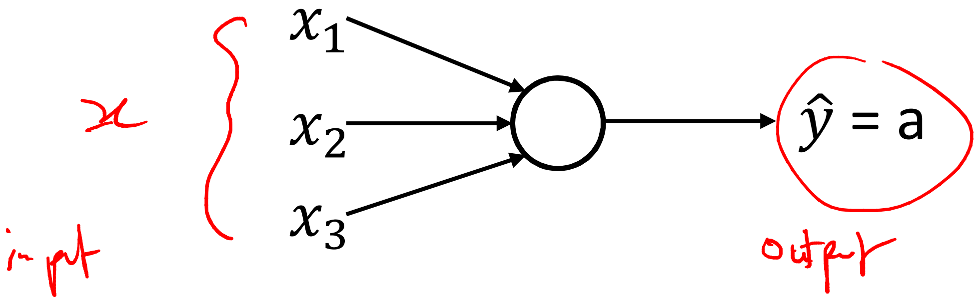
Deep learning

Dr. Aissa Boulmerka
a.boulmerka@centre-univ-mila.dz

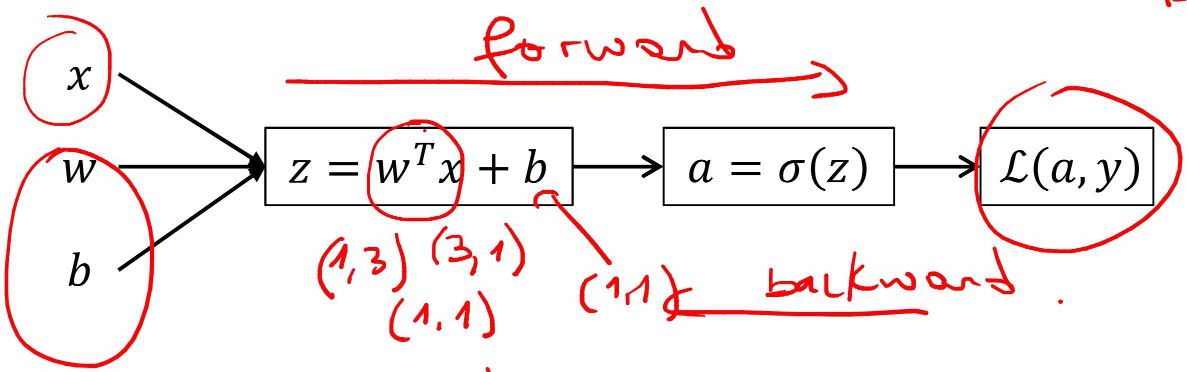
2023-2024

CHAPTER 2
SHALLOW NEURAL NETWORKS

What is a Neural Network?

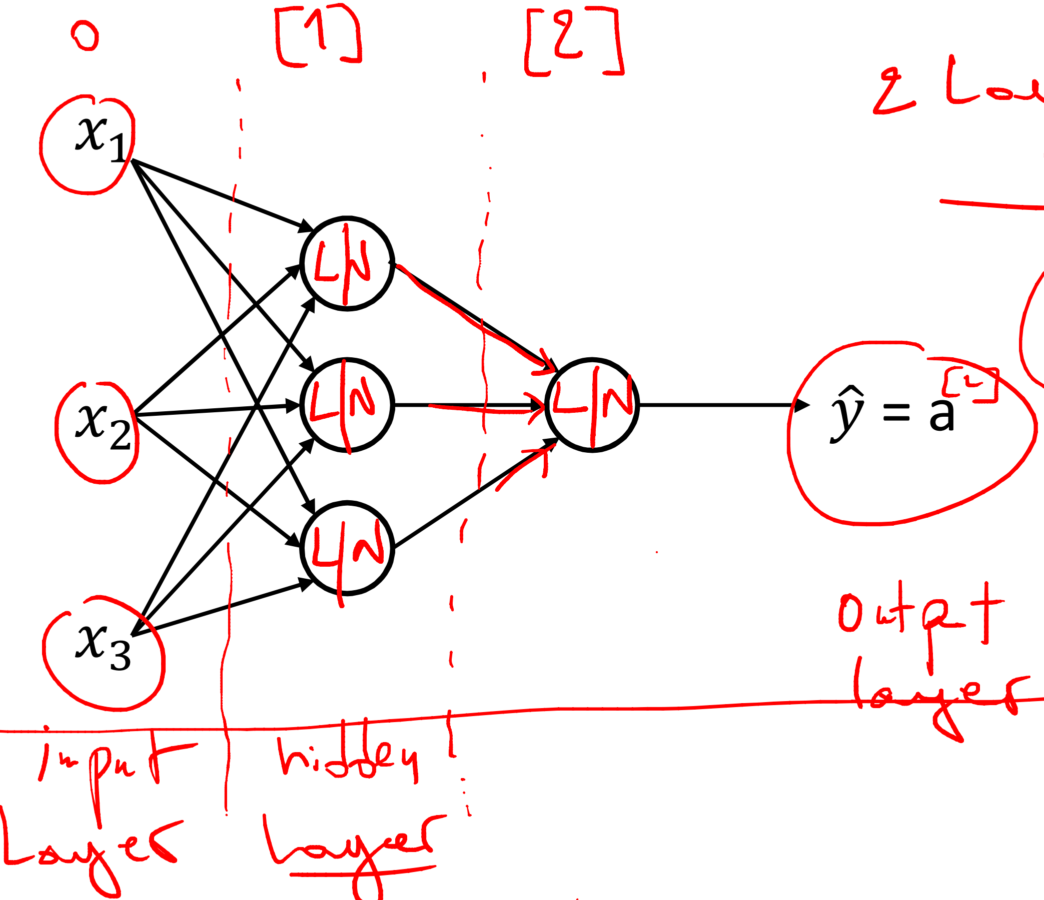


$$w^T x = \sum_{i=1}^{3} w_i x_i$$



$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \quad w = (w_1, w_2, w_3)$$

What is a Neural Network?



