1 & z \geq 0 \\ \text{undefined} & z = 0 \end{cases}$$

- Leaky ReLU



$$g(z) = \max(0.01z, z)$$

$$g'(z) = \begin{cases} 0.01 & z < 0 \\ 1 & z \geq 0 \end{cases}$$

(9) Gradient descent for Neural Networks (1 hidden layer)

- Parameters: $w^{[1]}$, $b^{[1]}$, $w^{[2]}$, $b^{[2]}$

$n_x = n^{[0]}$ (input features), $n^{[1]}$ (hidden unit), $n^{[2]}$ (output feature) (this case $n^{[2]} = 1$)

$\therefore$ $w^{[1]}_{(n^{[1]}, n^{[0]})}$, $b^{[1]}_{(n^{[1]}, 1)}$, $w^{[2]}_{(n^{[2]}, n^{[1]})}$, $b^{[2]}_{(n^{[2]}, 1)}$

... feature ... = 1 ...

$\therefore$ $w^{[1]}_{(n^{[1]}, n^{[0]})}$, $b^{[1]}_{(n^{[1]}, 1)}$, $w^{[2]}_{(n^{[2]}, n^{[1]})}$, $b^{[2]}_{(n^{[2]}, 0)}$

- Cost function:

$$J(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}) = \frac{1}{m}\sum_{i=1}^{n} \mathcal{L}(\hat{y}, y)$$

($\hat{y} = a^{[2]}$)

- Gradient descent

Repeat {

compute prediction ($\hat{y}^{(i)}$, i-1..,m)

$dw^{[1]} = \frac{dJ}{dw^{[1]}}$, $db^{[1]} = \frac{dJ}{db^{[1]}}$, ...

$w^{[1]} = w^{[1]} - \alpha\, dw^{[1]}$

$b^{[1]} = b^{[1]} - \alpha\, db^{[1]}$

$w^{[2]} = \ldots$

$b^{[2]} = \ldots$ }

- Formulas for computing derivatives

i. Forward propagation

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

$$A^{[1]} = g^{[1]}(z^{[1]})$$

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

$$A^{[2]} = g^{[2]}(z^{[2]}) = \sigma(z^{[2]})$$

ii. Back propagation

$$dz^{[2]} = A^{[2]} - y$$

$$dw^{[2]} = 1/m\; dz^{[2]} A^{[1]T}$$

$$db^{[2]} = \frac{1}{m}\, np.sum(dz^{[2]}, axis=1, keepdims=True)$$

(keepdims=True: to output $(n,1)$ instead of $(n,)$)

$$dz^{[1]} = w^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$$

($w^{[2]T} dz^{[2]}$: $(n^{[1]}, m)$; $g^{[1]\prime}(z^{[1]})$: $(n^{[1]}, m)$; * : element-wise product)

$$dw^{[1]} = \frac{1}{m}\, dz^{[1]} x^{T}$$

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

(10) Backpropagation intuition

11 Random Initialisation

- Random initialisation

$$W^{[1]} = np.random.randn((2,2)) * 0.01$$

small values preferred to avoid saturation of function

$$b^{[1]} = np.zeros((2,1))$$

$w^{[1]} = np.random.randn((2,2)) * 0.01$ to avoid saturation of function

$b^{[1]} = np.zeros((2,1))$

↳ initialising b to zero doesn't affect learning

$w^{[2]} = \ldots$

$b^{[2]} = 0$