

# DEEP LEARNING

## INTRODUCTION

---

Brendan Guillouet

2018

Institut National des Sciences Appliquées

# INTRODUCTION

---

Introduction

De Nouveaux Framework

CPU VS GPU

# QU'EST-CE QUE LE DEEP LEARNING ?

Livre de référence : Goodfellow et al. [2016]

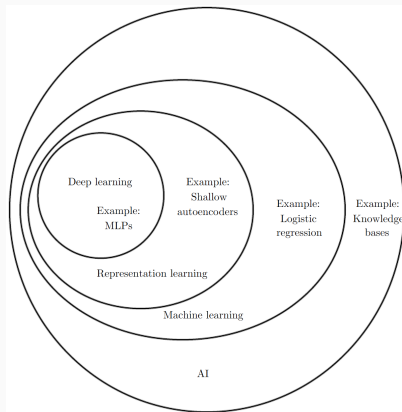


Figure 1.4 from Goodfellow et al. [2016]

# HIERARCHICAL FEATURES REPRESENTATION

Livre de référence : Goodfellow et al. [2016]

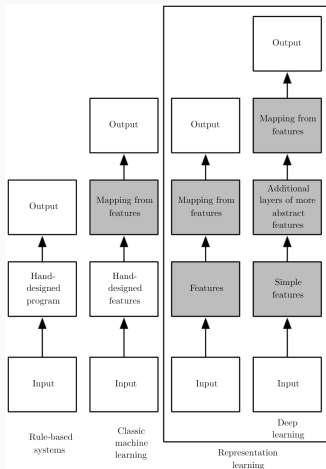


Figure 1.5 from Goodfellow et al. [2016]

# HIERARCHICAL FEATURES REPRESENTATION

Livre de référence : Goodfellow et al. [2016]

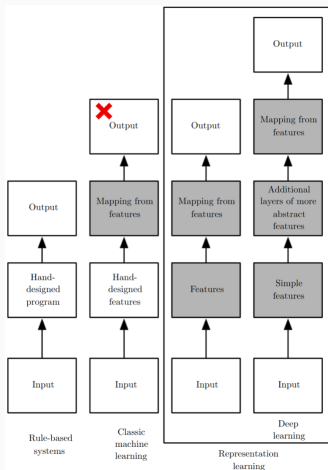


Figure 1.5 from Goodfellow et al. [2016]

# HIERARCHICAL FEATURES REPRESENTATION

Livre de référence : Goodfellow et al. [2016]

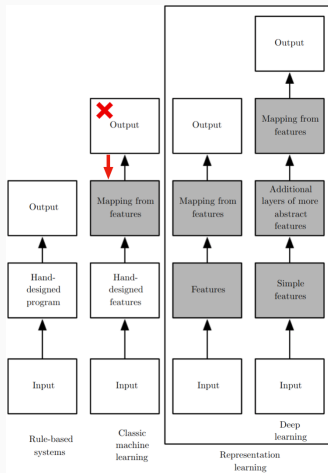


Figure 1.5 from Goodfellow et al. [2016]

# HIERARCHICAL FEATURES REPRESENTATION

Livre de référence : Goodfellow et al. [2016]

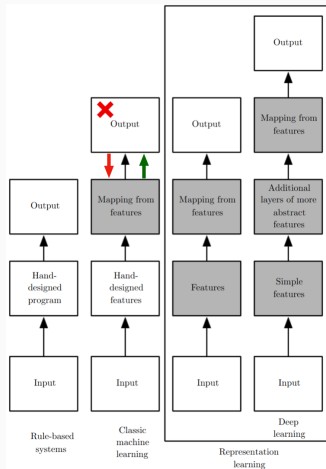


Figure 1.5 from Goodfellow et al. [2016]



# HIERARCHICAL FEATURES REPRESENTATION

Livre de référence : Goodfellow et al. [2016]

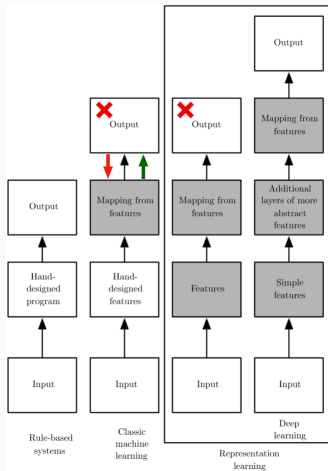


Figure 1.5 from Goodfellow et al. [2016]

