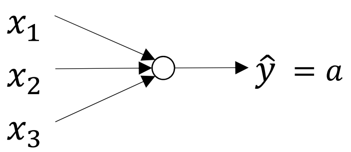
# Shallow Neural Network

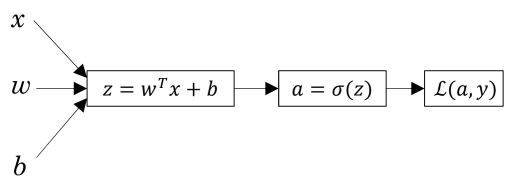
Most of this material is from Prof. Andrew Ng'and Chang's slides.

# What is a Neural Network?

- Logistic Regression

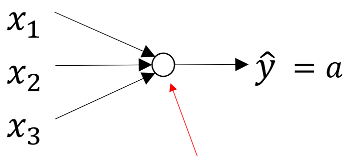



  - Computation Graph

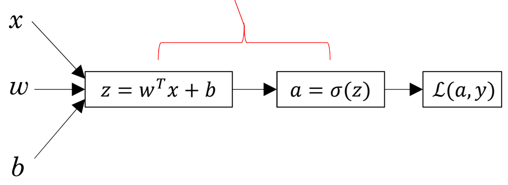


# What is a Neural Network?

- Logistic Regression

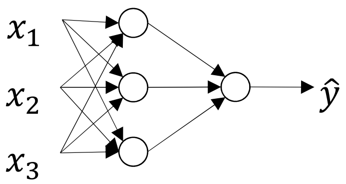



  - Computation Graph

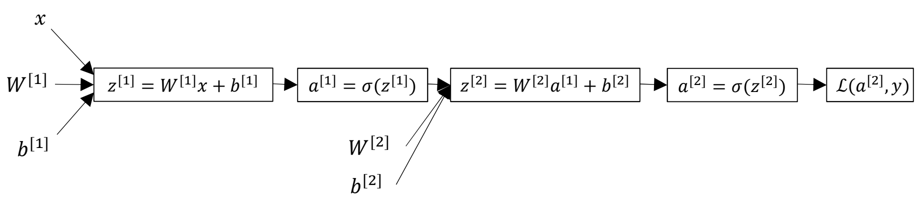



# What is a Neural Network?

- Neural Network

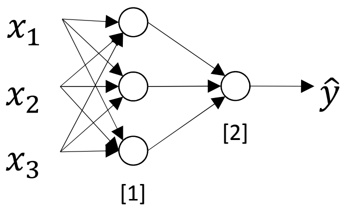



  - Computation Graph



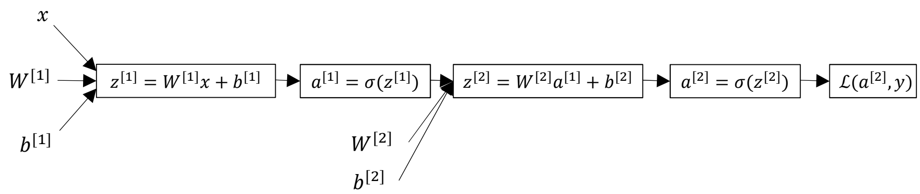
# What is a Neural Network?

- Neural Network



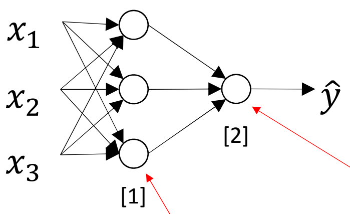

[i]: represent i-th layer
(i): represent i-th example

  - Computation Graph



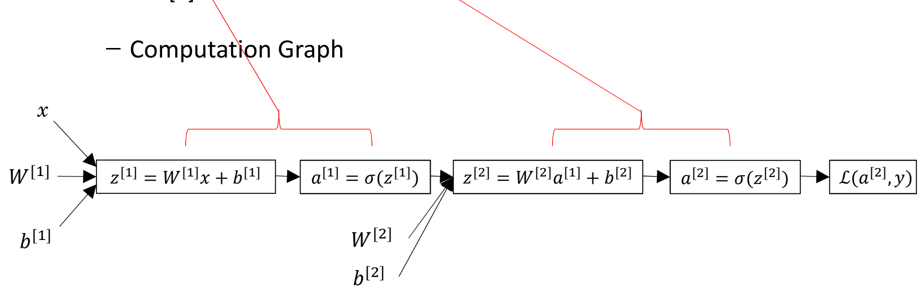

# What is a Neural Network?

- Neural Network



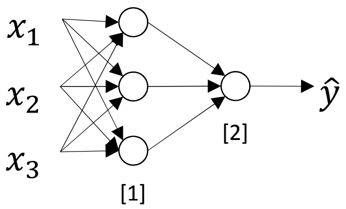

[i]: represent i-th layer
(i): represent i-th example

  - Computation Graph



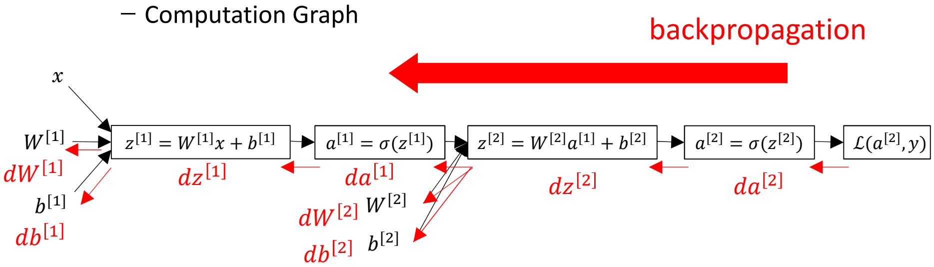
# What is a Neural Network?

