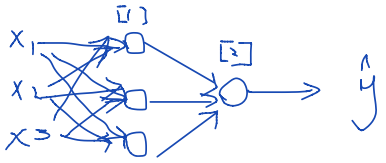
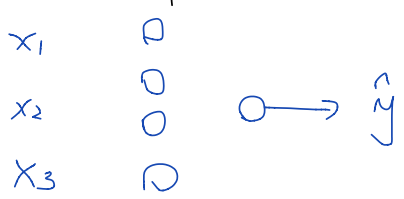
## Neural Network overview

### What is a Neural Network




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

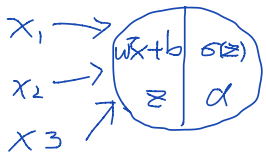
## Neural Network Representation




2 layer Neural Network
(don't count input layer as layer)

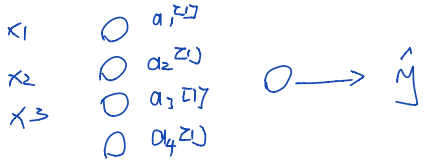
Input layer Hidden layer output layer

## Computing a Neural Networks output




← for a single neural

Vectorize.



$$z^{[1]} = \begin{bmatrix} \text{---} w_1^{[1]T} \text{---} \\ \text{---} w_2^{[1]T} \text{---} \\ \text{---} w_3^{[1]T} \text{---} \\ \text{---} w_4^{[1]T} \text{---} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ \vdots \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ \vdots \\ z_4^{[1]} \end{bmatrix}$$

$W^{[1]}$ $\quad$ $b^{[1]}$ $\quad$ $z^{[1]}$

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

- Vectorize across multiple examples

m training examples.

$$x \rightarrow a^{[2]} = \hat{y}$$

$$x^{(1)} \rightarrow a^{[2](1)} = \hat{y}^{(1)}$$

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

$$\vdots \qquad \vdots \qquad \vdots$$

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

for $i = 1$ to $m$:

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

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

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

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

How to vectorize above calculation and get rid of for loop?

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

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

row: different features/hidden neurons

column: different training examples

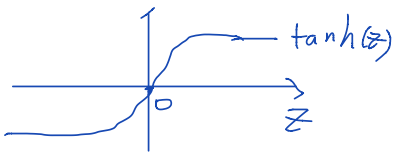
- Activation functions

Sigmoid function $\sigma$

more generally, the activation function can be $g$

A better choice of activation function: $g(z) = \tanh(z)$

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

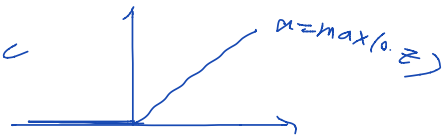



Advantage: compare to sigmoid func, the $\tanh(z)$ gives out put centered around average of 0

[*] Most of times in neural network $\tanh(z)$ is used instead of sigmoid func, but one exception is there when working on a classification problem, the output has to be $\hat{y} \in \{0, 1\}$, in this situation the output layer uses sigmoid func as the activation function.

ReLU function: Rectified Linear Unit $a = \max(0, z)$

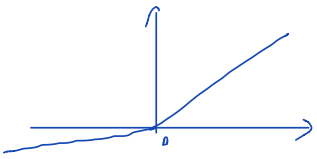
sigmoid func or tanh func slows down gradient descent




General Rule:

- Different layer can have different activation func
- Don't use sigmoid func as activation func unless for the output layer if binary classification
- Most of times people uses ReLU func

Leak ReLU




- Why do Neural Network need non-linear activation function ?

If assume $g(z) = z$ "linear activation function"

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

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

$= w'x + b'$

It's just doing linear regression, hidden unit does nothing!

☒ The only possible case of linear activation function is: all hidden layers use non-linear activation func (e.g. Relu, tanh...) but the output layer can use linear function (e.g. for linear regression as the last step)

- Derivatives of activation functions

  - Sigmoid activation function

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

- tanh activation function

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

- ReLU activation function

$$g(z) = \max(0, z) \qquad g'(z) = \begin{cases} 0 & \text{if } z<0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

- Leaky ReLU Activation function

$$g(z) = \max(0.01z, z) \qquad g'(z) = \begin{cases} 0.01, & \text{if } z<0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

# Gradient Descent for Neural Network: back propagation

for single hidden Unites

Parameters:

- Num of units in each layer: $n_x = n^{[0]}$, $n^{[1]}$, $n^{[2]}$. ($n^{[2]}=1$ for single hidden units)
- $W^{[1]}$: $n^{[1]} \times n^{[0]}$

  $b^{[1]}$: $n^{[1]} \times 1$

  $W^{[2]}$: $n^{[2]} \times n^{[1]}$

  $b^{[2]}$: $n^{[2]} \times 1$
- Cost function:

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

- Gradient descent

  randomly initialize $W$s, $b$s (non-zero)

  Repeat {

  Compute predictions ($\hat{y}^{(i)}$, $i=1,\dots m$)

  $dW^{[1]} = \frac{\partial J}{\partial W^{[1]}}$, $db^{[1]} = \frac{\partial J}{\partial b^{[1]}}$

  $W^{[1]} = W^{[1]} - \alpha dW^{[1]}$

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

  }

- Formulas for Computing derivatives

  Forward Propagation

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

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

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

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

Back propagation

"$dz^{[2]}$" $= A^{[2]} - Y$ (when $g$ = sigmoid(z))

"$dW^{[2]}$" $= \frac{1}{m}$ "$dz^{[2]}$" $A^{[1]T}$ (← broad casting)

"$db^{[2]}$" $= \frac{1}{m}$ np.Sum("$dz^{[2]}$", axis = 1, keepdims = True) ← $n^{[2]} \times 1$

"$dz^{[1]}$" $= W^{[2]T}$ "$dz^{[2]}$" $* g^{[1]'}(z^{[1]})$

↑ element-wise product

"$dW^{[1]}$" $= \frac{1}{m}$ "$dz^{[1]}$" $X^T$ (broadcasting)

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

- Random Initialization

- what happens if initialize all weights to 0?

  ~~Different~~ hidden Unit are identical.

- Have to initialize parameters randomly, to a SMALL value.

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

  why small: $z = W^T X + b$ $a = g(z)$

  if W to large, gradient will be small, especially for Sigmoid functions.