INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Deep Learning

Introduction

Brendan Guillouet

2018

Institut National des Sciences Appliquées

Introduction

Plan de Présentation

Qu'est-ce que le deep learning ?

Livre de référence : Goodfellow et al. [2016]

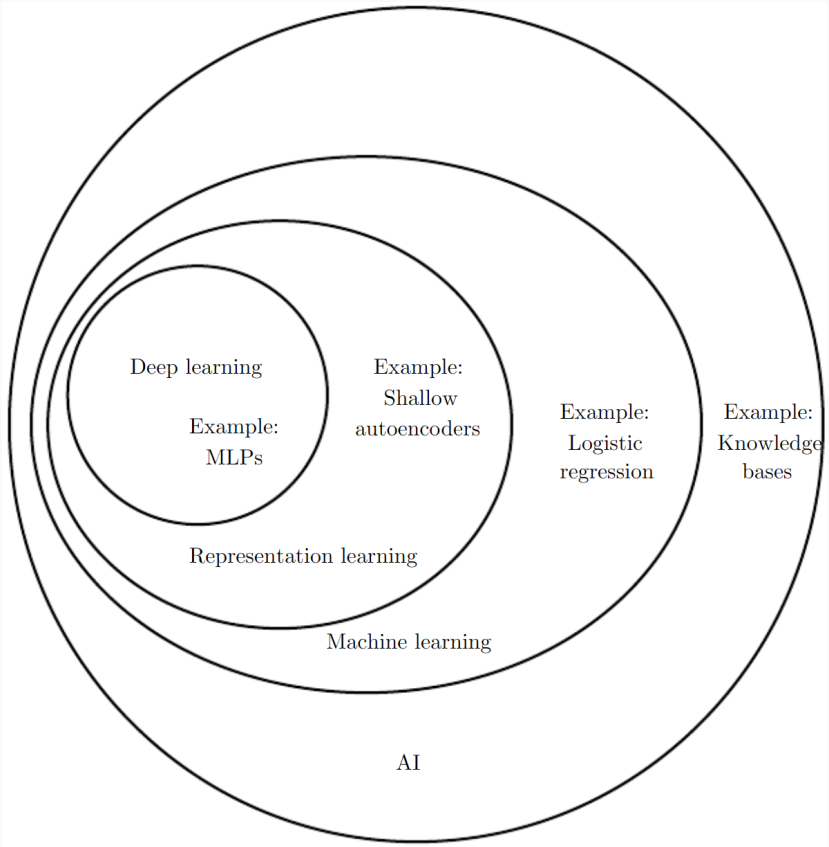


Figure 1.4 from Goodfellow et al. [2016]

Hierarchical features representation

Livre de référence : Goodfellow et al. [2016]

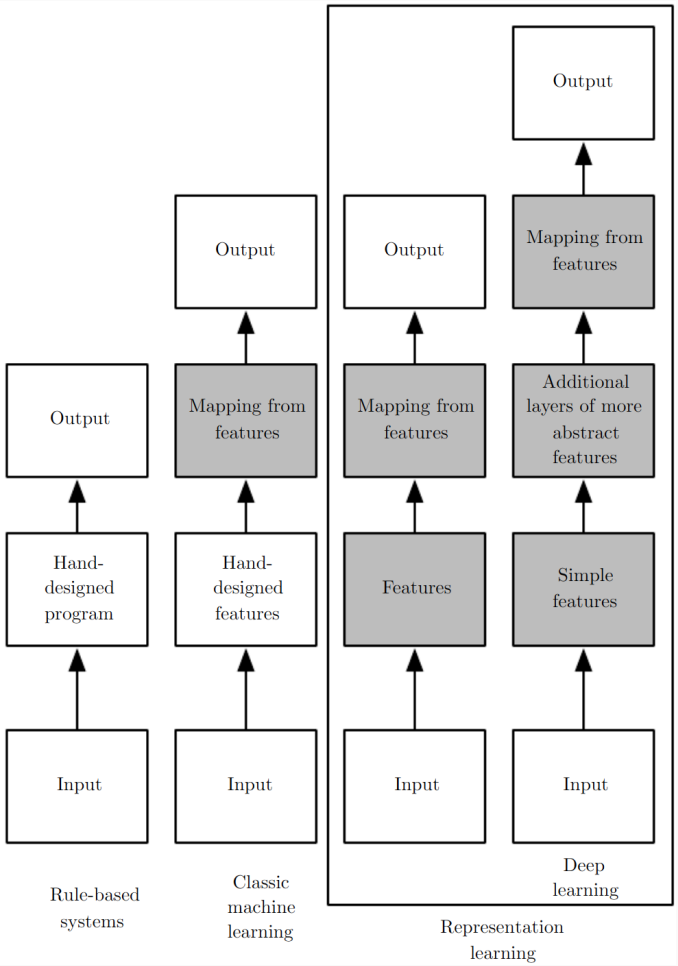


Figure 1.5 from Goodfellow et al. [2016]

Hierarchical features representation

Livre de référence : Goodfellow et al. [2016]

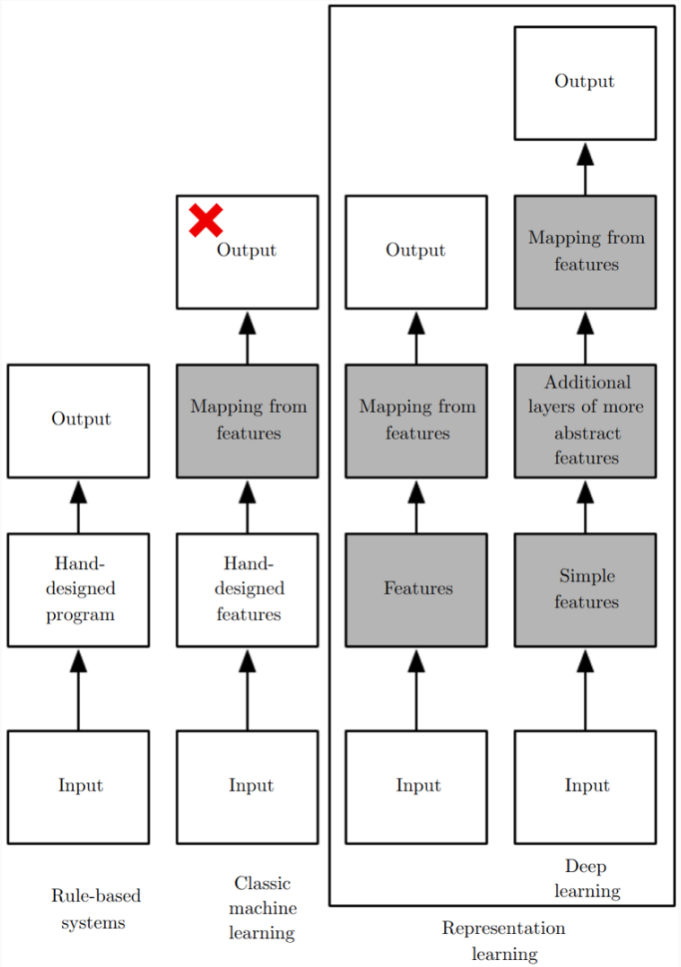


Figure 1.5 from Goodfellow et al. [2016]

Hierarchical features representation

Livre de référence : Goodfellow et al. [2016]

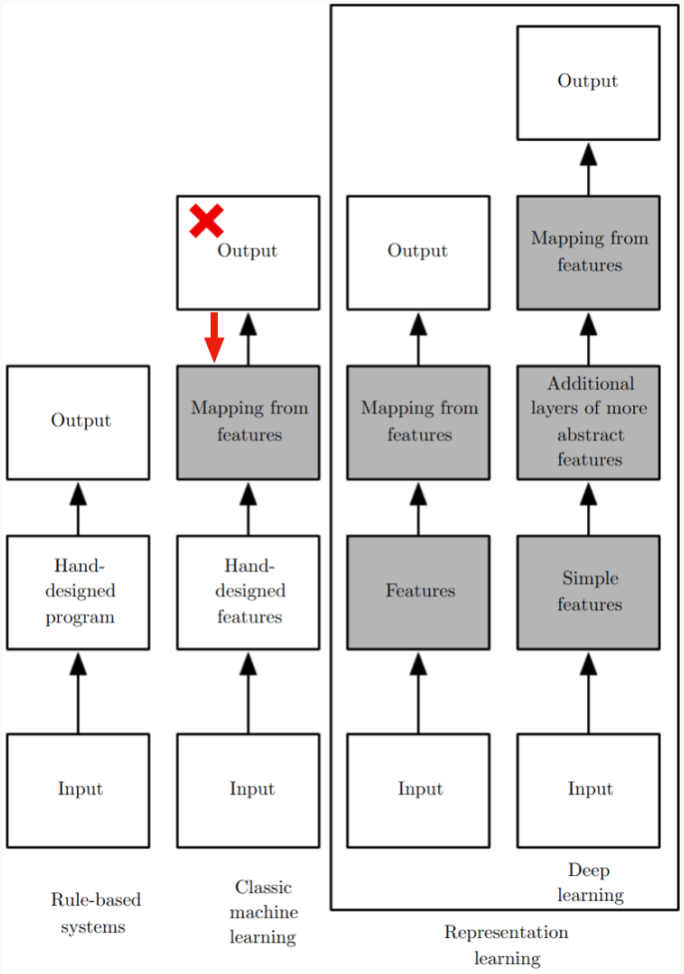


Figure 1.5 from Goodfellow et al. [2016]