- Neural Network

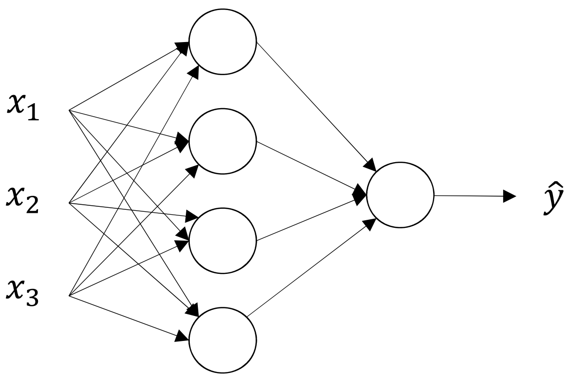



[i]: represent i-th layer
(i): represent i-th example

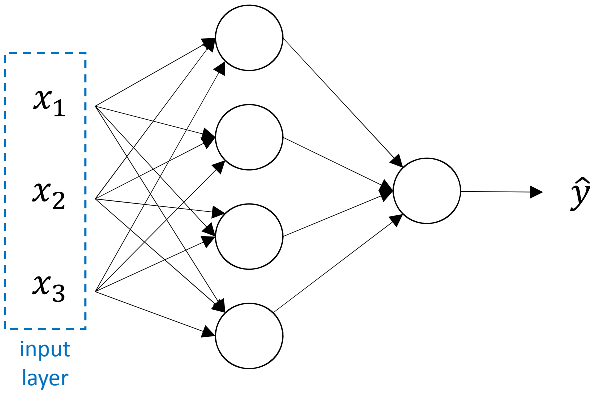
  - Computation Graph




# Neural Network Representation

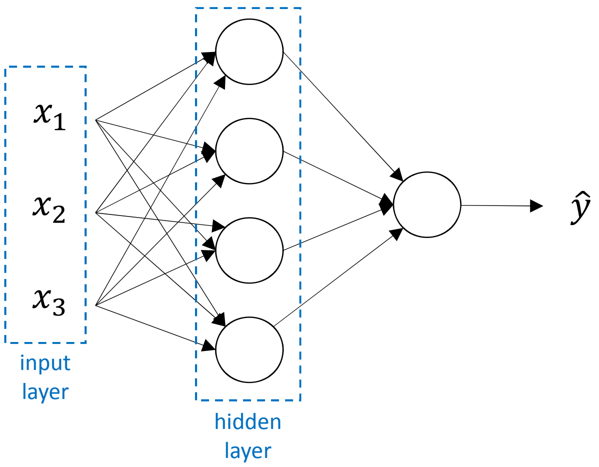


# Neural Network Representation



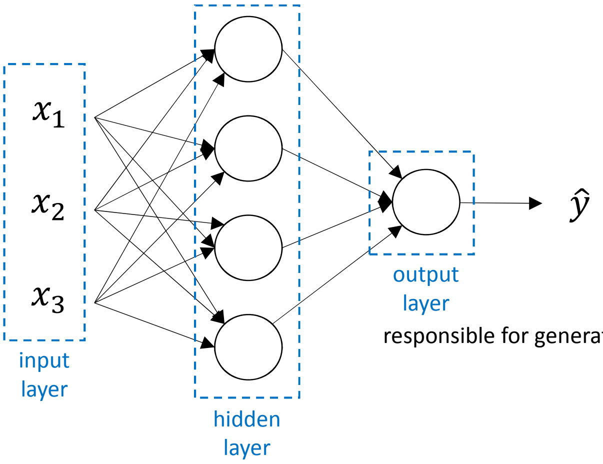

contains the inputs to the neural entwork

# Neural Network Representation



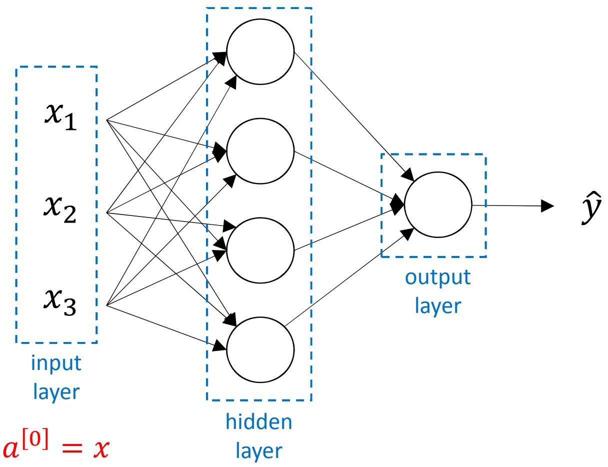

in supervised learning, a training set contains inputs as well as outputs
but, the true values for middle nodes are not observed (hidden!!)

## Neural Network Representation




responsible for generating the predicted value $\hat{y}$

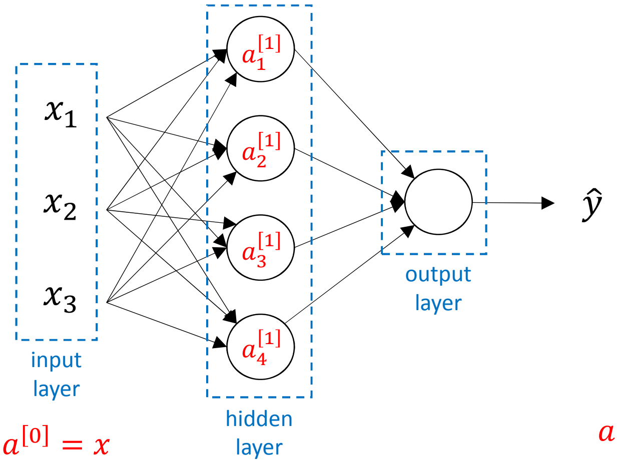
## Neural Network Representation




$a^{[0]} = x$

Notations

# Neural Network Representation



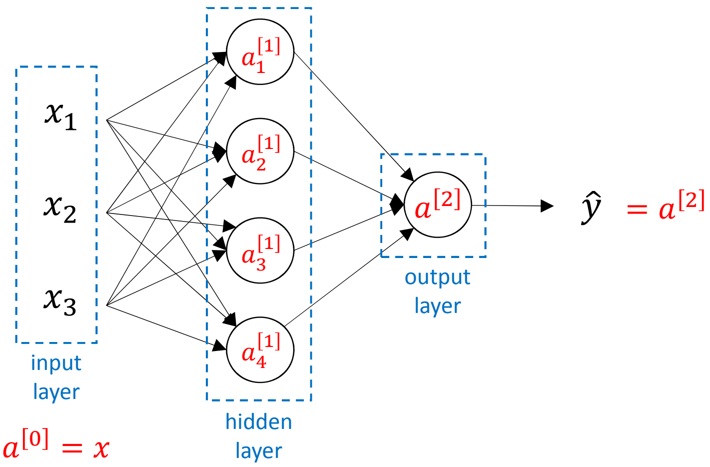

$a^{[0]} = x$

$a^{[1]}$

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

Notations

# Neural Network Representation

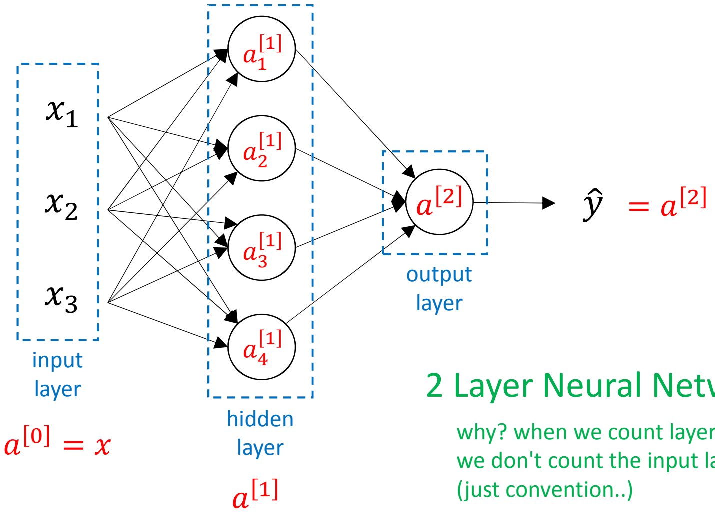



$a^{[0]} = x$

$a^{[1]}$

Notations

# Neural Network Representation



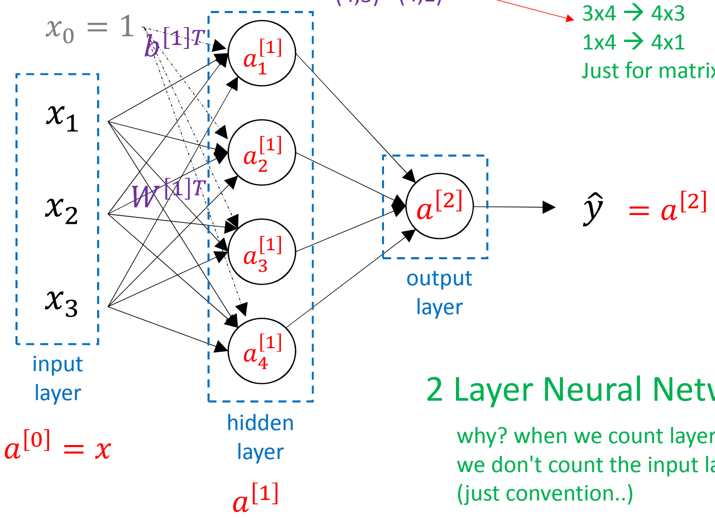

2 Layer Neural Network

why? when we count layers,
we don't count the input layer
(just convention..)

# Neural Network Representation

Parameters $\underbrace{W^{[1]T}}_{(4,3)}, \underbrace{b^{[1]T}}_{(4,1)}$

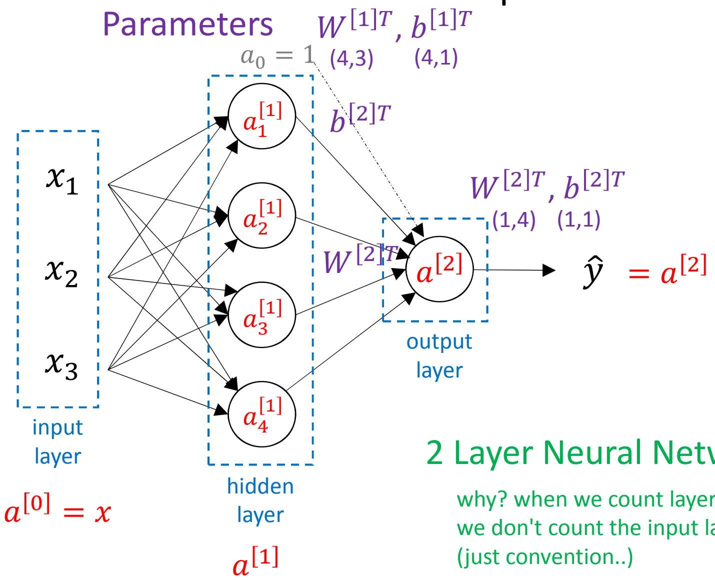
Transpose why?
3x4 → 4x3
1x4 → 4x1
Just for matrix multiplication!




2 Layer Neural Network

why? when we count layers,
we don't count the input layer
(just convention..)

# Neural Network Representation



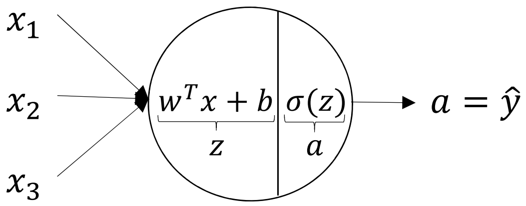

2 Layer Neural Network

why? when we count layers,
we don't count the input layer
(just convention..)