# HIERARCHICAL FEATURES REPRESENTATION

Livre de référence : Goodfellow et al. [2016]

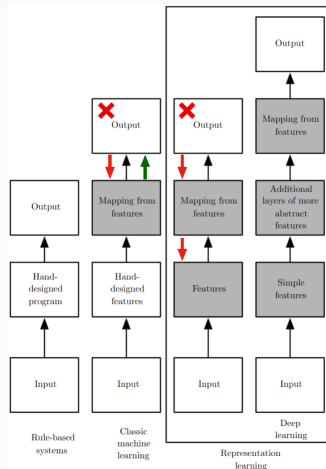


Figure 1.5 from Goodfellow et al. [2016]

# HIERARCHICAL FEATURES REPRESENTATION

Livre de référence : Goodfellow et al. [2016]

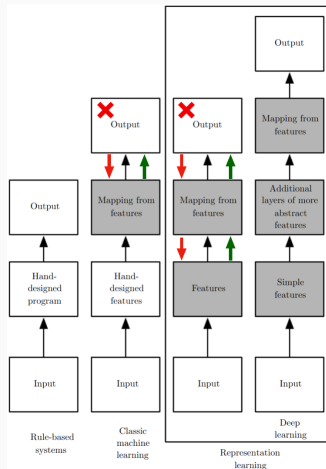


Figure 1.5 from Goodfellow et al. [2016]

# HIERARCHICAL FEATURES REPRESENTATION

Livre de référence : Goodfellow et al. [2016]

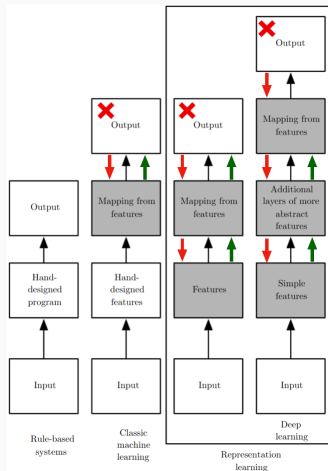
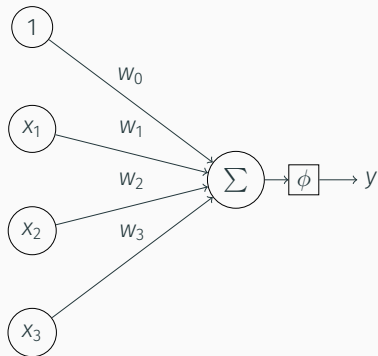


Figure 1.5 from Goodfellow et al. [2016]  $\Rightarrow$  HierarchicalFeatures

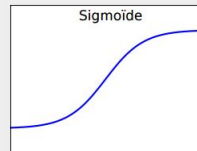
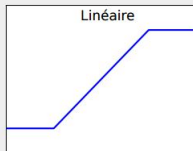
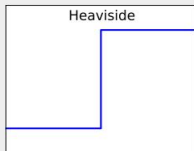
# RÉSEAUX DE NEURONES - PERCEPTRON (1957)



- $y = \phi(Wx)$
- $\phi$  : fonction d'activation
- Apprentissage par  
retro-propagation du gradient  
de l'erreur
- Simple séparatrice linéaire

# FONCTIONS D'ACTIVATION

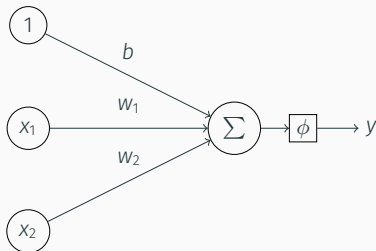
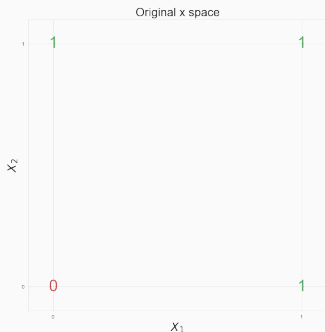
- Heaviside (seuil  $\theta$ )
  - Si  $x < \theta$  alors  $\phi(x) = 0$
  - Si  $x \geq \theta$  alors  $\phi(x) = 1$
- Linéaire - relu(seuils  $\theta_1, \theta_2$ )
  - Si  $x < \theta_1$  ou  $x > \theta_2$  alors  $\phi(x) = 0$
  - Sinon  $\phi(x) = x$
- Sigmoid
  - $\phi(x) = \frac{1}{1+\exp(x)}$



