# IG.3510-Machine Learning
## Lectures 5: Introduction to deep learning

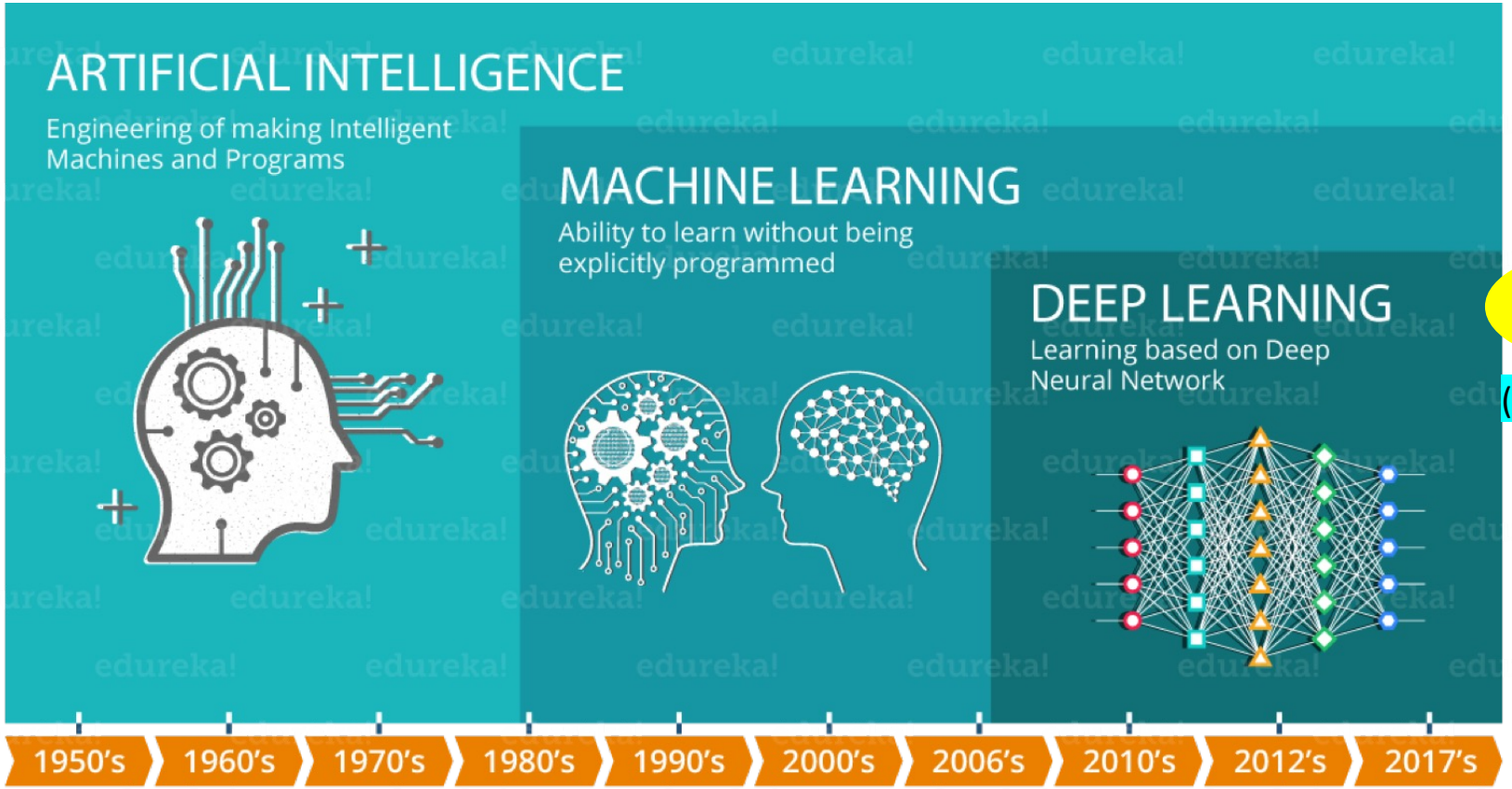
Dr. Patricia CONDE-CESPEDES

patricia.conde-cespedes@isep.fr

November 6th, 2023

# Introduction

# AI vs. Machine Learning vs. Deep Learning

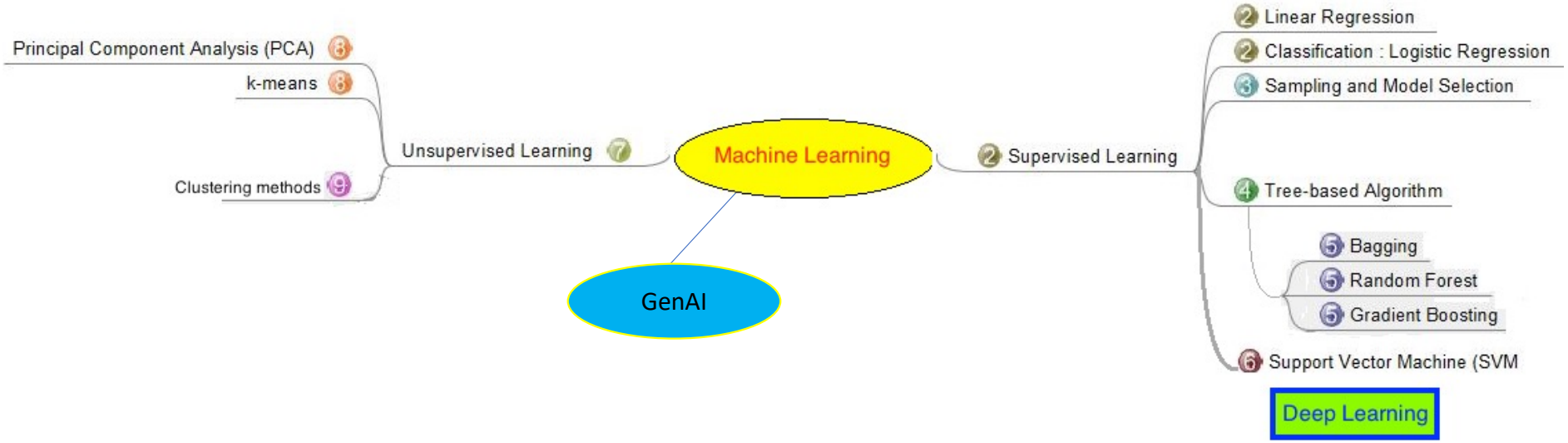



GenAI

(Since 2020s )

Source: https://www.edureka.co/blog/what-is-deep-learning

# What is the place of Deep Learning in Machine Learning?

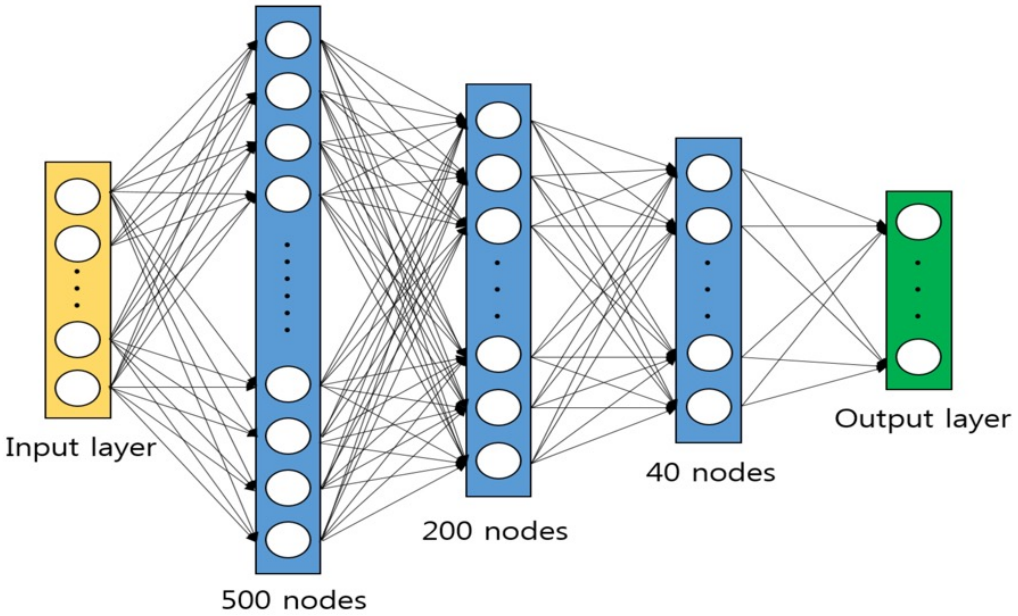


# What is Deep Learning?

**Deep Learning definition**

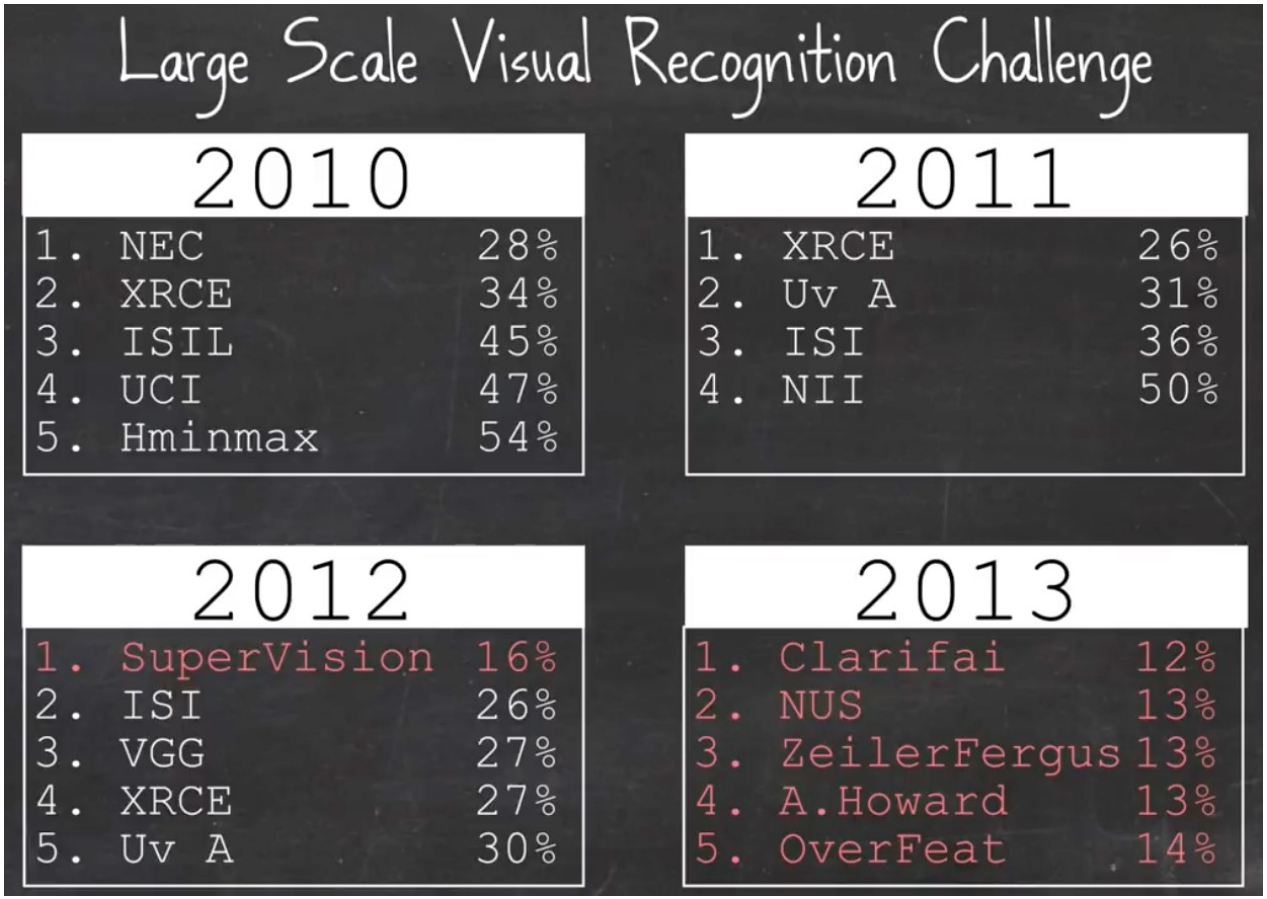
- A machine learning technique that learns features directly from data.

Data can be **images**, **text**, or **speech**:

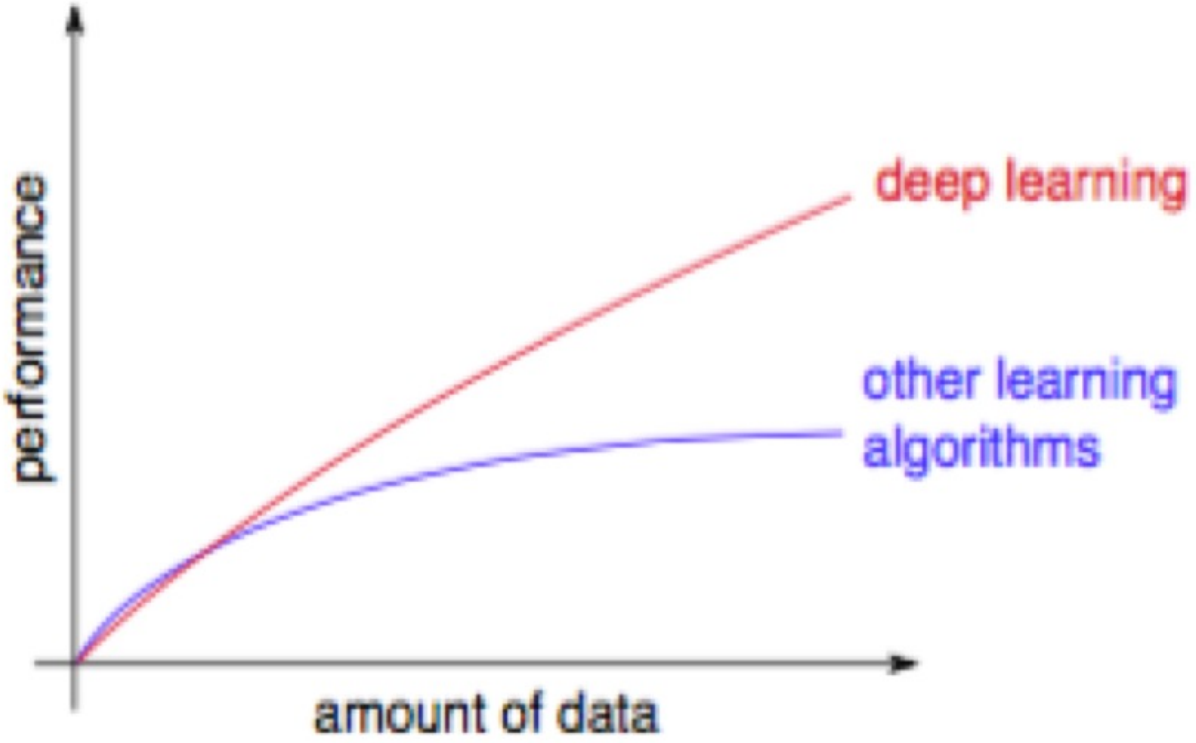
# Why Deep Learning now?