# Computing NN's Output

- Let's see details of exactly how NN computes outputs

- Logistic Regression

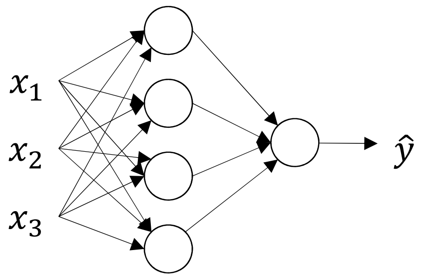



$z = w^T x + b$

$a = \sigma(z)$

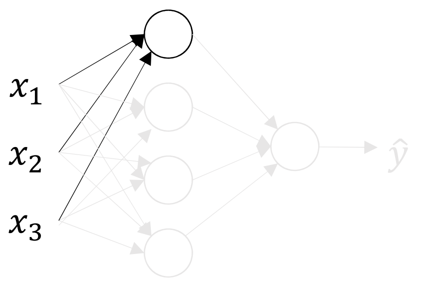
## Computing NN's Output

- Let's see details of exactly how NN computes outputs
- Neural Network



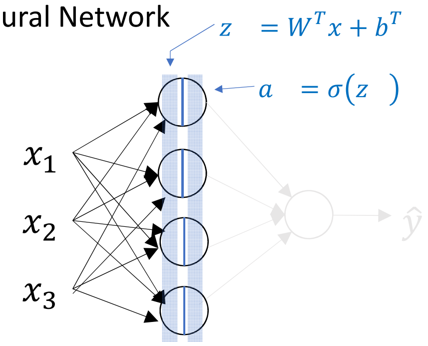

## Computing NN's Output

- Let's see details of exactly how NN computes outputs
- Neural Network



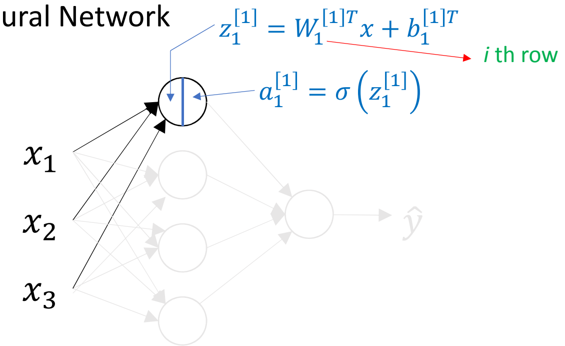
# Computing NN's Output

- Let's see details of exactly how NN computes outputs
- Neural Network




# Computing NN's Output

- Let's see details of exactly how NN computes outputs
- Neural Network



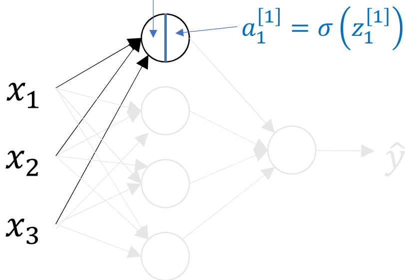
# Computing NN's Output

- Let's see details of exactly how NN computes outputs

- Neural Network

$z_1^{[1]} = W_1^{[1]T} x + b_1^{[1]T}$

$a_1^{[1]} = \sigma\left(z_1^{[1]}\right)$




layer

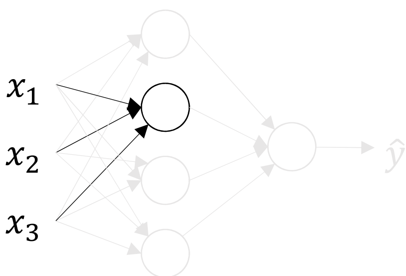
$a_i^{[l]}$

node in layer

# Computing NN's Output

- Let's see details of exactly how NN computes outputs

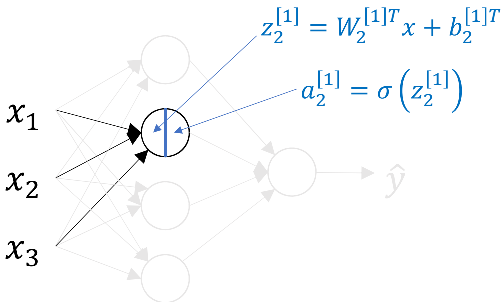
- Neural Network



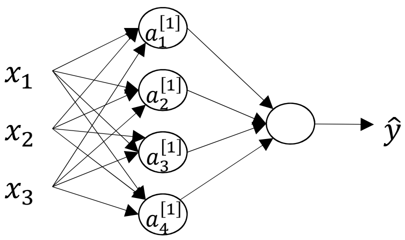
## Computing NN's Output

- Let's see details of exactly how NN computes outputs

- Neural Network

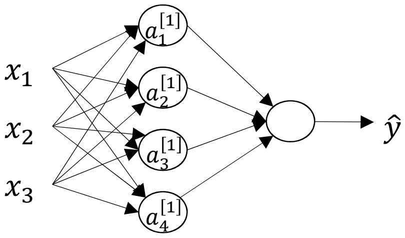



## Computing NN's Output



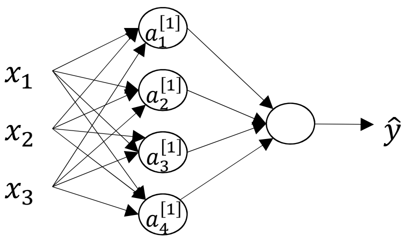

$$z_1^{[1]} = w_1^{[1]T}\, x + b_1^{[1]T},\ a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T}\, x + b_2^{[1]T},\ a_2^{[1]} = \sigma(z_2^{[1]})$$

# Computing NN's Output




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

# Computing NN's Output

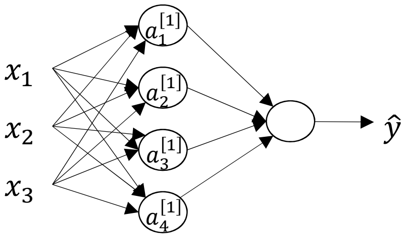



this is a vector

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

for-loop is inefficient

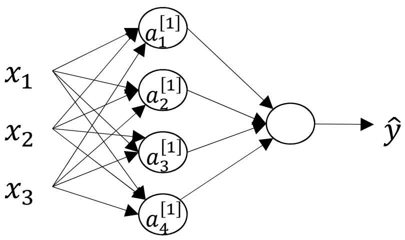
# Computing NN's Output




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

$$\underbrace{\overbrace{\begin{bmatrix} w_1^{[1]T} \\ w_2^{[1]T} \\ w_3^{[1]T} \\ w_4^{[1]T} \end{bmatrix}}^{W^{[1]}}}_{(4,3)} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} w_1^{[1]T}x \\ w_2^{[1]T}x \\ w_3^{[1]T}x \\ w_4^{[1]T}x \end{bmatrix}$$

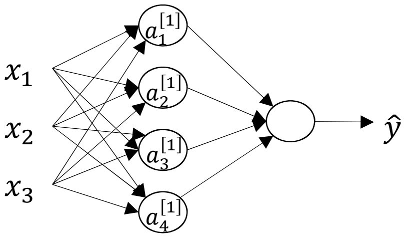
# Computing NN's Output




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

$$\underbrace{\overbrace{\begin{bmatrix} w_1^{[1]T} \\ w_2^{[1]T} \\ w_3^{[1]T} \\ w_4^{[1]T} \end{bmatrix}}^{W^{[1]}}}_{(4,3)} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \underbrace{\overbrace{\begin{bmatrix} b_1^{[1]T} \\ b_2^{[1]T} \\ b_3^{[1]T} \\ b_4^{[1]T} \end{bmatrix}}^{b^{[1]}}}_{(4,1)} = \begin{bmatrix} w_1^{[1]T}x + b_1^{[1]T} \\ w_2^{[1]T}x + b_2^{[1]T} \\ w_3^{[1]T}x + b_3^{[1]T} \\ w_4^{[1]T}x + b_4^{[1]T} \end{bmatrix}$$

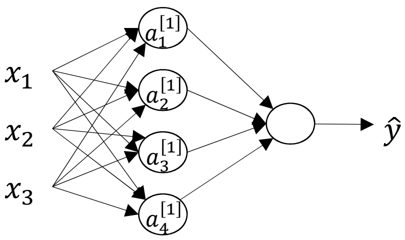
# Computing NN's Output




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

$$z^{[1]} = \underbrace{\overbrace{\begin{bmatrix} w_1^{[1]T} \\ w_2^{[1]T} \\ w_3^{[1]T} \\ w_4^{[1]T} \end{bmatrix}}^{W^{[1]}}}_{(4,3)} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \underbrace{\overbrace{\begin{bmatrix} b_1^{[1]T} \\ b_2^{[1]T} \\ b_3^{[1]T} \\ b_4^{[1]T} \end{bmatrix}}^{b^{[1]}}}_{(4,1)} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]T} \\ w_2^{[1]T} x + b_2^{[1]T} \\ w_3^{[1]T} x + b_3^{[1]T} \\ w_4^{[1]T} x + b_4^{[1]T} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

# Computing NN's Output



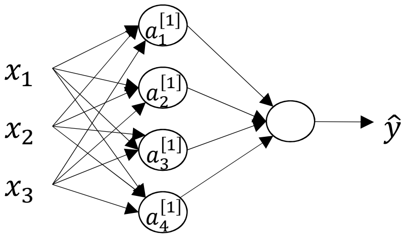

$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]T}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]T}, \quad a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]T}, \quad a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]T}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$$

Notice: No more transpose Notation !