Hierarchical features representation

Livre de référence : Goodfellow et al. [2016]

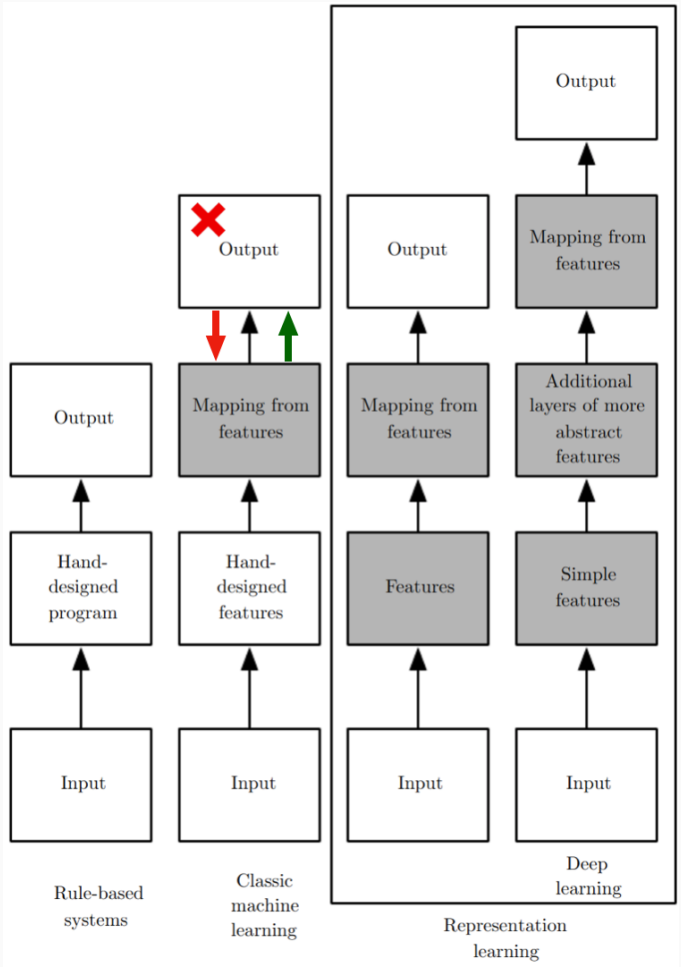


Figure 1.5 from Goodfellow et al. [2016]

Hierarchical features representation

Livre de référence : Goodfellow et al. [2016]

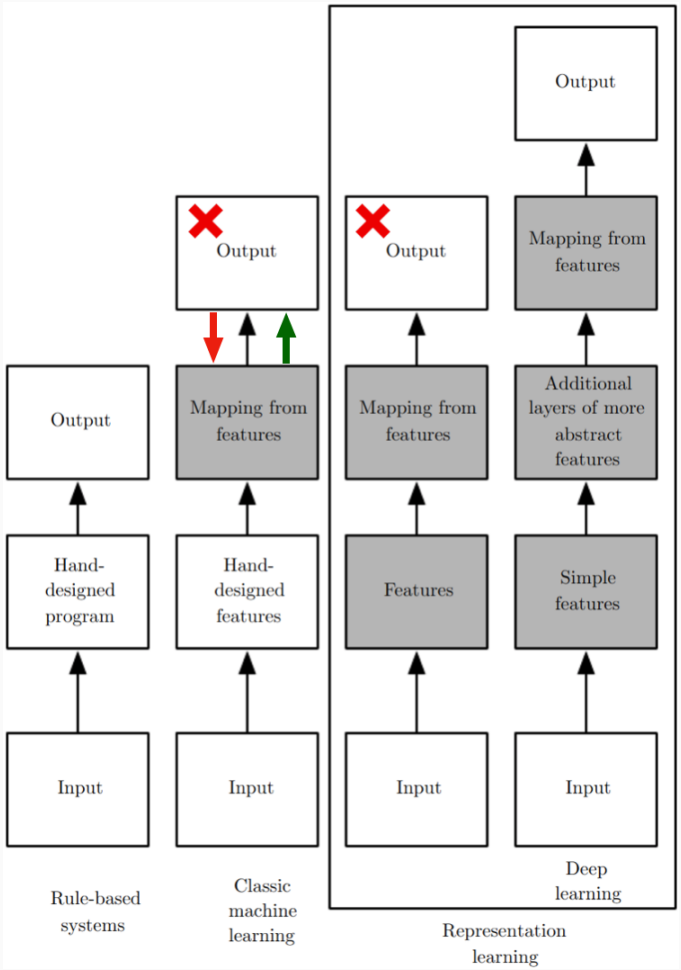


Figure 1.5 from Goodfellow et al. [2016]

Hierarchical features representation

Livre de référence : Goodfellow et al. [2016]

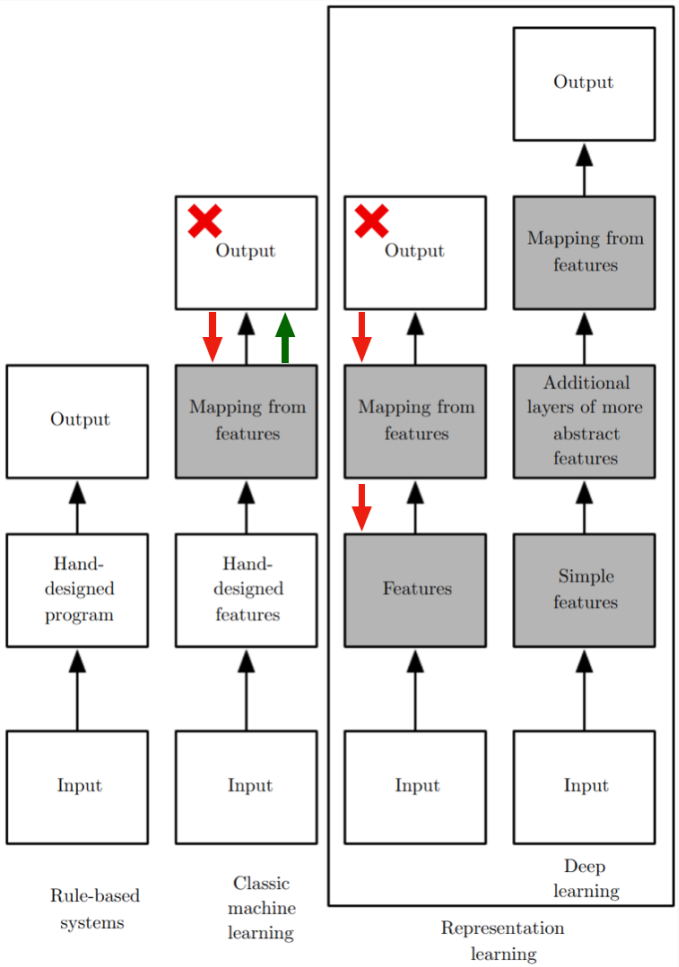


Figure 1.5 from Goodfellow et al. [2016]

Hierarchical features representation

Livre de référence : Goodfellow et al. [2016]

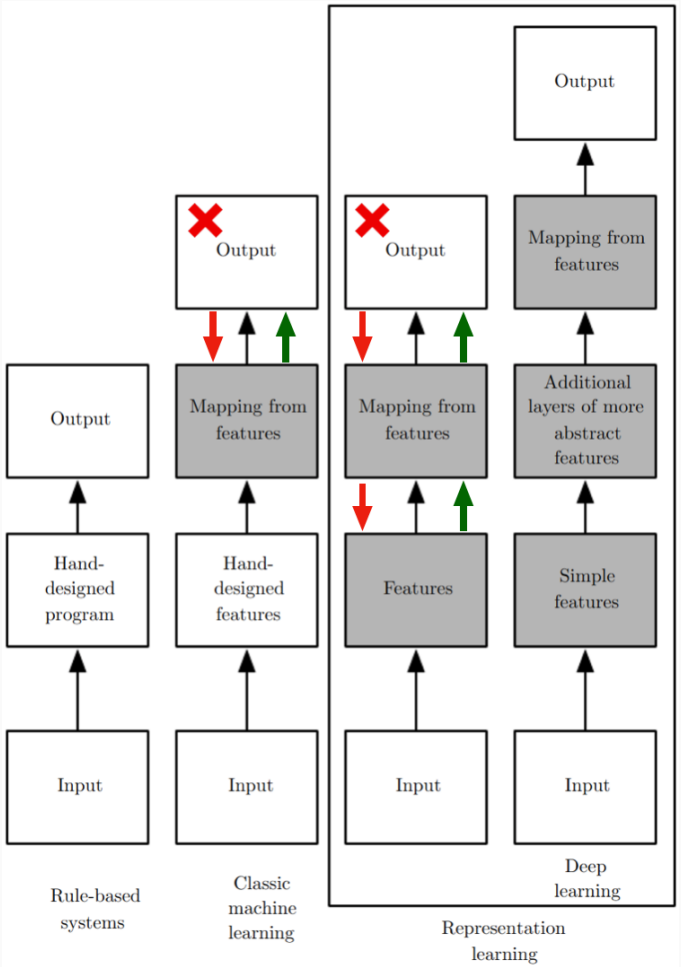


Figure 1.5 from Goodfellow et al. [2016]