(1998) Pioneer Yann Le Cun (At Meta since 2013)



# Why Deep Learning now?

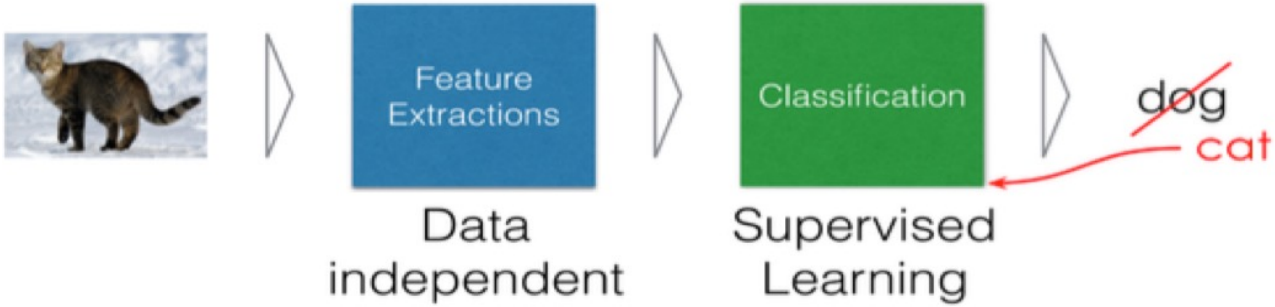
- Better algorithms and understanding
- Computer power (GPUs)
- More data available, i.e. big data
- Open source tools for models



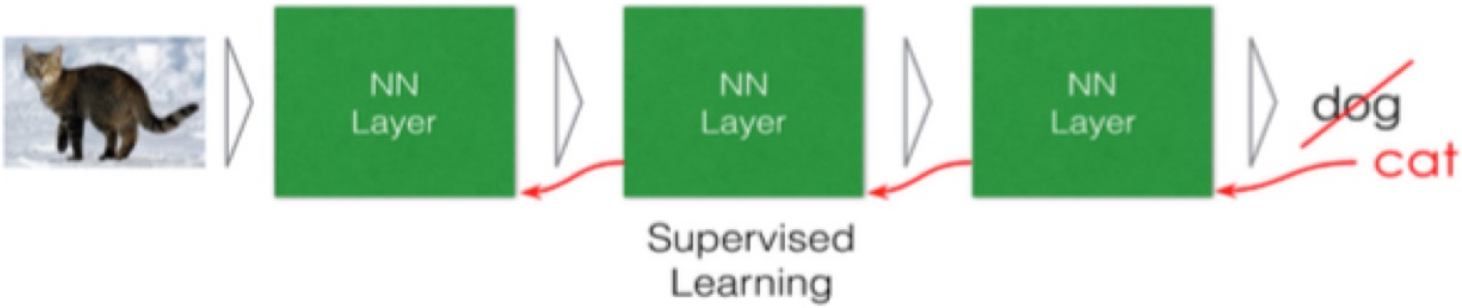

Source: Coursera ”Deep Learning specialization”.

# Machine Learning and Deep Learning

## Typical Machine Learning System




## Deep Learning System




Goal of this lecture: know how a deep learning system work. What is in those black boxes?

# Deep Learning in Computer vision (CV)

Image recognition


[Krizhevsky 2012]

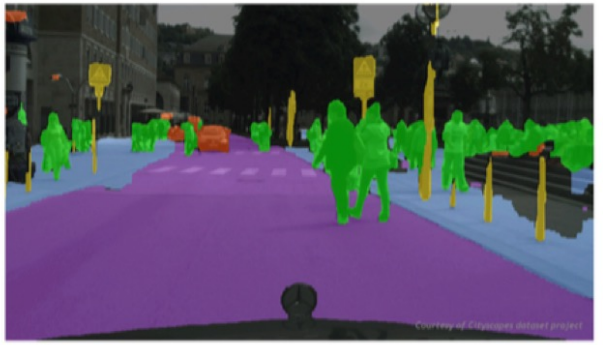
Handwriting recognition

[Ciresan et al. 2013]

Object Detection

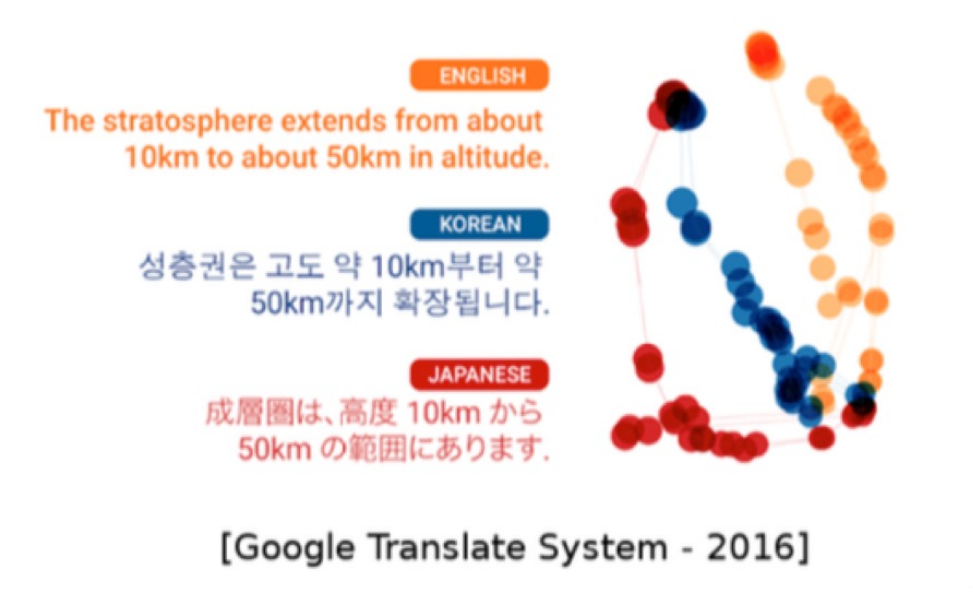
[Faster R-CNN - Ren 2015]

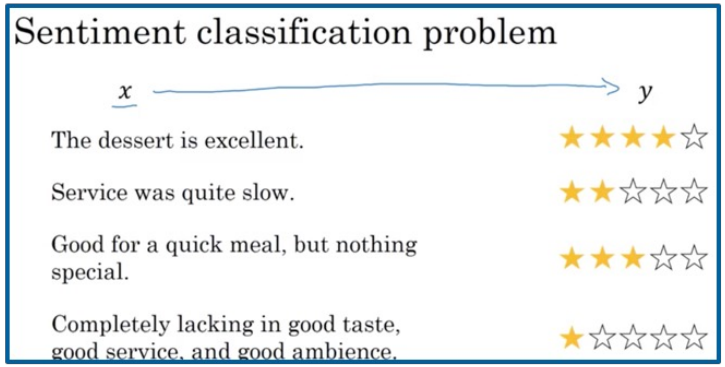
Image Segmentation



[NVIDIA dev blog]

# Deep Learning in Natural Language Processiong (NLP)



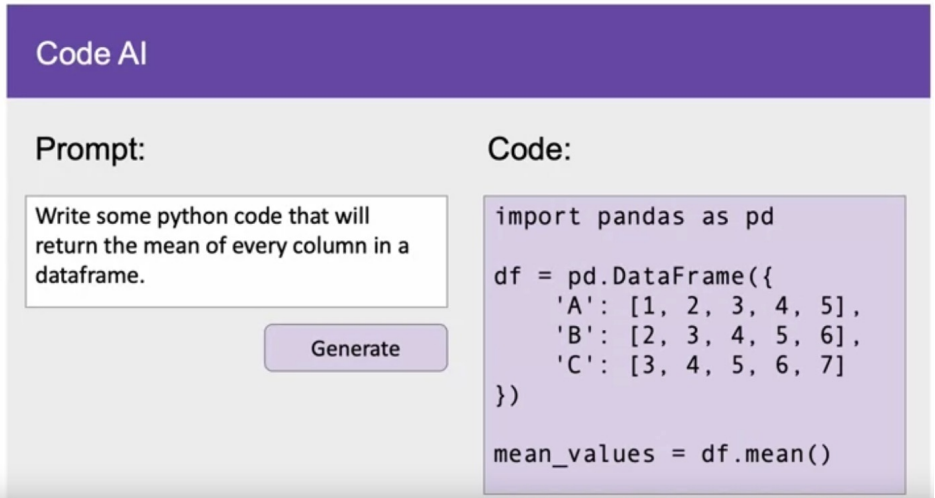

Speech recognition

"The quick brown fox jumped over the lazy dog."

# Generative AI (since 2020s)

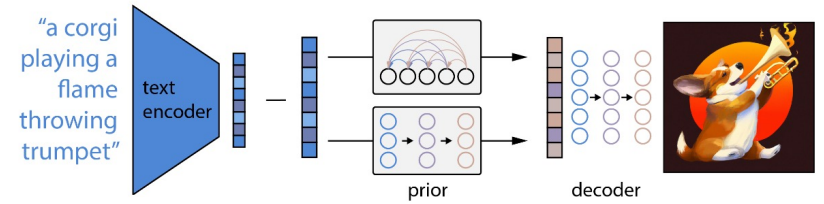
**Generative AI :** a set of Machine Learning models able to create content (image, text, music or speech) that mimics or approximates human ability.

LLMs (Large Language Models)




Source: Generative AI with Large Language Models

Text to image generation models :




# Pre-trained networks

- Training a model from scratch takes days to weeks.
- Many pre-trained models are publicly available: example ImageNet (over 15M anotated images).

**Transfer Learning:**

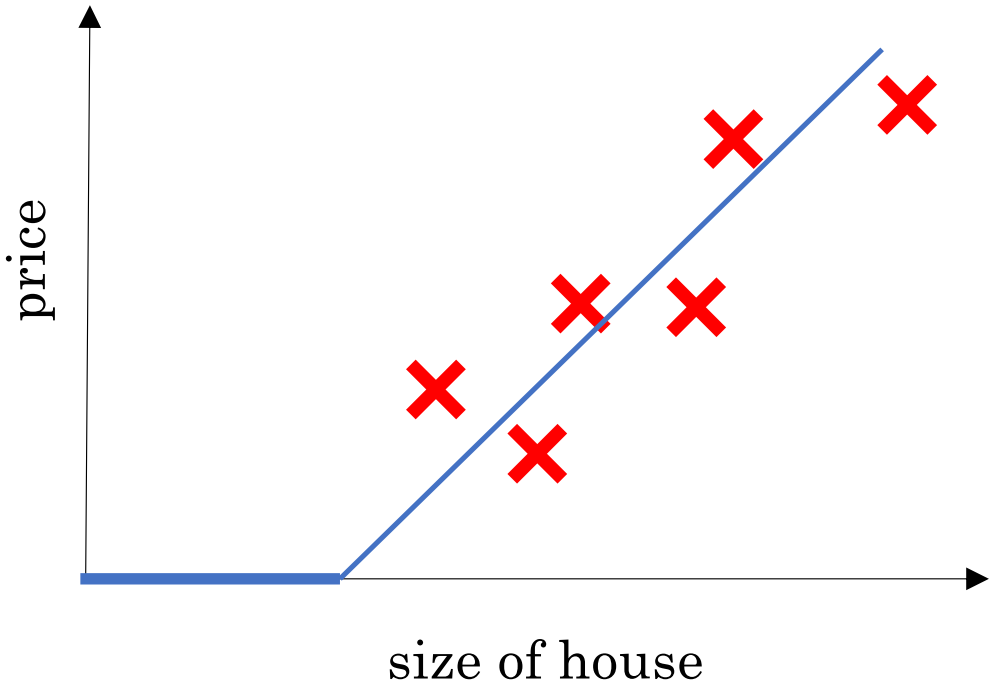
- Use pre-trained weights, modify last layers.
- Train a classification model from these features on a new classification task

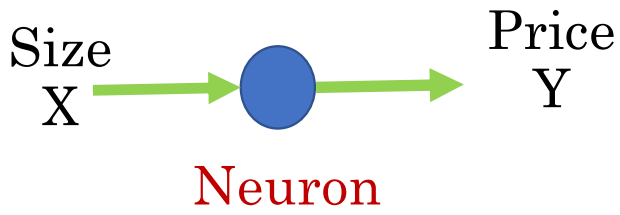

Andrew Ng "Deep Learning Specialization".
https://fr.coursera.org/specializations/deep-learning

# Neural Networks

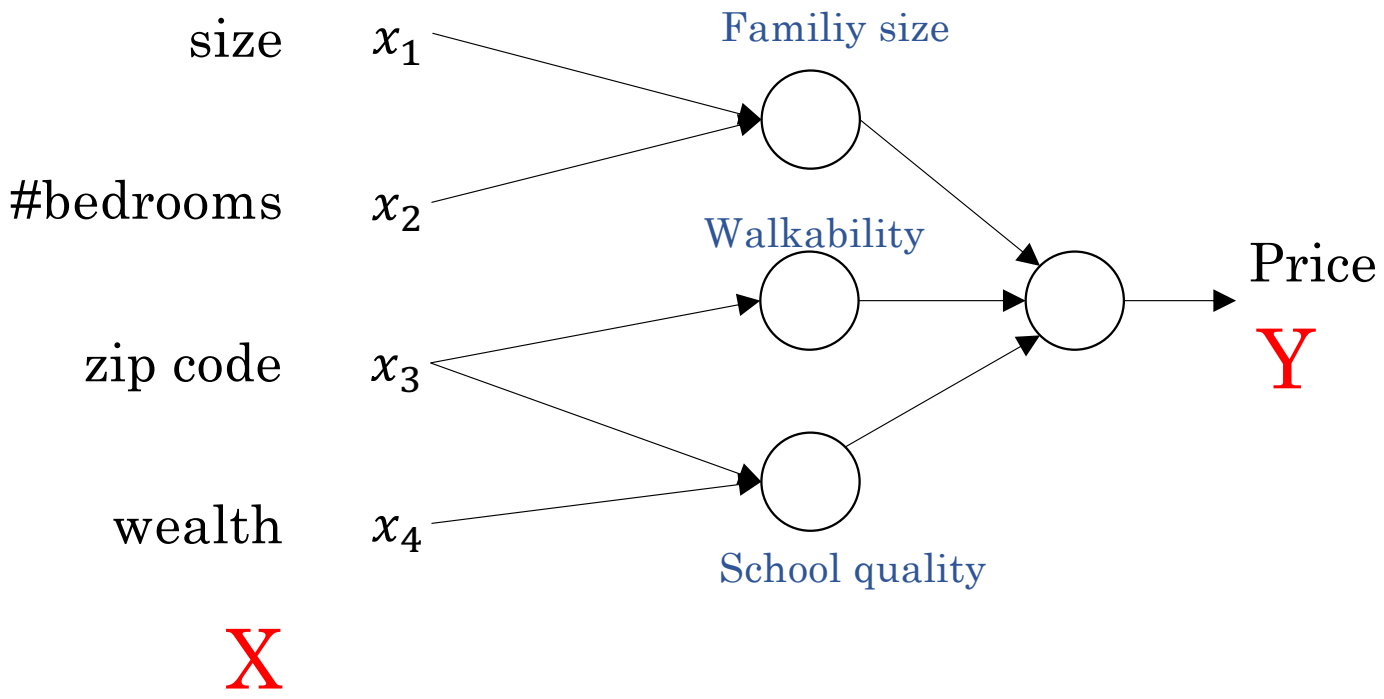
# What is a “Neural Network”?