## EXEMPLE - FONCTION "OU"

Soit la fonction logique suivante :

- $f(x_1 = 0, x_2 = 0) = 0$
- $f(x_1 = 1, x_2 = 0) = 1$
- $f(x_1 = 0, x_2 = 1) = 1$
- $f(x_1 = 1, x_2 = 1) = 1$



$\phi$  = Fonction de Heaviside,  $\theta = 0$ .

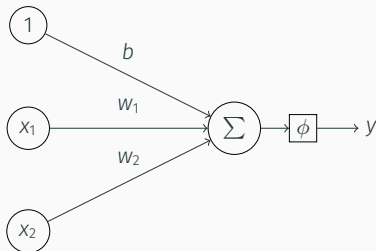
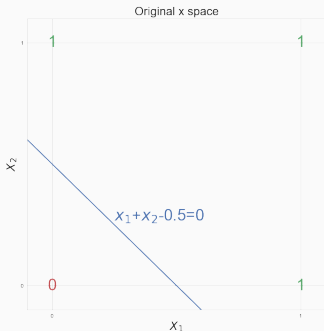
$$y = \phi(w^T x + b)$$

**QUESTION :** Quels poids  $w$  et  $b$  pour modéliser la fonction ?

## EXEMPLE - FONCTION "OU"

Soit la fonction logique suivante :

- $f(x_1 = 0, x_2 = 0) = 0$
- $f(x_1 = 1, x_2 = 0) = 1$
- $f(x_1 = 0, x_2 = 1) = 1$
- $f(x_1 = 1, x_2 = 1) = 1$



$\phi$  = Fonction de Heaviside,  $\theta = 0$ .

$$y = \phi(w^T x + b)$$

**QUESTION :** Quels poids  $w$  et  $b$  pour modéliser la fonction ?

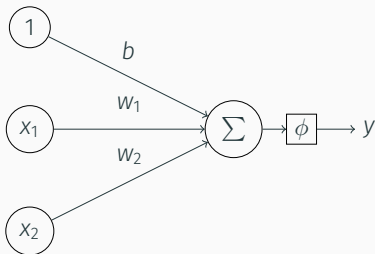
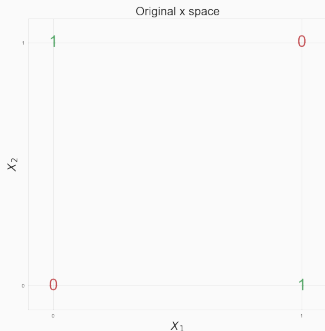
**RÉPONSE :**  $b = -.5, w_1 = 1, w_2 = 1$



## EXEMPLE - FONCTION "XOR"

Soit la fonction logique suivante :

- $f(x_1 = 0, x_2 = 0) = 0$
- $f(x_1 = 1, x_2 = 0) = 1$
- $f(x_1 = 0, x_2 = 1) = 1$
- $f(x_1 = 1, x_2 = 1) = 0$



$\phi$  = Fonction de Heaviside,  $\theta = 0$ .