2 Layer NN.

$y = \sigma(wx+b)$

$= z$

$a = \sigma(z)$

$z = wx+b$

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

$W^{[2]}, b^{[2]} \rightarrow z^{[2]} = w^{[2]}a^{[1]} + b^{[2]}$

$z^{[1]}$ $a^{[1]}$

$[i] \leftarrow$ Layer i

Neural Network Representation

2 layers neural network

$a^{[1]} = (a_1^{[1]}, a_2^{[1]}, a_3^{[1]}, a_4^{[1]})$

$\mathbf{w}^{[1]}, \mathbf{b}^{[1]}$

(4,3) (4,1)

$W^{[1]} =$

$\mathbf{a}^{[0]} = \mathbf{X}$

$\mathbf{a}^{[1]}$

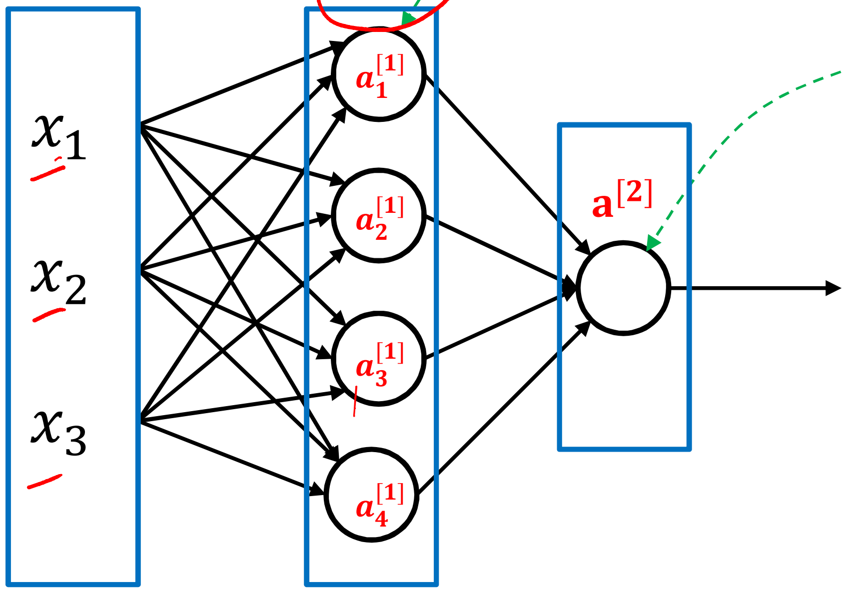
$\mathbf{w}^{[2]}, \mathbf{b}^{[2]}$

(1,4) (1,1)

$(4, 3)$

#nodes #inputs

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



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

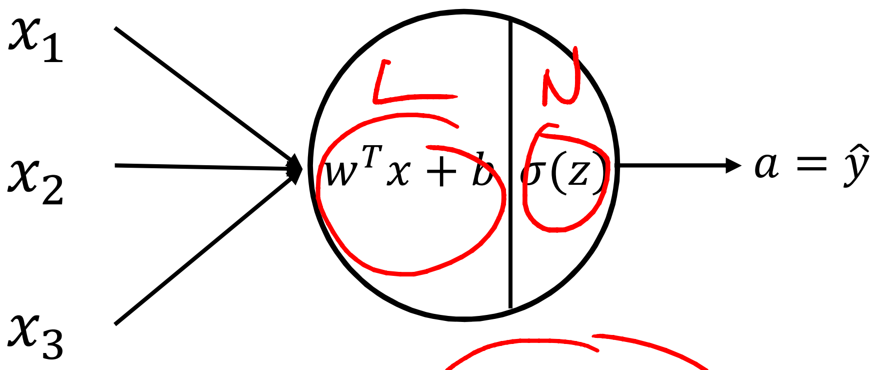
$$\mathbf{a}^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}$$