## Housing Price Prediction (1/3)

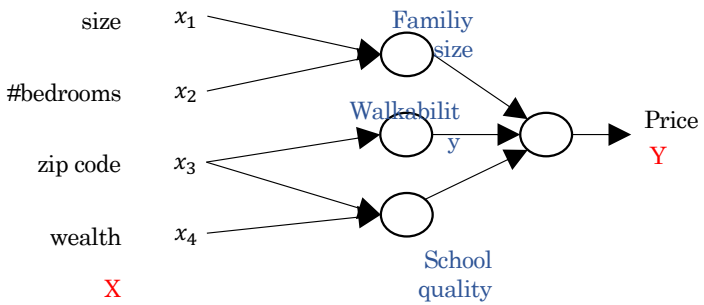


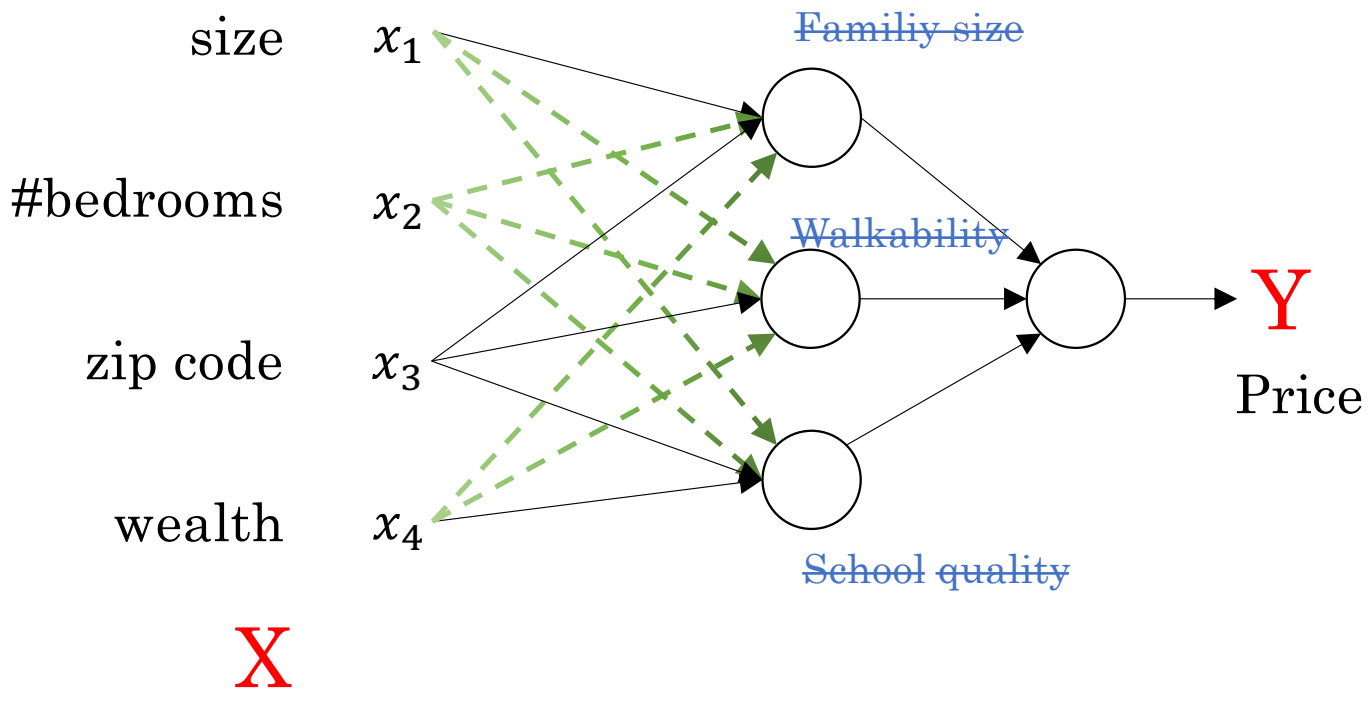



# Housing Price Prediction (2/3)



# Housing Price Prediction (3/3)



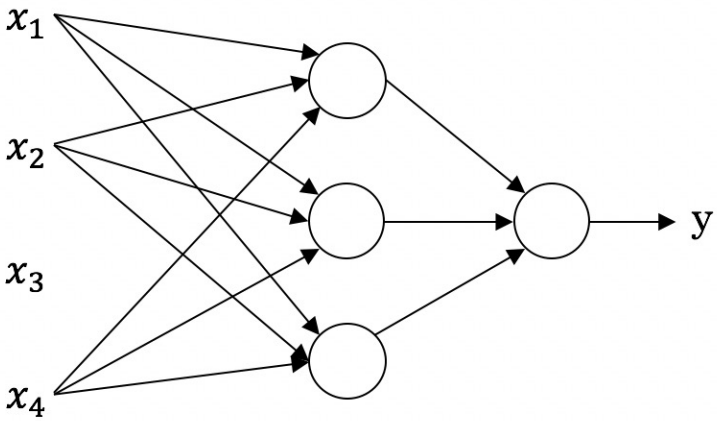



# Neural Networks and applications

| Input(x) | Output (y) | Application | |
|---|---|---|---|
| Home features | Price | Real Estate | Standard NN |
| Advertising, user info | Click on ad? (0/1) | Online Advertising | |
| Image | Object (1,…,1000) | Photo tagging | CNN |
| Audio clip | Text transcript | Speech recognition | RNN |
| English | Chinese | Machine translation | |
| Image, Radar info | Position of other cars | Autonomous driving | Hybrid/ custom NN |

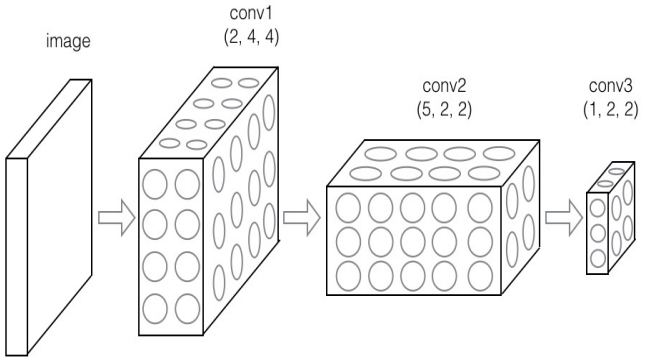
Andrew Ng "Deep Learning Specialization".
https://fr.coursera.org/specializations/deep-learning

# Neural Network examples



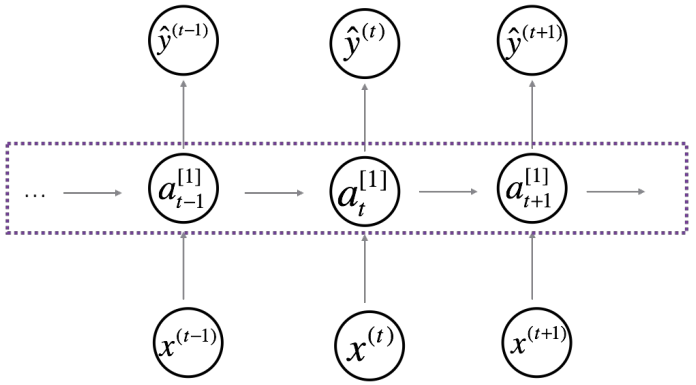

Standard NN

**Image**




Convolutional NN

**sequence data**




Recurrent NN

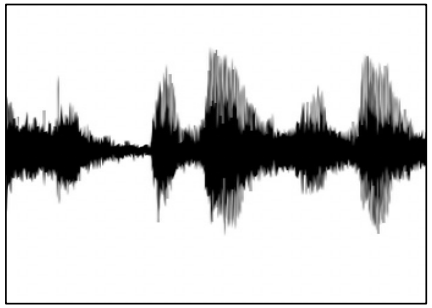
# Structured vs Unstructured data

## Structured Data

| Size | #bedrooms | ⋯ | Price (1000$s) |
|---|---|---|---|
| 2104 | 3 | | 400 |
| 1600 | 3 | | 330 |
| 2400 | 3 | | 369 |
| ⋮ | ⋮ | | ⋮ |
| 3000 | 4 | | 540 |

| User Age | Ad Id | ⋯ | Click |
|---|---|---|---|
| 41 | 93242 | | 1 |
| 80 | 93287 | | 0 |
| 18 | 87312 | | 1 |
| ⋮ | ⋮ | | ⋮ |
| 27 | 71244 | | 1 |

## Unstructured Data



Audio

Image

Four scores and seven years ago…

Text

# Logistic regression

# Binary Classification example

64

64

Blue
Green
Red

| | | | | |
|---|---|---|---|---|
| | | 255 | 134 | 93 | 22 |
| | 255 | 134 | 202 | 22 | 2 |
| 255 | 231 | 42 | 22 | 4 | 30 |
| 123 | 94 | 83 | 2 | 192 | 124 |
| 34 | 44 | 187 | 92 | 34 | 142 |
| 34 | 76 | 232 | 124 | 94 | |
| 67 | 83 | 194 | 202 | | |

Y

1 (cat) vs. 0 (non cat)

$$X = \begin{bmatrix} 255 \\ 231 \\ \vdots \\ 255 \\ 134 \\ \vdots \end{bmatrix}$$

$n_x = n = 64 \times 64 \times 3 = 12228$

Andrew Ng "Deep Learning Specialization".
https://fr.coursera.org/specializations/deep-learning

# Notation

Notation: superscript round bracket *(i)* to denote example *(i)*

$\mathbf{x} \in \mathbb{R}^{n_x}, \quad \mathbf{y} \in \{1, 0\}$

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

$$\mathbf{X} = \begin{bmatrix} | & | & \cdots & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & \cdots & | \end{bmatrix} \qquad \mathbf{Y} = \begin{bmatrix} \{y^{(1)}, y^{(2)}, \ldots, y^{(m)}\} \end{bmatrix}$$

$$\mathbf{X} \in \mathbb{R}^{n_x \times m} \qquad \mathbf{Y} \in \mathbb{R}^{1 \times m}$$

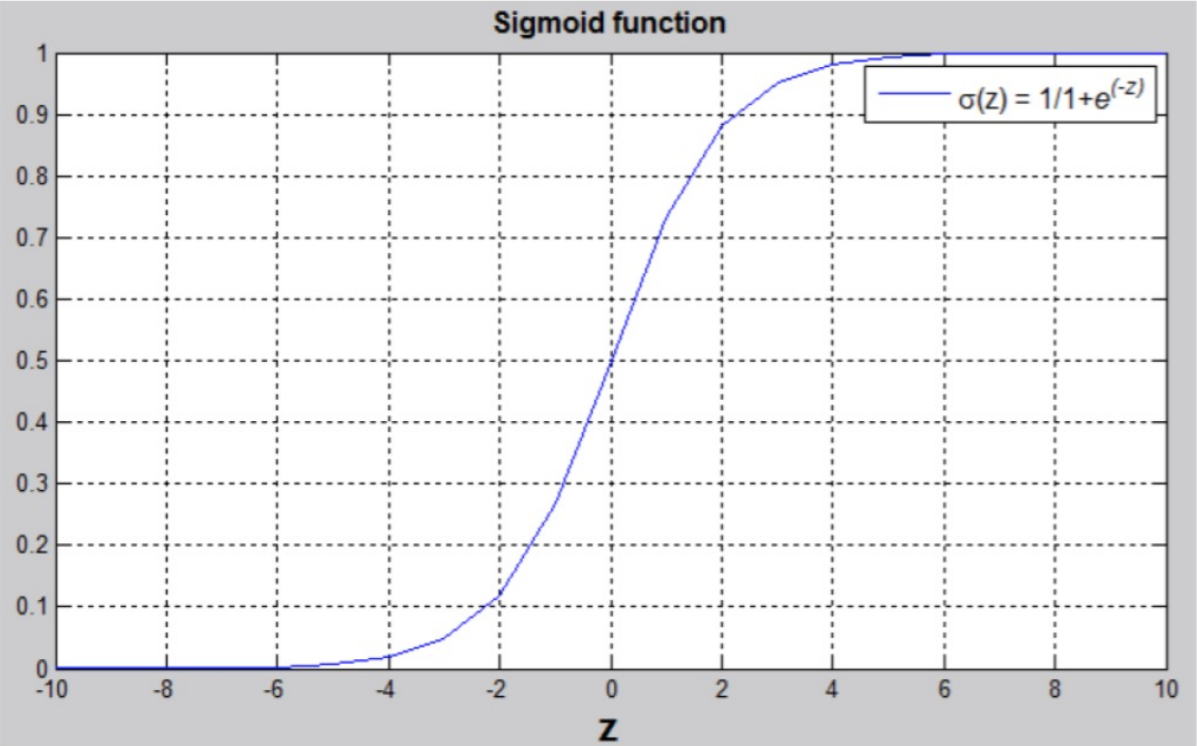
# Logistic Regression (1/2)

$$Given\ x\ ,\ \hat{y} = P(y = 1|x), \text{where } 0 \leq \hat{y} \leq 1$$

The parameters used in Logistic regression are:

- The input features vector: $x \in \mathbb{R}^{n_x}$, where $n_x$ is the number of features
- The training label: $y \in 0,1$
- The weights: $w \in \mathbb{R}^{n_x}$, where $n_x$ is the number of features
- The threshold: $b \in \mathbb{R}$
- The output: $\hat{y} = \sigma(w^T x + b)$
- Sigmoid function: s = $\sigma(w^T x + b) = \sigma(z) = \frac{1}{1+e^{-z}}$

# Logistic Regression (2/2)




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

- If $z$ large $\sigma(z) \approx 1$
- If $z$ large negative $\sigma(z) \approx 0$

The Sigmoid function is bounded between [0,1].

Andrew Ng "Deep Learning Specialization".
https://fr.coursera.org/specializations/deep-learning

# Logistic Regression cost function

$\hat{y} = \sigma(w^T x + b)$, where $\sigma(z) = \frac{1}{1+e^{-z}}$

Given $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$, want $\hat{y}^{(i)} \approx y^{(i)}$.