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

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

# Computing NN's Output




Given input x:

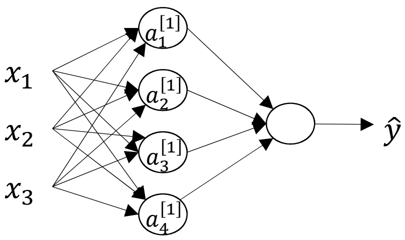
$$z^{[1]} = W^{[1]}x + b^{[1]}$$

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

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

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

# Computing NN's Output




Given input x:

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

(4,1) (4,3) (3,1) (4,1)

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

(4,1) (4,1)

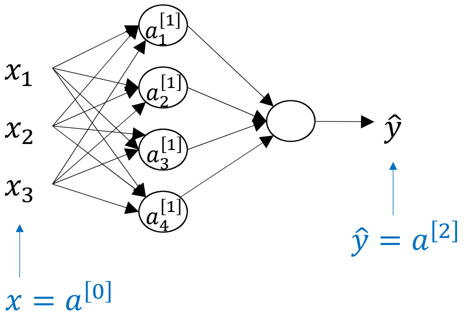
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

(1,1) (1,4) (4,1) (1,1)

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

(1,1) (1,1)

# Computing NN's Output




Given input x:

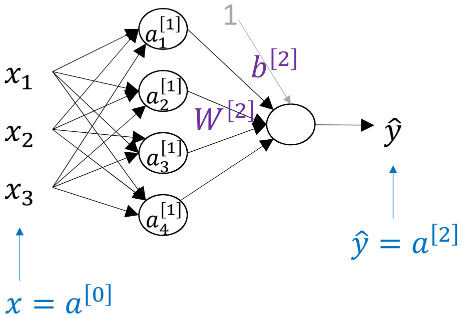
$$\underset{(4,1)}{z^{[1]}} = \underset{(4,3)}{W^{[1]}}\underset{(3,1)}{x} + \underset{(4,1)}{b^{[1]}}$$

$$\underset{(4,1)}{a^{[1]}} = \sigma(\underset{(4,1)}{z^{[1]}})$$

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

$$\underset{(1,1)}{a^{[2]}} = \sigma(\underset{(1,1)}{z^{[2]}})$$

# Computing NN's Output




Given input x:

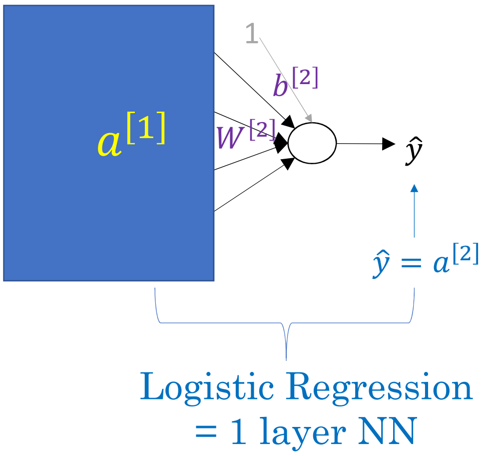
$$\underset{(4,1)}{z^{[1]}} = \underset{(4,3)}{W^{[1]}}\underset{(3,1)}{x} + \underset{(4,1)}{b^{[1]}}$$

$$\underset{(4,1)}{a^{[1]}} = \sigma(\underset{(4,1)}{z^{[1]}})$$

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

$$\underset{(1,1)}{a^{[2]}} = \sigma(\underset{(1,1)}{z^{[2]}})$$

# Computing NN's Output




Given input x:

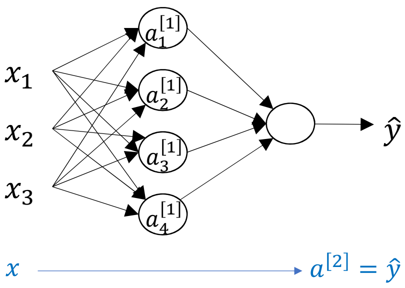
$$\underset{(4,1)}{z^{[1]}} = \underset{(4,3)}{W^{[1]}}\underset{(3,1)}{x} + \underset{(4,1)}{b^{[1]}}$$

$$\underset{(4,1)}{a^{[1]}} = \sigma(\underset{(4,1)}{z^{[1]}})$$

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

$$\underset{(1,1)}{a^{[2]}} = \sigma(\underset{(1,1)}{z^{[2]}})$$

# Vectorizing across multiple examples



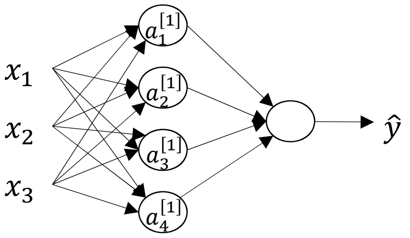

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

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

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

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

## Vectorizing across multiple examples




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

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

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

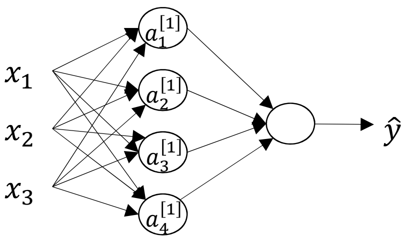
$x^{(2)} \longrightarrow a^{[2](2)} = \hat{y}^{(2)}$

$\vdots \qquad \vdots$

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

number of training data

## Vectorizing across multiple examples




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

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

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

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

$\vdots \qquad \vdots$

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

non-vectorized implementation:

for i = 1 to m,

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

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

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

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

## Vectorizing across multiple examples

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

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

## Vectorizing across multiple examples

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

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

$$\underset{(n,m)}{\mathrm{X}} = \begin{bmatrix} x^{(1)} & x^{(2)} & \cdots & x^{(m)} \end{bmatrix}$$

$$Z^{[1]} = \begin{bmatrix} z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \end{bmatrix}$$

$$\vdots$$

## Vectorizing across multiple examples

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

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

$$\underset{(n,m)}{\mathrm{X}} = \begin{bmatrix} x^{(1)} & x^{(2)} & \cdots & x^{(m)} \end{bmatrix}$$

$$Z^{[1]} = \begin{bmatrix} z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \end{bmatrix}$$

$$\vdots$$

training examples

(hidden) units


## Vectorizing across multiple examples

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

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


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


$$\underset{(n,m)}{\mathrm{X}} = \begin{bmatrix} x^{(1)} & x^{(2)} & \cdots & x^{(m)} \end{bmatrix}$$

$$Z^{[1]} = \begin{bmatrix} z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \end{bmatrix}$$

$$\vdots$$

training examples

(hidden) units

# Vectorizing across multiple examples

$$z^{[1](1)} = W^{[1]}x^{(1)} + b^{[1]} \qquad z^{[1](2)} = W^{[1]}x^{(2)} + b^{[1]} \qquad z^{[1](3)} = W^{[1]}x^{(3)} + b^{[1]} \quad \cdots$$

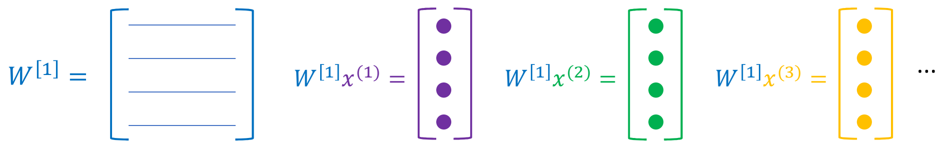
# Vectorizing across multiple examples

$$z^{[1](1)} = W^{[1]}x^{(1)} + \cancel{b^{[1]}} \qquad z^{[1](2)} = W^{[1]}x^{(2)} + \cancel{b^{[1]}} \qquad z^{[1](3)} = W^{[1]}x^{(3)} + \cancel{b^{[1]}} \quad \cdots$$

$$W^{[1]} = \begin{bmatrix} \text{———————} \\ \text{———————} \\ \text{———————} \\ \text{———————} \end{bmatrix}$$

