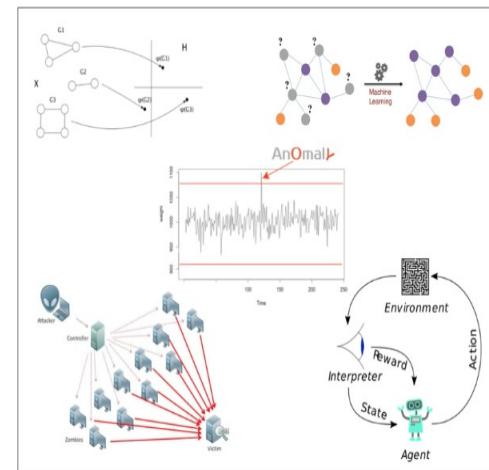
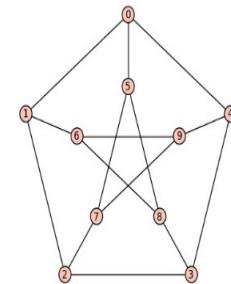


# Introduction au Deep Learning

Pierre Pereira

Université Côte d'Azur / Inria Coati

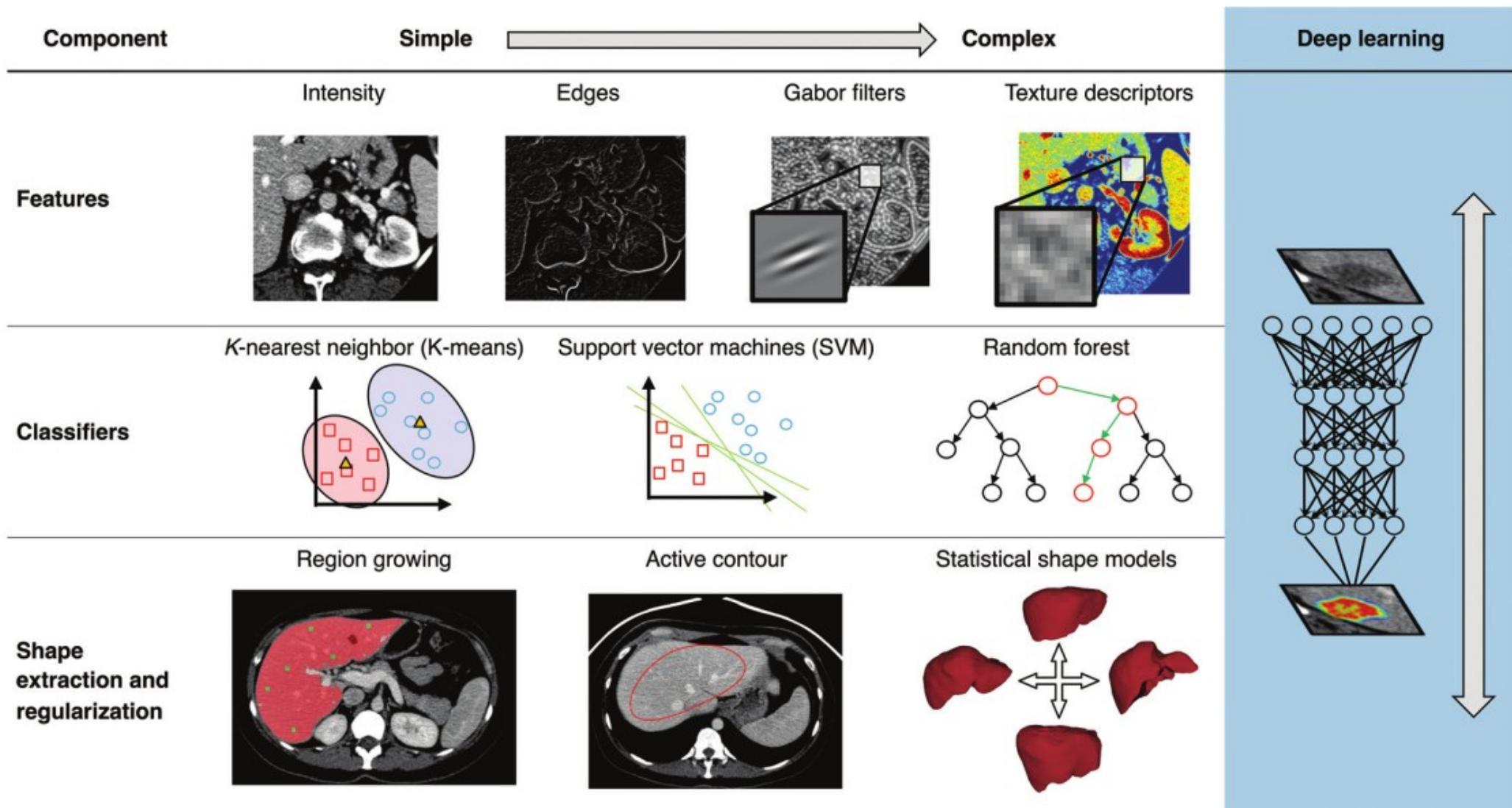
pierre.pereira@inria.fr



Sources multiples, principalement  
Polytechnique Montréal

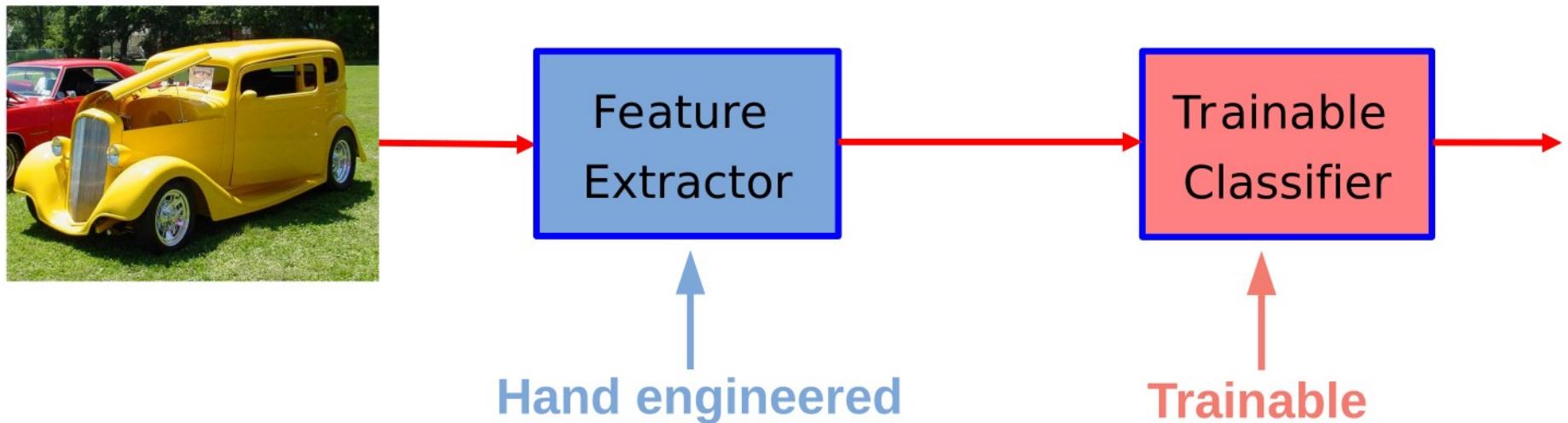
$$\begin{aligned} \min \quad & \sum_{e \in \mathcal{E}} y_e \\ \text{s.t.} \quad & \sum_{a \in A_i^+(u)} f_a^i - \sum_{a \in A_i^-(u)} f_a^i = \begin{cases} |V_i| - 1 & \text{if } u = s_i \\ -1 & \text{if } u \neq s_i \end{cases} \quad \forall u \in V_i, V_i \in C \\ & f_a^i \leq |V_i| \cdot x_a, \quad \forall V_i \in C, a \in A \\ & x_{(u,v)} \leq y_{uv}, \quad \forall uv \in \mathcal{E} \\ & x_{(v,u)} \leq y_{uv}, \quad \forall uv \in \mathcal{E} \end{aligned}$$

# From Pattern Recognition and Machine Learning to Deep Learning



# The Standard Paradigm of Pattern Recognition

- ▶ ...since the 1960s
- ▶ ...and “traditional” Machine Learning
- ▶ until the “Deep Learning Revolution” (circa 2012)