**Loss (error) function "cross-entropy"** for an observation $(i)$:

$$L(\hat{y}^{(i)}, y^{(i)}) = -(y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}))$$

Intuition:

- If $y^{(i)} = 1$: $L(\hat{y}^{(i)}, y^{(i)}) = -\log(\hat{y}^{(i)})$ where $\log(\hat{y}^{(i)})$ and $\hat{y}^{(i)}$ should be close to 1
- If $y^{(i)} = 0$: $L(\hat{y}^{(i)}, y^{(i)}) = -\log(1 - \hat{y}^{(i)})$ where $\log(1 - \hat{y}^{(i)})$ and $\hat{y}^{(i)}$ should be close to 0

**Cost function** for the entire dataset:

$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^{m} [(y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

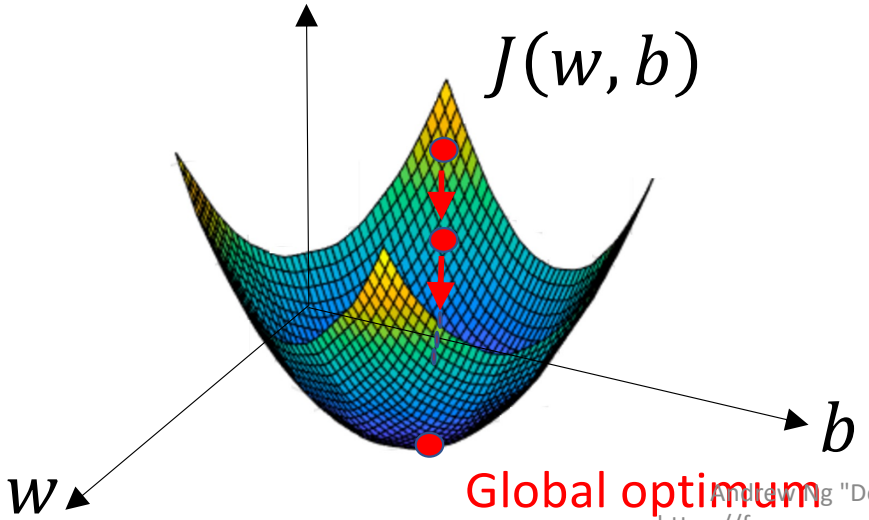
Andrew Ng "Deep Learning Specialization", https://fr.coursera.org/specializations/deep-learning

# Gradient Descent (1/2)

Recap: $\hat{y} = \sigma(w^T x + b), \ \sigma(z) = \frac{1}{1+e^{-z}}$

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

Want to find $w, b$ that minimize $J(w,b)$

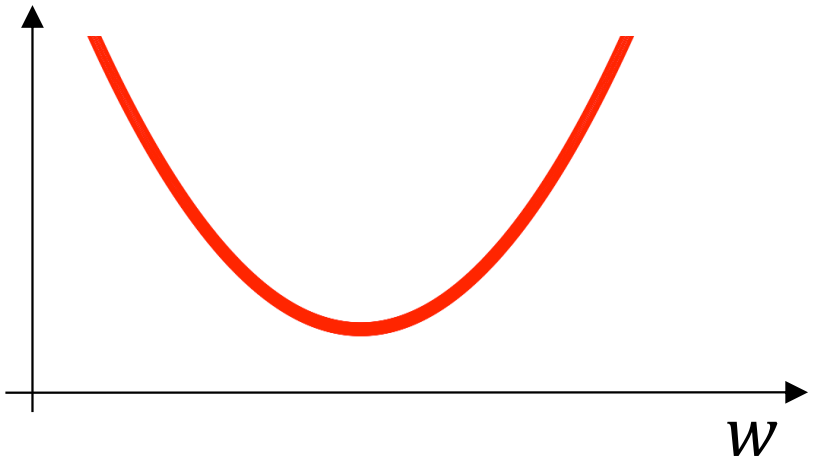



**J is a convex function**

*Randomly initialize w and b to some initial value. and then takes a step in the steepest downhill direction*

# Gradient Descent (2/2)

What gradient descent essentially does




$\alpha$ learning rate

Repeat {

$$w := w - \alpha \frac{dJ}{dw}$$

}

In the case of logistic regression there are 2 parameters:

$$w := w - \alpha \frac{dJ}{dw} \quad and \quad b := b - \alpha \frac{dJ}{db}$$

**Convention:**

$\frac{dJ}{dw}$ denoted dw

$\frac{dJ}{db}$ denoted db

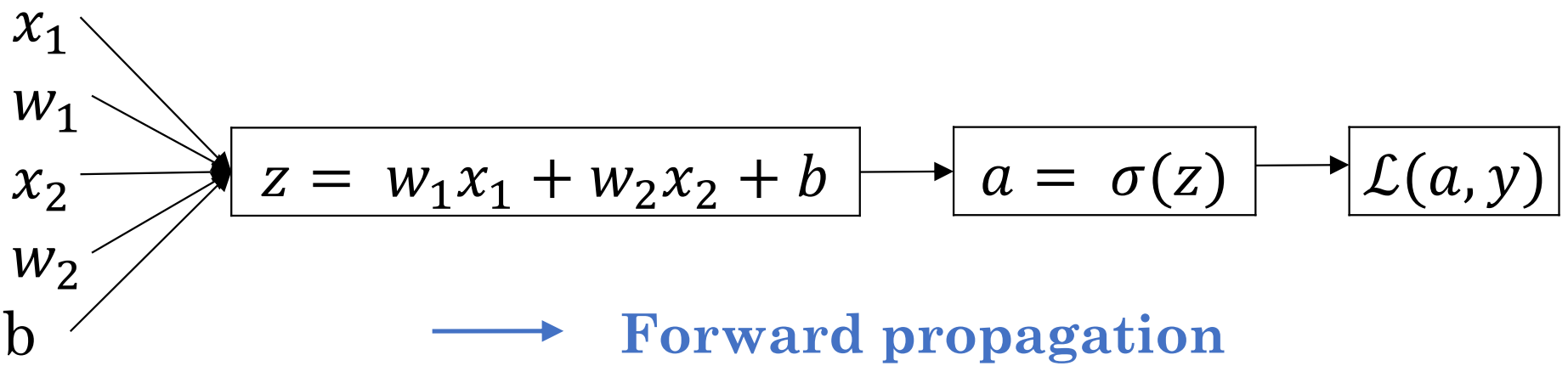
# Logistic Regression Gradient descent (1/2)

**Logistic regression recap**

$$z = w^T x + b$$

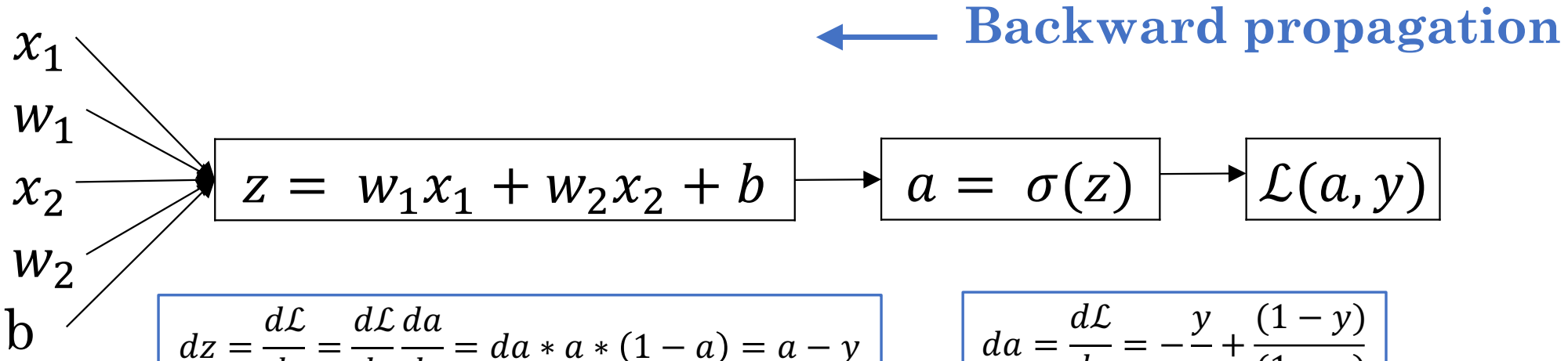
$$\hat{y} = a = \sigma(z)$$

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




**Forward propagation**

# Logistic Regression Gradient descent (2/2)




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

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

$$dw_1 = \frac{d\mathcal{L}}{da}\frac{da}{dz}\frac{dz}{dw_1} = x_1 * (dz)$$

$$dw_2 = \frac{d\mathcal{L}}{da}\frac{da}{dz}\frac{dz}{dw_2} = x_2 * (dz)$$

$$db = \frac{d\mathcal{L}}{da}\frac{da}{dz}\frac{dz}{db} = (dz)$$

$$w_1 := w_1 - \alpha\,(dw_1)$$
$$w_2 := w_2 - \alpha\,(dw_2)$$
$$b := b - \alpha\,(db)$$

# Gradient descent on $m$ examples (1/2)

Cosider the cost function:

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

where $\hat{y}^{(i)} = a^{(i)} = \sigma\left(z^{(i)}\right) = \sigma(w_1 x_1 + w_2 x_2 + b)$
for observation $(x^{(i)}, y^{(i)})$

The derivative of a sum is the sum of derivatives:

$$\frac{\partial J(w,b)}{\partial w_1} = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial \mathcal{L}(\hat{y}^{(i)}, y^{(i)})}{\partial w_1}$$

# Algorithm Gradient descent on $m$ examples (2/2)

Initialize: $J = 0, dw_1 = 0, dw_2 = 0, db = 0$ and $w_1, w_2, b$ randomly

**For** i **in** $1 \ldots m$:

{

$z^{(i)} = w_1 x_1^{(i)} + w_2 x_2^{(i)} + b,$

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

$J \mathrel{+}= -(y^{(i)} \log(a^{(i)}) + (1 - y^{(i)}) \log(1 - a^{(i)}))$

$dz^{(i)} \mathrel{+}= (a^{(i)} - y^{(i)})$

$dw_1 \mathrel{+}= x_1^{(i)} dz$

$dw_2 \mathrel{+}= x_2^{(i)} dz$

$db \mathrel{+}= dz^{(i)}$

}

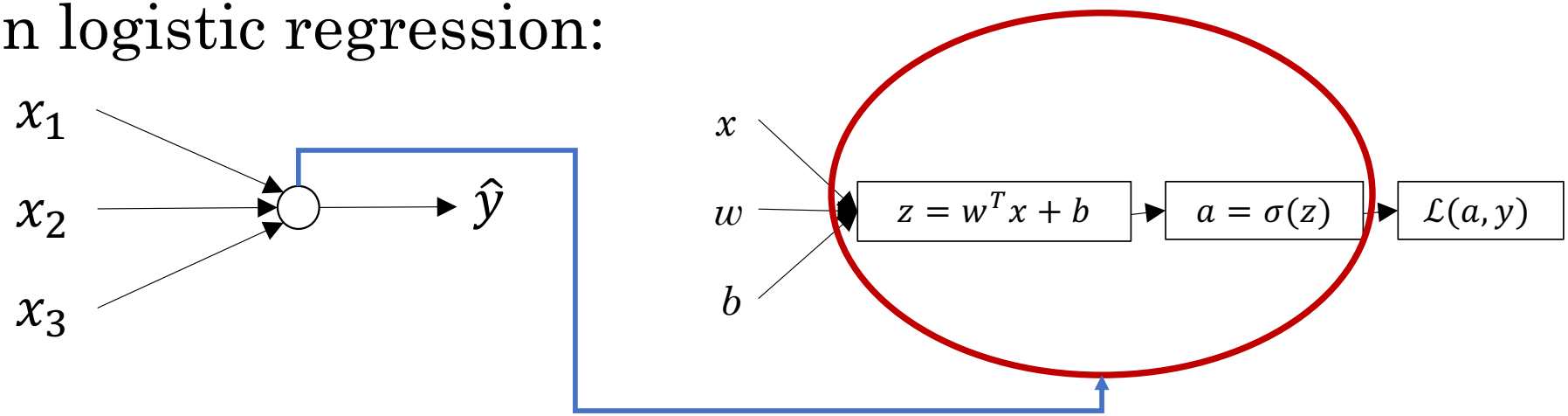
$$J := \frac{J}{m}\ ; dw_1 := \frac{dw_1}{m}\ ; dw_2 := \frac{dw_2}{m}; db := \frac{db}{m}$$

Update:

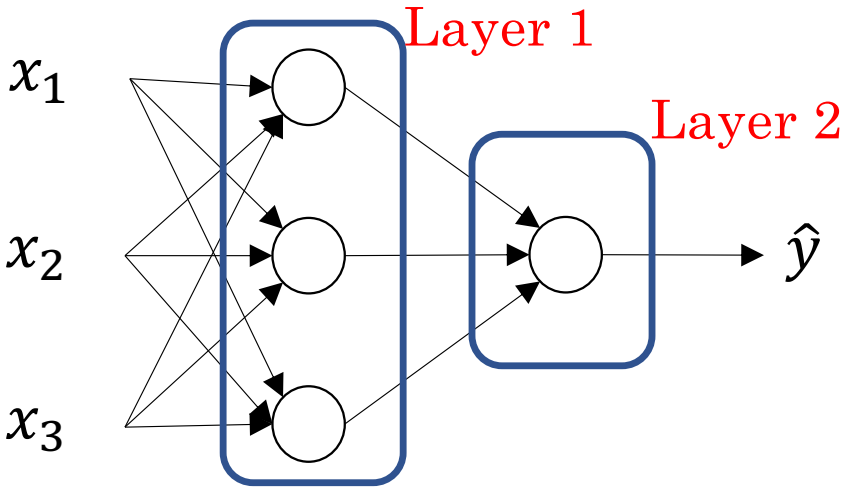
$$w_1 := w_1 - \alpha\ (dw_1)$$
$$w_2 := w_2 - \alpha\ (dw_2)$$
$$b := b - \alpha\ (db)$$

# One hidden layer Neural Network

In logistic regression:




In a neural network:

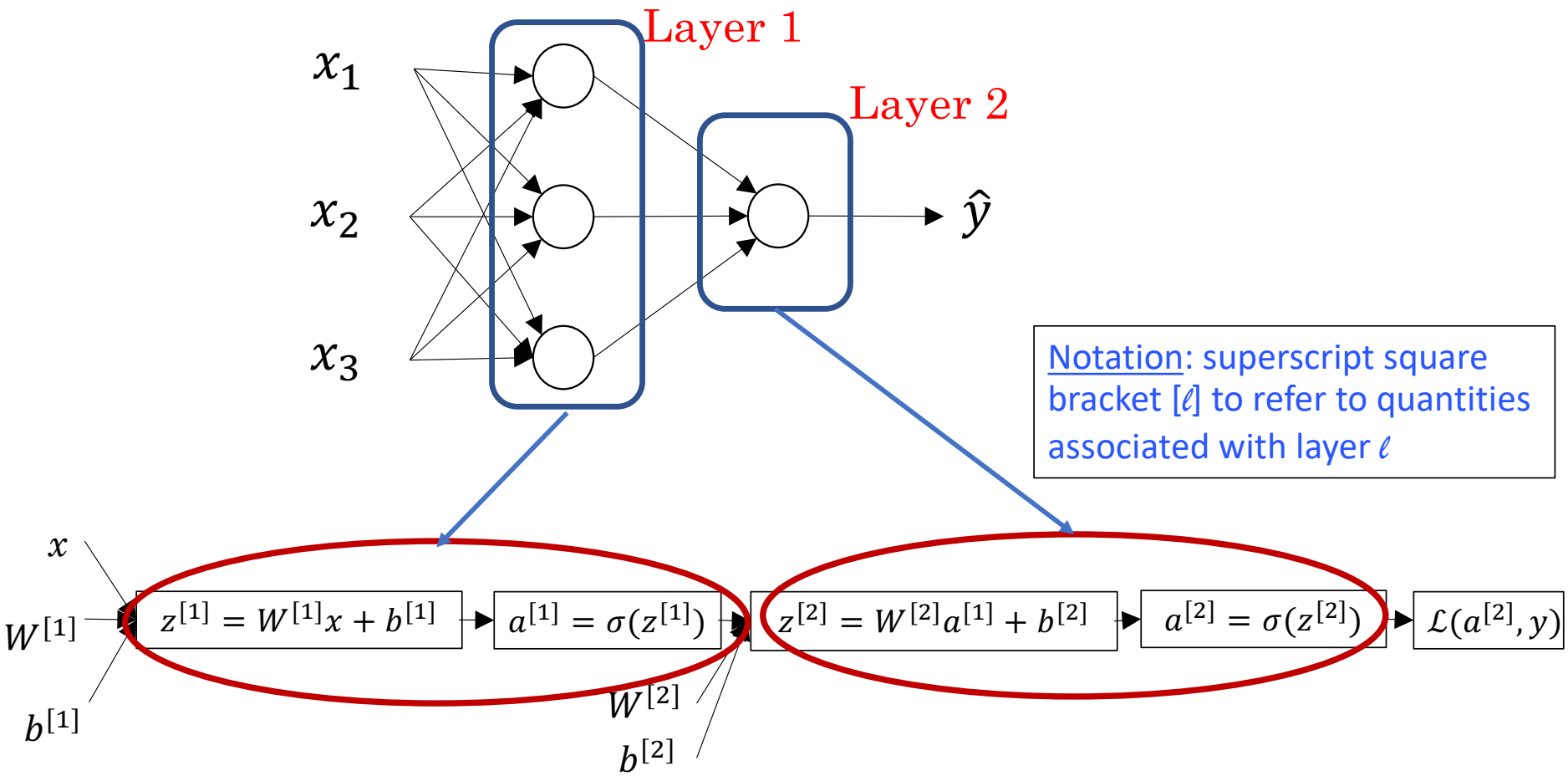



Each stack of nodes, called **layer** represents the **2** steps of calculation!