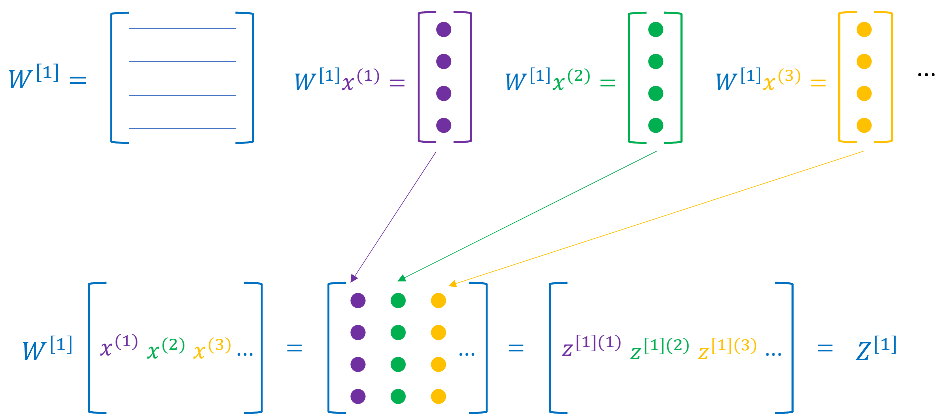
# Vectorizing across multiple examples

$$z^{[1](1)} = W^{[1]}x^{(1)} + \cancel{b^{[1]}} \quad z^{[1](2)} = W^{[1]}x^{(2)} + \cancel{b^{[1]}} \quad z^{[1](3)} = W^{[1]}x^{(3)} + \cancel{b^{[1]}} \quad \cdots$$



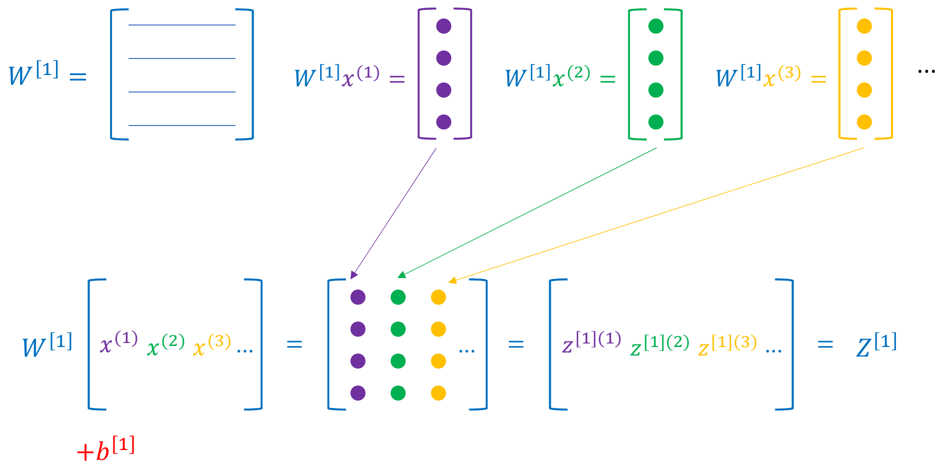

# Vectorizing across multiple examples

$$z^{[1](1)} = W^{[1]}x^{(1)} + \cancel{b^{[1]}} \quad z^{[1](2)} = W^{[1]}x^{(2)} + \cancel{b^{[1]}} \quad z^{[1](3)} = W^{[1]}x^{(3)} + \cancel{b^{[1]}} \quad \cdots$$



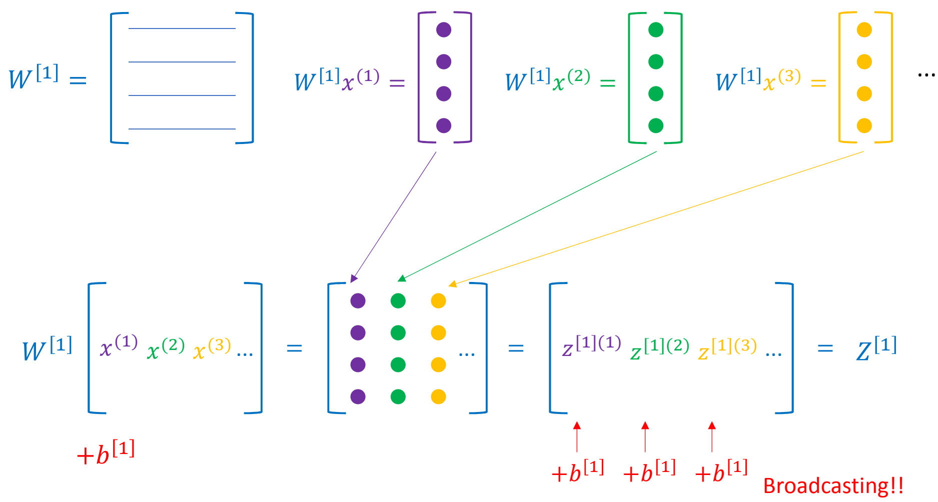
# Vectorizing across multiple examples

$z^{[1](1)} = W^{[1]}x^{(1)} + b^{[1]}$ $\quad z^{[1](2)} = W^{[1]}x^{(2)} + b^{[1]}$ $\quad z^{[1](3)} = W^{[1]}x^{(3)} + b^{[1]}$ $\quad \cdots$

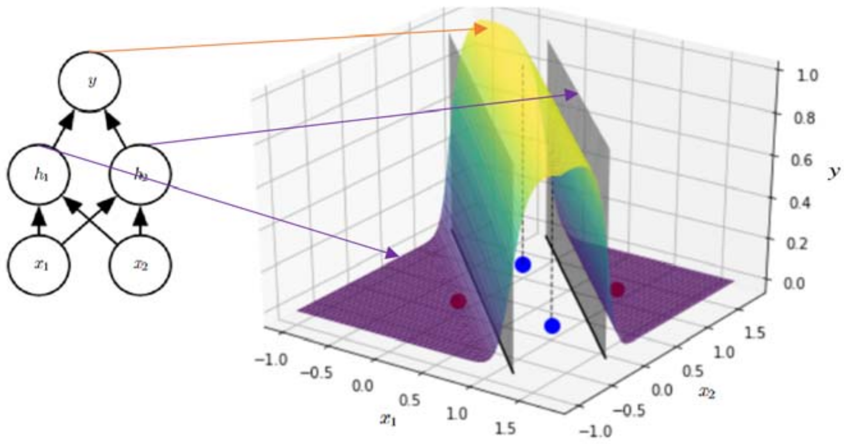



# Vectorizing across multiple examples

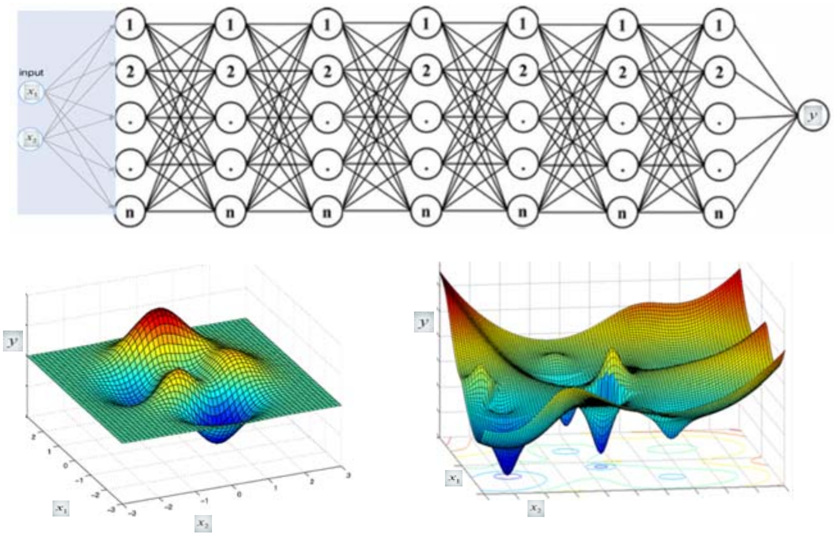
$z^{[1](1)} = W^{[1]}x^{(1)} + b^{[1]}$ $\quad z^{[1](2)} = W^{[1]}x^{(2)} + b^{[1]}$ $\quad z^{[1](3)} = W^{[1]}x^{(3)} + b^{[1]}$ $\quad \cdots$