Input layer

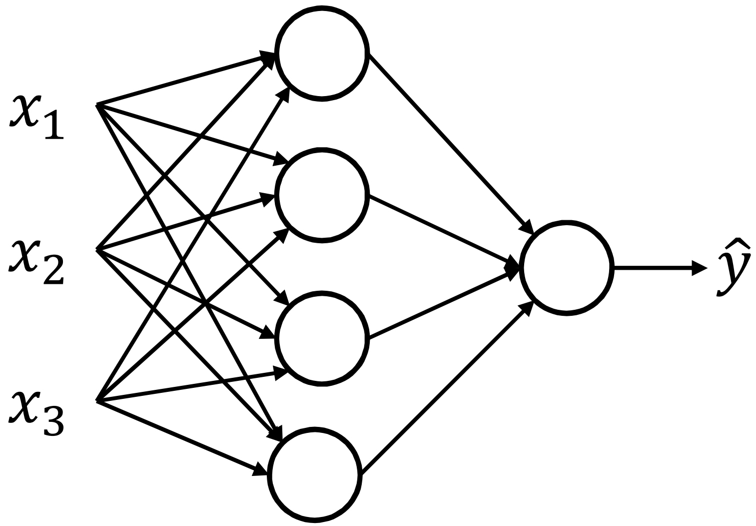
Hidden layer

Output layer

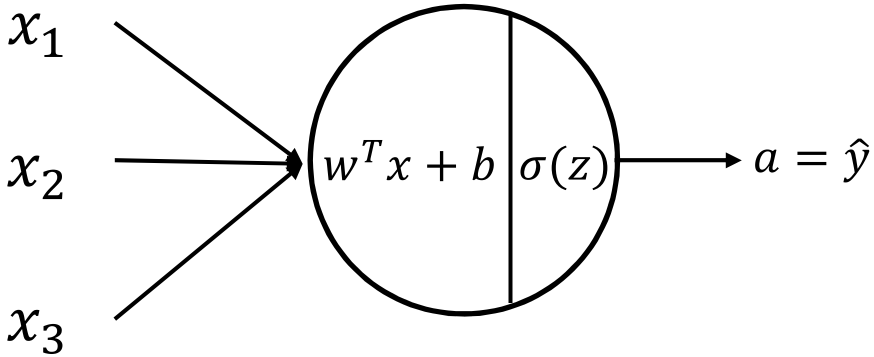
Neural Network Representation



$$z = w^T x + b$$
$$a = \sigma(z)$$



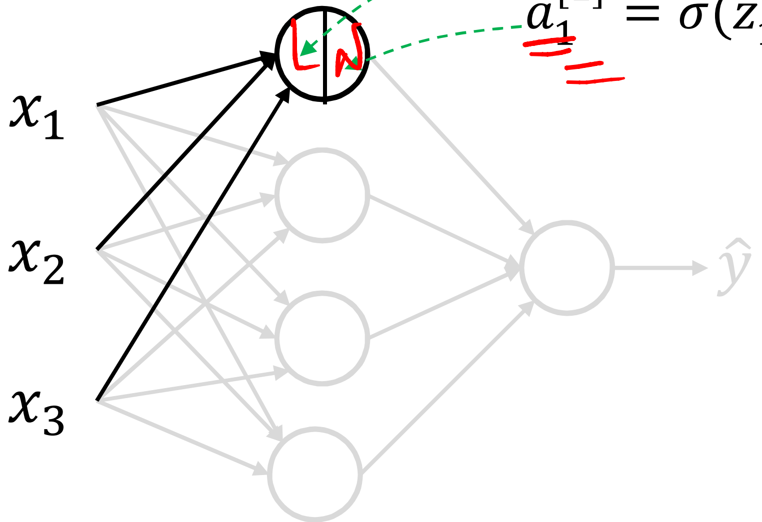
Neural Network Representation



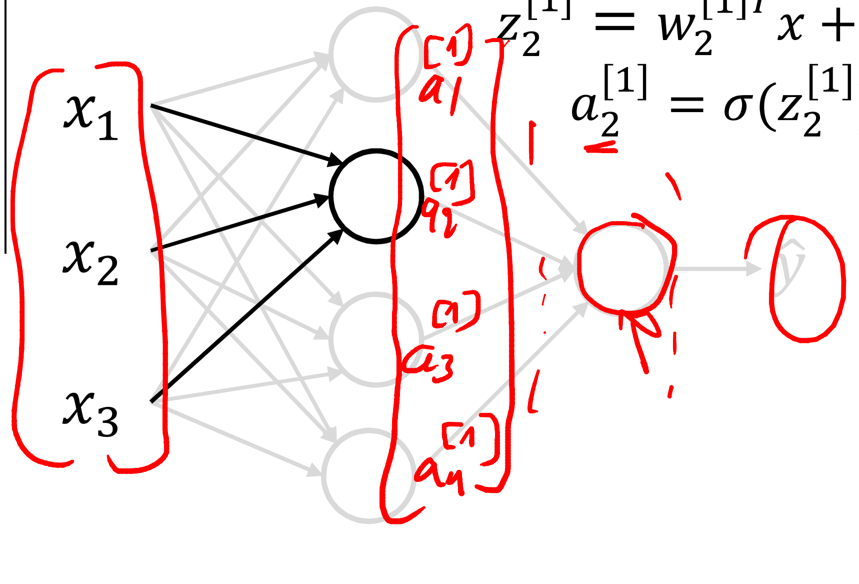
$$z = w^T x + b$$
$$a = \sigma(z)$$

$$z_1^{[2]} = w^{[2]} a^{[1]} + b^{[2]}$$
$$\hat{y} = a_1^{[2]} = \sigma(z_1^{[2]})$$

