Deep Learning Par la Pratique

François-Marie Giraud

https://www.orsys.fr/

Deep Learning par la Pratique

Présentation & coordonnées

Nom	François-Marie Giraud
Courriel	giraud.francois@gmail.com
Activité	Consultant/Formateur indépendant
Spécialité	Intelligence Artificielle
Parcours	Master Intelligence Artificielle et Décision (Paris 6)

Description

Cette formation présente les **fondamentaux** du **deep learning** à travers des travaux pratiques.

Description

Prérequis

- Bonne connaissances du Machine Learning de ses principes de fonctionnement **théorique** comme **pratiques**
- Avoir un **compte Google** afin de pouvoir faire les TPs dans Google Colaboratory

Objectifs pédagogiques

- Comprendre l'évolution des réseaux de neurones vers le deep learning
- Utiliser TensorFlow/Keras
- Comprendre les principes de conceptions, les outils de diagnostic et les effets des différents verrous et leviers à disposition
- Mettre en pratique sur des problèmes réels

Ressources

Je vous ferai parvenir les ressources informatiques utilisées à chaque début de cours. Elles sont aussi accessibles via My Orsys.

Emploi du temps

- 3 jours de 9h à 12h30 et de 14h à 17h30
- Le dernier jour, à 17h00 on fini de remplir les documents administratifs.

Tour de table : présentez-vous !

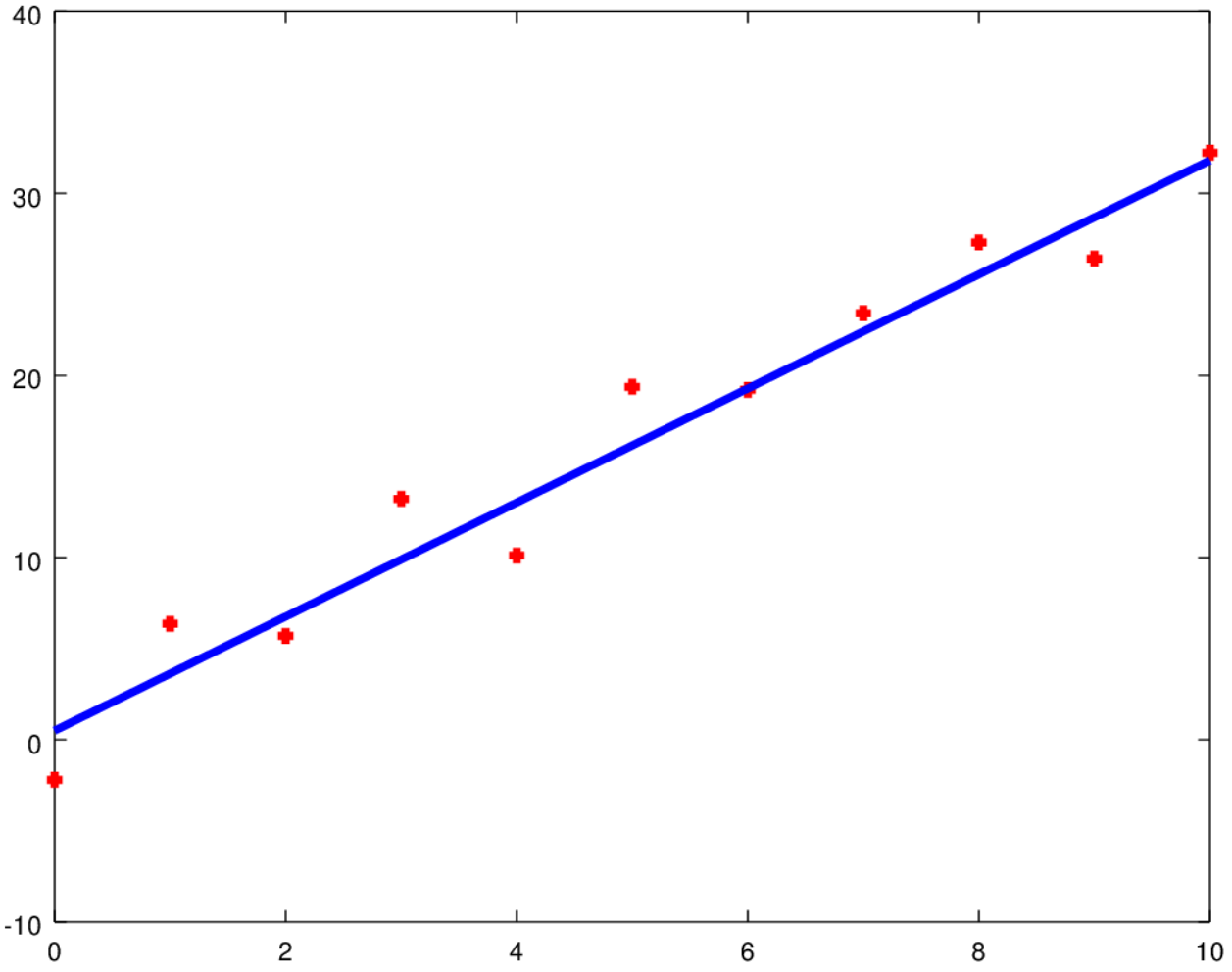
- Votre nom
- Votre métier
- Votre société client si appliquable
- Vos compétences dans les domaines liés à cette formation
- Vos objectifs et vos attentes vis-à-vis de cette formation