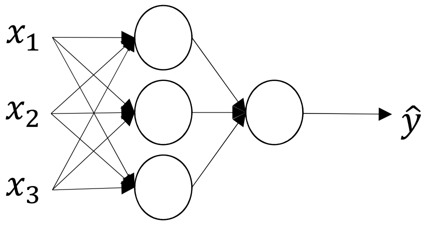
A Quick Example:
2 Layered (shallow) Neural Network by Gradient Descent




Another Example:
Deep (Layered) Neural Network by Gradient Descent



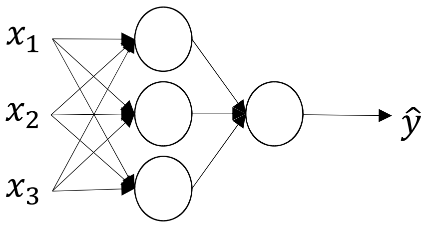
## Activation functions




Given x:

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

## Activation functions

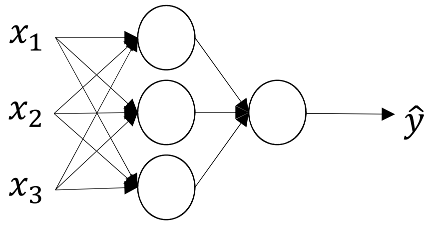


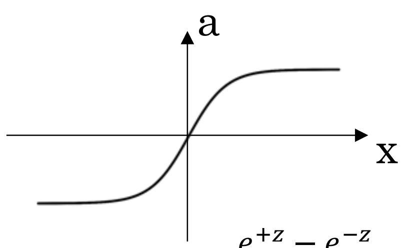

In general, sigmoid function is replaced with other non-linear activation functions

Given x:

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

# Activation functions



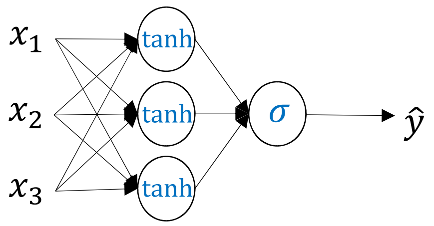



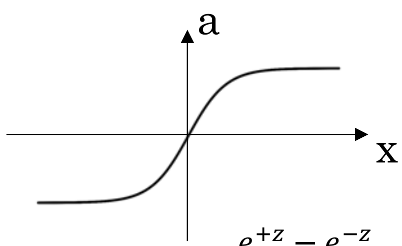

tanh: $a = \dfrac{e^{+z} - e^{-z}}{e^{+z} + e^{-z}}$

Given x:

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

# Activation functions




tanh: $a = \dfrac{e^{+z} - e^{-z}}{e^{+z} + e^{-z}}$

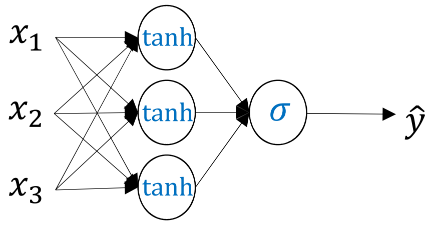
Given x:

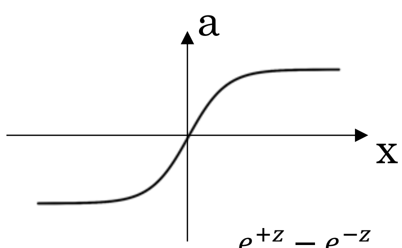
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \boxed{g^{[1]}}(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \boxed{g^{[2]}}(z^{[2]})$$

**tanh (hyperboic tangent)**

- mathematically a sifted version of the sigmoid function
- almost always works better than the sigmoid function (make learning easier)

# Activation functions




$$\text{tanh:} \quad a = \frac{e^{+z} - e^{-z}}{e^{+z} + e^{-z}}$$

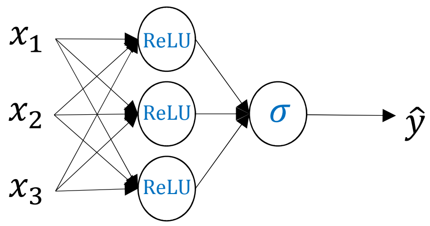
Given x:

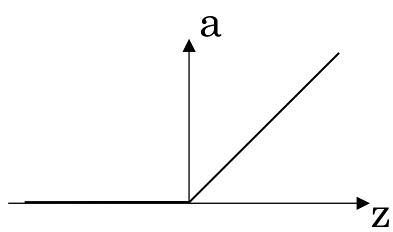
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = g^{[1]}(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = g^{[2]}(z^{[2]})$$

**tanh (hyperboic tangent)**

- mathematically a sifted version of the sigmoid function
- almost always works better than the sigmoid function (make learning easier)
- if z is either very large or very small, then the gradient (derivative) becomes very small, and this slows down gradient descent → vanishing gradient

# Activation functions




$$\text{ReLU:} \quad a = \max(0, z)$$

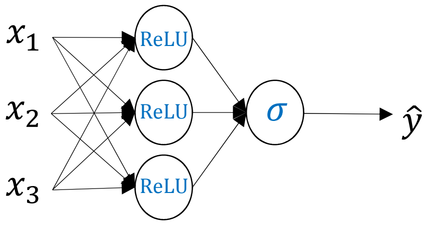
Given x:

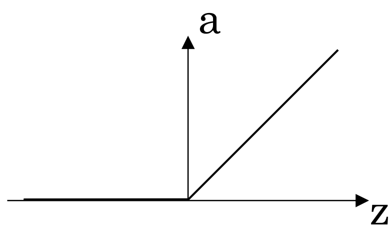
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = g^{[1]}(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = g^{[2]}(z^{[2]})$$

**ReLU (rectified linear unit)**

- solves vanishing gradient problem
- don't need to worry about derivative at origin (just select 1 or 0)

# Activation functions




ReLU: $a = \max(0, z)$

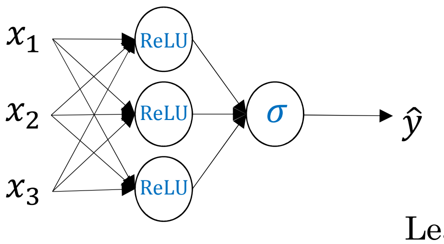
Given x:

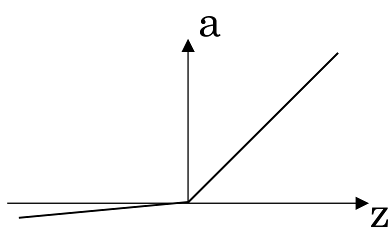
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \boxed{g^{[1]}}(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \boxed{g^{[2]}}(z^{[2]})$$

**Rule of Thumb**

- if your output is 0/1 (binary classification), then the sigmoid activation function is very natural for the output layer
- for all other units, ReLU is increasingly the default choice of activation functions

# Activation functions




Leaky ReLU: $a = \max(0.01z, z)$

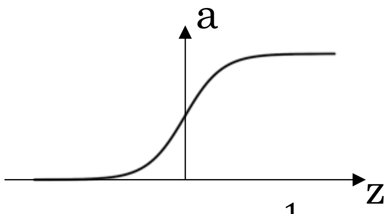
Given x:

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

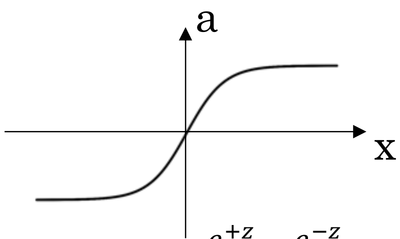
**Rule of Thumb**

- if your output is 0/1 (binary classification), then the sigmoid activation function is very natural for the output layer
- for all other units, ReLU is increasingly the default choice of activation functions
- derivative is equal to 0.01 when z is negative → leaky ReLU

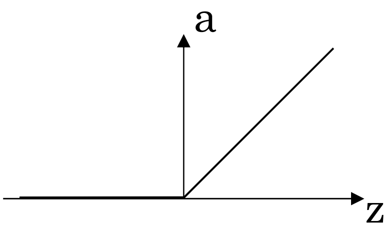
# Recap: Activation functions



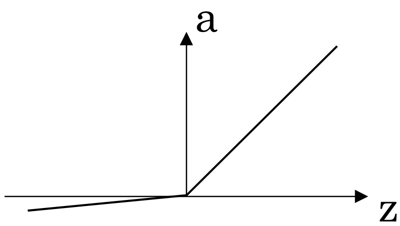
sigmoid: $a = \frac{1}{1+e^{-z}}$



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



ReLU: $a = \max(0, z)$



Leaky ReLU: $a = \max(0.01z, z)$

# Why does neural network need non-linear activation function?

- For your neural network to compute interesting functions, you do need to take a non-linear function
- Try simple linear activation function (identity function)