Hierarchical features representation

Livre de référence : Goodfellow et al. [2016]

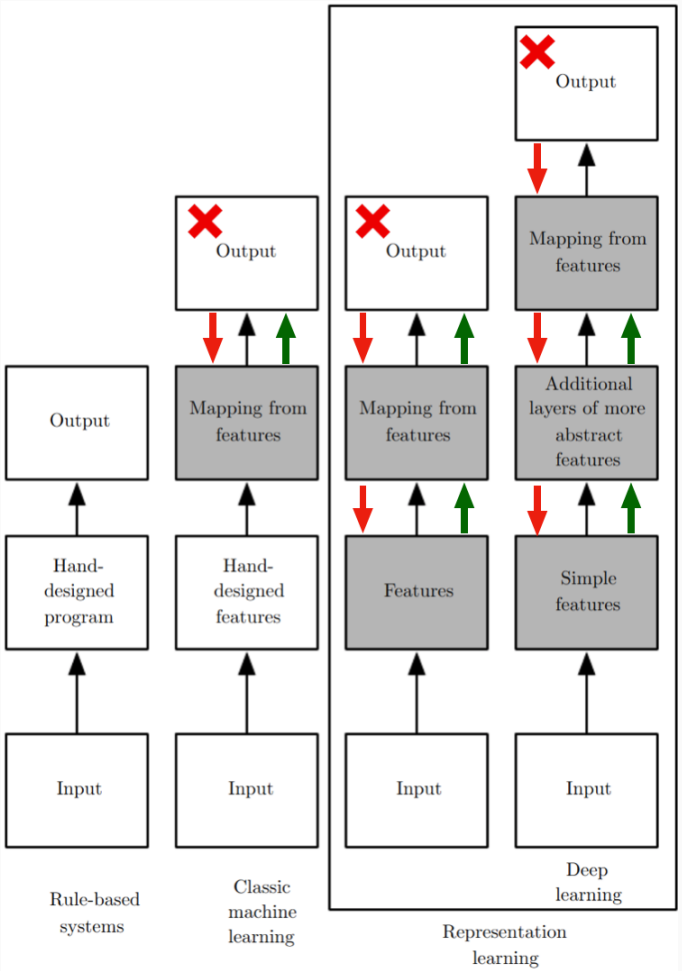
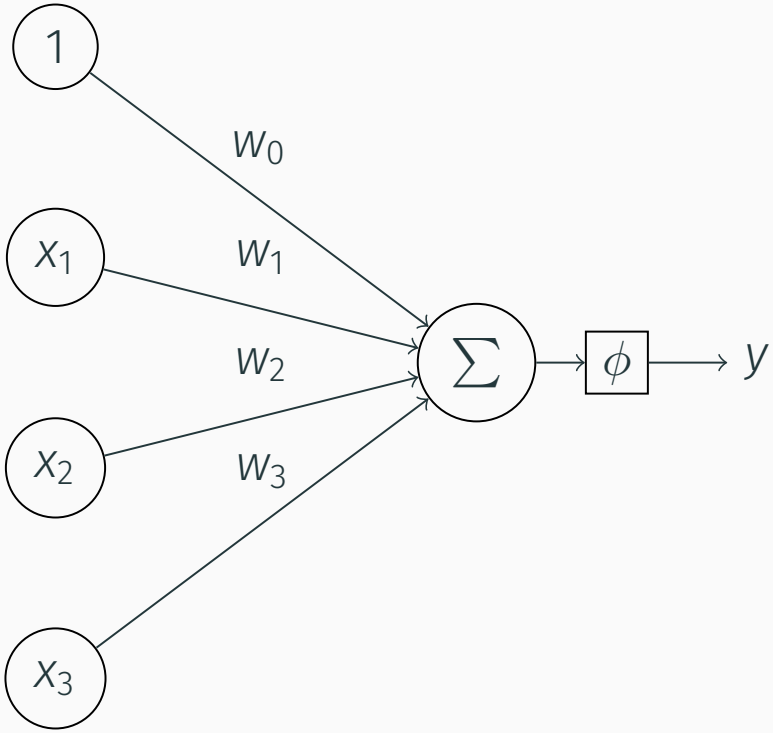


Figure 1.5 from Goodfellow et al. [2016] $\Longrightarrow$ *HierarchicalFeatures*

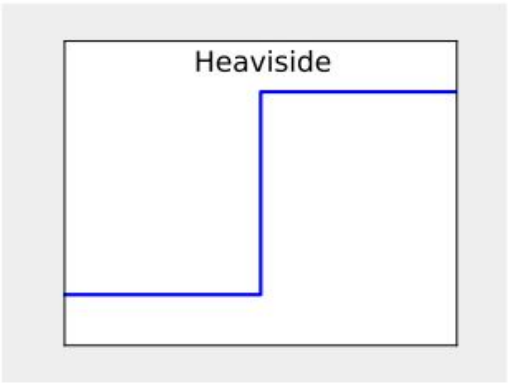
Réseaux de neurones - Perceptron (1957)

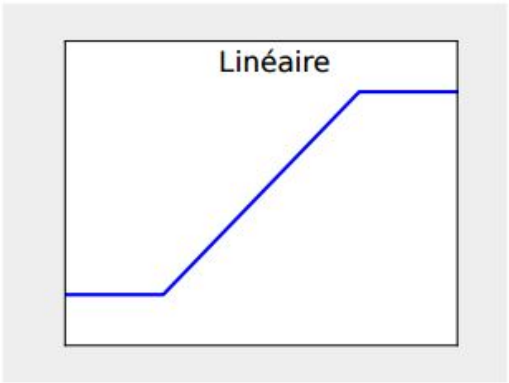


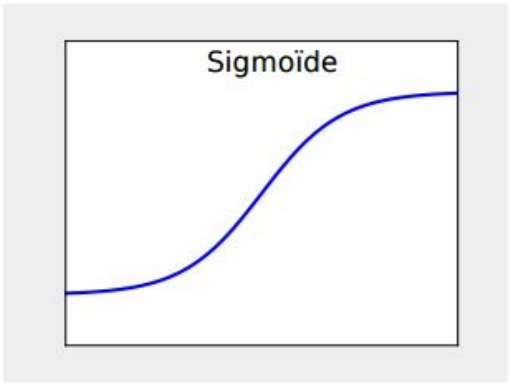
- $y = \phi(Wx)$
- ϕ : fonction d'activation
- Apprentissage par retro-propagation du gradient de l'erreur
- Simple séparatrice linéaire

Fonctions d'activation

- Heaviside (seuil θ)
  - Si $x < \theta$ alors $\phi(x) = 0$
  - Si $x \geq \theta$ alors $\phi(x) = 1$
- Linéaire - relu(seuils θ_1, θ_2)
  - Si $x < \theta_1$ ou $x > \theta_2$ alors $\phi(x) = 0$
  - Sinon $\phi(x) = x$
- Sigmoïd
  - $\phi(x) = \frac{1}{1+exp(x)}$