Réseaux de neurones

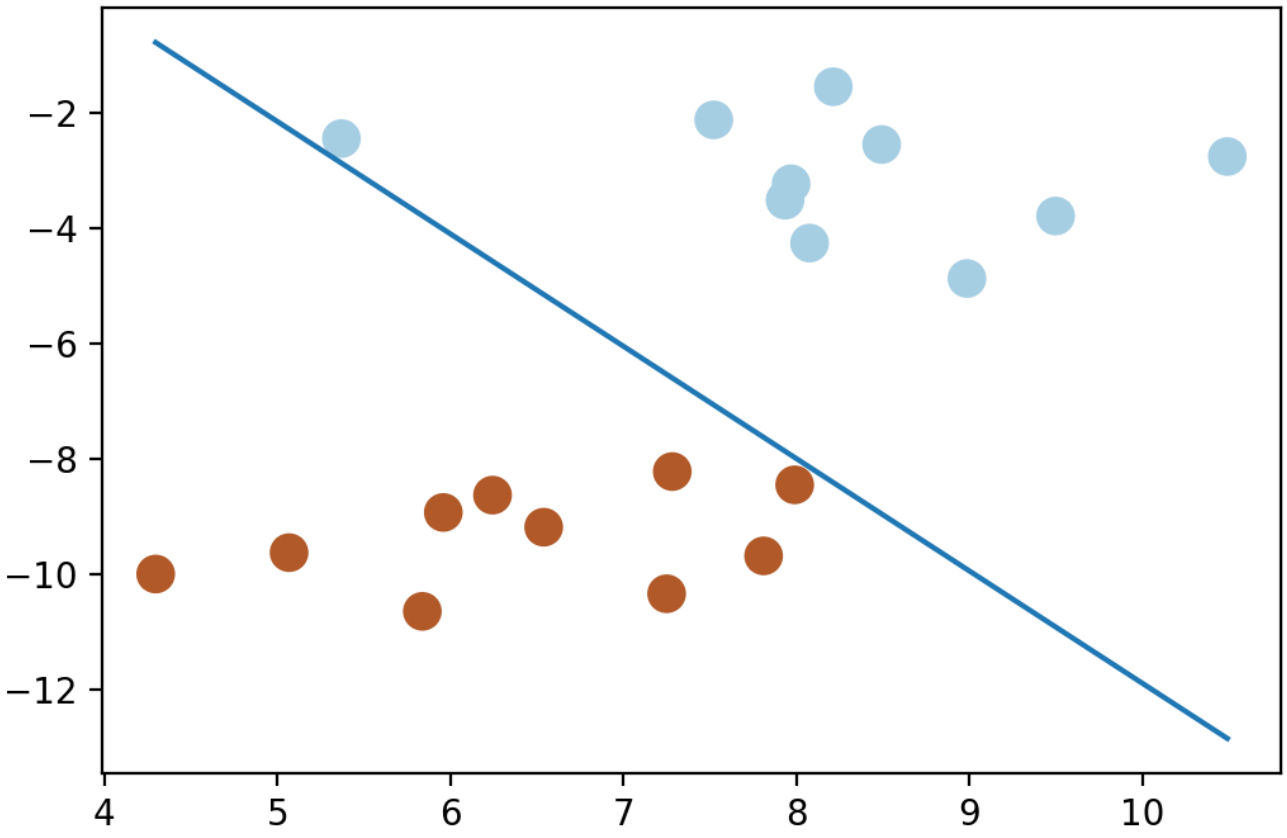
Réseaux de neurones

Introduction

Lien avec la régression linéaire