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

# Why does neural network need non-linear activation function?

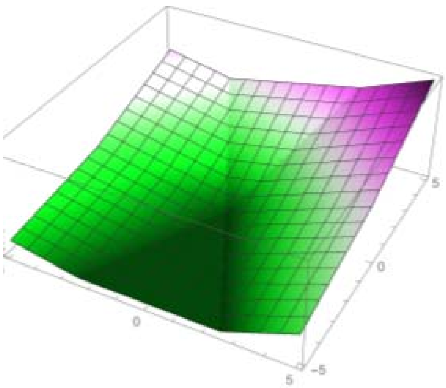
- For your neural network to compute interesting functions, you do need to take a non-linear function
- Try simple linear activation function (identity function)
  → neural network just outputs a linear function of the input

$$
\begin{aligned}
z^{[1]} &= W^{[1]}x + b^{[1]} \\
a^{[1]} &= g^{[1]}(z^{[1]}) = z^{[1]} \\
z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\
a^{[2]} &= g^{[2]}(z^{[2]}) = z^{[2]}
\end{aligned}
$$

$$
\begin{aligned}
a^{[1]} &= z^{[1]} = W^{[1]}x + b^{[1]} \\
a^{[2]} &= z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \\
&= W^{[2]}(W^{[1]}x + b^{[1]}) + b^{[2]} \\
&= \underbrace{(W^{[2]}W^{[1]})}_{W'}x + \underbrace{(W^{[2]}b^{[1]} + b^{[2]})}_{b'} \\
&= W'x + b'
\end{aligned}
$$

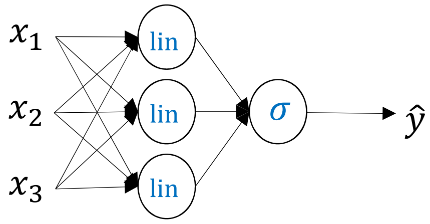
# Why does neural network need non-linear activation function?

- For your neural network to compute interesting functions, you do need to take a non-linear function
- Try simple linear activation function (identity function)
  → neural network just outputs a linear function of the input



$$
\begin{aligned}
a^{[1]} &= z^{[1]} = W^{[1]}x + b^{[1]} \\
a^{[2]} &= z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \\
&= W^{[2]}(W^{[1]}x + b^{[1]}) + b^{[2]} \\
&= \underbrace{(W^{[2]}W^{[1]})}_{W'}x + \underbrace{(W^{[2]}b^{[1]} + b^{[2]})}_{b'} \\
&= W'x + b'
\end{aligned}
$$

# Why does neural network need non-linear activation function?



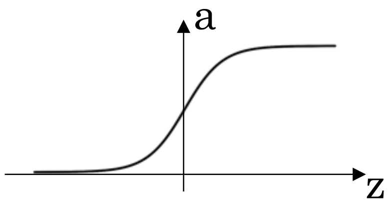

how about this?

this model may be no much more expressive than standard logistic regression

# Derivatives of activation functions

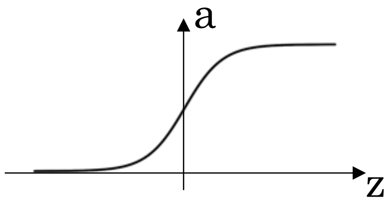
- When you implement back-propagation for neural network, you need to compute the slope or the derivative of the activation functions

## Sigmoid activation function




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


## Sigmoid activation function



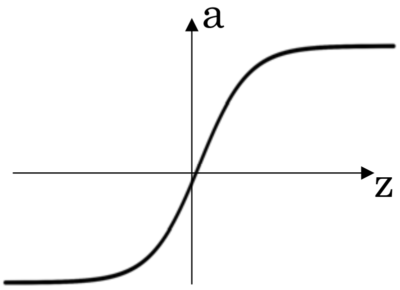
$$g(z) = \frac{1}{1+e^{-z}}$$


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

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

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

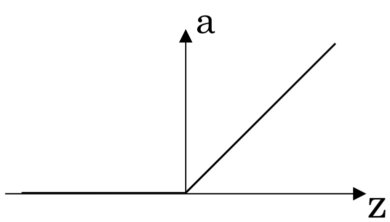
# Tanh activation function




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

$$g'(z) = \frac{d}{dz} g(z) = \text{ slope of } g(x) \text{ at } z$$
$$= 1 - \left(\tanh(z)\right)^2$$
$$= 1 - g(z)^2 = (1 + g(z))(1 - g(z))$$

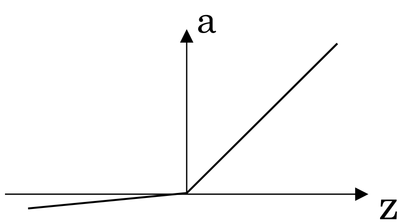
# ReLU and Leaky ReLU




ReLU

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

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




Leaky ReLU

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

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

# Gradient descent for neural networks

For neural networks with a single hidden layer

# of units: $n^{[0]}, n^{[1]}, n^{[2]}$

dimension of training data
= input dimension

# Gradient descent for neural networks

For neural networks with a single hidden layer

# of units: $n^{[0]}, n^{[1]}, n^{[2]}$

Parameters: $\underset{(n^{[1]},n^{[0]})}{W^{[1]}}, \underset{(n^{[1]},1)}{b^{[1]}}, \underset{(n^{[2]},n^{[1]})}{W^{[2]}}, \underset{(n^{[2]},1)}{b^{[2]}}$

# Gradient descent for neural networks

For neural networks with a single hidden layer

# of units: $n^{[0]}, n^{[1]}, n^{[2]}$

Parameters: $\underset{(n^{[1]},n^{[0]})}{W^{[1]}},\ \underset{(n^{[1]},1)}{b^{[1]}},\ \underset{(n^{[2]},n^{[1]})}{W^{[2]}},\ \underset{(n^{[2]},1)}{b^{[2]}}$

Cost function: $J(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\overset{a^{[2]}}{\hat{y}}, y)$

# Gradient descent for neural networks

For neural networks with a single hidden layer

# of units: $n^{[0]}, n^{[1]}, n^{[2]}$

Parameters: $\underset{(n^{[1]},n^{[0]})}{W^{[1]}},\ \underset{(n^{[1]},1)}{b^{[1]}},\ \underset{(n^{[2]},n^{[1]})}{W^{[2]}},\ \underset{(n^{[2]},1)}{b^{[2]}}$

Cost function: $J(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\overset{a^{[2]}}{\hat{y}}, y)$

Gradient descent: Repeat {

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

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

Update parameters

$W^{[1]} := W^{[1]} - \alpha \cdot dW^{[1]}$
$b^{[1]} := b^{[1]} - \alpha \cdot db^{[1]}$
$W^{[2]} := W^{[2]} - \alpha \cdot dW^{[2]}$
$b^{[2]} := b^{[2]} - \alpha \cdot db^{[2]}$

} parameters converged;

# Gradient descent for neural networks

For neural networks with a single hidden layer

# of units: $n^{[0]}, n^{[1]}, n^{[2]}$

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

$(n^{[1]}, n^{[0]})$ $(n^{[1]}, 1)$ $(n^{[2]}, n^{[1]})$ $(n^{[2]}, 1)$

$a^{[2]}$

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

Gradient descent: Repeat {

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

back-propagation: Compute gradients $dW^{[1]} = \frac{\partial J}{\partial W^{[1]}},\ b^{[1]} = \frac{\partial J}{\partial b^{[1]}}, \dots$

Update parameters

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

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

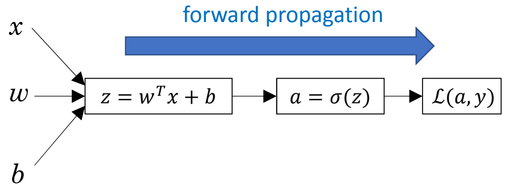
$W^{[2]} := W^{[2]} - \alpha \cdot dW^{[2]}$

$b^{[2]} := b^{[2]} - \alpha \cdot db^{[2]}$

} parameters converged;