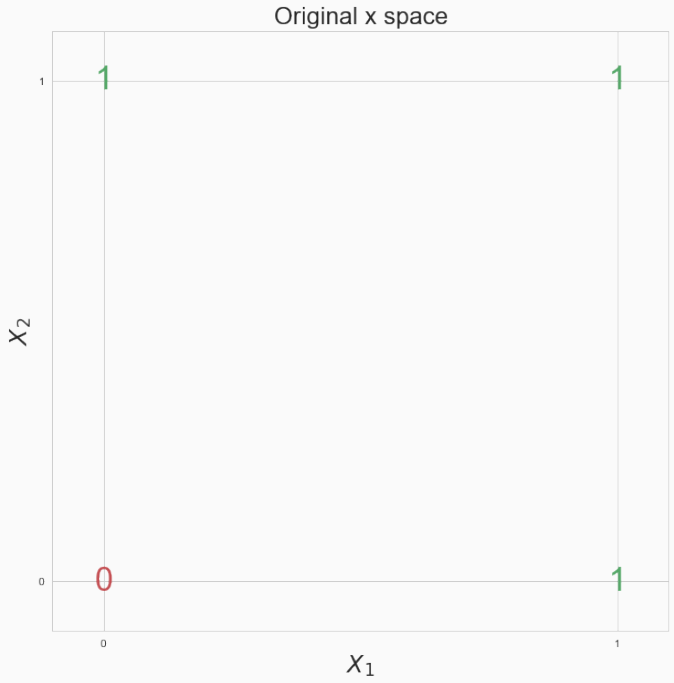


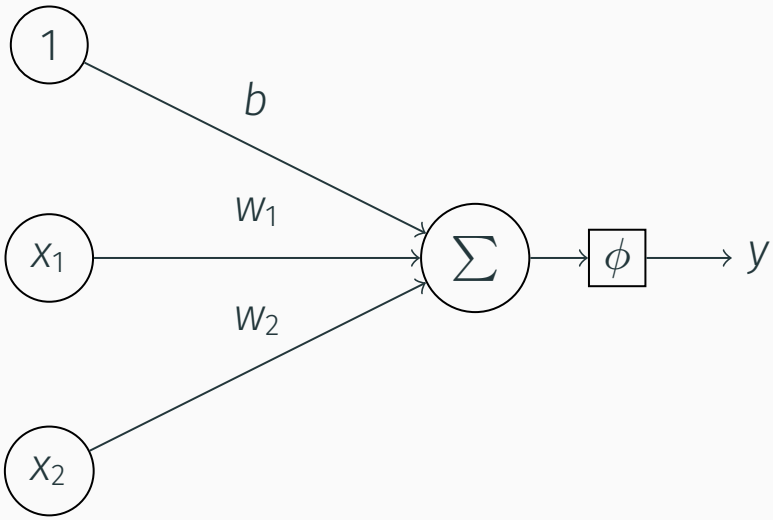


Exemple - Fonction "ou"

Soit la fonction logique suivante :

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





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

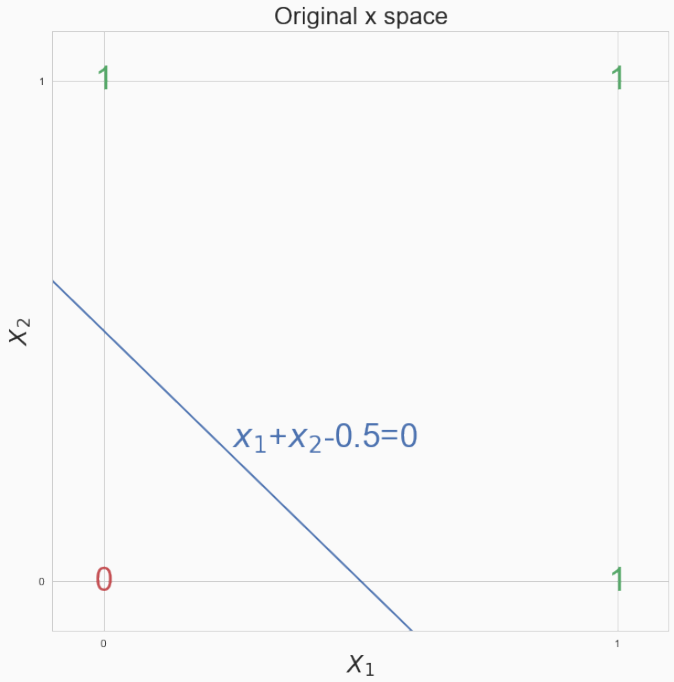
$y = \phi(w^T x + b)$

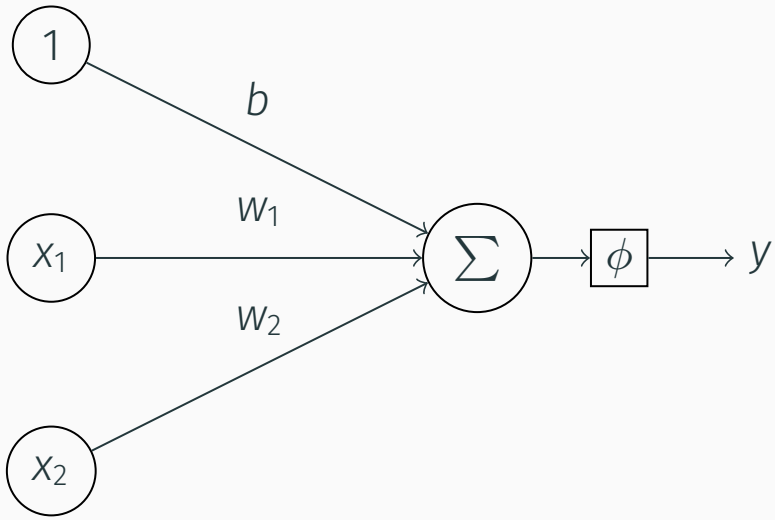
Question : Quels poids w et b pour modéliser la fonction ?

Exemple - Fonction "ou"

Soit la fonction logique suivante :

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





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

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

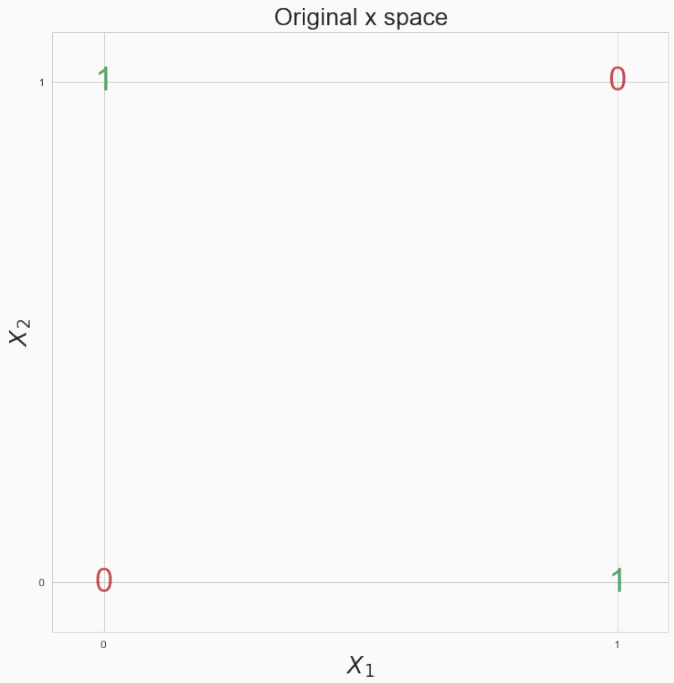
Question : Quels poids w et b pour modéliser la fonction ?

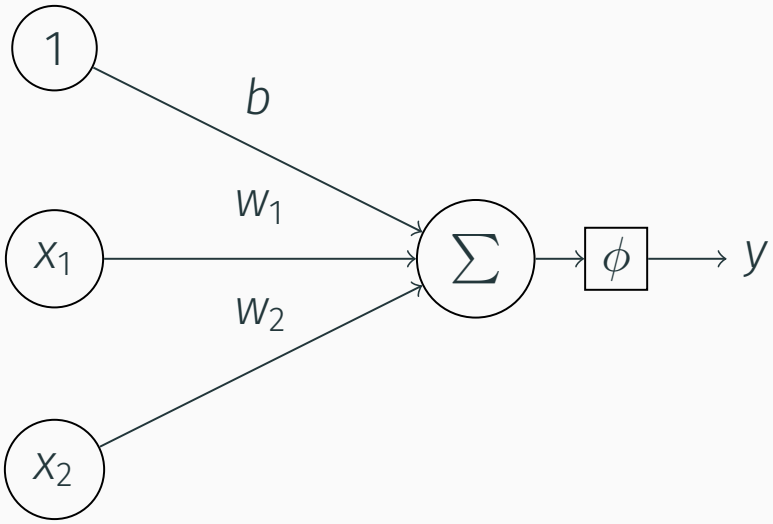
Réponse : $b = -.5, w_1 = 1, w_2 = 1$

Exemple - Fonction "xor"

Soit la fonction logique suivante :