# Multilayer Neural Nets and Deep Learning

## ► Traditional Machine Learning

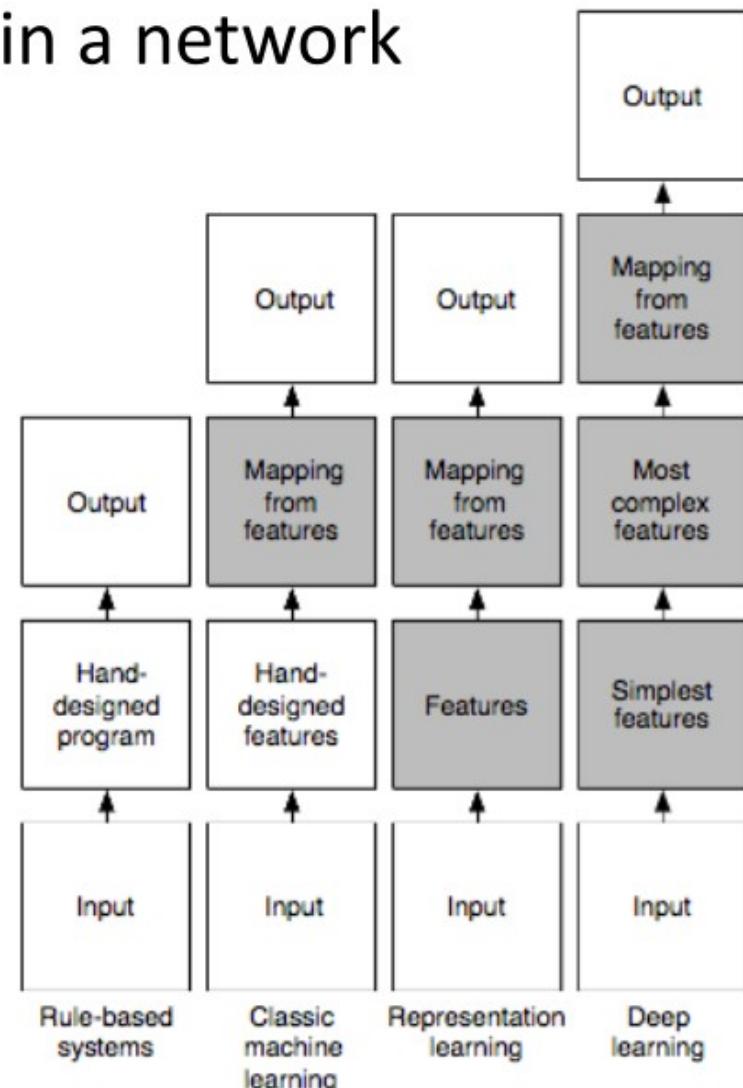


## ► Deep Learning

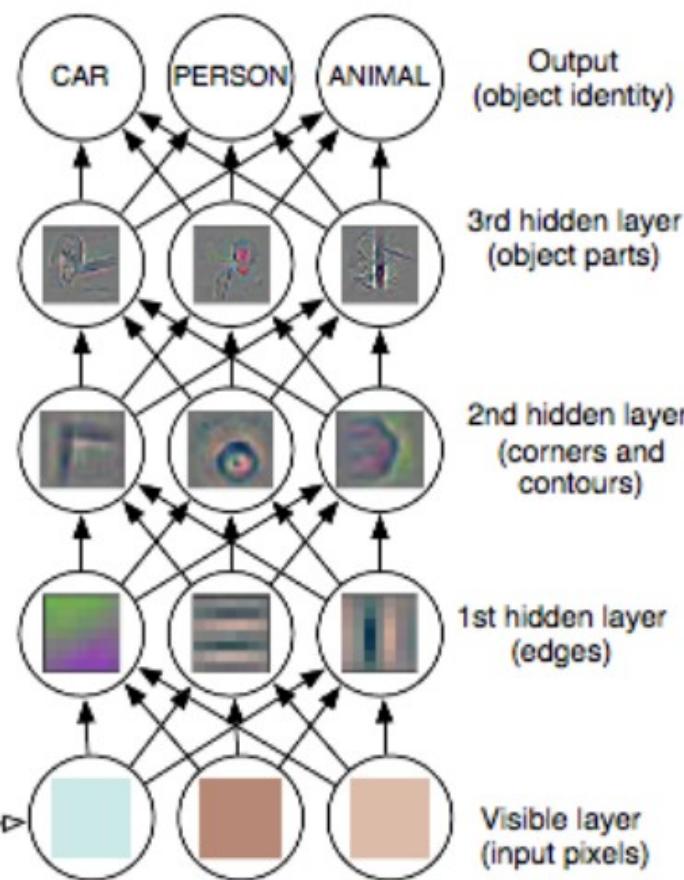


# Why does Deep Learning ‘Work’ so well

- Deep Learning methods seek to learn representations through a sequence of transformations of the data, typically thought of as layers in a network
- Flow charts from Bengio et al. (2014) “showing how the different parts of an AI system relate to each other within different AI disciplines.
- Shaded boxes indicate components that are able to learn from data”



# Intuitions about Deep Learning



- Deep Learning methods seek to learn representations through a sequence of transformations of the data, typically thought of as layers in a network

# **GPU computing is basically essential for training and experiment with models**

- Convolution operations and matrix multiplies are parallelize very well using GPUs
- Model training can be 30-60x faster using GPUs
- Example: Experiment using the famous Oxford VGG-convolutional neural network
- The model was trained on 1.28 million images

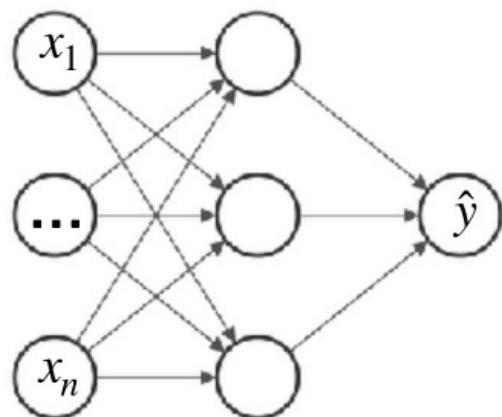
CPU: 2x Xeon e5-2699v4 server takes **55 days**

GPU: NVIDIA DGX-1 Trains in 23 hours or **1 day**

# Aperçu d'un réseau de neurones

## Réseau de neurones

Ensemble de neurones organisées en un réseau composé de différents niveaux (*layers*)



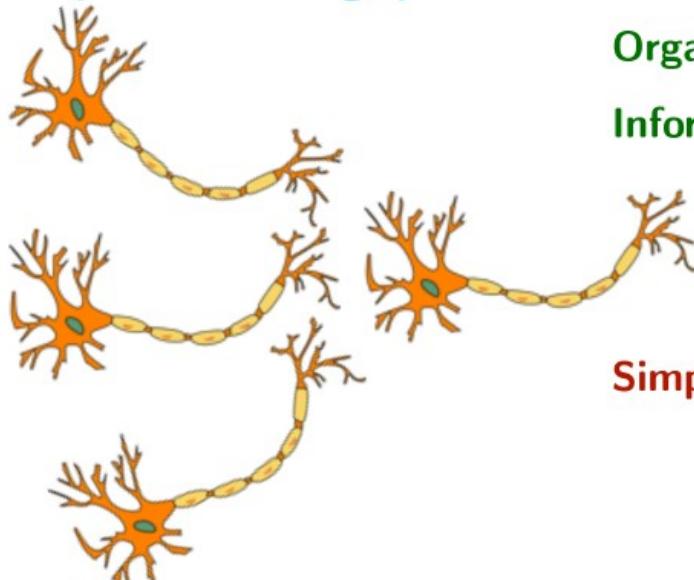
Chaque neurone a ses propres paramètres  $w$  et  $b$

Chaque neurone apporte sa propre contribution pour la prédiction

Dit autrement, les neurones apprennent une fonction différentes, qui sont ensuite composées ensemble pour faire une prédiction plus riche