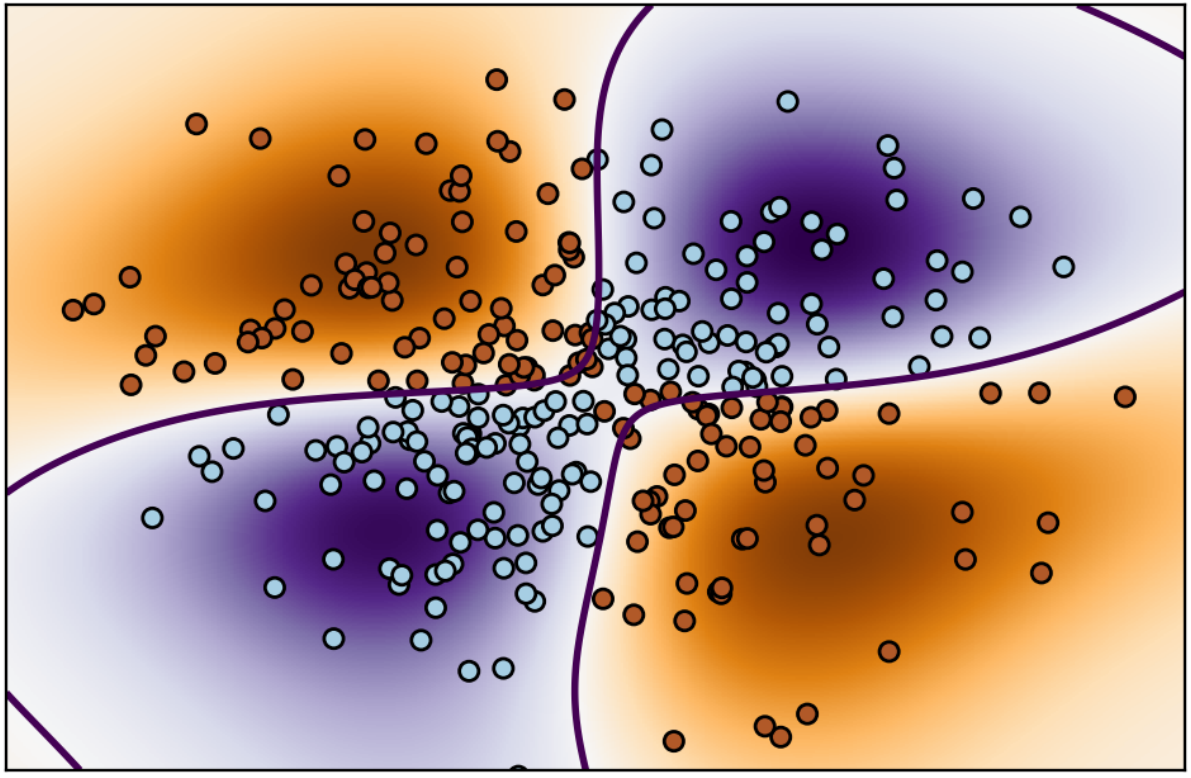


Lien avec la régression linéaire



Séparateur linéaire, F.-M. Giraud & H. Mougard, CC-BY-SA-4.0.