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





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

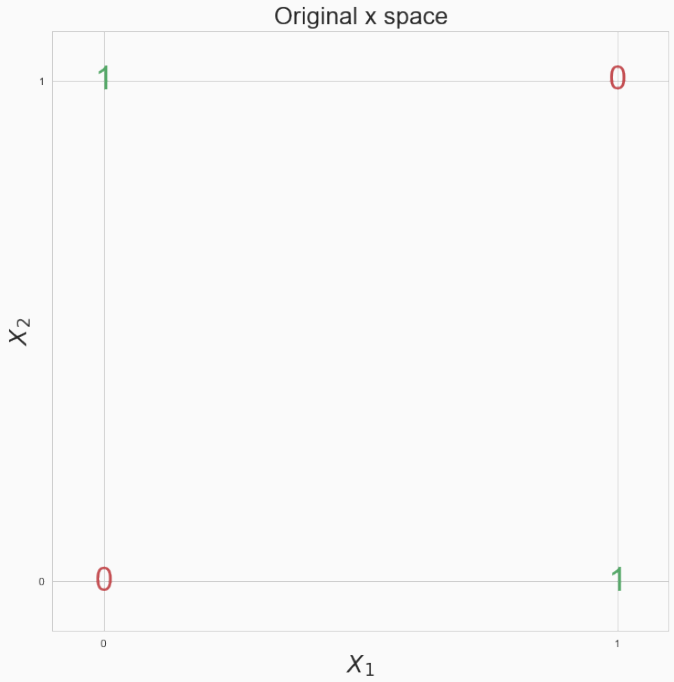
$y = \phi(w^T x + b)$

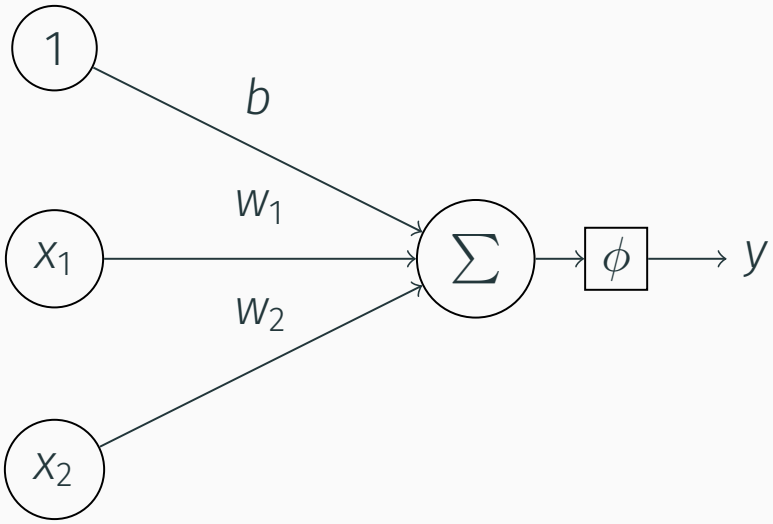
Question : Quels poids w et b pour modéliser la fonction ?

Exemple - Fonction "xor"

Soit la fonction logique suivante :

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





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

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

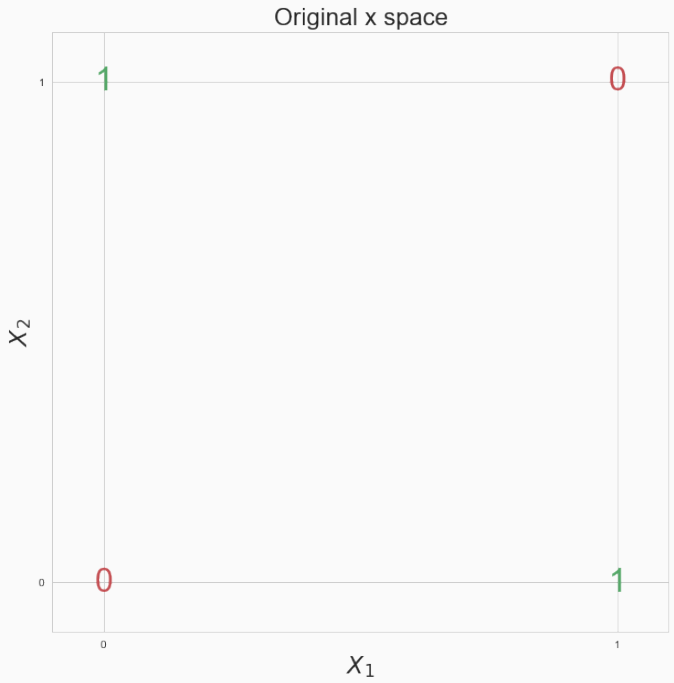
Question : Quels poids w et b pour modéliser la fonction ?

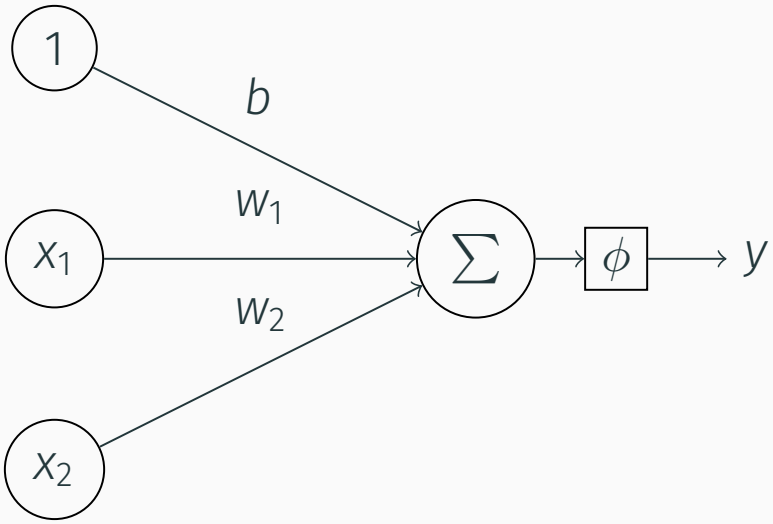
Réponse : Impossible avec cette structure

Exemple - Fonction "xor"

Soit la fonction logique suivante :

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





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

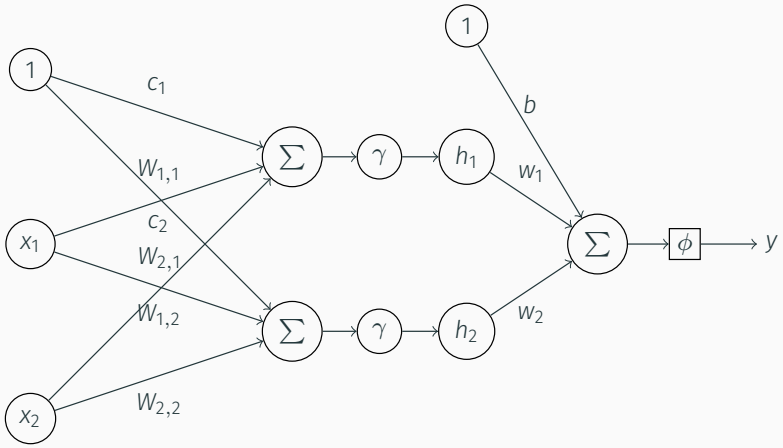
$y = \phi(w^T x + b)$

Question : Quels poids w et b pour modéliser la fonction ?

Réponse : Impossible avec cette structure

Solution : Changer la structure !

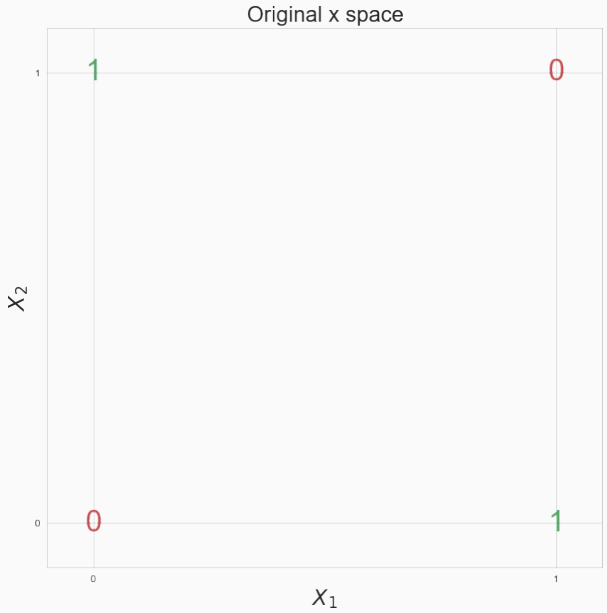
Exemple - Fonction "xor"



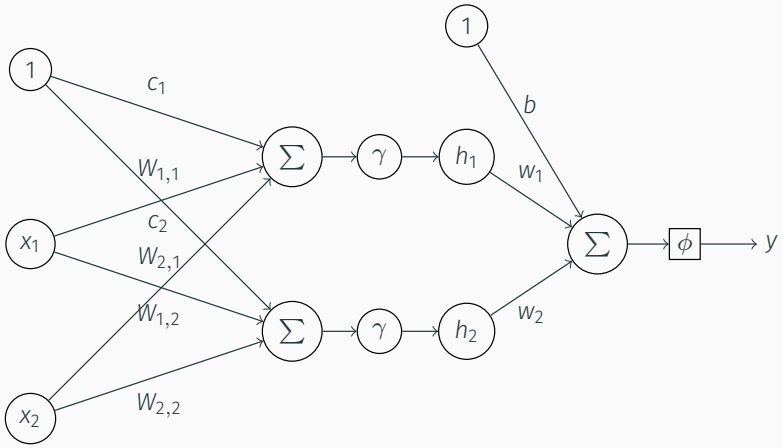
$\phi =$ Fonction de *Heaviside*.