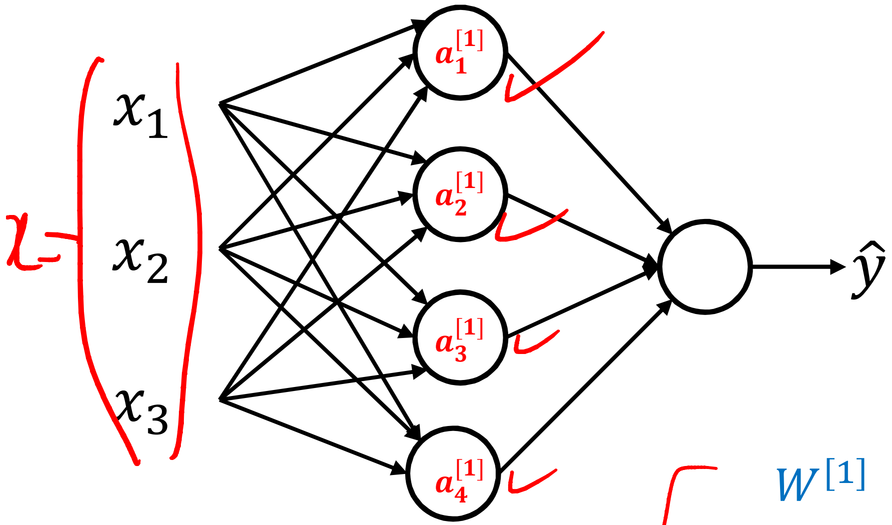
$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}$$
$$a_1^{[1]} = \sigma(z_1^{[1]})$$



$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}$$
$$a_2^{[1]} = \sigma(z_2^{[1]})$$



Neural Network Representation



$$z_1^{[1]} = w_1^{[1]T}x + b_1^{[1]}, \quad a_1^{[1]} = \sigma\left(z_1^{[1]}\right)$$

$$z_2^{[1]} = w_2^{[1]T}x + b_2^{[1]}, \quad a_2^{[1]} = \sigma\left(z_2^{[1]}\right)$$

$$z_3^{[1]} = w_3^{[1]T}x + b_3^{[1]}, \quad a_3^{[1]} = \sigma\left(z_3^{[1]}\right)$$

$$z_4^{[1]} = w_4^{[1]T}x + b_4^{[1]}, \quad a_4^{[1]} = \sigma\left(z_4^{[1]}\right)$$

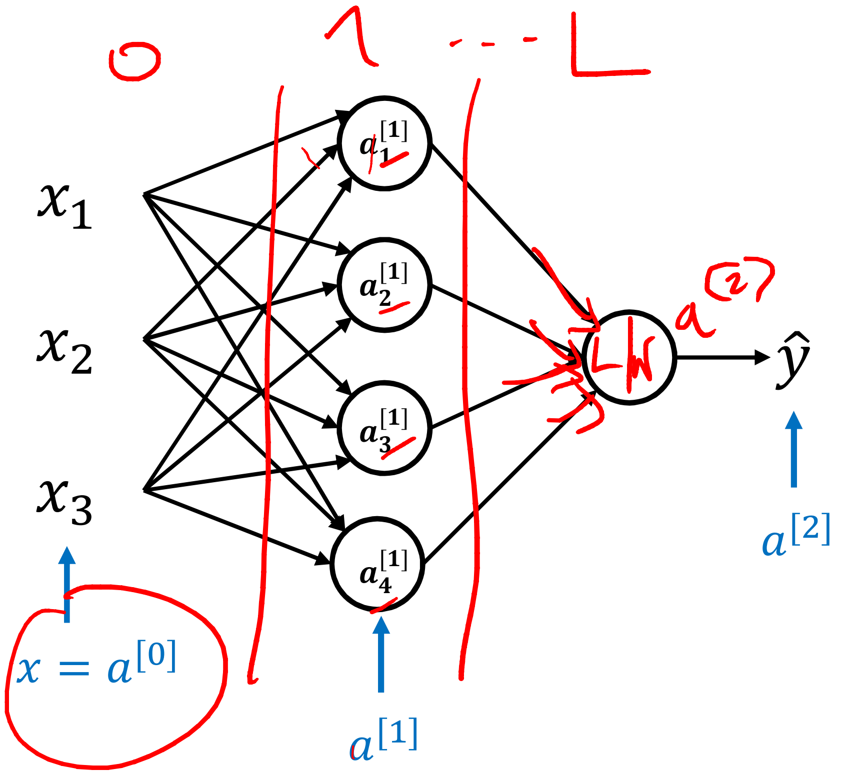
$$\underset{(4,1)}{z^{[1]}} = \underbrace{\overbrace{\begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_2^{[1]T} — \\ — w_2^{[1]T} — \end{bmatrix}}^{W^{[1]}}}_{(4,3)} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{(3,1)} + \underbrace{\overbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]}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

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

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

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix} = \sigma(Z^{[1]})$$

Neural Network Representation Learning



for i:=1 to L

$$z^{[i]} = w^{[i]} \cdot a^{[i-1]} + b^{[i]}$$

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

Given input x:

L $z^{[1]} = W^{[1]}x + b^{[1]}$ (x = $a^{[0]}$)