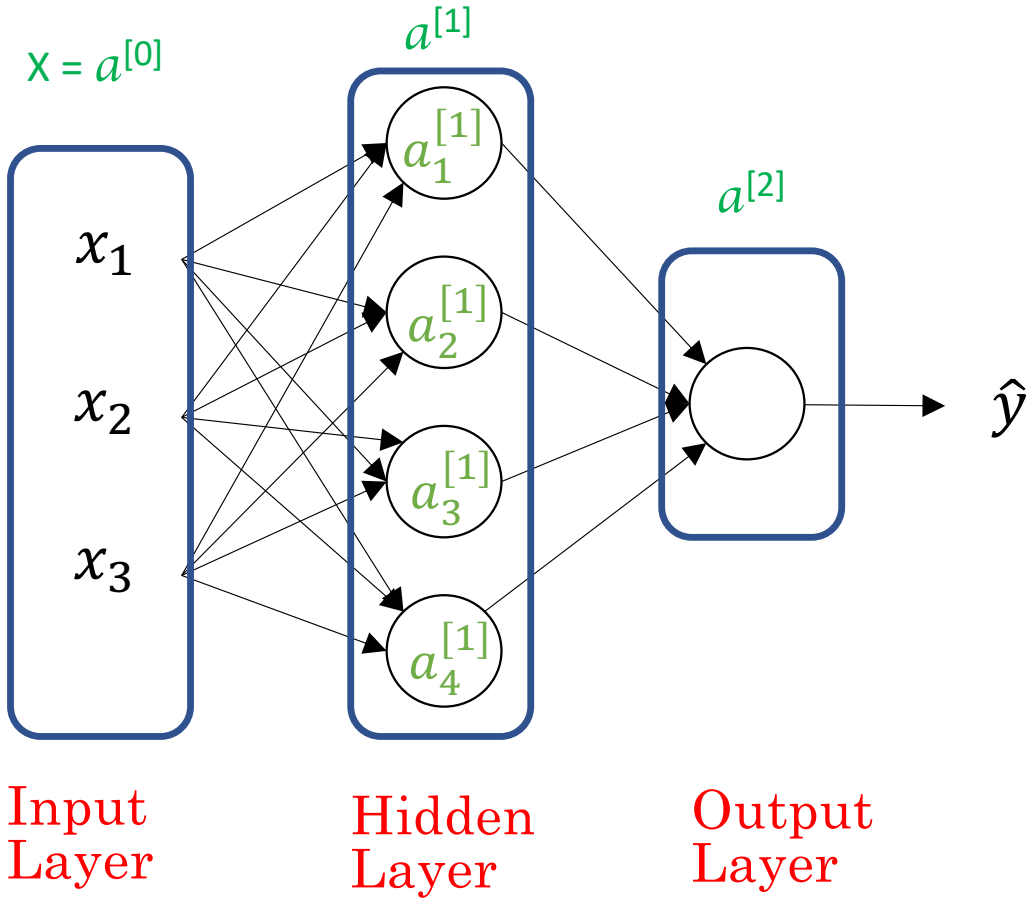


Andrew Ng "Deep Learning Specialization".
https://fr.coursera.org/specializations/deep-learning

# Neural network details



# Example: 2-layer NN




$a$: activation

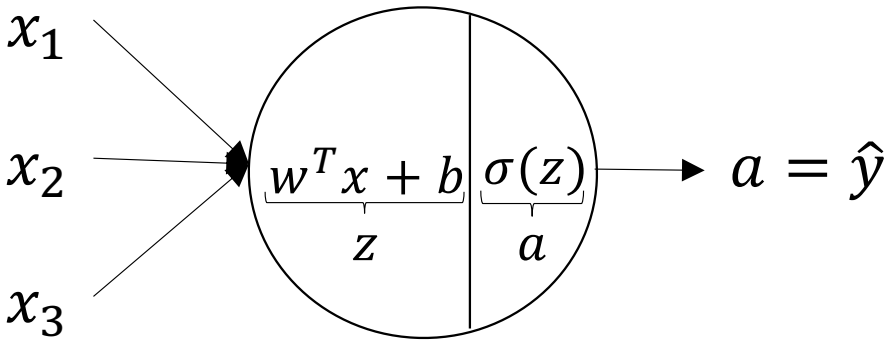
Notation:

$a_i^{[\ell]}$ node $i$ from layer $\ell$

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

# Computing a Neural Network's Output (1/4)

Logistic regression

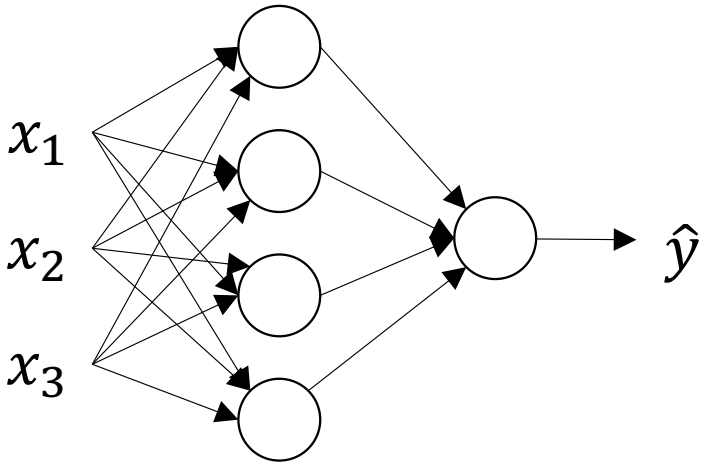



$$z = w^T x + b$$

$$a = \sigma(z)$$

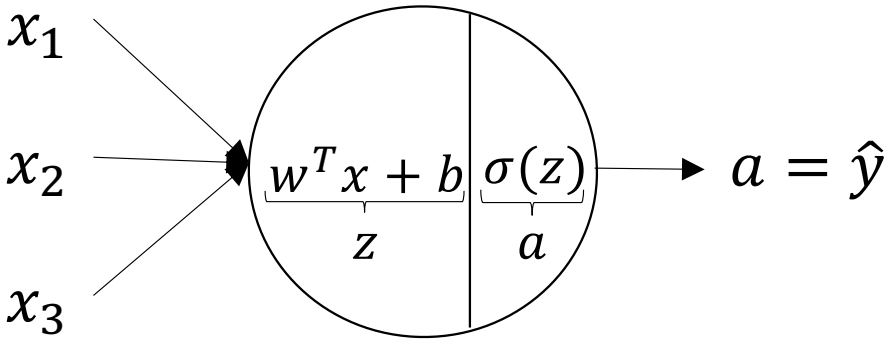
2 steps of calculation

2 − layer NN



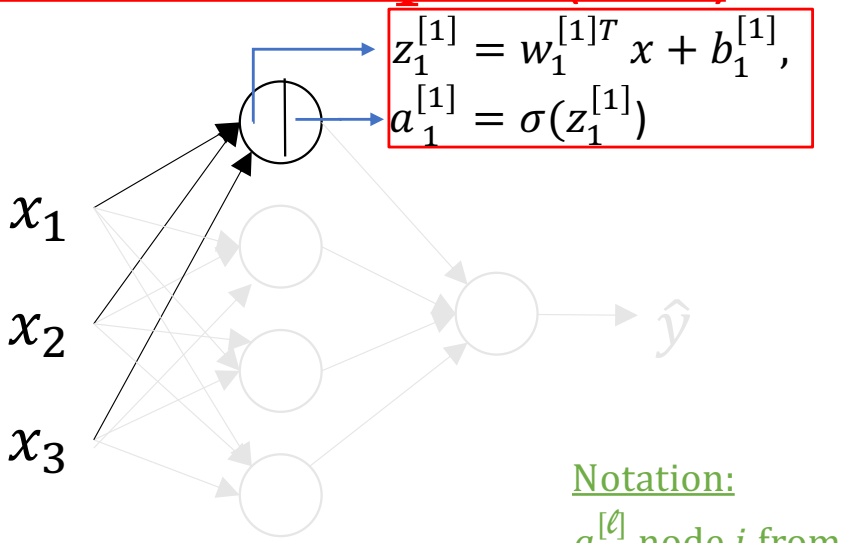

A NN repeats this process lot more times

# Computing a Neural Network's Output (2/4)



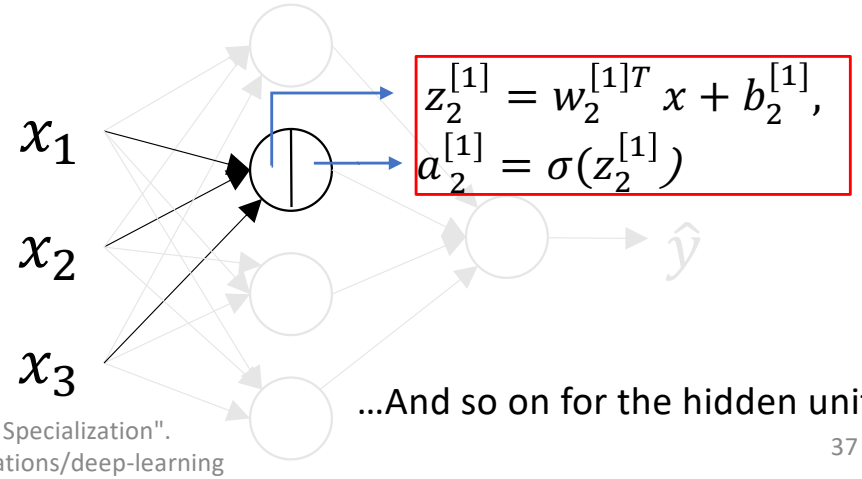

$$z = w^T x + b$$

$$a = \sigma(z)$$



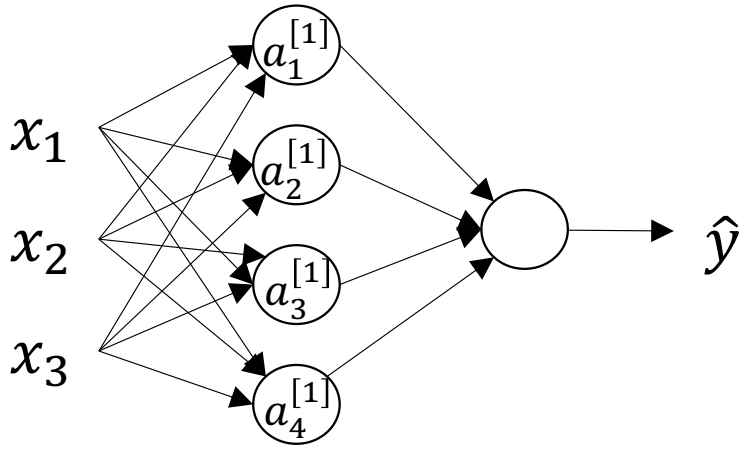

Notation:

$a_i^{[\ell]}$ node $i$ from layer $\ell$




...And so on for the hidden units 3 and 4.

# Computing a Neural Network's Output (3/4)




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

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

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

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

=

*Matrix notation:*

$$z^{[1]} = \begin{pmatrix} w_1^{[1]T} \\ w_2^{[1]T} \\ w_3^{[1]T} \\ w_4^{[1]T} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{pmatrix} = \begin{pmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{pmatrix} = \begin{pmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{pmatrix}$$

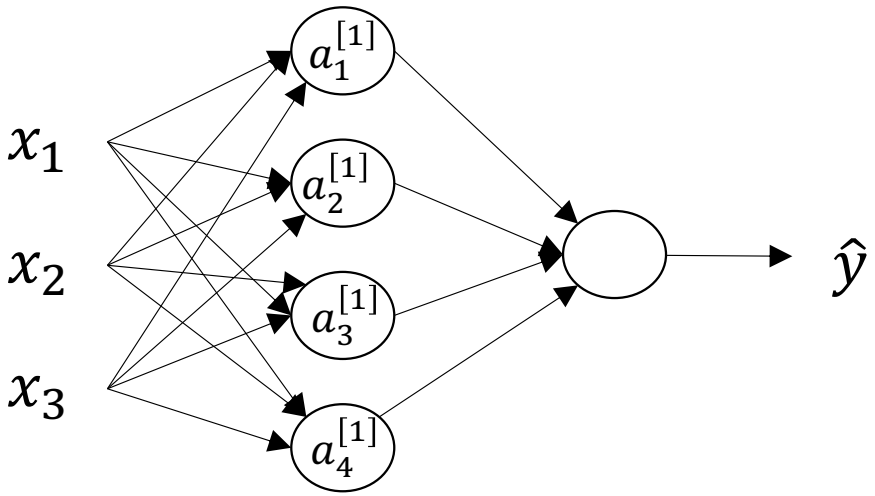
$$a^{[1]} = \sigma \begin{pmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{pmatrix} = \begin{pmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{pmatrix}$$

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

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

Andrew Ng "Deep Learning Specialization".
https://fr.coursera.org/specializations/deep-learning

# Computing a Neural Network's output (4/4)




Given input x:

$a^{[0]}$

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

(4,1) (4,3)x(3,1) + (4,1)

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

(4,1) (4,1)

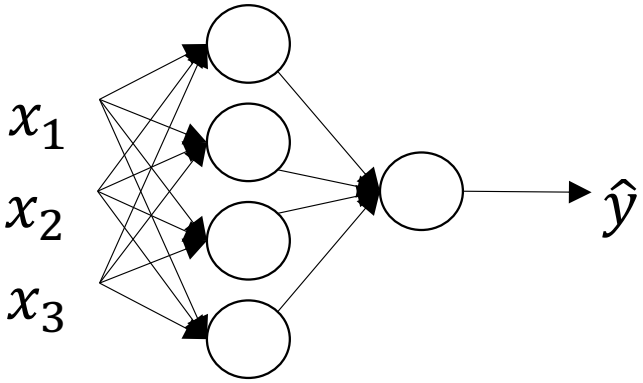
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

(1,1) (1,4)x(4,1) + (1,1)

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

(1,1) (1,1)

# 2-layer NN with $m$ examples (1/2)




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

For only one training example

In the case of $m$ examples: **for loop**

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

Notation:

$a^{[\ell](i)}$

layer $\ell$ example $i$

# 2-layer NN with $m$ examples (2/2)

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

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

Recall: matrix $\mathbf{X}$

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \qquad \mathbf{X} \in \mathbb{R}^{n_x \times m}$$

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

Similarly stack $\mathbf{z}$ **and** $a$ in columns:

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

Hidden units

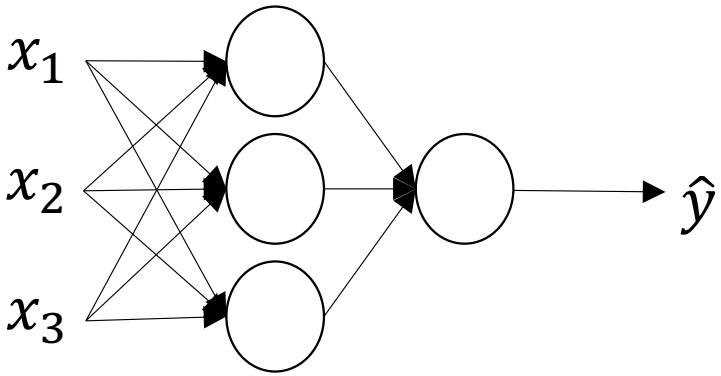
$$\mathrm{A}^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