On a ainsi, un modèle plus expressif, non limité aux régressions logistiques

## Métaphore biologique



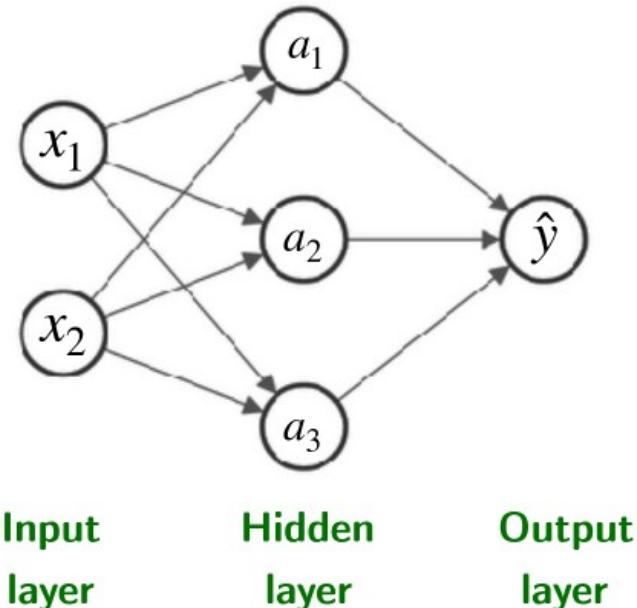
Organisation d'un cerveau

Information (signal électrique) transmise de neurones en neurones

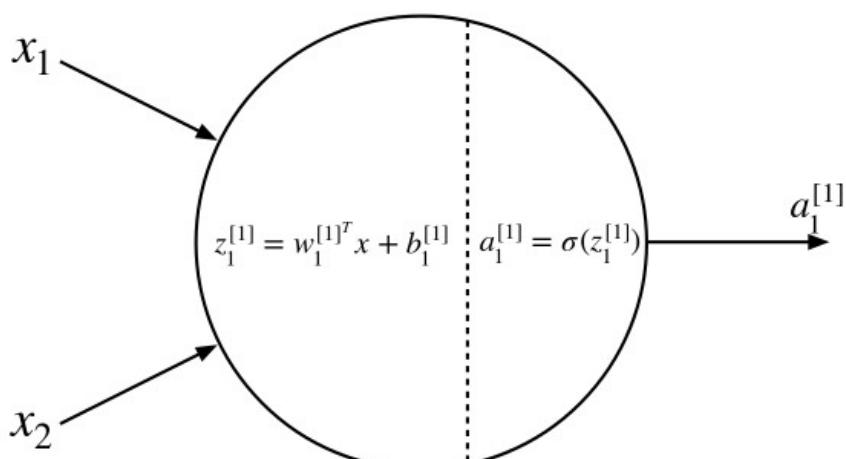
Simplement une métaphore, et non une correspondance exacte

# Réseau de neurones: notations

## 2-layers neural network



## Zoom sur un neurone



## Layers

**Input layer:** couche qui reprend les features de la donnée  $x$

**Hidden layer:** couche dédiée aux calculs intermédiaires  $a$

**Output layer:** couche dédiée à la prédiction finale  $\hat{y}$

**L'input layer n'a pas de paramètres devant être appris**

Par convention, **l'input layer est la couche 0**

## Notations

$x_j$  : Feature  $j$  pour une donnée spécifique

$a_j^{[i]}$  : output du neurone  $j$  au layer  $i$

$\hat{y}$  : prediction pour les features données

$n^{[i]}$  : nombre de neurones au layer  $i$

$w_j^{[i]}, b_j^{[i]}$  : paramètres du neurone  $j$  au layer  $i$

**Notez que  $w_j^{[i]}$  est un vecteur de dimension  $n^{[i-1]} \times 1$**

## Intuition

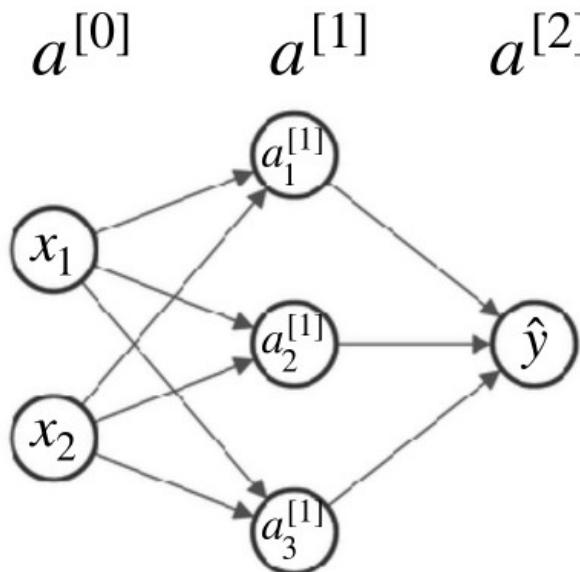
**Chaque neurone fait une régression logistique avec ses propres paramètres ( $w, b$ )**

$$a_1^{[1]} = \sigma(w_1^{[1]T}x + b_1^{[1]}) \quad a_3^{[1]} = \sigma(w_3^{[1]T}x + b_3^{[1]})$$

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

# Réseau de neurones: équations fondamentales

## Réseau de neurones



## Forward pass equations

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

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

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

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

L'input et l'output sont mis en évidence

## Ou de manière similaire

$$z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}$$

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

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

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

Forme plus générique



Combien de paramètres à t-on dans ce réseau ?

$$W^{[1]} : n^{[1]} \times n^{[0]} = 3 \times 2 \rightarrow 6$$

$$b^{[1]} : n^{[1]} \times 1 = 3 \times 1 \rightarrow 3$$

$$W^{[2]} : n^{[2]} \times n^{[1]} = 1 \times 3 \rightarrow 3$$

$$b^{[2]} : n^{[2]} \times 1 = 1 \times 1 \rightarrow 1$$

Total: 13 paramètres devant être entraînés



"Les dimensions des matrices, et le nombre de paramètres, tu maîtriseras"

# Forme matricielle pour plusieurs prédictions

## Comment implémenter cela ?

### Solution naïve

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

?

Est-ce que c'est correct ?

Oui !

?

Est-ce que c'est efficace ?

Non !

Opérations matricielles sont plus efficaces  
qu'une boucle (surtout avec des GPUs)

Regroupement en matrice des instances

$$\left( \begin{array}{|c|} \hline | \\ \hline x^{(1)} \\ \hline | \\ \hline \end{array} \right), \dots, \left( \begin{array}{|c|} \hline | \\ \hline x^{(m)} \\ \hline | \\ \hline \end{array} \right) \rightarrow \left( \begin{array}{|c| \dots |c|} \hline x^{(1)} & \dots & x^{(m)} \\ \hline | & \dots & | \\ \hline \end{array} \right) = X$$

$n^{[0]} \times 1 \quad n^{[0]} \times 1$        $n^{[0]} \times m$

#features × #instances

## Forme matricielle

Une colonne contient les informations spécifiques pour une instance

Les valeurs  $z$ , et  $a$  de chaque neurone devient également une matrice

$$Z^{[1]} = \begin{pmatrix} - & w_1^{[1]T} & - \\ - & \dots & - \\ - & w_{n^{[1]}}^{[1]T} & - \end{pmatrix} \begin{pmatrix} | & \dots & | \\ x^{(1)} & \dots & x^{(m)} \\ | & \dots & | \end{pmatrix} + \begin{pmatrix} b_1^{[1]} \\ \vdots \\ b_{n^{[1]}}^{[1]} \end{pmatrix}$$
$$= \begin{pmatrix} w_1^{[1]T}x^{(1)} + b_1 & \dots & w_1^{[1]T}x^{(m)} + b_1 \\ \dots & \dots & \dots \\ w_{n^{[1]}}^{[1]T}x^{(1)} + b_{n^{[1]}} & \dots & w_{n^{[1]}}^{[1]T}x^{(m)} + b_{n^{[1]}} \end{pmatrix}$$

Broadcasting  
de  $b$

$$= \begin{pmatrix} | & \dots & | \\ z^{[1](1)} & \dots & z^{[1](m)} \\ | & \dots & | \end{pmatrix}$$