$(4,1) \quad (4,3)(3,1) \quad (4,1)$

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

$(4,1) \quad (4,1)$

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

$(1,1) \quad (1,4)(4,1) \quad (1,1)$

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

$(1,1) \quad (1,1)$

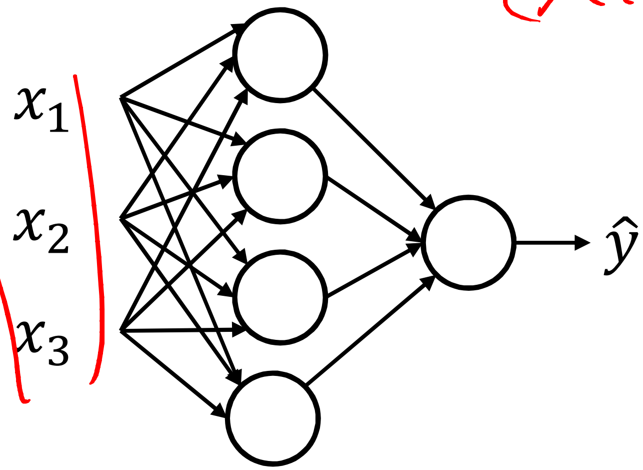
$x^{(1)}, x^{(2)} \ldots x^{(m)}$

$(x_1^{(1)}, x_2^{(1)}, x_3^{(1)})$

For loop across multiple examples

$(X, Y), \; X = \{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

$x =$



forward

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

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

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

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

… … …

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

$a^{[2](i)}$

Layer 2 ---- Example (i)

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

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

$z^{[1](i)}$

Vectorizing across multiple examples

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

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

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

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

(n_x, m)

$n_x = 3$

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

m

$4 \begin{bmatrix} \ \end{bmatrix}$

training examples (m)

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

hidden units

$m = 1000$

Justification for vectorized implementation

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

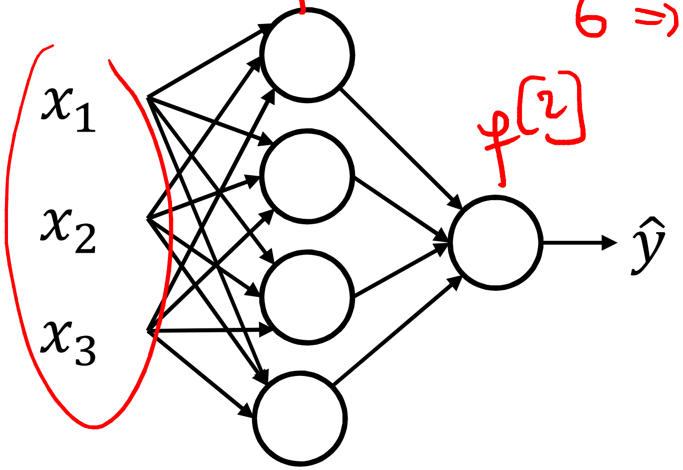
$W^{[1]}x^{(1)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}$ $W^{[1]}x^{(2)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}$ $W^{[1]}x^{(3)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}$

$$W^{[1]}\underbrace{\begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & x^{(3)} \\ | & | & | \end{bmatrix}}_{X} + b^{[1]} = \begin{bmatrix} | & | & | \\ W^{[1]}x^{(1)} & W^{[1]}x^{(2)} & W^{[1]}x^{(3)} \\ | & | & | \\ + & + & + \\ b^{[1]} & b^{[1]} & b^{[1]} \end{bmatrix} = \underbrace{\begin{bmatrix} | & | & | \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} \\ | & | & | \end{bmatrix}}_{Z^{[1]}}$$

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

$m=3$

Recap of vectorizing across multiple examples



$6 \Rightarrow f^{[1]}$

$f = [6, 8]$

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

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

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

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

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

activation

$x =$

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

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

$A^{[0]}$

$$A^{[1]} = \sigma\left(Z^{[1]}\right)$$

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

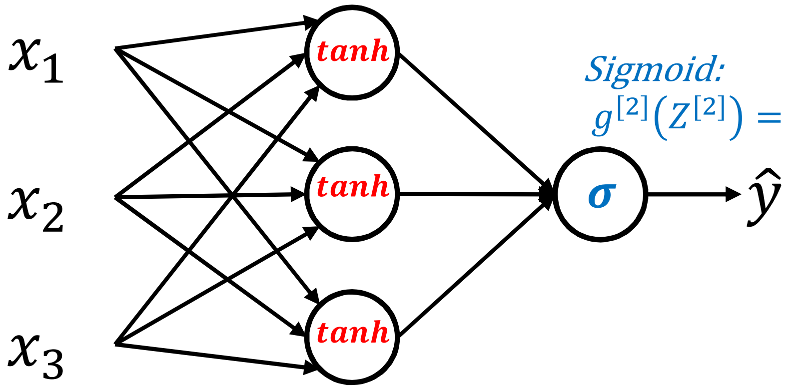
$$A^{[2]} = \sigma\left(Z^{[2]}\right)$$

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

Activation functions

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

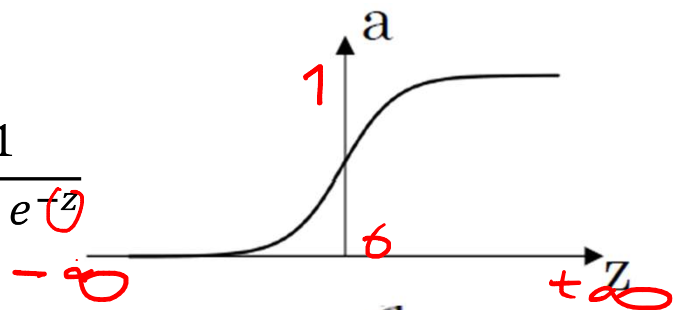
Sigmoid:
$g^{[2]}(Z^{[2]}) = \sigma(Z^{[2]})$