Hidden units

training examples

# Activation functions

# Activation functions




$g^{[\ell]}$ activation function of layer $\ell$

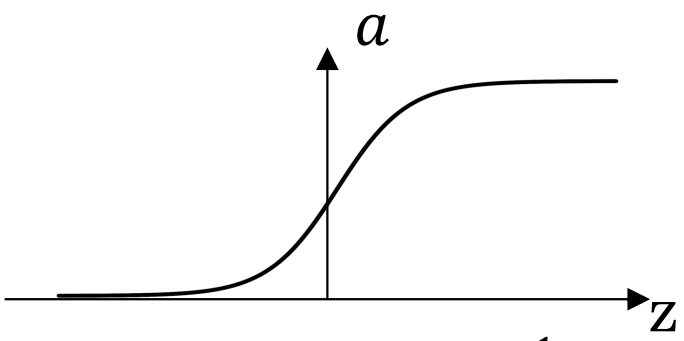
Given x:

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

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

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

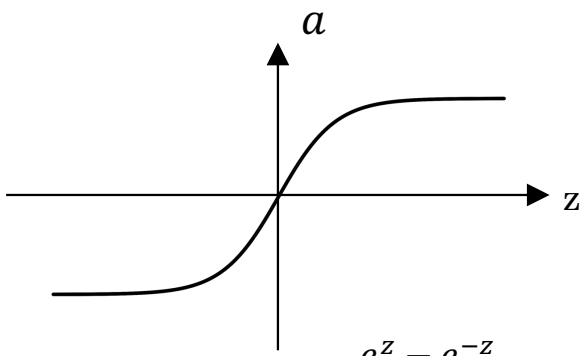
$$a^{[2]} = g^{[2]}(z^{[2]})$$




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

# *tanh* and ReLU Activation functions

**tanh function (**hyperbolic tangent function)

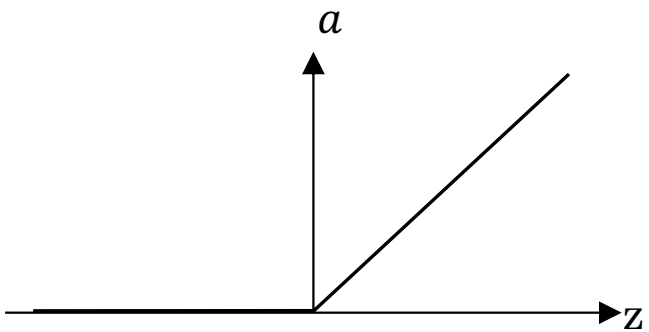



tanh: $a = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

- **Shifted version** of the sigmoid function
- For hidden units almost always works better than sigmoid function. Because the output values have nearly mean 0.
- Prefer sigmoid when the output value must lay between 0 and 1, like in the output layer

Downsides of both, when *z* very large or very small, the derivative becomes very small and this can **slow down** gradient descent!

**ReLU** (Rectified Linear unit)

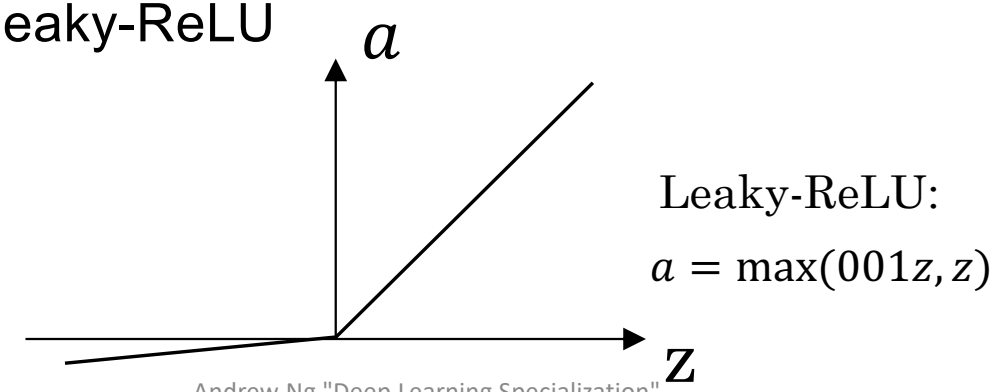



ReLU: $a = \max(0, z)$

- The derivative is 1 if Z>0 and derivative or the slope is 0 when Z<0.
- The NN learns much faster
- Very popular in Machine Learning

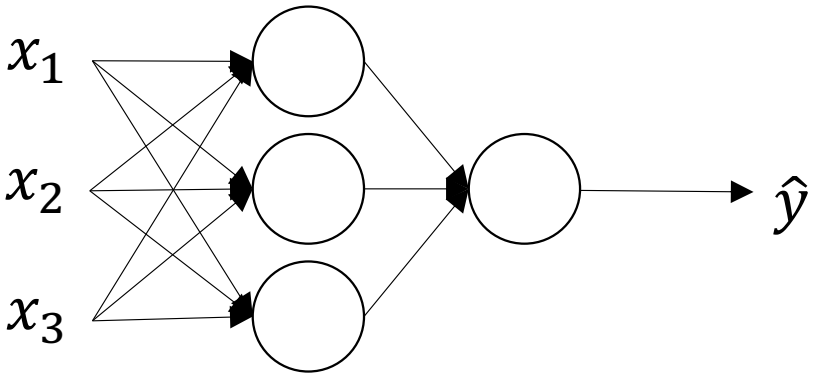
# What activation function to choose?

- **Sigmoid activation function**: use this for the output layer for binary classification.
- ***tanh* activation function*:*** pretty much strictly superior than sigmoid in hidden layers.
- ***ReLU:*** default the most commonly used activation function.
- You can also try leaky-ReLU




Andrew Ng "Deep Learning Specialization".
https://fr.coursera.org/specializations/deep-learning

# Why not linear activation functions?




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

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

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

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

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

Given x:

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

*Linear activation function:*

$g(z) = z$

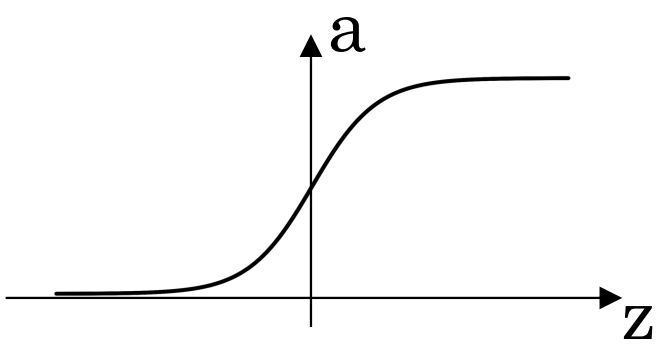
The composition of two linear functions is itself a linear function!

hidden layers are not necessary!

EXCEPTION: for regression, linear activation function in the output layer ONLY.

# Derivatives of activation functions (1/3)

## Sigmoid activation function



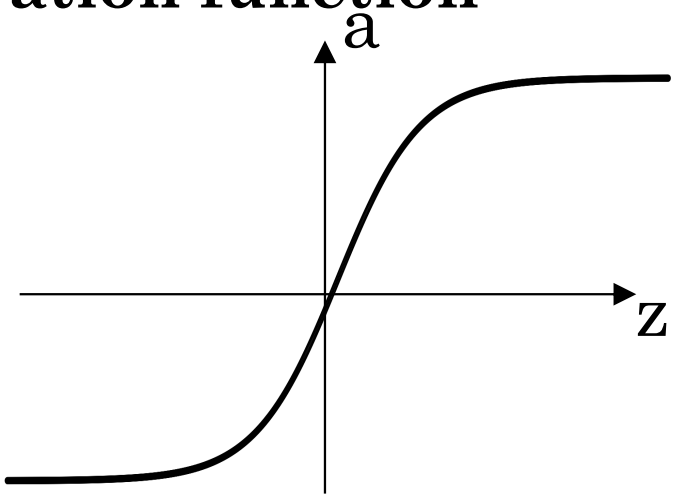

$$g(z) = \frac{1}{1+e^{-z}}$$

$$g'(z) = \frac{dg}{dz} = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right)$$
$$= g(z)(1-g(z))$$
$$= a(1-a)$$

$$g'(z) = a(1-a)$$

# Derivatives of activation functions (2/3)

**_tanh_ activation function**




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

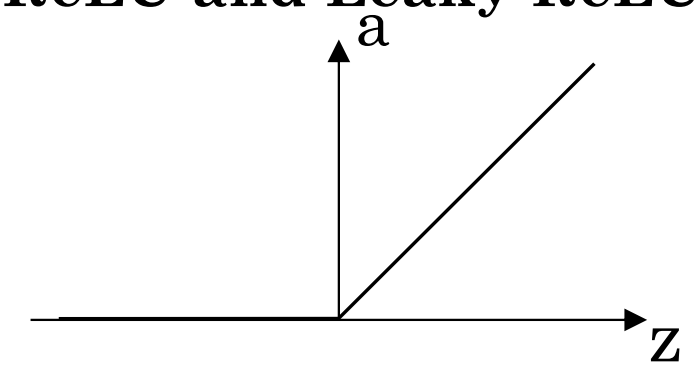
$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

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

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

# Derivatives of activation functions (3/3)

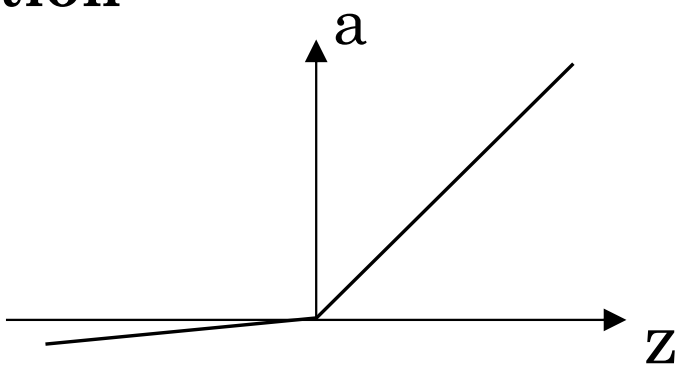
**ReLU and Leaky ReLU activation function**




ReLU

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

$$g'(z) = \begin{cases} 1 \text{ if } z > 0 \\ 0 \text{ if } z < 0 \end{cases}$$




Leaky ReLU

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

$$g'(z) = \begin{cases} 1 \text{ if } z > 0 \\ 0.01 \text{ if } z < 0 \end{cases}$$

The derivative is not defined for 0, but for implementation set it to 0

# Gradient descent for neural networks

# Gradient descent for neural networks

Parameters of our NN with one single hidden layer:

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

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

Output units of layers

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

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

In order to minimize the cost function, we will use ***Gradient descent:***

**Repeat**:{

Calculate $\hat{y}^{(i)}$ for i = 1,....,m

$$dw^{[1]} = \frac{\partial \mathcal{L}}{\partial w^{[1]}}, db^{[1]} = \frac{\partial \mathcal{L}}{\partial b^{[1]}}$$

Update $w^{[1]} := w^{[1]} - \alpha\left(dw^{[1]}\right)$ and $b^{[1]} := b^{[1]} - \alpha\left(db^{[1]}\right)$

Idem for layer 2...}

Andrew Ng "Deep Learning Specialization".
https://fr.coursera.org/specializations/deep-learning

# Formulas for computing derivatives for 2-layer NN example

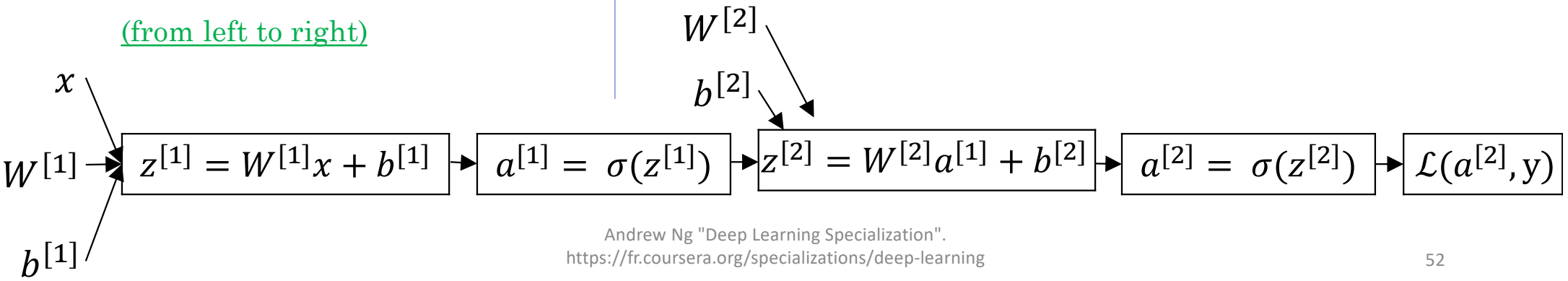
## Forward propagation

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

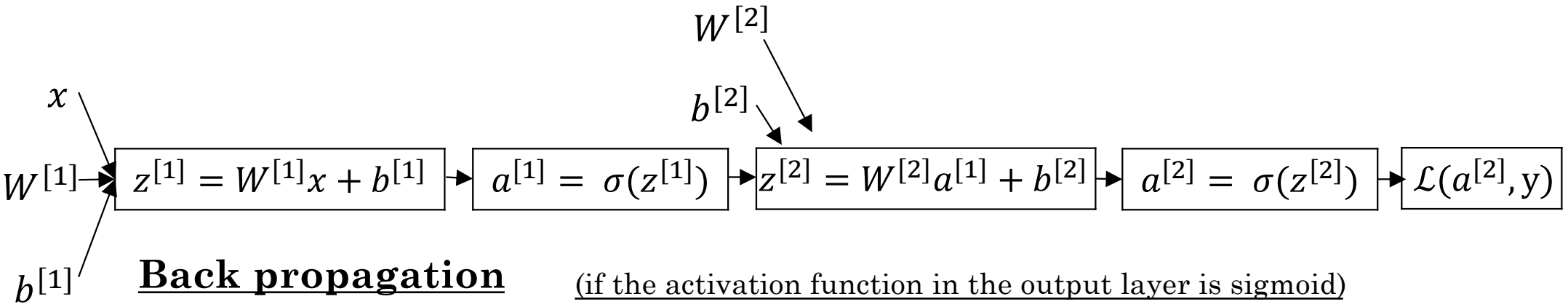
(from left to right)

## Back propagation

(from right to left)



# Formulas for computing derivatives for 2-layer NN example




## Back propagation

(if the activation function in the output layer is sigmoid)

$Y = (y^{(1)}, \dots, y^{(m)})$

Layer 2

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

$$dW^{[2]} = \frac{dZ^{[2]}A^{[1]T}}{m} \qquad db^{[2]} = \frac{\sum_{i=1}^{m} dZ^{[2](i)}}{m}$$

elementwise product

Layer 1

$$dZ^{[1]} = \left(W^{[2]T}dZ^{[2]}\right) * g'^{[1]}(Z^{[1]})$$

$$dW^{[1]} = \frac{dZ^{[1]}A^{[0]T}}{m} = \frac{dZ^{[1]}X^{T}}{m} \qquad db^{[1]} = \frac{\sum_{i=1}^{m} dZ^{[1](i)}}{m}$$