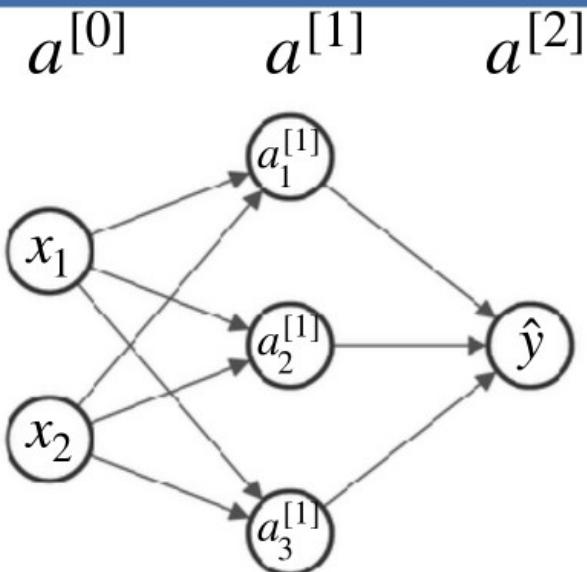
Dimension :  $n^{[1]} \times m$   
#neurones × #instances

$$A^{[1]} = \begin{pmatrix} | & \dots & | \\ \sigma(z^{[1](1)}) & \dots & \sigma(z^{[1](m)}) \\ | & \dots & | \end{pmatrix}$$
$$= \begin{pmatrix} | & \dots & | \\ a^{[1](1)} & \dots & a^{[1](m)} \\ | & \dots & | \end{pmatrix}$$

Application de la  
sigmoïde à chaque valeur

Dimension :  $n^{[1]} \times m$   
#neurones × #instances

# Illustration de la représentation matricielle



**Equations pour une  
seule prédition**

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

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

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

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

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

**Equations pour une  
plusieurs prédition**

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

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

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

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

$$\hat{Y} = A^{[2]}$$



## Illustration

**Problème de l'étudiant caractérisé par deux features**

# heures d'étude :  $x_1$

# heures de repos :  $x_2$

**Quatre données d'entraînement**

$$D : \left\{ (x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), (x^{(4)}, y^{(4)}) \right\}$$

**Données en forme matricielle**

$$X = \begin{pmatrix} x_1^{(1)} & x_1^{(2)} & x_1^{(3)} & x_1^{(4)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(3)} & x_2^{(4)} \end{pmatrix}$$

**Que représente  $x_2^{(3)}$  ?**

Nombre d'heures de repos  
de la troisième donnée

**Output du premier layer**

$$A^{[1]} = \begin{pmatrix} a_1^{[1](1)} & a_1^{[1](2)} & a_1^{[1](3)} & a_1^{[1](4)} \\ a_2^{[1](1)} & a_2^{[1](2)} & a_2^{[1](3)} & a_2^{[1](4)} \\ a_3^{[1](1)} & a_3^{[1](2)} & a_3^{[1](3)} & a_3^{[1](4)} \end{pmatrix}$$

**Que représente  $a_3^{[1](2)}$  ?**

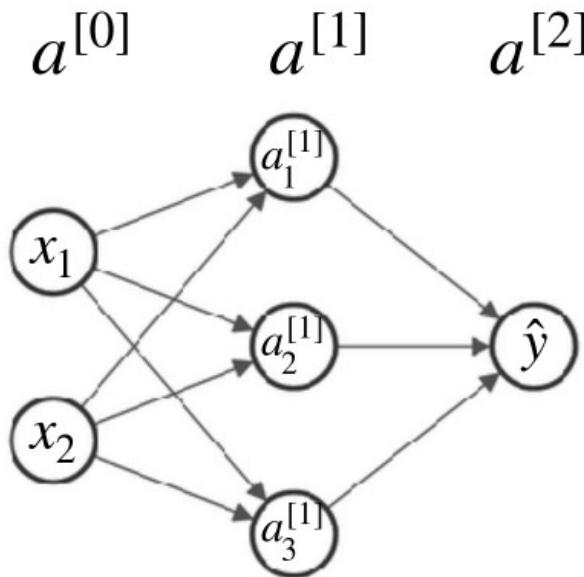
Activation du troisième neurone de la  
couche 1 pour la deuxième donnée

**Prédiction finale (une valeur pour chaque donnée)**

$$A^{[2]} = (a_1^{[2](1)} \ a_1^{[2](2)} \ a_1^{[2](3)} \ a_1^{[2](4)})$$

$$\hat{Y} = (\hat{y}^{(1)} \ \hat{y}^{(2)} \ \hat{y}^{(3)} \ \hat{y}^{(4)})$$

# Apprentissage supervisé par un réseau de neurones



Exactement pareil que la régression logistique, sauf qu'on apprend une fonction plus complexe, avec plus de paramètres

**Forward pass equations**

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

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

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

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

$$\hat{Y} = A^{[2]}$$

**Paramètres**

$$W^{[1]} : n^{[1]} \times n^{[0]} = 3 \times 2$$

$$b^{[1]} : n^{[1]} \times 1 = 3 \times 1$$

$$W^{[2]} : n^{[2]} \times n^{[1]} = 1 \times 3$$

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

13 paramètres à déterminer

**Fonction de coût**

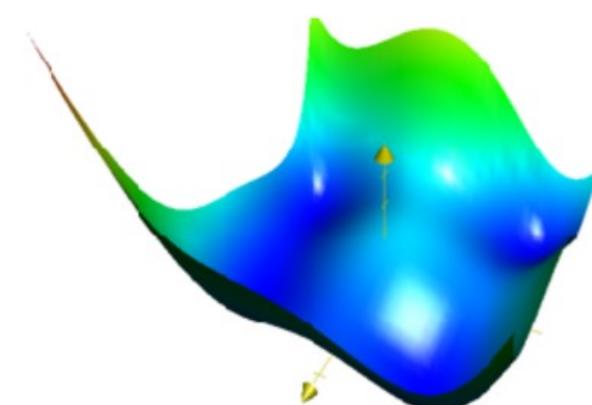
$$J(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}, y)$$

Flexible avec la loss function utilisée

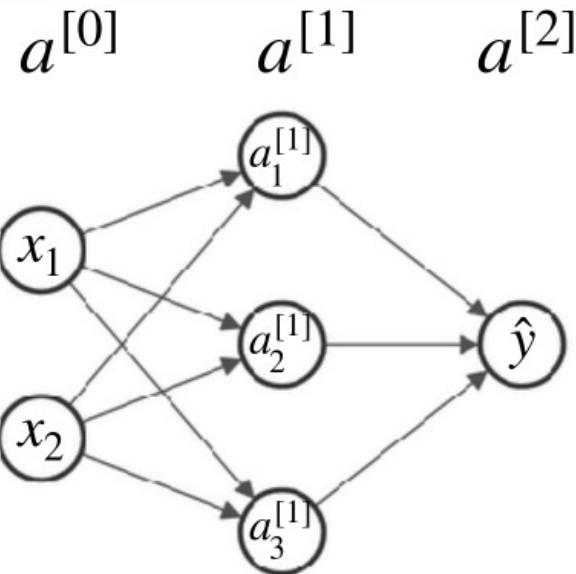
Tout le réseau est différentiable

Peut être minimisée par une descente de gradient

Mais la fonction n'est plus convexe... risque de minimum local



# Descente de gradient dans un réseau de neurones



## Backward pass (Backpropagation)

$$\frac{\partial J}{\partial b^{[1]}} = \frac{1}{m} \left( \frac{\partial L(\hat{y}^{(1)}, y^{(1)})}{\partial b^{[1]}} + \dots + \frac{\partial L(\hat{y}^{(m)}, y^{(m)})}{\partial b^{[1]}} \right)$$

$$\frac{\partial J}{\partial b^{[2]}} = \frac{1}{m} \left( \frac{\partial L(\hat{y}^{(1)}, y^{(1)})}{\partial b^{[2]}} + \dots + \frac{\partial L(\hat{y}^{(m)}, y^{(m)})}{\partial b^{[2]}} \right)$$

$$\frac{\partial J}{\partial W^{[1]}} = \frac{1}{m} \left( \frac{\partial L(\hat{y}^{(1)}, y^{(1)})}{\partial W^{[1]}} + \dots + \frac{\partial L(\hat{y}^{(m)}, y^{(m)})}{\partial W^{[1]}} \right)$$

$$\frac{\partial J}{\partial W^{[2]}} = \frac{1}{m} \left( \frac{\partial L(\hat{y}^{(1)}, y^{(1)})}{\partial W^{[2]}} + \dots + \frac{\partial L(\hat{y}^{(m)}, y^{(m)})}{\partial W^{[2]}} \right)$$

Dérivation: [www.efavdb.com/backpropagation-in-neural-networks](http://www.efavdb.com/backpropagation-in-neural-networks)

Calculé avec la règle du chaînage

sur un computation graph

Quentin Cappart

## Forward pass equations

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

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

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

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

$$\hat{Y} = A^{[2]}$$

## Algorithme d'apprentissage

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

repeat :

compute  $\hat{Y}$

compute  $\frac{\partial J}{\partial W^{[2]}}, \frac{\partial J}{\partial b^{[2]}}, \frac{\partial J}{\partial W^{[1]}}, \frac{\partial J}{\partial b^{[1]}}$