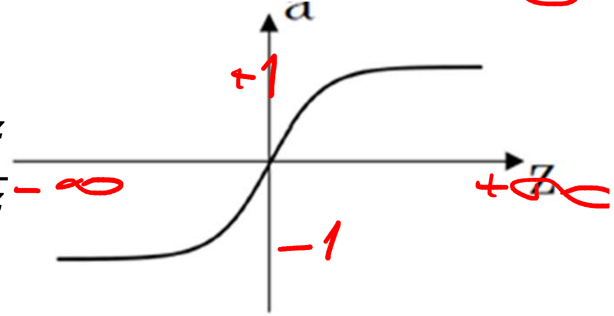
Given X:

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

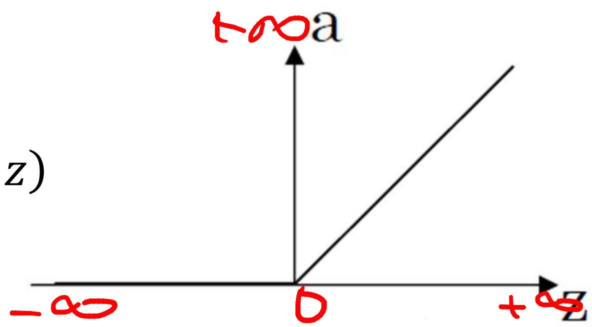
Sigmoid:
$$a = \frac{1}{1 + e^{-z}}$$



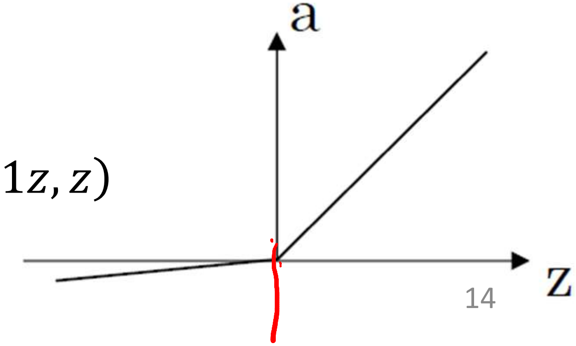
tanh:
$$a = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



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



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

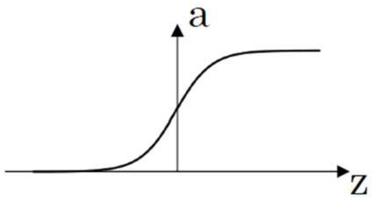


Pros and cons of activation functions

Sigmoid activation function:

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

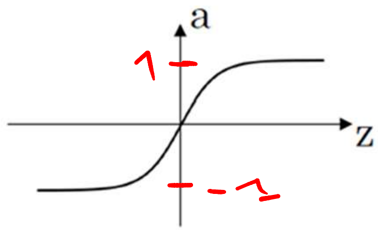
- Never use this, except for the output layer.
- if you are doing binary classification, or maybe almost never use this.



tanh activation function:

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

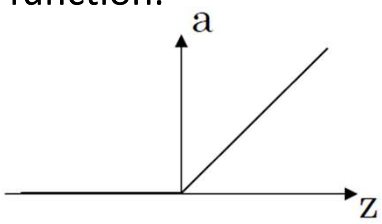
- The tanh is much strictly superior.



ReLU activation function:

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

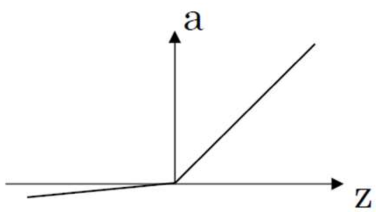
- The default and the most commonly used activation function is the ReLU.
- So if you're not sure what else to use, use the ReLU function.



Leaky ReLU activation function:

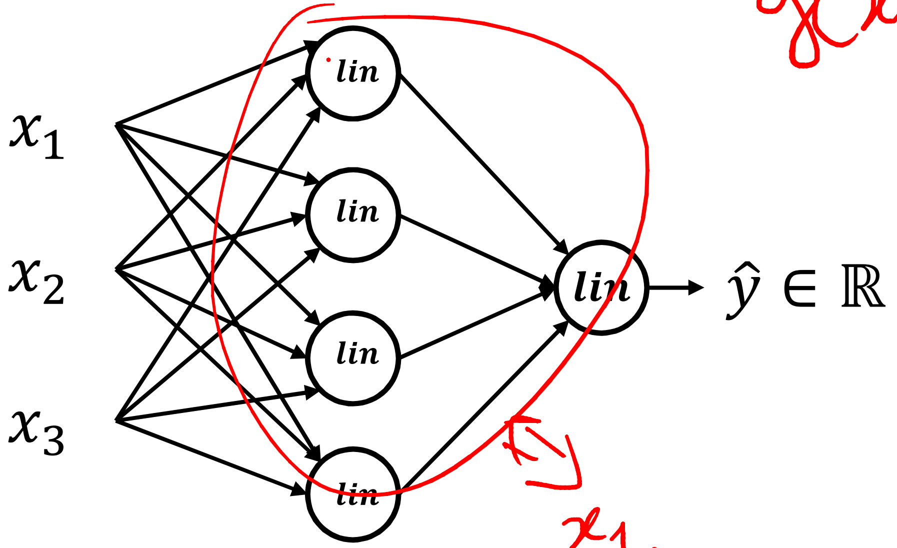
$$a = \max(0.01z, z)$$

- You can also try the leaky ReLU function.