# Summary of derivatives formulas for 2-layer NN example

## Forward propagation

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

(from left to right) ⟶

## Back propagation

(if the activation function in the output layer is sigmoid)

$$dZ^{[2]} = (A^{[2]} - Y) \qquad Y = (y^{(1)}, \dots, y^{(m)})$$

(1,m)

$$dW^{[2]} = \frac{dZ^{[2]}A^{[1]T}}{m}$$ Dimension $(n^{[2]}, n^{[1]})$

$$db^{[2]} = \frac{\sum_{i=1}^{m} dZ^{[2](i)}}{m}$$ Dimension $(n^{[2]}, 1)$

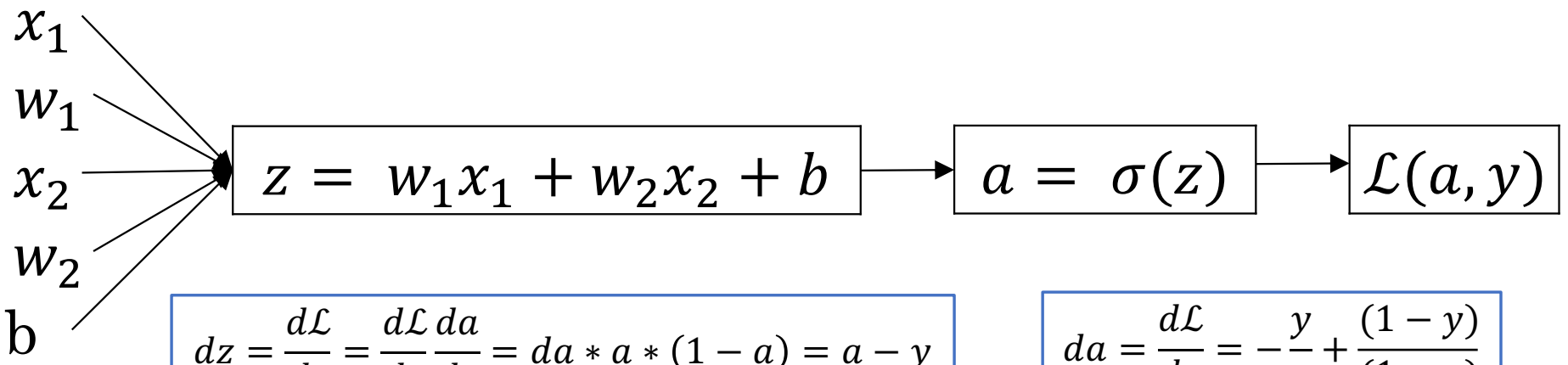
$$dZ^{[1]} = \left(W^{[2]T}dZ^{[2]}\right) * g'^{[1]}(Z^{[1]})$$

$$dW^{[1]} = \frac{dZ^{[1]}A^{[0]T}}{m} = \frac{dZ^{[1]}X^{T}}{m}$$

$$db^{[1]} = \frac{\sum_{i=1}^{m} dZ^{[1](i)}}{m}$$

# Backpropagation intuition proof (1/3)

*Reminder* logistic regression backpropagation




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

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

$$dw_1 = \frac{d\mathcal{L}}{da}\frac{da}{dz}\frac{dz}{dw_1} = x_1 * (dz)$$

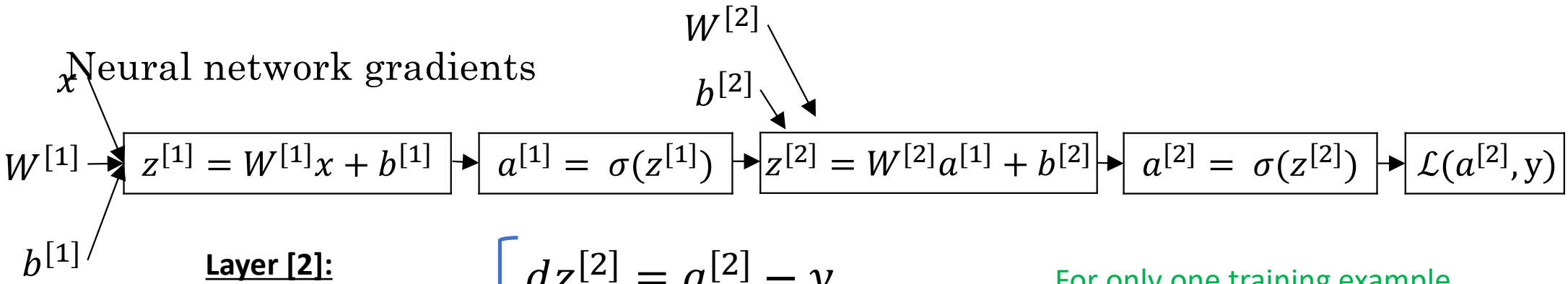
$$dw_2 = \frac{d\mathcal{L}}{da}\frac{da}{dz}\frac{dz}{dw_2} = x_2 * (dz)$$

$$db = \frac{d\mathcal{L}}{da}\frac{da}{dz}\frac{dz}{db} = (dz)$$

Computation of derivatives for the 2nd layer of our NN is similar

# Backpropagation intuition proof (2/3)

Neural network gradients




**Layer [2]:**
Very similar to the logistic regression network

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

For only one training example

**Layer [1]**

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

# Backpropagation intuition proof (3/3)

**For one observation**

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

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

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

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

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

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

**For *m* observations**

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

$$dW^{[2]} = \frac{dZ^{[2]} A^{[1]T}}{m} \qquad (n^{[2]}, n^{[1]})$$

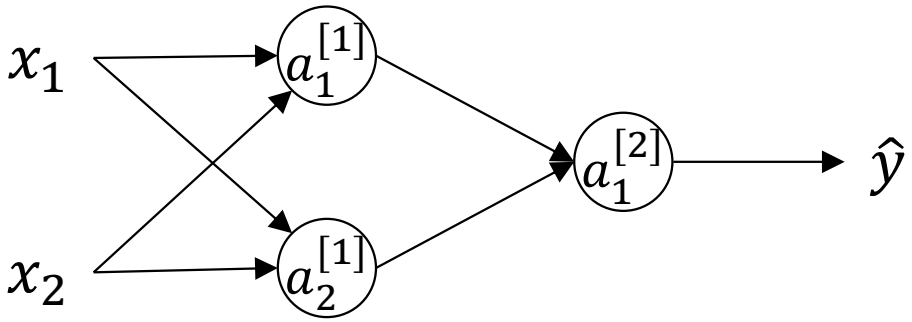
$$db^{[2]} = \frac{\sum_{i=1}^{m} dZ^{[2](i)}}{m} \qquad (n^{[2]}, 1)$$

$$dZ^{[1]} = \left(W^{[2]T} dZ^{[2]}\right) * g'^{[1]}(Z^{[1]}) \qquad (n^{[1]}, m)$$

$$dW^{[1]} = \frac{dZ^{[1]} A^{[0]T}}{m} = \frac{dZ^{[1]} X^{T}}{m} \qquad (n^{[1]}, n^{[0]})$$

$$db^{[1]} = \frac{\sum_{i=1}^{m} dZ^{[1](i)}}{m} \qquad (n^{[1]}, 1)$$

# Initialization: what happens if you initialize weights to zero?




PROBLEM: For any example $a_2^{[1]} = a_1^{[1]}$. During backpropagation, $dz_1^{[1]} = dz_2^{[1]}$ (by symmetry). All the rows of matrix: W will take on the same value.

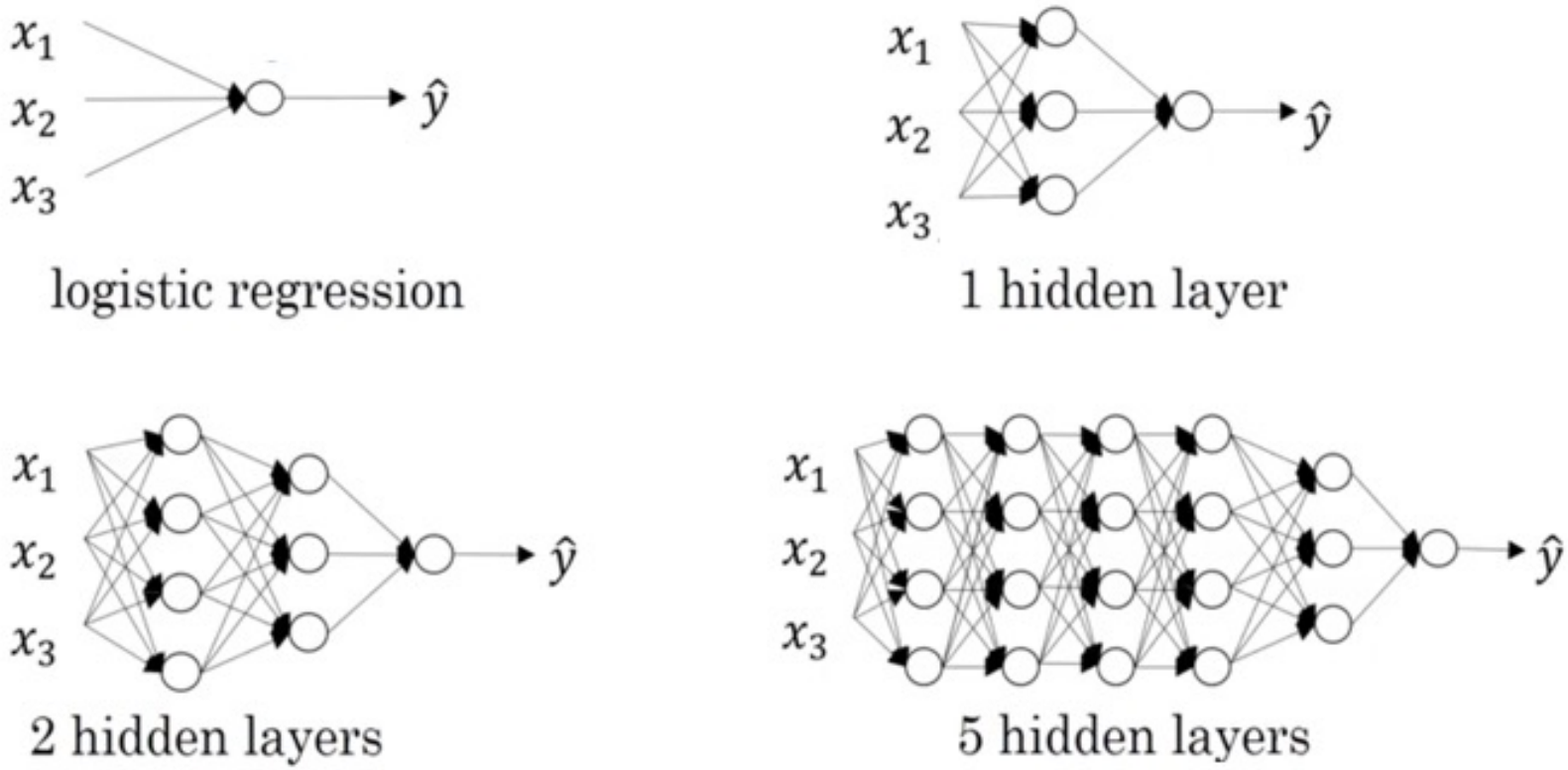
After many iterations, all the hidden units will still be computing exactly the same function.

So in this case, what is the interest in having more than one single hidden unit?

SOLUTION: Initialize parameters randomly

# Deep neural networks

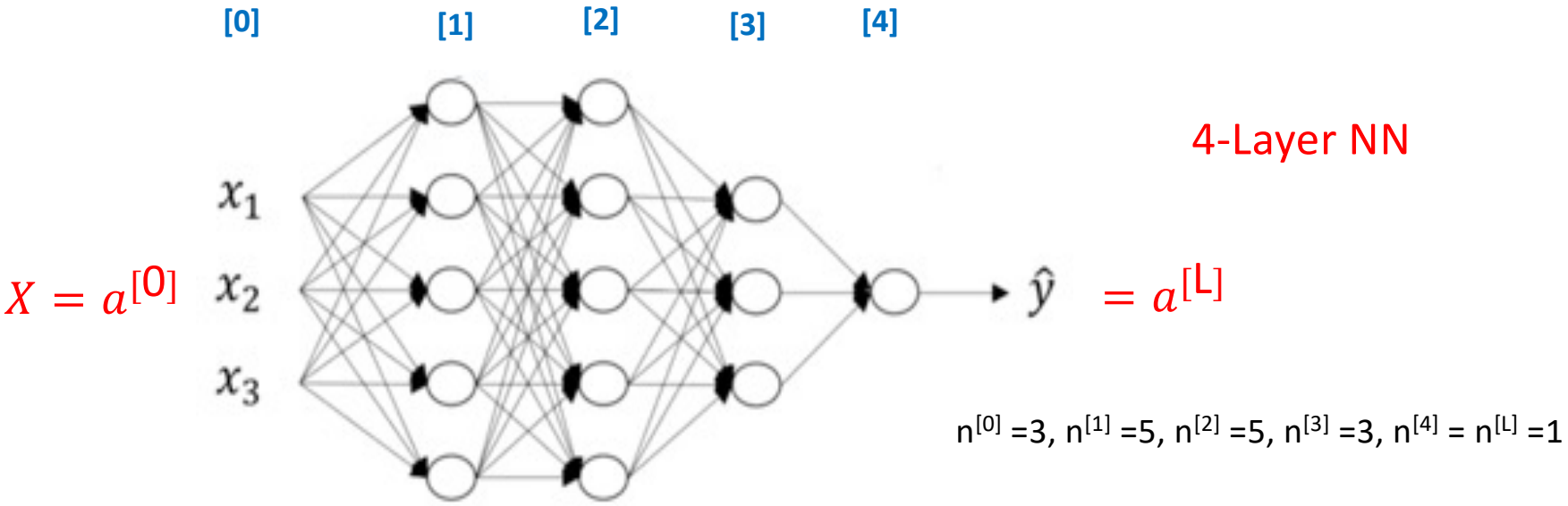
# Deep-L-layer-neural-network




Number of hidden layers is a hyper parameter

Andrew Ng "Deep Learning Specialization".
https://fr.coursera.org/specializations/deep-learning

# Deep Neural network notation

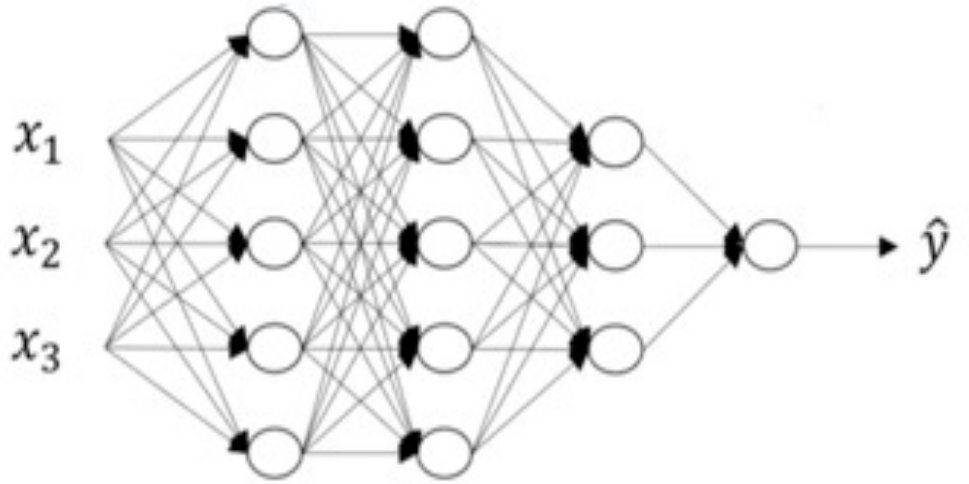



L : Number of layers (starts at 0 and ends at L)

$n^{[\ell]}$ : number of hidden units in layer $\ell$

$z^{[\ell]}, a^{[\ell]}$ : activation, parameters $W^{[\ell]}$ and $b^{[\ell]}$, $g^{[\ell]}$ activation function of layer $\ell$.