$W^{[2]} = W^{[2]} - \alpha \frac{\partial J}{\partial W^{[2]}}$

$b^{[2]} = b^{[2]} - \alpha \frac{\partial J}{\partial b^{[2]}}$

$W^{[1]} = W^{[1]} - \alpha \frac{\partial J}{\partial W^{[1]}}$

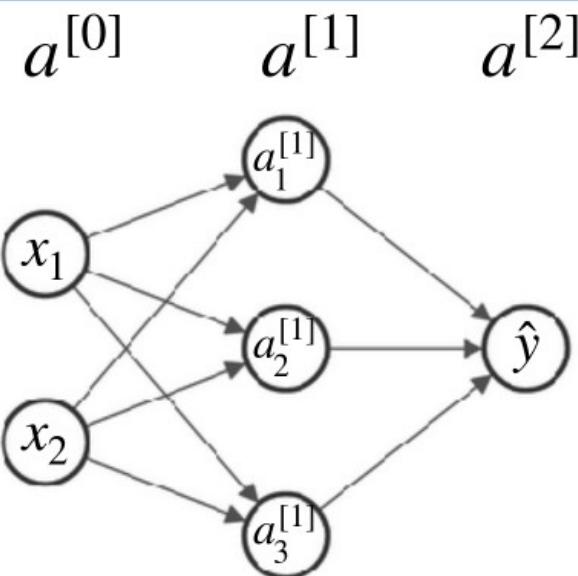
$b^{[1]} = b^{[1]} - \alpha \frac{\partial J}{\partial b^{[1]}}$

Forward pass

Backpropagation

Descente de gradient

# Initialisation aléatoire



? Est-il nécessaire de faire une initialisation aléatoire ?  
Pourquoi ne pas initialiser les paramètres à 0 ?

Essayons cette idée...

Pour le hidden layer, on a

$$W^{[1]} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \quad b^{[1]} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \quad a^{[1]} = \sigma(W^{[1]}x + b^{[1]})$$

Que peut-on dire des outputs ( $a$ ) du hidden layer durant la forward pass ?

$$a_1^{[1]} = a_2^{[1]} = a_3^{[1]}$$

Ils sont tous identiques

Qu'est ce que cela implique ?

Rien ne différencie les neurones, comme leur output est identique

Durant la back-propagation, les gradients vont avoir le même update

On est pas plus expressif qu'un seul neurone dans le layer

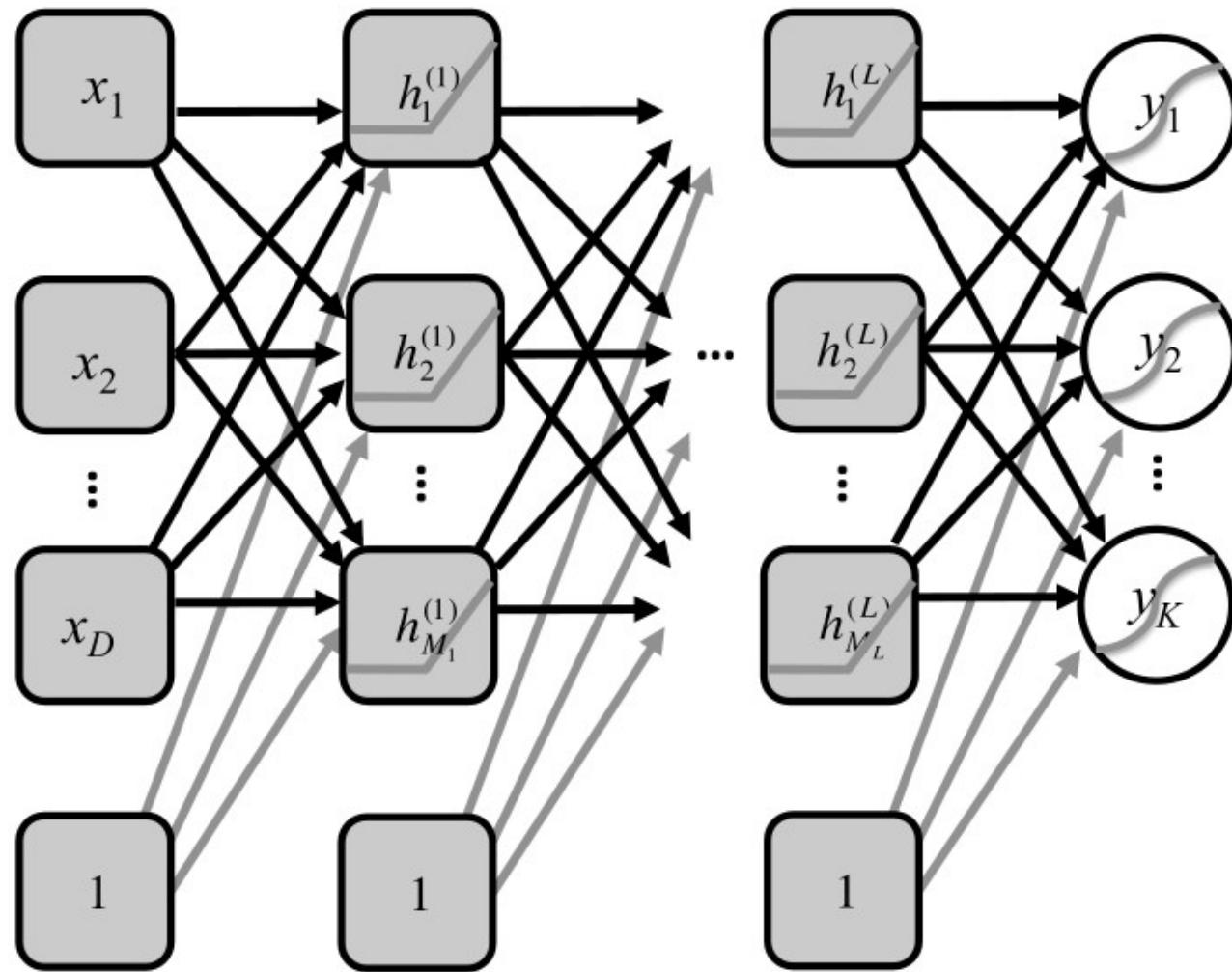
$$\frac{\partial J}{\partial W^{[1]}} = \begin{pmatrix} u & v \\ u & v \\ u & v \end{pmatrix}$$

Peut être facilement résolu en initialisant les poids aléatoirement

Note 1: il n'y a aucun problème à initialiser les biais ( $b$ ) à zéro

Note 2: il existe des initialisations aléatoires plus efficaces que d'autres

# Deep feedforward networks



# Deep neural network architectures

- Compose computations performed by many layers
- Denoting the output of hidden layers by  $\mathbf{h}^{(l)}(\mathbf{x})$ , the computation for a network with  $L$  hidden layers is:

$$\mathbf{f}(\mathbf{x}) = \mathbf{f}\left[\mathbf{a}^{(L+1)}\left(\mathbf{h}^{(L)}\left(\mathbf{a}^{(L)}\left(\dots\left(\mathbf{h}^{(2)}\left(\mathbf{a}^{(2)}\left(\mathbf{h}^{(1)}\left(\mathbf{a}^{(1)}(\mathbf{x})\right)\right)\right)\right)\right)\right)\right)\right]$$