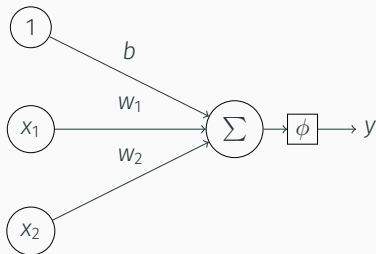
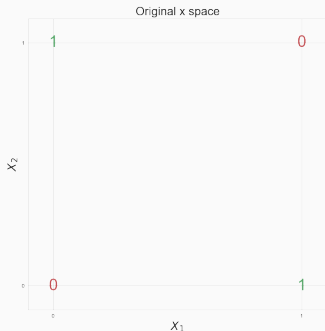
$$y = \phi(w^T x + b)$$

**QUESTION :** Quels poids  $w$  et  $b$  pour modéliser la fonction ?

## EXEMPLE - FONCTION "XOR"

Soit la fonction logique suivante :

- $f(x_1 = 0, x_2 = 0) = 0$
- $f(x_1 = 1, x_2 = 0) = 1$
- $f(x_1 = 0, x_2 = 1) = 1$
- $f(x_1 = 1, x_2 = 1) = 0$



$\phi$  = Fonction de Heaviside,  $\theta = 0$ .  
 $y = \phi(w^T x + b)$

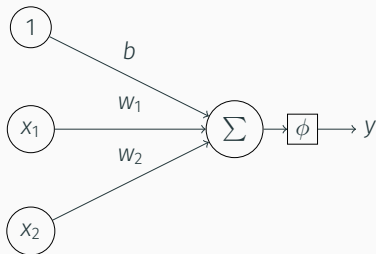
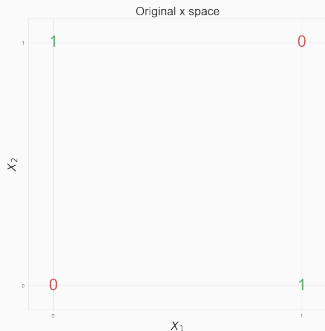
**QUESTION :** Quels poids  $w$  et  $b$  pour modéliser la fonction ?

**RÉPONSE :** Impossible avec cette structure

## EXEMPLE - FONCTION "XOR"

Soit la fonction logique suivante :

- $f(x_1 = 0, x_2 = 0) = 0$
- $f(x_1 = 1, x_2 = 0) = 1$
- $f(x_1 = 0, x_2 = 1) = 1$
- $f(x_1 = 1, x_2 = 1) = 0$



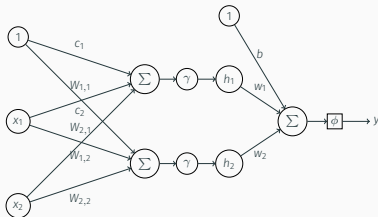
$\phi$  = Fonction de Heaviside,  $\theta = 0$ .  
 $y = \phi(w^T x + b)$

**QUESTION :** Quels poids  $w$  et  $b$  pour modéliser la fonction ?

**RÉPONSE :** Impossible avec cette structure

**SOLUTION :** Changer la structure !

# EXEMPLE - FONCTION "XOR"

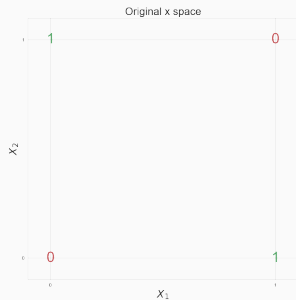


$\phi$  = Fonction de Heaviside.

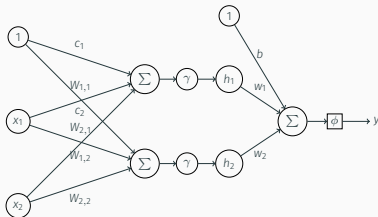
$\gamma$  = Fonction *ReLU*

$$y = \phi(w^T \cdot \gamma(W^T x + c) + b)$$

$$= \phi(w^T h + b)$$



# EXEMPLE - FONCTION "XOR"



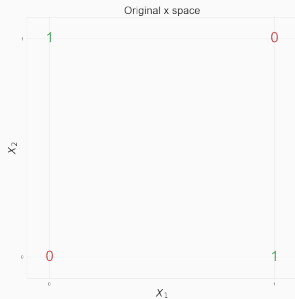
$\phi$  = Fonction de Heaviside.