# Forward propagation in a deep neural network




**For a single observation:**

*For the layer* $\ell = 1,\ldots,L$

$$z^{[\ell]} = W^{[\ell]} a^{[\ell-1]} + b^{[\ell]}$$

$$a^{[\ell]} = g^{[\ell]}(z^{[\ell]})$$

**For *m* observations:**

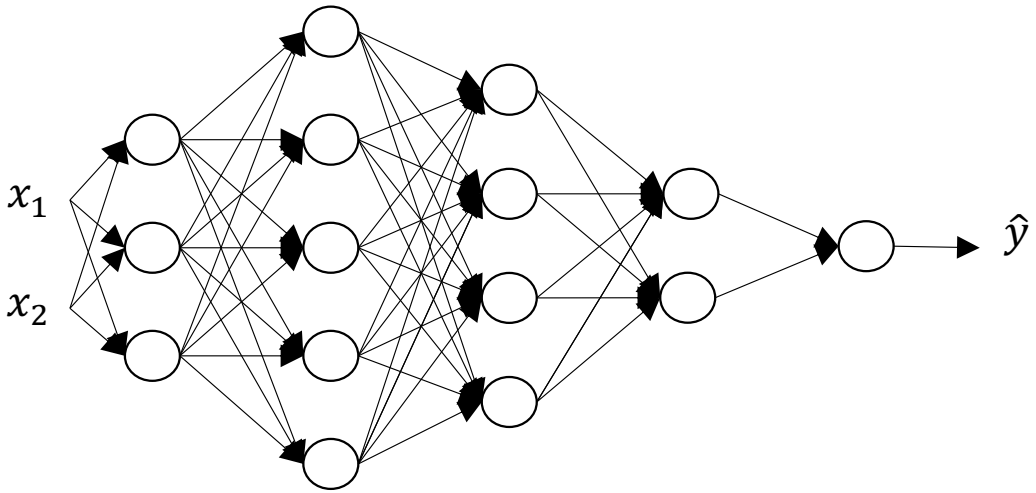
*For the layer* $\ell = 1,\ldots,L$

$$Z^{[\ell]} = W^{[\ell]} A^{[\ell-1]} + b^{[\ell]}$$

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

Andrew Ng "Deep Learning Specialization".
https://fr.coursera.org/specializations/deep-learning

# Getting your matrix dimensions right




Warning:
$Z^{[\ell]} = W^{[\ell]} Z^{[\ell-1]} + b^{[\ell]}$

Duplicated and converted to $(n^{[\ell]}, m)$

Parameters $W^{[\ell]}$ and $b^{[\ell]}$

$W^{[\ell]} : (n^{[\ell]}, n^{[\ell-1]})$ and $b^{[\ell]}: (n^{[\ell]}, 1)$
$dW^{[\ell]} : (n^{[\ell]}, n^{[\ell-1]})$ and $db^{[\ell]}: (n^{[\ell]}, 1)$

(derivatives have of course same dimensions as parameters!)

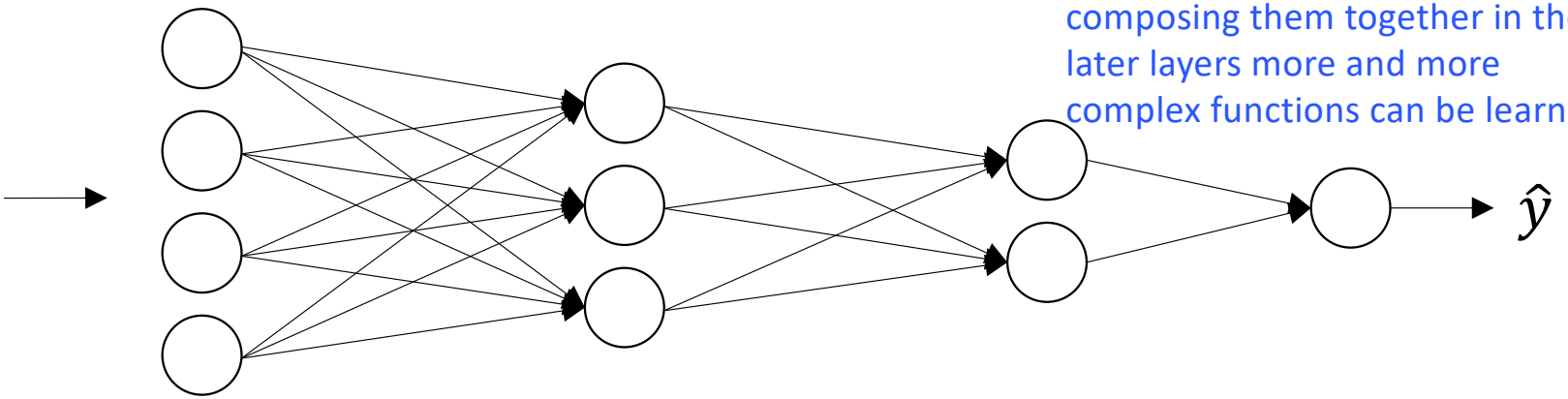
Dimensions of $Z^{[\ell]}$ and $A^{[\ell]}$

For a single example $z^{[\ell]}$, $a^{[\ell]}: (n^{[\ell]}, 1)$

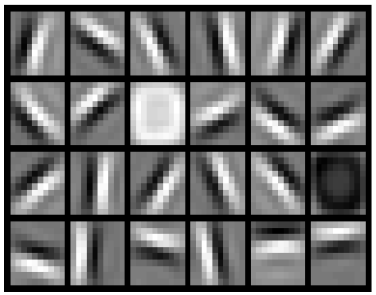
For *m* examples
$Z^{[\ell]} : (n^{[\ell]}, m)$ , $A^{[\ell]} : (n^{[\ell]}, m)$

# Why deep representations?

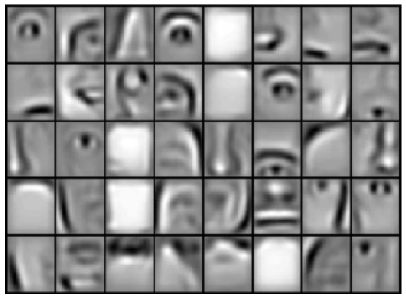
**Example1:** “Face recognition or Face detection system”




Earlier layers are detecting simple functions (edges). Then, composing them together in the later layers more and more complex functions can be learned .



edge detector



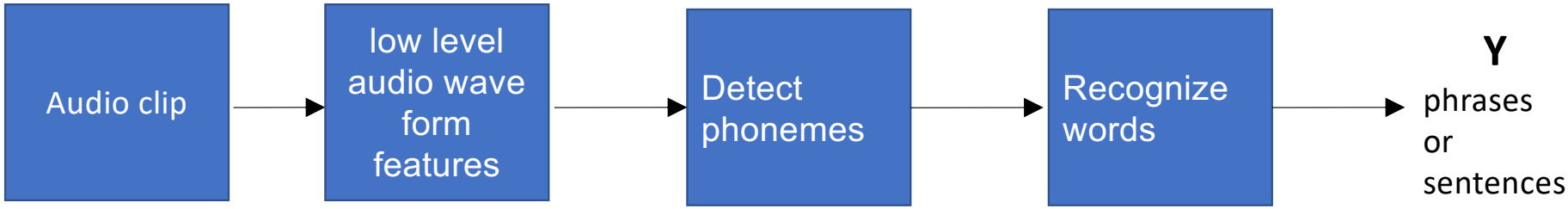
Detects parts of faces

Detects types of faces

Andrew Ng "Deep Learning Specialization".
https://fr.coursera.org/specializations/deep-learning

# Why deep representations?

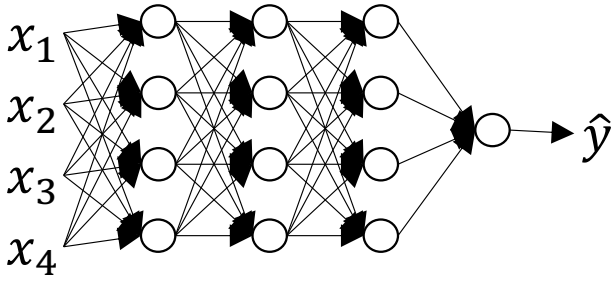
**Example1:** “Speech recognition system ”



# Forward and backward propagation for Deep neural networks

# Forward and backward propagation

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







Update: $W^l := W^l - \alpha\,(dW^l)$ and $b^l := b^l - \alpha\,(db^l)$

# Forward and backward propagation functions

For $m$ examples

## Forward

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = g^{[2]}(Z^{[2]})$$
$$\vdots$$
$$Z^{[\ell]} = W^{[\ell]}A^{[\ell-1]} + b^{[\ell]}$$
$$A^{[\ell]} = g^{[\ell]}(Z^{[\ell]})$$
$$\vdots$$
$$Z^{[L]} = W^{[L]}A^{[L-1]} + b^{[L]}$$
$$A^{[L]} = g^{[L]}(Z^{[L]}) = \hat{Y}$$

## Backward

(If last activation function is "sigmoid")

$$dZ^{[L]} = A^{[L]} - Y$$
$$dW^{[L]} = \frac{1}{m} dZ^{[L]} A^{[L-1]^T} \quad db^{[L]} = \frac{\sum_{i=1}^{m} dZ^{[L](i)}}{m}$$
$$dZ^{[L-1]} = W^{[L]^T} dZ^{[L]} * g'^{[L-1]}(Z^{[L-1]})$$
$$\vdots$$
$$dZ^{[\ell]} = W^{[\ell+1]^T} dZ^{[\ell+1]} * g'^{[\ell]}(Z^{[\ell]})$$
$$dW^{[\ell]} = \frac{1}{m} dZ^{[\ell]} A^{[\ell-1]^T} \quad db^{[\ell]} = \frac{\sum_{i=1}^{m} dZ^{[\ell](i)}}{m}$$
$$\vdots$$
$$dZ^{[1]} = W^{[2]^T} dZ^{[2]} * g'^{[1]}(Z^{[1]})$$
$$dW^{[1]} = \frac{1}{m} dZ^{[1]} A^{[0]^T} \quad db^{[1]} = \frac{\sum_{i=1}^{m} dZ^{[1](i)}}{m}$$

Andrew Ng "Deep Learning Specialization".
https://fr.coursera.org/specializations/deep-learning

# Backward propagation for layer $l$ (cheating sheet)

→ Input $da^{[l]}$

→ Output $da^{[l-1]}, dW^{[l]}, db^{[l]}$

$$dz^{[l]} = da^{[l]} * g^{[l]\prime}(z^{[l]})$$

$$dW^{[l]} = dz^{[l]} \cdot a^{[l-1]}$$

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

$$da^{[l-1]} = W^{[l]T} \cdot dz^{[l]}$$

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

$$dz^{[l]} = dA^{[l]} * g^{[l]\prime}(Z^{[l]})$$

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

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

$$dA^{[l-1]} = W^{[l]T} \cdot dz^{[l]}$$

# Parameters vs Hyperparameters

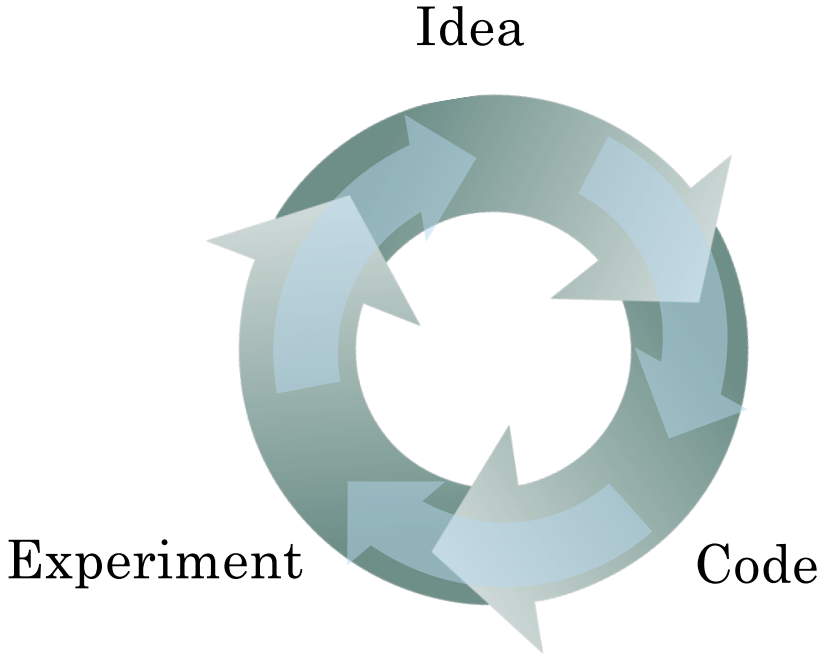
Parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]}$ ...

Hyperparameters:

- Learning rate $\alpha$
- Number of iterations
- Number of hidden layers $L$
- Number of hidden units $n^{[1]}, \ldots, n^{[L]}$
- Choice of the activation function: $\sigma, tanh, RELU$

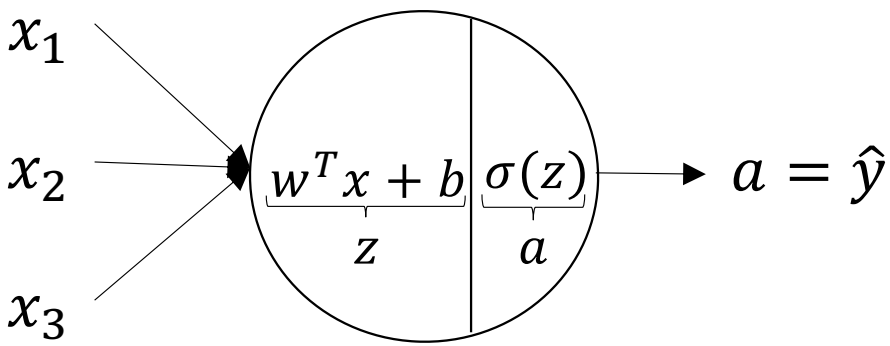
- Many other

Applied deep learning is a very empirical process

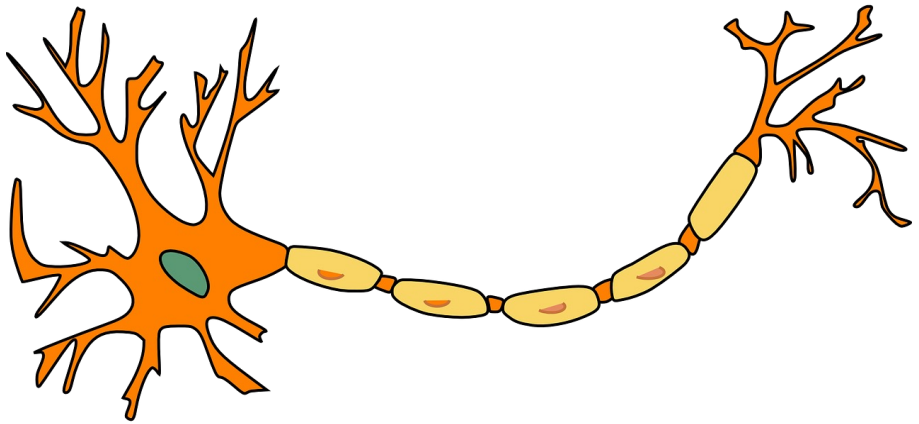


# What does this have to do with the brain?

logistic regression




biological neuron



# References

This course is based on:

Mooc Coursera : https://fr.coursera.org , « Deep Learning specialization ».
Course 1: Neural Networks and Deep Learning. Site web:
https://fr.coursera.org/specializations/deep-learning , 2018