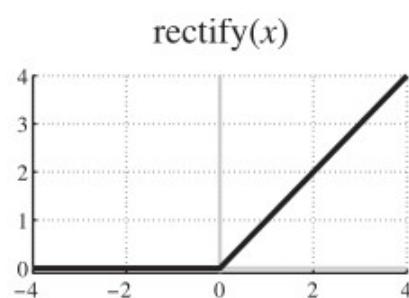
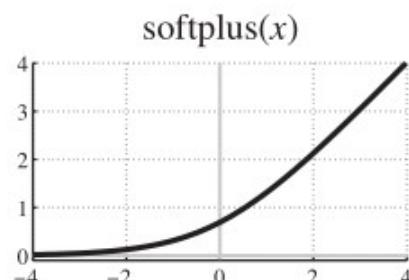
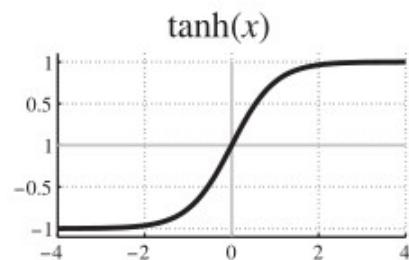
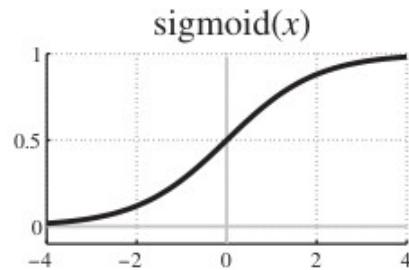
- Where *pre-activation functions*  $\mathbf{a}^{(l)}(\mathbf{x})$  are typically linear, of the form  $\mathbf{a}^{(l)}(\mathbf{x}) = \mathbf{W}^{(l)}\mathbf{x} + \mathbf{b}^{(l)}$  with matrix  $\mathbf{W}^{(l)}$  and bias  $\mathbf{b}^{(l)}$
- This formulation can be expressed using a single parameter matrix  $\theta$  with the trick of defining  $\hat{\mathbf{x}}$  as  $\mathbf{x}$  with a 1 appended to the end of the vector; we then have

$$\mathbf{a}^{(l)}(\hat{\mathbf{x}}) = \theta^{(l)}\hat{\mathbf{x}} \quad , l=1$$

$$\mathbf{a}^{(l)}(\hat{\mathbf{h}}^{(l-1)}) = \theta^{(l)}\hat{\mathbf{h}}^{(l-1)} \quad , l>1$$

# Activation functions

## Name and Graph



## Function

$$h(x) = \frac{1}{1 + \exp(-x)}$$

$$h(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

$$h(x) = \log(1 + \exp(x))$$

$$h(x) = \max(0, x)$$

## Derivative

$$h'(x) = h(x)[1 - h(x)]$$

$$h'(x) = 1 - h(x)^2$$

$$h'(x) = \frac{1}{1 + \exp(-x)}$$

$$h'(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

# Universalité des réseaux de neurones



Pourquoi veut-on avoir une fonction d'activation non linéaire ?



## Universalité des réseaux de neurones

Un NN avec des activations non-linéaires peut théoriquement approximer avec une certaine précision n'importe quelle fonction réelle

Approximation by superpositions of a sigmoidal function [Cybenko, 1989]



Que se passerait-il avec une activation linéaire ?

Soit l'activation identité:  $g(z) = z$

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

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

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



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

$$A^{[1]} = Z^{[1]}$$

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



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

$$Z^{[2]} = W'X + b'$$

$$A^{[2]} = Z^{[2]}$$

Le réseau s'effondre en une grosse combinaison linéaires des features en input

Cela arrive car la composition de deux fonctions linéaires est aussi linéaire

Sans non-linéarité, le modèle n'est pas plus expressif qu'une régression logistique

# Activation du dernier layer

## Activation du dernier layer

Cette action a une importante particularité

Elle définit le type de prédiction que l'on souhaite obtenir

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

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

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

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

$$\hat{Y} = A^{[2]}$$

## Classification binaire

On veut obtenir une probabilité de succès (comprise entre 0 et 1)

La fonction sigmoïde est un bon choix

$$g^{[2]}(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Une fonction de coût adaptée est ensuite la cross-entropy loss



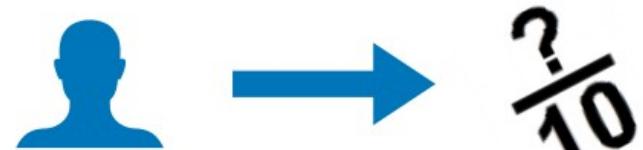
## Régression

On veut obtenir une valeur réelle

La fonction identité est un bon choix

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

Une fonction de coût adaptée est ensuite la mean-squared-error



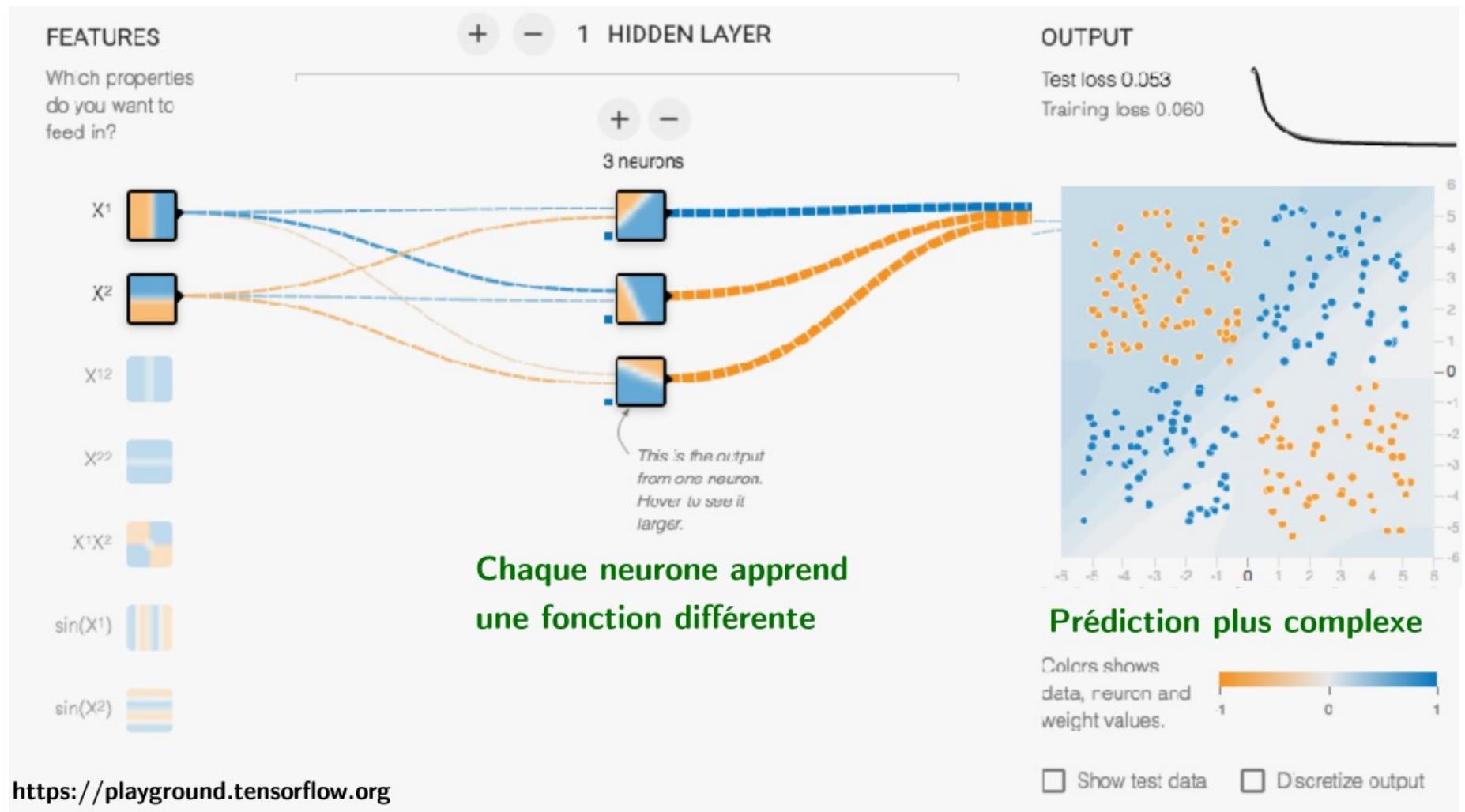
## Classification multi-classes

On a plus que deux classes de prédiction

Regardez du côté de la fonction softmax



# Visualisation de la prédiction



Deux features en input: coordonnées en x, et en y

Activation non-linéaire avec tanh

Learning rate de 0.01

Essayez par vous-même :-)

# Apprentissage profond — deep learning

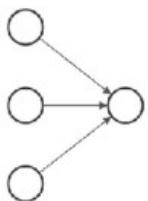


## Concept du deep learning

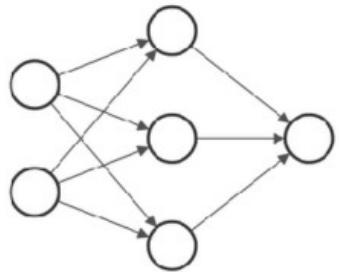
Le terme *deep* fait référence seulement à la profondeur du réseau de neurones