$\gamma$  = Fonction ReLU

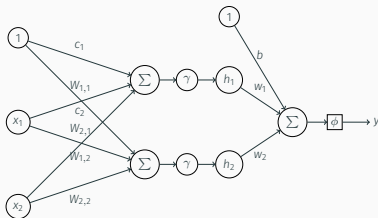
$$y = \phi(w^T \cdot \gamma(W^T x + c) + b)$$

$$= \phi(w^T h + b)$$

**QUESTION :** Quels poids  $W$ ,  $c$  pour que  $h$  soit un espace où un modèle linéaire peut résoudre le problème ?



# EXEMPLE - FONCTION "XOR"



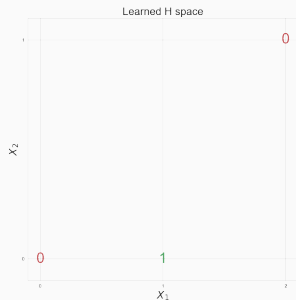
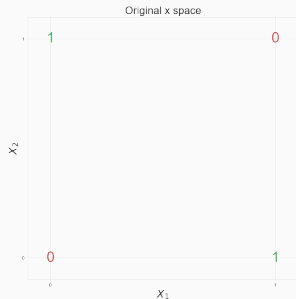
$\phi$  = Fonction de Heaviside.

$\gamma$  = Fonction ReLU

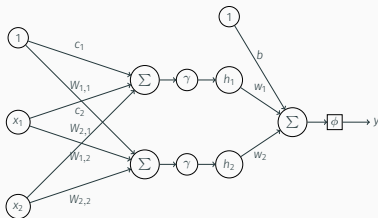
$$y = \phi(w^T \cdot \gamma(W^T x + c) + b) \\ = \phi(w^T h + b)$$

**QUESTION :** Quels poids  $W, c$  pour que  $h$  soit un espace où un modèle linéaire peut résoudre le problème ?

**SOLUTION :**  $W = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, c = (0, 1)$



# EXEMPLE - FONCTION "XOR"



$\phi$  = Fonction de Heaviside.

$\gamma$  = Fonction ReLU

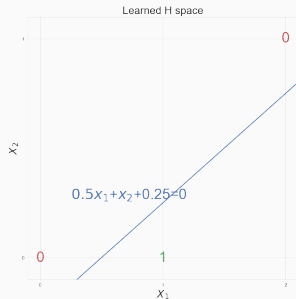
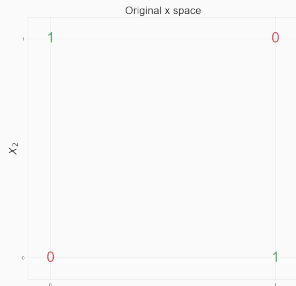
$$y = \phi(w^T \cdot \gamma(W^T x + c) + b)$$

$$= \phi(w^T h + b)$$

**QUESTION :** Quels poids  $W, c$  pour que  $h$  soit un espace où un modèle linéaire peut résoudre le problème ?

**SOLUTION :**  $W = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, c = (0, 1)$

**SOLUTION FINALE :**  $w = [-0.5, 1]^T, b = 0.25$



# POURQUOI MAINTENANT ?

Des modèles anciens :

- Perceptron (1958, *Rosenblatt [1958]*),
- perceptron multicouche, rétro-propagation du gradient (1984, *Rumelhart et al. [1985]*),
- théorème d'approximation universelle (1989, *Cybenko [1989]*, 1991 *Hornik [1991]*),
- réseaux de convolution (1998, (*Le-Net5*) *LeCun et al. [1998]*).

:



# POURQUOI MAINTENANT ?

Des modèles anciens :

- Perceptron (1958, *Rosenblatt [1958]*),
- perceptron multicouche, rétro-propagation du gradient (1984, *Rumelhart et al. [1985]*),
- théorème d'approximation universelle (1989, *Cybenko [1989]*, 1991 *Hornik [1991]*),
- réseaux de convolution (1998, (*Le-Net5*) *LeCun et al. [1998]*).

Avancées académiques récente :

- Utilisation de Relu contre le "vanishing gradient",
- Drop out.

:

# POURQUOI MAINTENANT ?

Des modèles anciens :

- Perceptron (1958, *Rosenblatt [1958]*),
- perceptron multicouche, rétro-propagation du gradient (1984, *Rumelhart et al. [1985]*),
- théorème d'approximation universelle (1989, *Cybenko [1989]*, 1991 *Hornik [1991]*),
- réseaux de convolution (1998, (*Le-Net5*) *LeCun et al. [1998]*).