$\gamma =$ Fonction*ReLu*

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



Exemple - Fonction "xor"

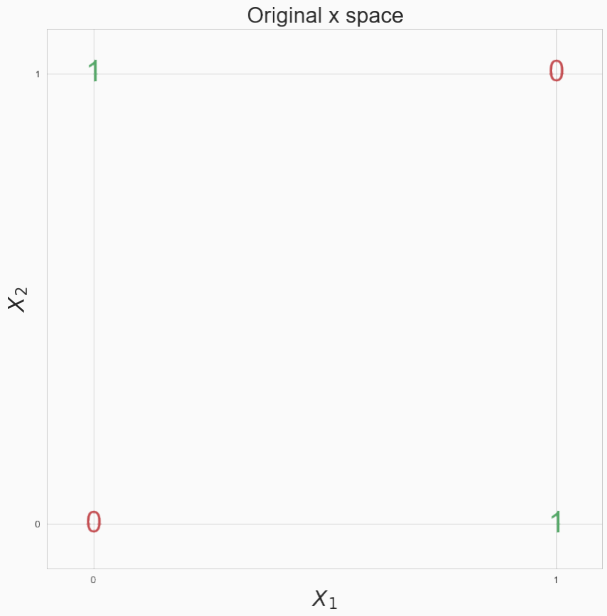


$\phi =$ Fonction de *Heaviside*.

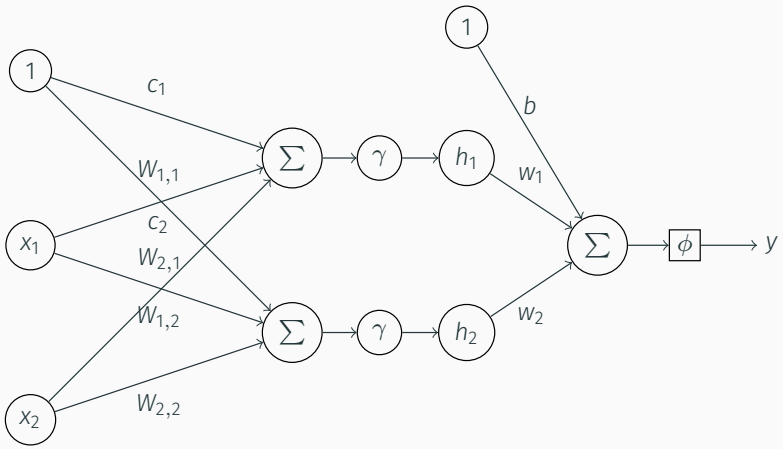
$\gamma =$ Fonction*ReLu*

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

Question : Quels poids W, c pour que h soit un espace ou un modèle linéaire peut résoudre le problème ?



Exemple - Fonction "xor"



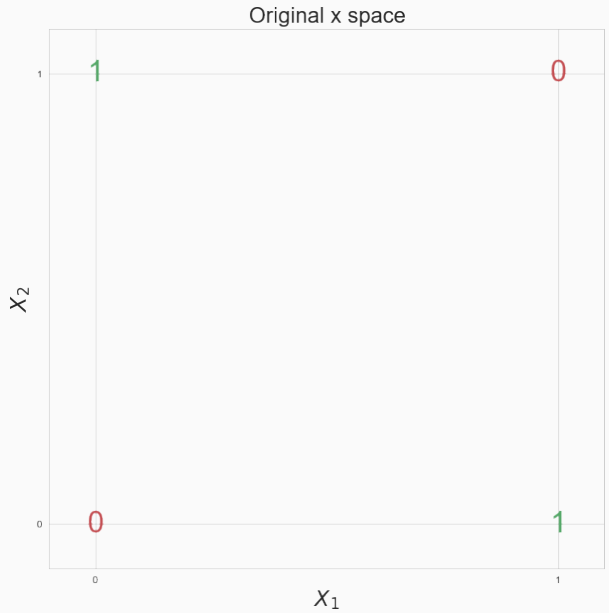
$\phi =$ Fonction de *Heaviside*.

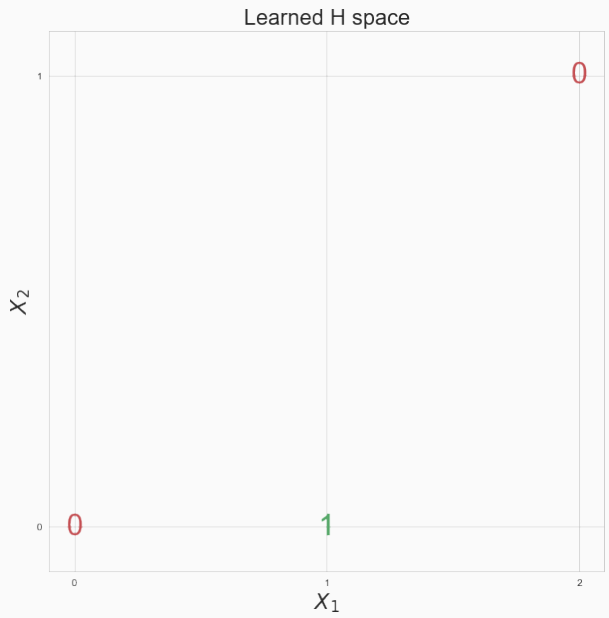
$\gamma =$ Fonction*ReLu*

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

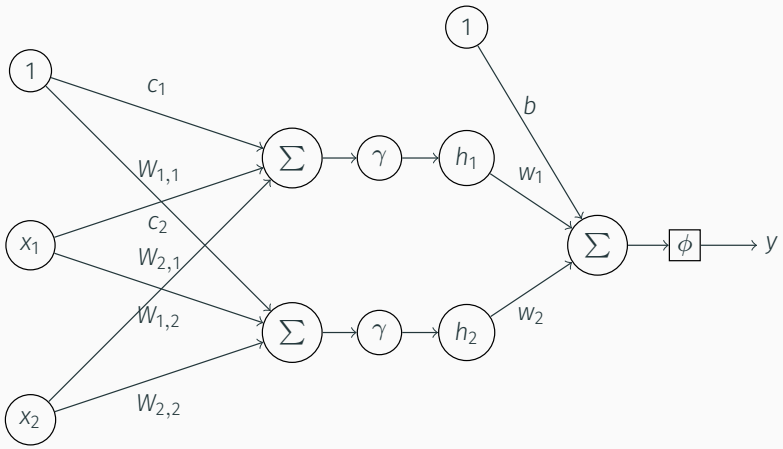
Question : Quels poids W, c pour que h soit un espace ou un modèle linéaire peut résoudre le problème ?

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





Exemple - Fonction "xor"



$\phi =$ Fonction de *Heaviside*.

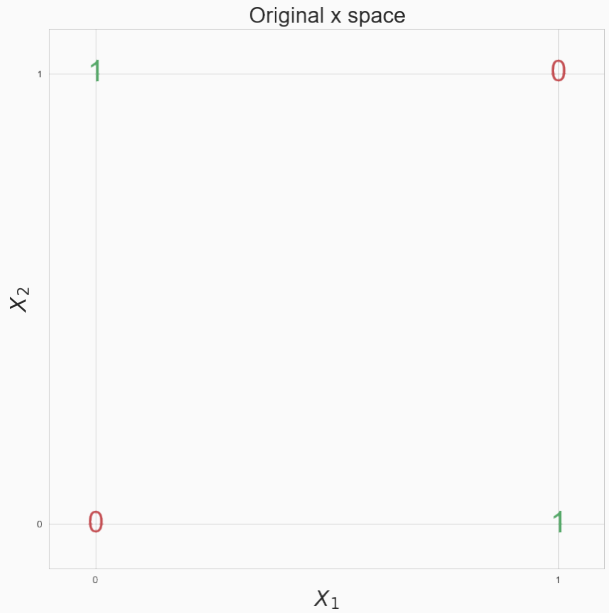
$\gamma =$ Fonction*ReLu*

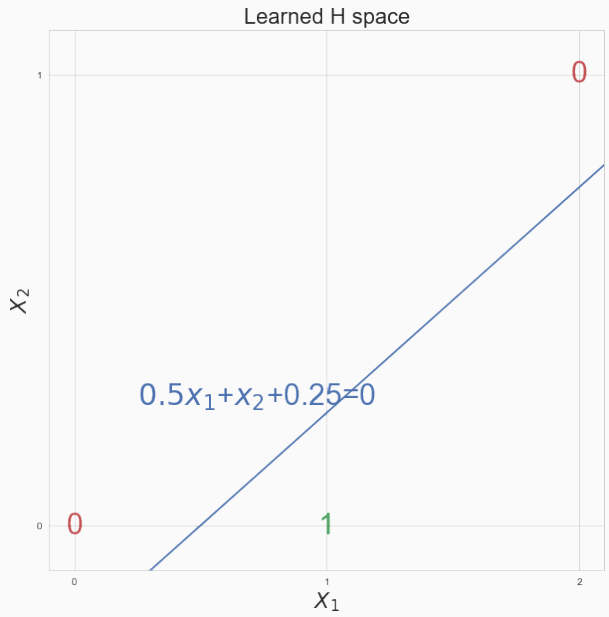
$$y = \phi(w^T.\gamma(W^T x + c) + b)$$
$$= \phi(w^T h + b)$$

Question : Quels poids W, c pour que h soit un espace ou 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.