Avoir plus de deux hidden layers est considéré comme du deep learning

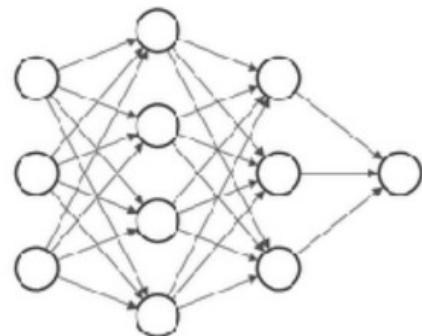
1-layer NN  
(simple régression)



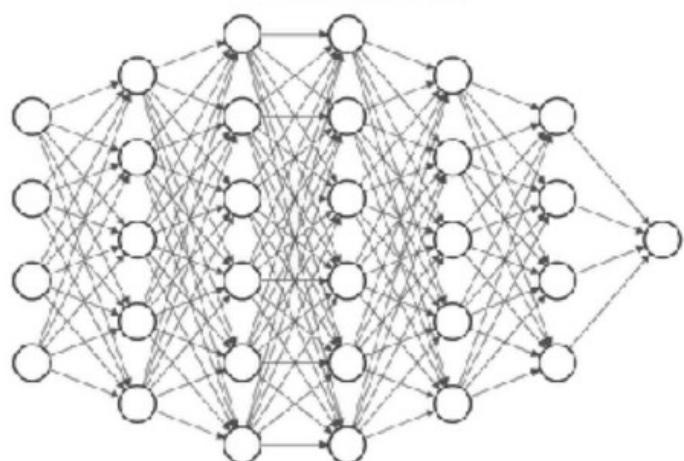
2-layers NN



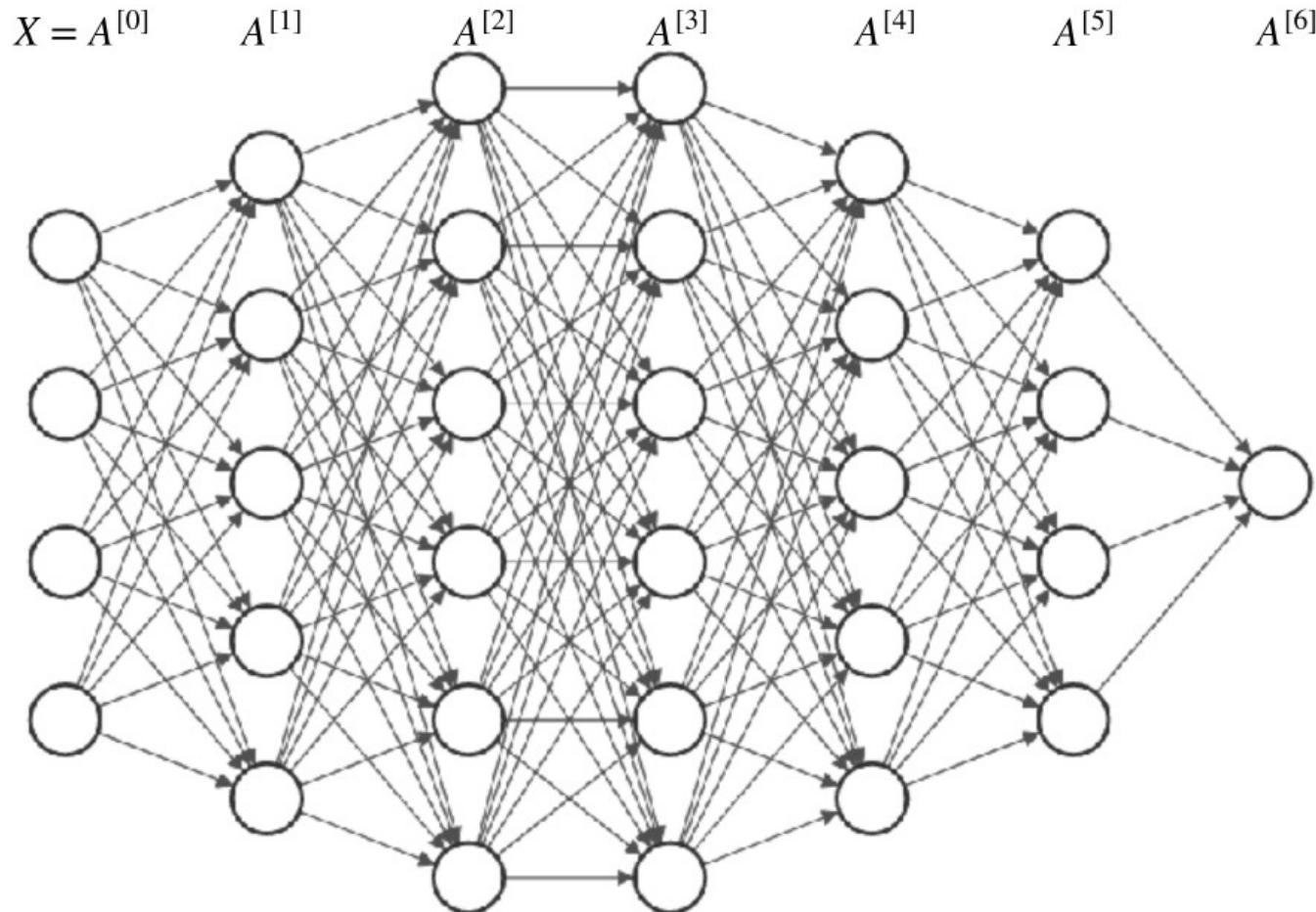
3-layers NN



6-layers NN



# Notations pour le deep learning



## Notations

$L$  : nombre de layers (à l'exception de celui d'input)

$n^{[l]}$  : nombre de neurones pour le layer  $l$

$X$  : features d'input (forme matricielle)

$A^{[l]}$  : output du layer  $l$

$Z^{[l]}$  : calcul intermédiaire du layer  $l$  (avant l'activation)

$W^{[l]}, b^{[l]}$  : paramètres du layer  $l$

$\hat{Y}$  : output du réseau (prédiction)

## Exercice

Donnez les dimensions de chaque matrice

Considérez 10 instances d'entraînement

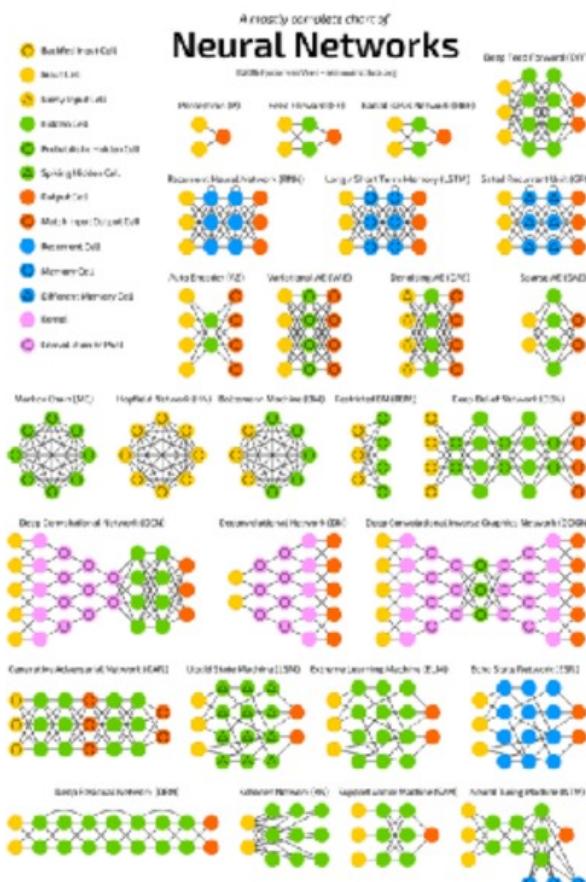
# Intérêt du deep learning



Quel est l'avantage du deep learning par rapport aux autres techniques d'apprentissage ?