Avancées académiques récente :

- Utilisation de Relu contre le "vanishing gradient",
- Drop out.

Plus de données (*BigData*) :

# POURQUOI MAINTENANT ?

Des modèles anciens :

- Perceptron (1958, *Rosenblatt [1958]*),
- perceptron multicouche, rétro-propagation du gradient (1984, *Rumelhart et al. [1985]*),
- théorème d'approximation universelle (1989, *Cybenko [1989]*, 1991 *Hornik [1991]*),
- réseaux de convolution (1998, (*Le-Net5*) *LeCun et al. [1998]*).

Avancées académiques récente :

- Utilisation de Relu contre le "vanishing gradient",
- Drop out.

Plus de données (*BigData*) :

Avancées technologiques :

- Capacités de calcul accrues (*cluster, GPU*),
- API simples à utiliser.



Keras

A deep learning library

gensim

theano

PYTORCH

Microsoft

CNTK

mxnet



Caffe2

spaCy



Keras

A deep learning library

gensim

theano

PYTORCH

Microsoft  
CNTK

mxnet

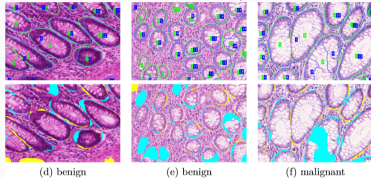


Caffe2

spaCy

- Mise à disposition de modèle pré-entraîné...
- ... de (très) nombreuses nouvelles applications.

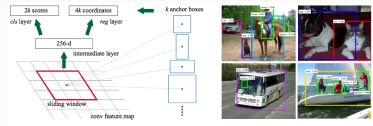
## Tumor Detection (*Kainz et al. [2015]*)



## Sampled Celebrities (*Karras et al. [2017]*)



## Faster R-CNN (*Ren et al. [2015]*)



Neu-

## ral Style (*Gatys et al. [2015]*)



# LES APPLICATIONS : VISION + NLP

## Stack Gan ++ (Zhang et al. [2017])



Image

## Captioning (Karpathy and Fei-Fei [2015])



"man in black shirt is playing guitar."

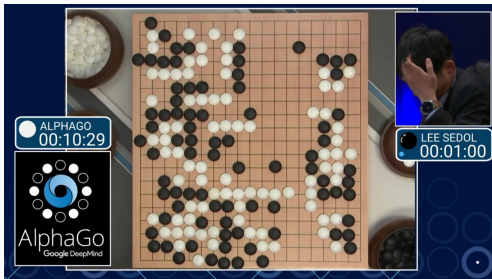


"construction worker in orange safety vest is working on road."

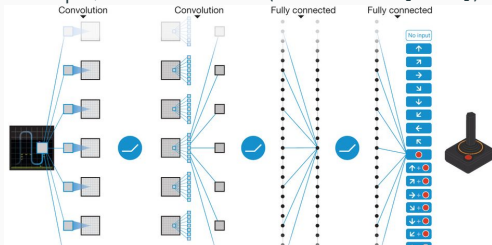


"two young girls are playing with lego toy."