Lien avec la régression linéaire



Classification de données obtenues grâce à XOR, F.-M. Giraud & H. Mougard, CC-BY-SA-4.0.

Lien (ténu) avec la biologie

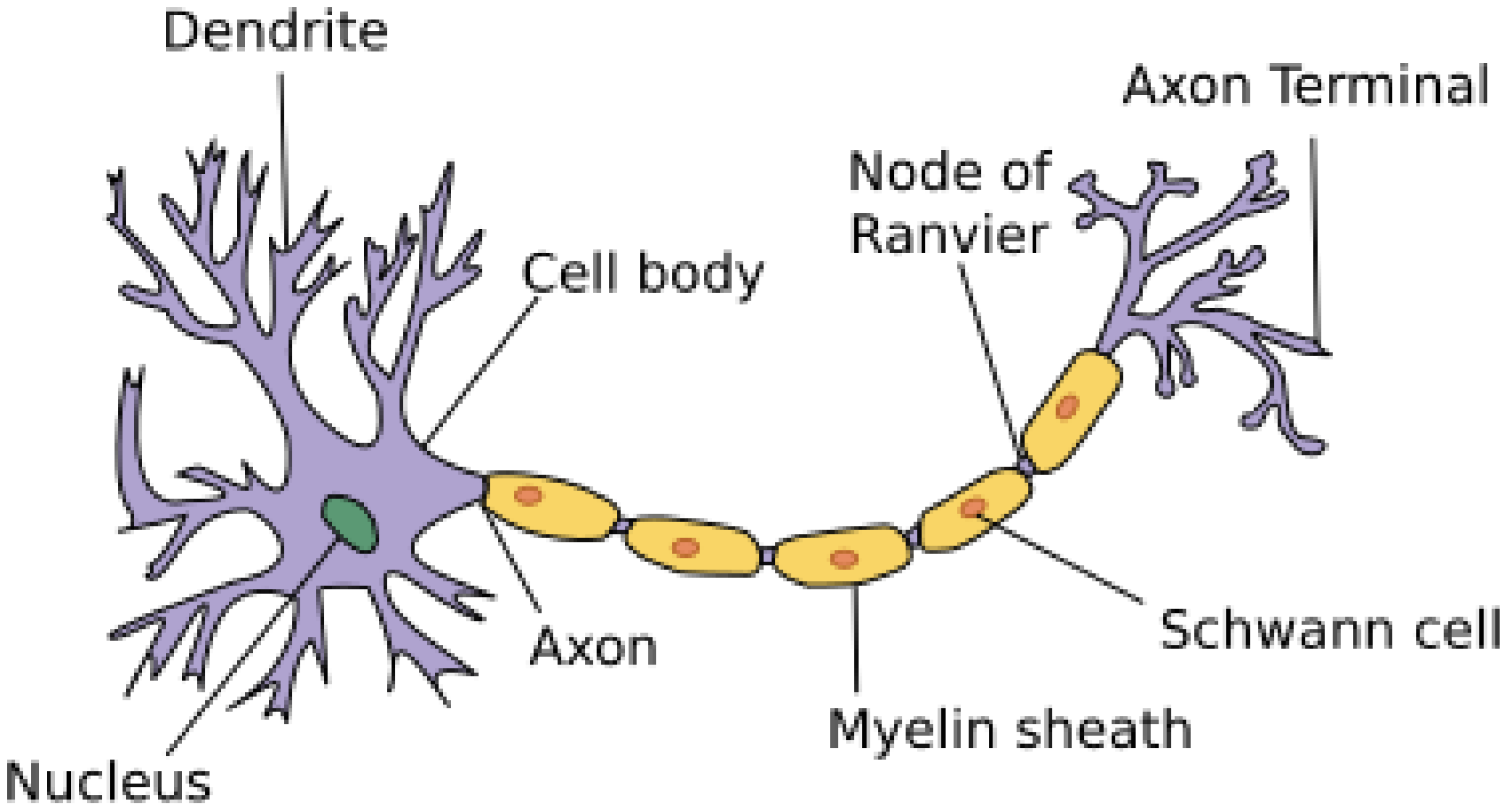
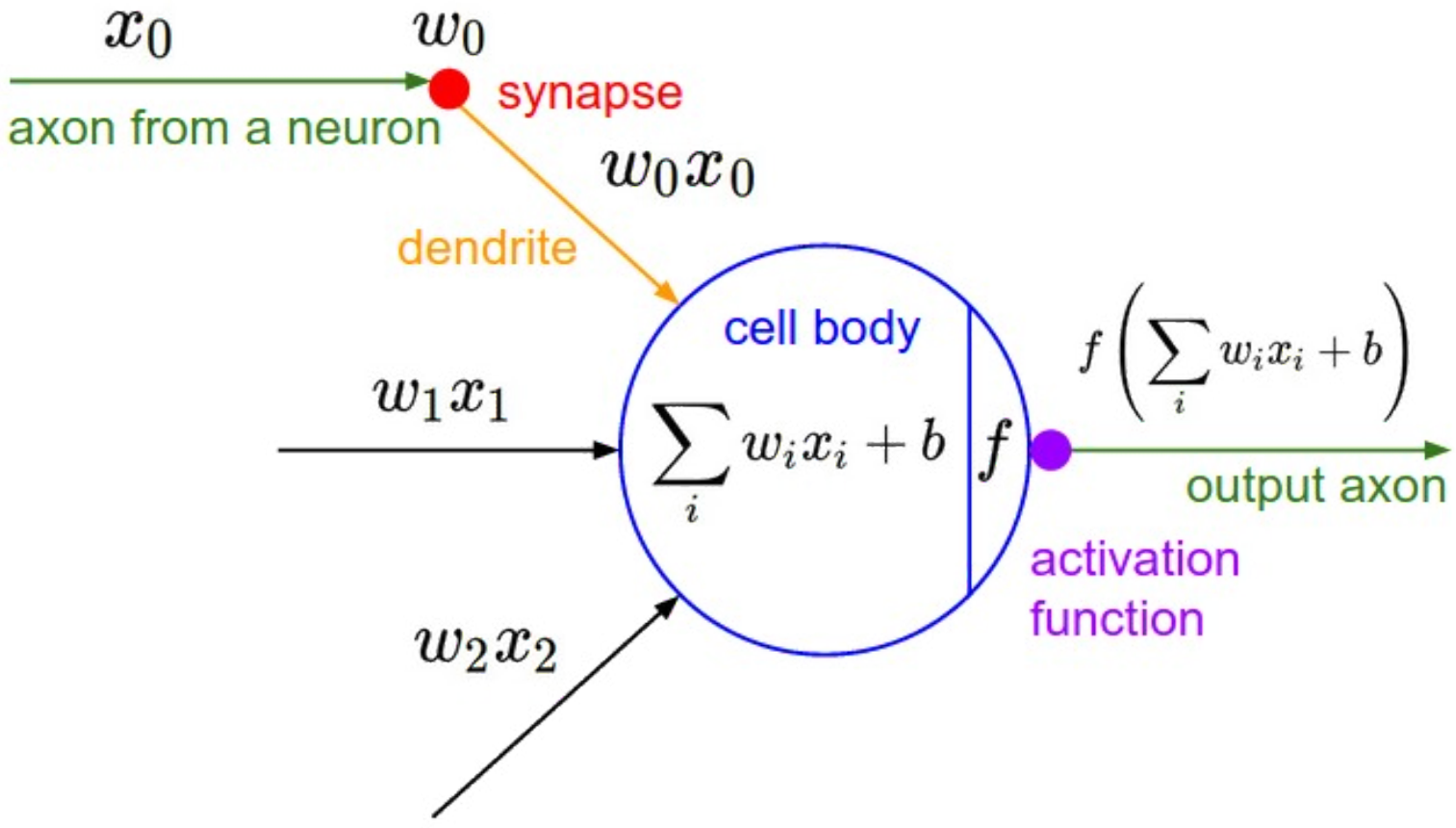


Diagram of a Neuron, membre de Wikimedia Dhp1080, CC BY-SA 3.0.