Raison 1: les réseaux de neurones peuvent gérer toutes sortes de données différentes

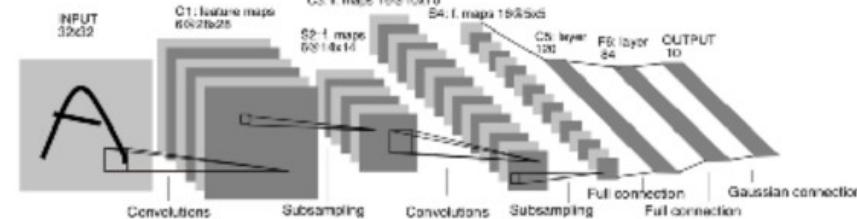
Beaucoup de recherches scientifiques sont consacrées à la construction de nouvelles architectures



[https://www.asimovinstitute.org/  
author/fjodorvanveen/](https://www.asimovinstitute.org/author/fjodorvanveen/)

## Convolutional neural networks

Spécialisés pour l'apprentissage sur des images



Gradient-based learning applied to document recognition [LeCun, Bengio et al., 1998]

## Recurrent neural networks

Spécialisés pour de l'apprentissage sur des données séquentielles



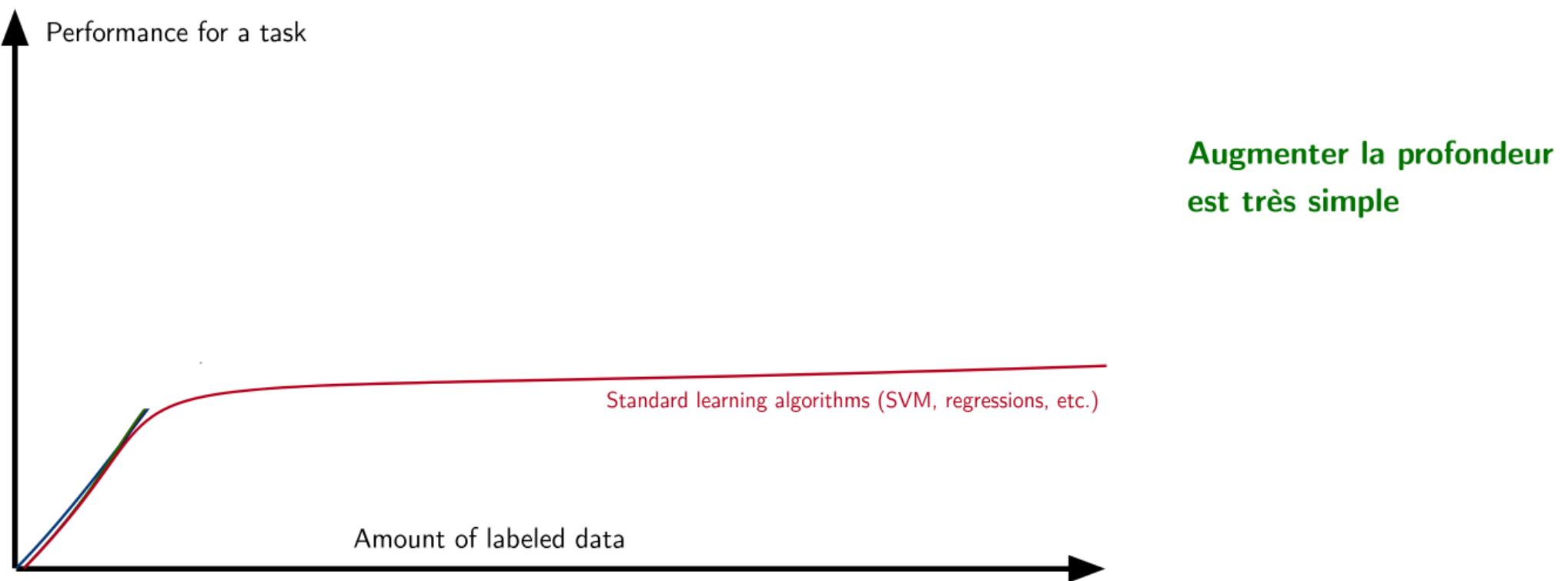
Neural networks and physical systems with emergent collective computational abilities [Hopfield, 1982]

Long short-term memory (LSTM)  
[Hochreiter and Schmidhuber, 1997]

# Intérêt du deep learning

Raison 2: les réseaux de neurones peuvent ingérer un très grand nombre de données

Au plus on a de données, au mieux c'est



Adapté du cours d'Andrew Ng (<https://www.coursera.org/specializations/deep-learning>)

Raison 3: méthode très facile d'utilisation

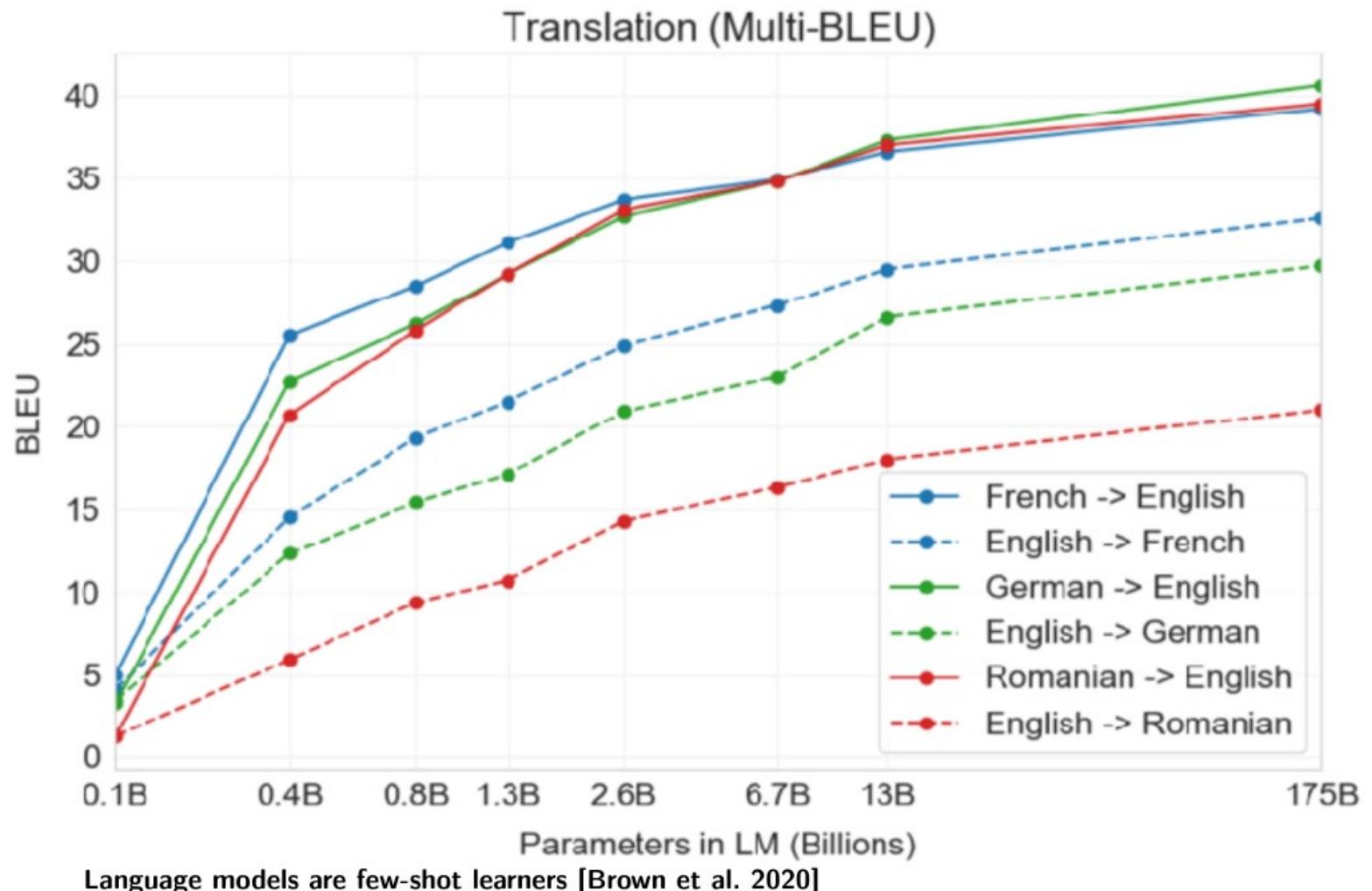
Fondations mathématiques pas trop lourdes (analyse, algèbre, probabilités)

Librairies disponibles pour implémenter des modèles (Keras, Pytorch, etc.)

Connaissances d'un modèle peuvent être réutilisées et transférées pour d'autres tâches

# A quel point un réseau peut-être profond ?

## GPT-3 (language model) pour des tâches de traductions



Réponse: très très profond :-)