Why do you need non linear activation functions?

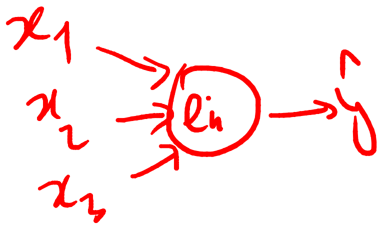
$g(x) = x.$ $g(x) = ax + b$



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

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

$$a^{[2]} = W'x + b'$$



Given X:

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

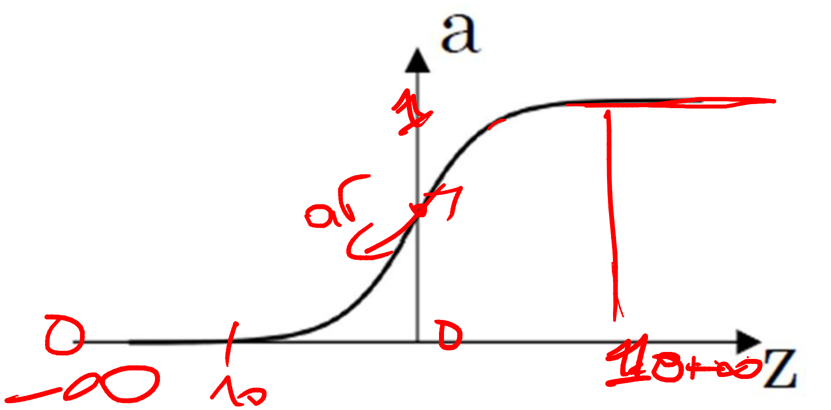
Linear activation function:
$g(z) = z$

Derivatives of activation functions

$\sigma(10) \simeq 1$

$\sigma'(10) \simeq 0$

Sigmoid activation function:



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

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

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

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

$a = g(z), \quad g'(z) = a(1-a)$

- $z = 10 \Longrightarrow g(z) \approx 1$

  $\Longrightarrow \frac{d}{dz} g(z) \approx 1(1-1) \approx 0$

- $z = -10 \Longrightarrow g(z) \approx 0$

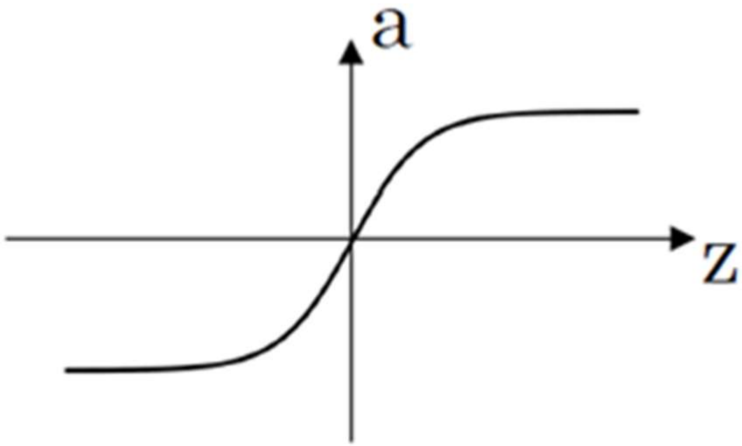
  $\Longrightarrow \frac{d}{dz} g(z) \approx 0(1-0) \approx 0$

- $z = 0 \Longrightarrow g(z) = 1/2$

  $\Longrightarrow \frac{d}{dz} g(z) = \frac{1}{2}\left(1 - \frac{1}{2}\right) = 1/4$

Derivatives of activation functions

Tanh activation function:



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

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

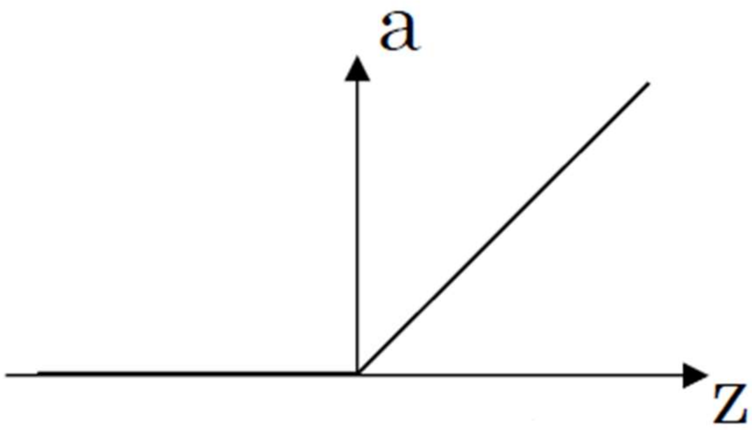
$$= 1 - (\tanh(z))^2$$

$$a = g(z), \qquad g'(z) = 1 - a^2$$

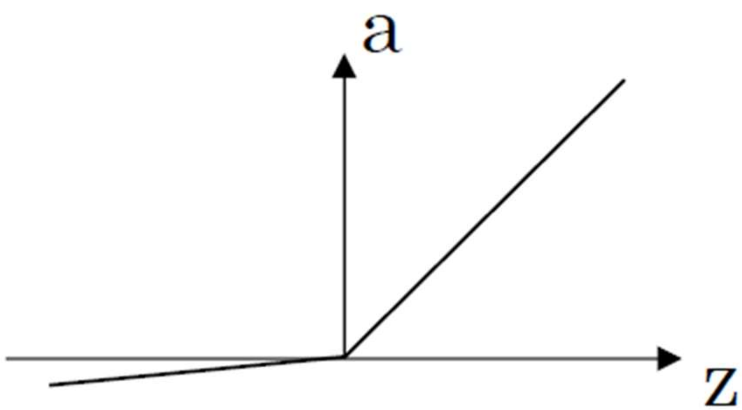
- $z = 10 \implies tanh(z) \approx 1$
  $\implies g'(z) \approx 0$
- $z = -10 \implies g(z) \approx -1$
  $\implies g'(z) \approx 0$
- $z = 0 \implies g(z) = 0$
  $\implies g'(z) = 1$

Derivatives of activation functions

ReLU and Leaky ReLU :



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



$$a = max(0.01z, z)$$

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

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

Gradient descent for neural networks

Parameters :

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

with $n_x = n^{[0]}, n^{[1]}, n^{[2]} = 1$

Cost function:

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

output

Gradient descent:

Initialization of $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}$.

```
Repeat {
```

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

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

$$\left.\begin{aligned} W^{[1]} &= W^{[1]} - \alpha dW^{[1]} \\ b^{[1]} &= b^{[1]} - \alpha db^{[1]} \end{aligned}\right\} \text{MAJ}$$

$dW^{[2]} = \frac{\partial J}{\partial W^{[2]}}, db^{[2]} = \frac{\partial J}{\partial b^{[2]}}$ derivées

$$\left.\begin{aligned} W^{[2]} &= W^{[2]} - \alpha dW^{[2]} \\ b^{[2]} &= b^{[2]} - \alpha db^{[2]} \end{aligned}\right\} \text{MAJ}$$

```
}
```

Convergence.

Formulas for computing derivatives

Forward propagation:

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

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

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

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

Back propagation:

$\hat{y} - y$

$L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$

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

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$

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

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

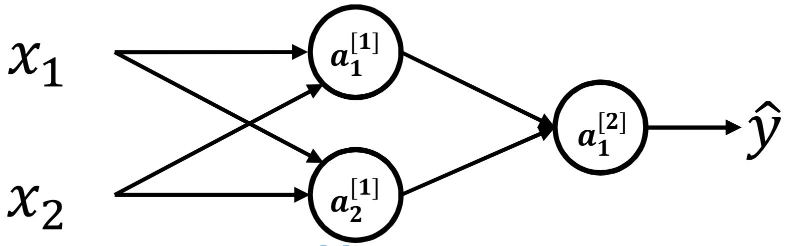
Element wise product

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$

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

reshape

What happens if you initialize weights to zero?



$n^{[0]} = 2$ $\quad n^{[1]} = 2$

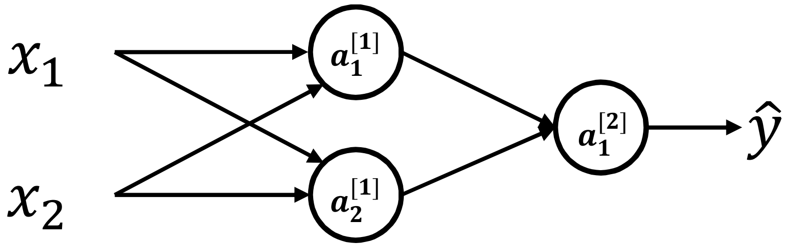
$$W^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\boldsymbol{a_1^{[1]}} = \boldsymbol{a_2^{[1]}} \text{(symmetric)} \qquad \boldsymbol{dz_1^{[1]}} = \boldsymbol{dz_2^{[1]}}$$

$$dw = \begin{bmatrix} u & v \\ u & v \end{bmatrix} \qquad W^{[1]} = W^{[1]} - \alpha dw$$

- The bias terms b can be initialized by 0, but initializing W to all 0s is a problem:
  - The two activations $\boldsymbol{a_1^{[1]}}$ and $\boldsymbol{a_2^{[1]}}$ will be the same, because both of these hidden units are computing exactly the same function.
  - After every single iteration of training the two hidden units are still computing exactly the same function.

Random initialization



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

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

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

$$b^{[1]} = 0$$

Vectorization demo

jupyter demo (modifié)

Entrée [3]:

```
import time

a = np.random.rand(10000000)
b = np.random.rand(10000000)

tic = time.time()
c = np.dot(a,b)
toc = time.time()

print("{:.6f}".format(c))
print("Vectorized version:" + str(1000*(toc-tic))+"ms")

tic = time.time()
c = 0
for i in range(10000000):
    c = c + a[i]*b[i]
toc = time.time()

print("{:.6f}".format(c))
print("For loop:" + str(1000*(toc-tic)) + "ms")
```

```
2499584.436480
Vectorized version:4.6520233154296875ms
2499584.436480
For loop:4441.97678565979ms
```

References

- Andrew Ng. Deep learning. Coursera.
- Geoffrey Hinton. Neural Networks for Machine Learning.
- Kevin P. Murphy. Probabilistic Machine Learning An Introduction. MIT Press, 2022.