Modélisation d'un neurone



Modélisation d'un réseau de neurones

Agencement de beaucoup de neurones :

En parallèle Calculent des résultats indépendamment dans la même couche

En série Prennent en entrée les résultats des neurones de la couche précédente

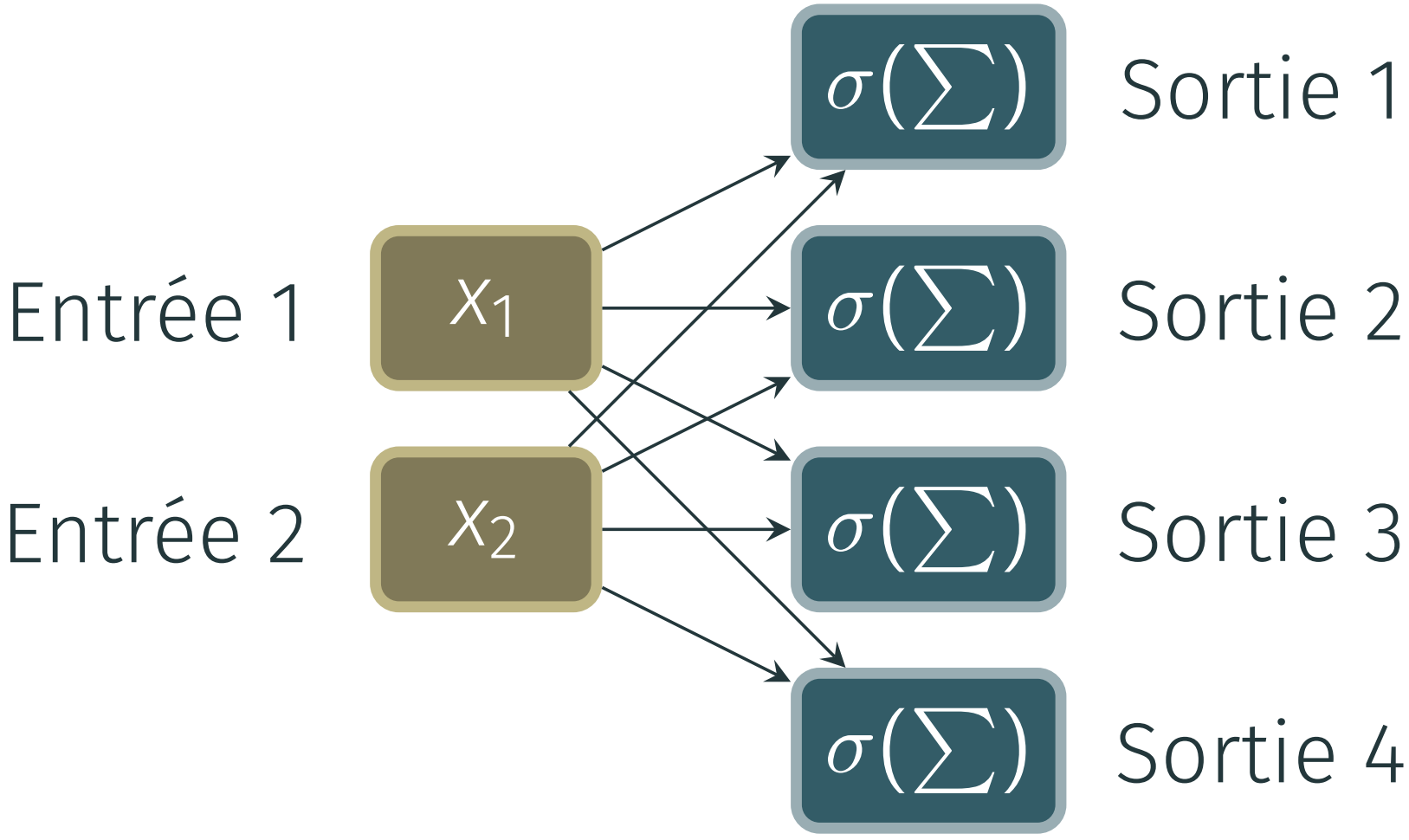
Deux types de neurones

On distingue deux types de neurones :