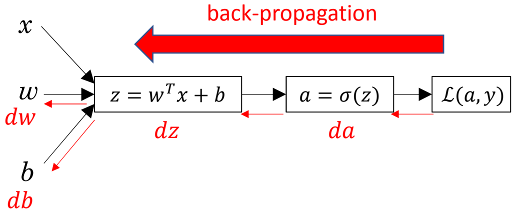
# Computing gradients

- Logistic regression



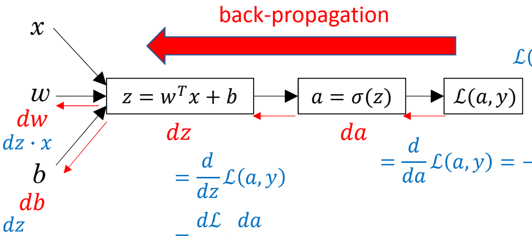
# Computing gradients

- Logistic regression




# Computing gradients

- Logistic regression




$$\mathcal{L}(a, y) = -y \log a - (1 - y) \log(1 - a)$$

$dw = dz \cdot x$

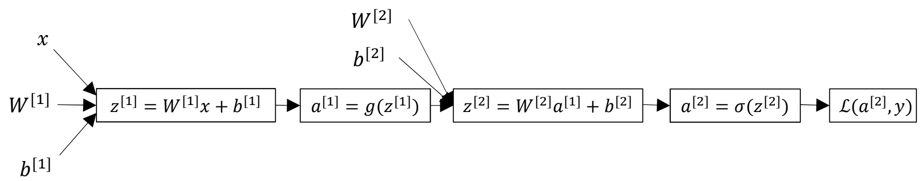
$db = dz$

$$da = \frac{d}{da}\mathcal{L}(a, y) = -\frac{y}{a} + \frac{1 - y}{1 - a}$$

$$\begin{aligned}
dz &= \frac{d}{dz}\mathcal{L}(a, y) \\
&= \frac{d\mathcal{L}}{da} \cdot \frac{da}{dz} \\
&= da \cdot \sigma'(z) \\
&= da \cdot \sigma(z) \cdot \big(1 - \sigma(z)\big) \\
&= da \cdot a \cdot (1 - a) \\
&= a - y
\end{aligned}$$

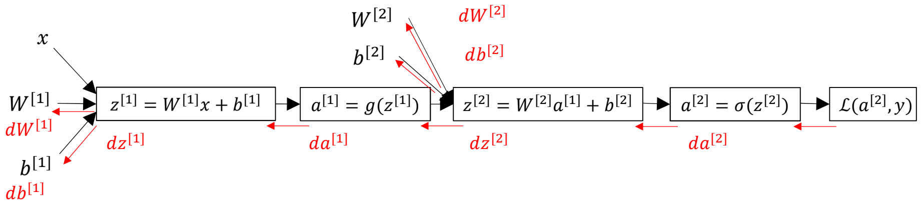
# Computing gradients

- Two-layer neural network




# Computing gradients

- Two-layer neural network



# Computing gradients

- Two-layer neural network

$x$, $W^{[1]}$, $b^{[1]}$ → $z^{[1]} = W^{[1]}x + b^{[1]}$ → $a^{[1]} = g(z^{[1]})$ → ($W^{[2]}$, $b^{[2]}$) → $z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$ → $a^{[2]} = \sigma(z^{[2]})$ → $\mathcal{L}(a^{[2]}, y)$

$dW^{[2]} = dz^{[2]}a^{[1]T}$ $\quad (n^{[2]}, n^{[1]}) = (n^{[2]}, 1) \times (1, n^{[1]})$

$db^{[2]} = dz^{[2]}$ $\quad (n^{[2]}, 1)$

$da^{[2]} = -\frac{y}{a^{[2]}} + \frac{1-y}{1-a^{[2]}}$Z

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

$da^{[1]} = W^{[2]T}dz^{[2]}$

$(n^{[1]}, 1) = (n^{[1]}, n^{[2]}) \times (n^{[2]}, 1)$

$dW^{[1]}$

$db^{[1]}$

$dz^{[1]} = da^{[1]} * g'(z^{[1]})$

element-wise product

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

$(n^{[1]}, 1) = (n^{[1]}, n^{[2]}) \times (n^{[2]}, 1) * (n^{[2]}, 1)$

$dW^{[1]} = dz^{[1]} \cdot x^T$ $\quad (n^{[1]}, n^{[0]}) = (n^{[1]}, 1) \times (1, n^{[0]})$

$db^{[1]} = dz^{[1]}$ $\quad (n^{[1]}, 1)$

# Computing gradients

- Two-layer neural network

$x$, $W^{[1]}$, $b^{[1]}$ → $z^{[1]} = W^{[1]}x + b^{[1]}$ → $a^{[1]} = g(z^{[1]})$ → ($W^{[2]}$, $b^{[2]}$) → $z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$ → $a^{[2]} = \sigma(z^{[2]})$ → $\mathcal{L}(a^{[2]}, y)$

$dW^{[2]} = dz^{[2]}a^{[1]T}$ $\quad (n^{[2]}, n^{[1]}) = (n^{[2]}, 1) \times (1, n^{[1]})$

$db^{[2]} = dz^{[2]}$ $\quad (n^{[2]}, 1)$

$da^{[2]} = -\frac{y}{a^{[2]}} + \frac{1-y}{1-a^{[2]}}$Z

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

$da^{[1]} = W^{[2]T}dz^{[2]}$

$(n^{[1]}, 1) = (n^{[1]}, n^{[2]}) \times (n^{[2]}, 1)$

$dW^{[1]}$

$db^{[1]}$

$dz^{[1]} = da^{[1]} * g'(z^{[1]})$

element-wise product

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

$(n^{[1]}, 1) = (n^{[1]}, n^{[2]}) \times (n^{[2]}, 1) * (n^{[1]}, 1)$

$dW^{[1]} = dz^{[1]} \cdot x^T$ $\quad (n^{[1]}, n^{[0]}) = (n^{[1]}, 1) \times (1, n^{[0]})$

$db^{[1]} = dz^{[1]}$ $\quad (n^{[1]}, 1)$

## Summary of gradient descent

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

$dW^{[2]} = dz^{[2]} a^{[1]^T}$

$db^{[2]} = dz^{[2]}$

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

$dW^{[1]} = dz^{[1]} x^T$

$db^{[1]} = dz^{[1]}$

## Summary of gradient descent

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

$dW^{[2]} = dz^{[2]} a^{[1]^T}$

$db^{[2]} = dz^{[2]}$

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

$dW^{[1]} = dz^{[1]} x^T$

$db^{[1]} = dz^{[1]}$

$dZ^{[2]} = A^{[2]} - Y$

$dW^{[2]} = \frac{1}{m}(A^{[2]} - Y) A^{[1]^T}$

$db^{[2]} = \frac{1}{m}(A^{[2]} - Y) I$

$dZ^{[1]} = W^{[2]T}(A^{[2]} - Y) * g^{[1]\prime}(Z^{[1]})$

$dW^{[1]} = \frac{1}{m}(W^{[2]T}(A^{[2]} - Y) * g^{[1]\prime}(Z^{[1]})) X^T$

$db^{[1]} = \frac{1}{m}(W^{[2]T}(A^{[2]} - Y) * g^{[1]\prime}(Z^{[1]})) I$

# Summary of gradient descent

- computed only by matrix operations
  → parallel processing in GPU
- required to code efficiently in Tensorflow

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

$$dW^{[2]} = dz^{[2]}a^{[1]^T}$$

$$db^{[2]} = dz^{[2]}$$

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

$$dW^{[1]} = dz^{[1]}x^T$$

$$db^{[1]} = dz^{[1]}$$

$$\underbrace{dZ^{[2]}}_{(n^{[2]},m)} = \underbrace{A^{[2]} - Y}_{(n^{[2]},m)}$$

$$\underbrace{dW^{[2]}}_{(n^{[2]},n^{[1]})} = \frac{1}{m}\underbrace{(A^{[2]}-Y)}_{=(n^{[2]},m)}\underbrace{A^{[1]^T}}_{(m,n^{[1]})}$$

$$\underbrace{db^{[2]}}_{(n^{[2]},1)} = \frac{1}{m}\underbrace{(A^{[2]}-Y)}_{(n^{[2]},m)}\underbrace{I}_{(m,1)}$$

$$\underbrace{dZ^{[1]}}_{(n^{[1]},m)} = \underbrace{W^{[2]T}(A^{[2]}-Y)}_{(n^{[1]},m)} * \underbrace{g^{[1]\prime}(Z^{[1]})}_{(n^{[1]},m)}$$

$$\underbrace{dW^{[1]}}_{(n^{[1]},n^{[0]})} = \frac{1}{m}\underbrace{(W^{[2]T}(A^{[2]}-Y) * g^{[1]\prime}(Z^{[1]}))}_{=\ (n^{[1]},m)}\underbrace{X^T}_{(m,n^{[0]})}$$

$$\underbrace{db^{[1]}}_{(n^{[1]},1)} = \frac{1}{m}\underbrace{(W^{[2]T}(A^{[2]}-Y) * g^{[1]\prime}(Z^{[1]}))}_{=\ (n^{[1]},m)}\underbrace{I}_{(m,1)}$$