## AlphaGo



## DeepQ - AtariGames (Mnih et al. [2015])





# DE NOUVEAUX FRAMEWORK

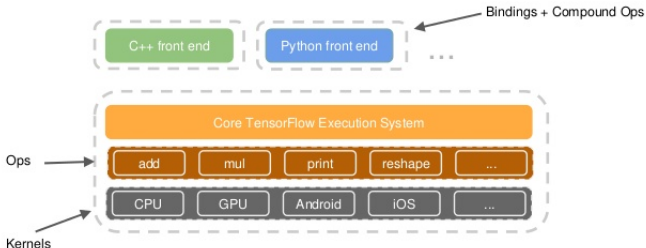
---

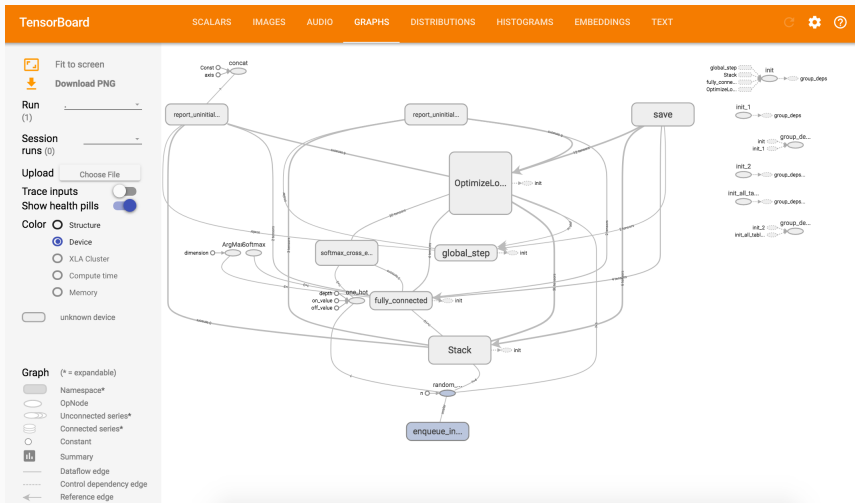
Un framework open source développé par *Google Brain* (2015).

- Implémentation du noyau en C++/CUDA
- Différentes API (Python, Java, C++, Go)
- Différentes API de "Haut niveau" (Keras)



## TensorFlow Architecture





# KERAS - POURQUOI UNE TELLE API ?

Pour plus de simplicité !

Définition d'une couche de convolution en :



- TENSORFLOW

```
kernel = tf.Variable(tf.truncated_normal([3, 3, 64, 64], type=tf.float32, stddev=1e-1),  
name='weights')  
conv = tf.nn.conv2d(self.conv1_1, kernel, [1, 1, 1, 1], padding='SAME')  
biases = tf.Variable(tf.constant(0.0, shape=[64], dtype=tf.float32), trainable=True,  
name='biases')  
out = tf.nn.bias_add(conv, biases)  
self.conv1_2 = tf.nn.relu(out, name='block1_conv2')
```

- KERAS

```
x = Convolution2D(64, 3, 3, activation='relu', border_mode='same', name='block1_conv2')(x)
```

Officiellement supporté par Google :

Github : <https://github.com/fchollet/keras>

Il existe deux façons de construire des modèles de réseaux de neurones avec KERAS :

- SEQUENTIAL :

```
model = Sequential()  
model.add(Dense(32, input_shape=(500,)))  
model.add(Dense(10, activation='softmax'))  
...
```

- MODEL API :

```
a = Input(shape=(32,))  
b = Dense(32)(a)  
model = Model(inputs=a, outputs=b)
```

Il existe deux façons de construire des modèles de réseaux de neurones avec KERAS :

- **SEQUENTIAL** : Utiliser pour les TP

```
model = Sequential()  
model.add(Dense(32, input_shape=(500,)))  
model.add(Dense(10, activation='softmax'))  
...
```

- **MODEL API** :

```
a = Input(shape=(32,))  
b = Dense(32)(a)  
model = Model(inputs=a, outputs=b)
```

Les fonctions LAYERS permettent d'ajouter différentes couches à un modèle SEQUENTIAL, parmi lesquelles :

- DENSE
  - *Fully connected layer*
- ACTIVATION
  - *Relu, sigmoid, tanh*
- DROPOUT
- FLATTEN, RESHAPE
- CONV2D
  - *Réseau de convolution*
- MAXPOOLING2D
- LSTM
  - *Long Short-Term Memory*
- ...

Keras permet d'utiliser plusieurs modèles de classification d'images entraînés sur la base *ImageNet*.