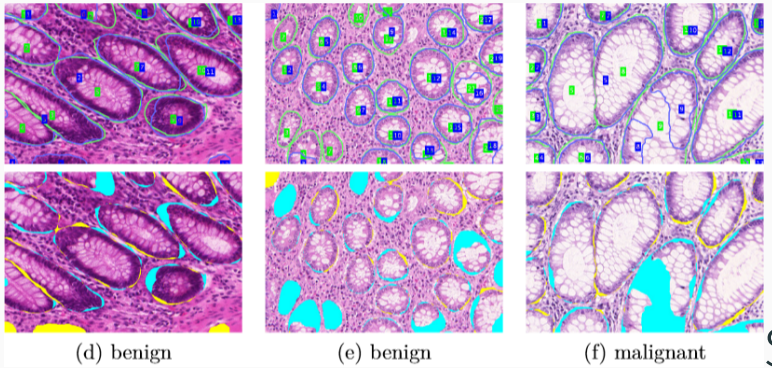
Librairies et framework de DeepLearning

Librairies et framework de DeepLearning

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

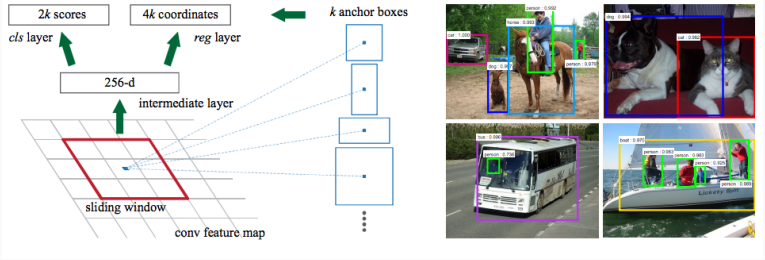
Les applications : Vision

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



Sampled Celebreities (*Karras et al. [2017]*)

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



Neural Style (*Gatys et al. [2015]*)

Les applications : Vision + NLP

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

Image Captionning (*Karpathy and Fei-Fei [2015]*)

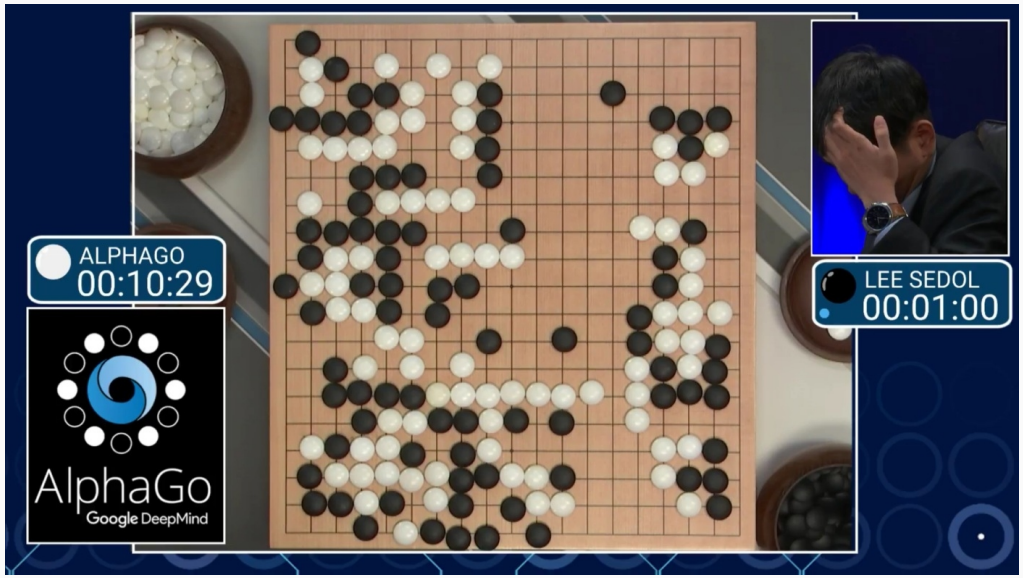

"man in black shirt is playing guitar."


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

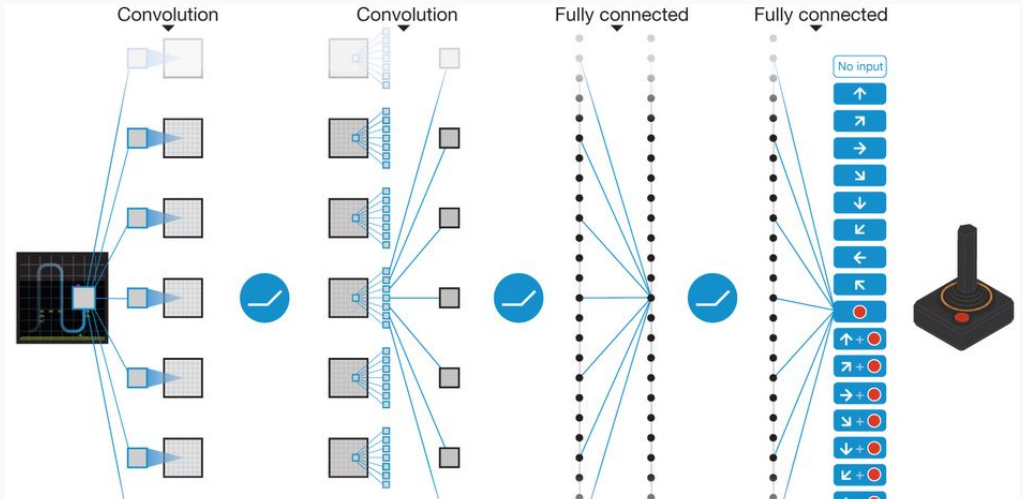

"two young girls are playing with lego toy."

Les applications : Reinforcement Learning

AlphaGo



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

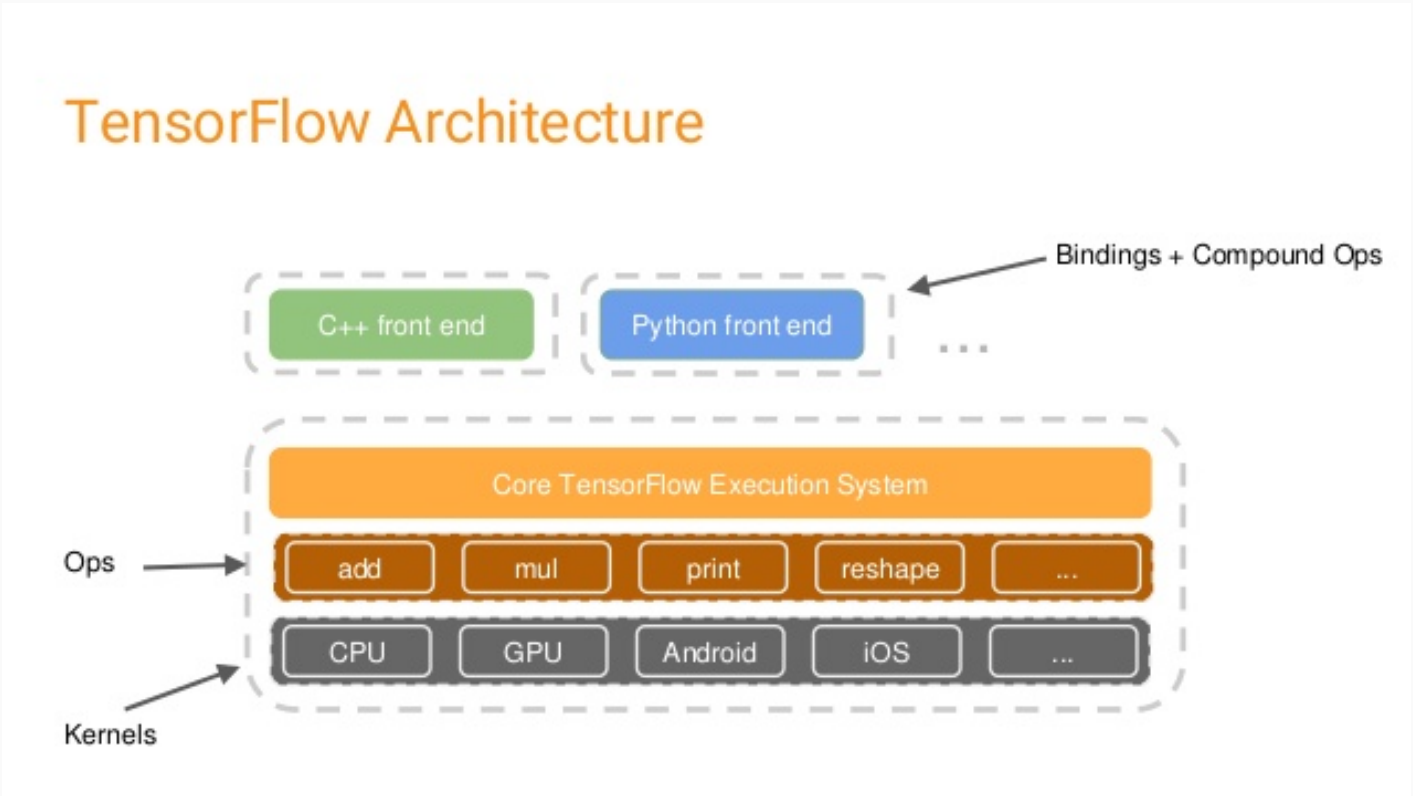


De Nouveaux Framework

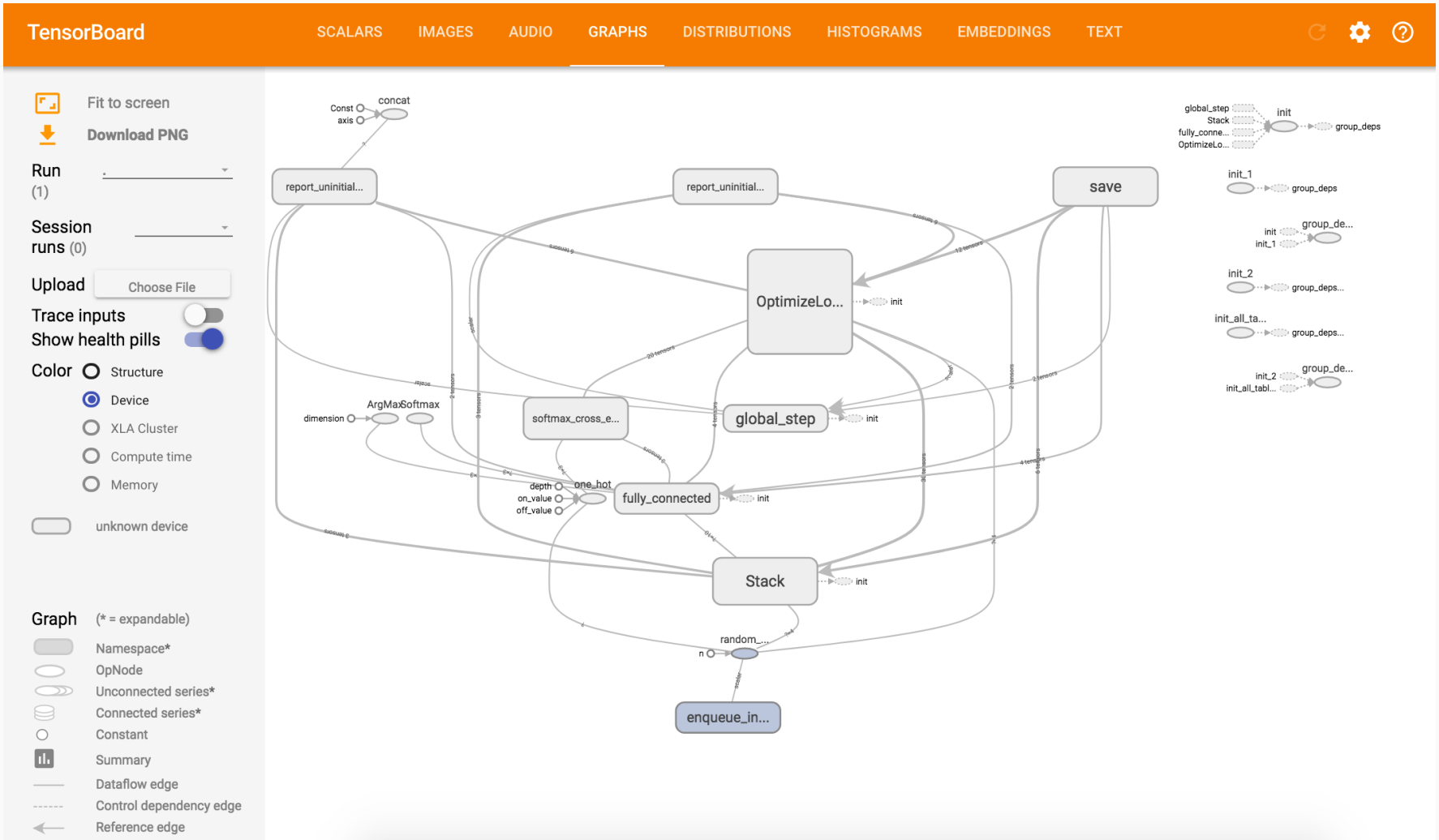
Tensorflow

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)



TensorBoard



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

Keras - Models

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)
```

Keras - Models

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)
```

Keras - Layers

Les fonctions Layers permettent d'ajouter différentes couches à un modèle Sequential, parmi lesquelles :

- Dense
  - *Fully connected layer*
- Activation
  - *Relu, sigmoïd, tanh*
- Dropout
- Flatten, Reshape
- Conv2D
  - *Réseau de convolution*
- MaxPooling2D
- LSTM
  - *Long Short-Term Memory*

...

Keras - Applications

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

- Xception
- VGG16
- VGG19
- ResNet50
- InceptionV3
- InceptionResNetV2
- MobileNet

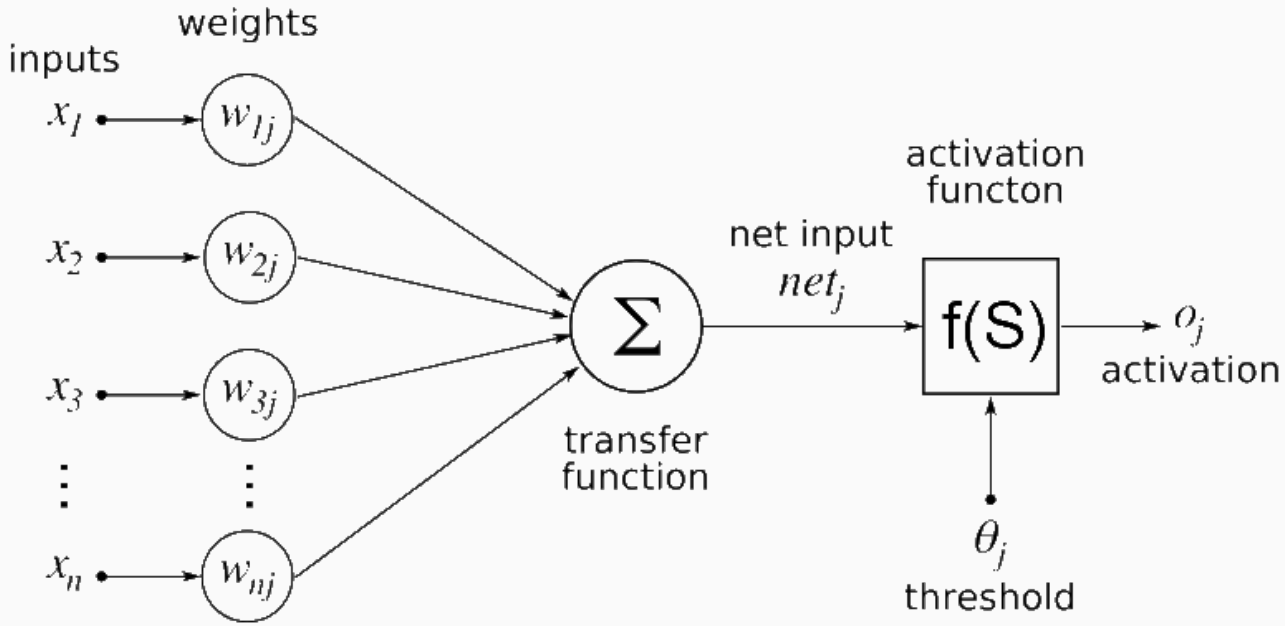
Keras - Exemple

```
# 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

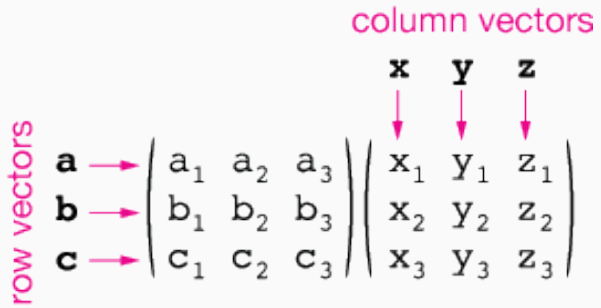
L'apprentissage d'un réseau de neurone est composé de deux opérations principales :

- ***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.



$\Longrightarrow$ Essentiellement des multiplications de matrices.

Essentiellement des multiplications de matrice :



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.

$\Longrightarrow$ Taches séquentielles.

GPU :

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

$\Longrightarrow$ 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

Remarques

- 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/m2dsupsdlclass/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.