- XCEPTION
- VGG16
- VGG19
- RESNET50
- INCEPTIONV3
- INCEPTIONRESNETV2
- MOBILENET



```
# Définition du réseau
model = km.Sequential()
model.add(kl.Dense(512, activation='relu', input_shape=(784,)))
model.add(kl.Dropout(0.2))
model.add(kl.Dense(N_classes, activation='softmax'))

# Compilation
model.compile(loss='categorical_crossentropy',
              optimizer=ko.RMSprop(),
              metrics=['accuracy'])

# Apprentissage
history = model.fit(X_train, Y_train_cat,
                   batch_size=batch_size,
                   epochs=epochs,
                   verbose=1,
                   validation_data=(X_test, Y_test_cat))

# Evaluation
score_mpl = model.evaluate(X_test, Y_test_cat, verbose=0)
```

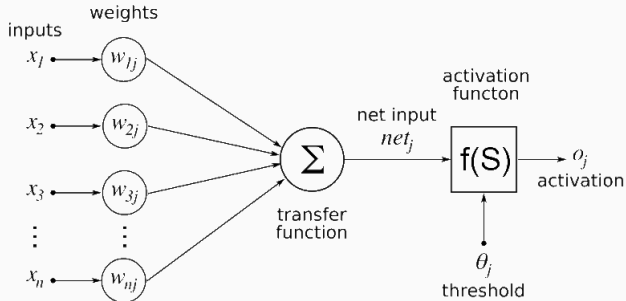
# CPU VS GPU

---

# DEEP LEARNING ET APPRENTISSAGE

L'apprentissage d'un réseau de neurone est composé de deux opérations principales :

- **Forward Pass** : Les *input* passent à travers le réseau entier jusqu'à obtenir une valeur en sortie.
- **Backward Pass** Les poids de chaque neurone sont mis à jour à partir de l'erreur obtenue pendant l'étape forward.



⇒ Essentiellement des multiplications de matrices.

Essentiellement des multiplications de matrice :

$$\begin{array}{c} \text{row vectors} \\ \mathbf{a} \rightarrow \\ \mathbf{b} \rightarrow \\ \mathbf{c} \rightarrow \end{array} \begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{pmatrix} \begin{array}{c} \text{column vectors} \\ \mathbf{x} \quad \mathbf{y} \quad \mathbf{z} \\ \downarrow \quad \downarrow \quad \downarrow \\ \begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{pmatrix} \end{array}$$

**PROBLÈME :** Calcul simple mais en très grand nombre.

Modèle VGG : 138,357,544 paramètres à optimiser à chaque itération !

**SOLUTION :** Le calcul GPU.

# CPU VS GPU

## CPU :

- Peu de coeur....
- .. mais très complexe et rapide,
- mémoire partagée avec le système.

⇒ Taches séquentielles.

## GPU :

- Beaucoup de coeur...
- ...mais peu complexe et peu rapide,
- mémoire séparée du système.

⇒ Taches en parallèles.

## A L'INSA - GEI 103

CPU : INTEL XEON E5-1620 v4		GPU : GEFORCE GTX 1080
Cores	4 (8 threads)	2560
Fréquence	3,5/3,8Ghz	1,6/1,73Ghz

- Pour les autres algorithmes d'apprentissage le GPU n'est pas forcément nécessaire.
  - *Complexité moindre*
- Le temps de chargement des données vers le GPU peut-être coûteux!
  - *GPU n'est utile que si le temps de calcul est supérieur au temps de chargement.*
  - *i.e. utile pour des modèles complexes*

# SOURCES

- Goodfellow et al. [2016]
- <https://github.com/m2dsupsdclass/lectures-labs>

# RÉFÉRENCES

---

- George Cybenko. Approximations by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2 :183–192, 1989.
- Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv :1508.06576*, 2015.
- Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.
- Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2) :251–257, 1991.



## REFERENCES II

- Philipp Kainz, Michael Pfeiffer, and Martin Urschler. Semantic segmentation of colon glands with deep convolutional neural networks and total variation segmentation. *arXiv preprint arXiv :1511.06919*, 2015.
- Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3128–3137, 2015.
- Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality. *Stability, and Variation. arXiv preprint*, 2017.
- Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11) :2278–2324, 1998.

## REFERENCES III

- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540) :529, 2015.
- Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn : Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.
- Frank Rosenblatt. The perceptron : a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6) :386, 1958.

## REFERENCES IV

- David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams.  
Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.
- Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris Metaxas. Stackgan++ : Realistic image synthesis with stacked generative adversarial networks. *arXiv preprint arXiv :1710.10916*, 2017.