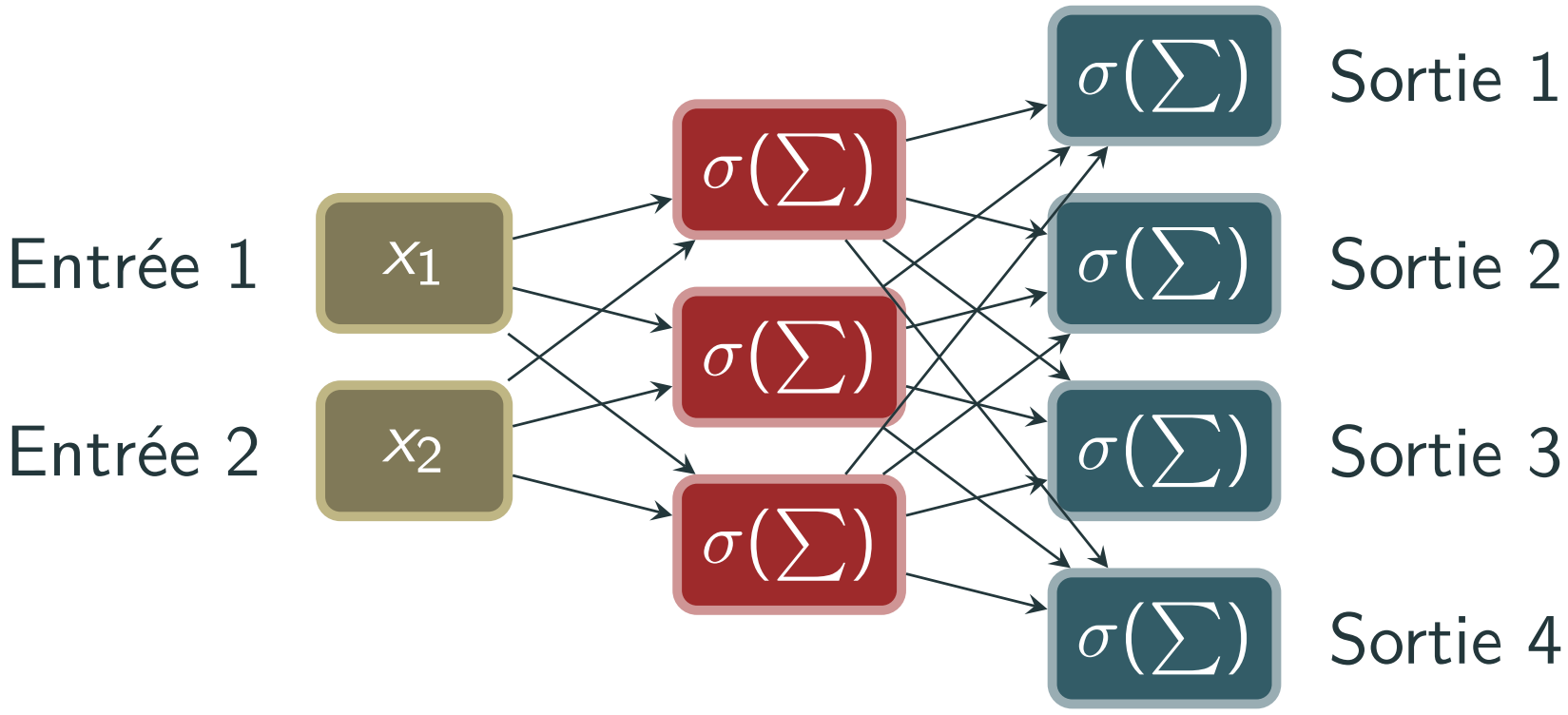
Neurones cachés Neurones des couches intermédiaires. Améliorent l'expressivité du modèle

Neurones de sortie Neurones de la couche finale. Contraints par le type de sortie attendu

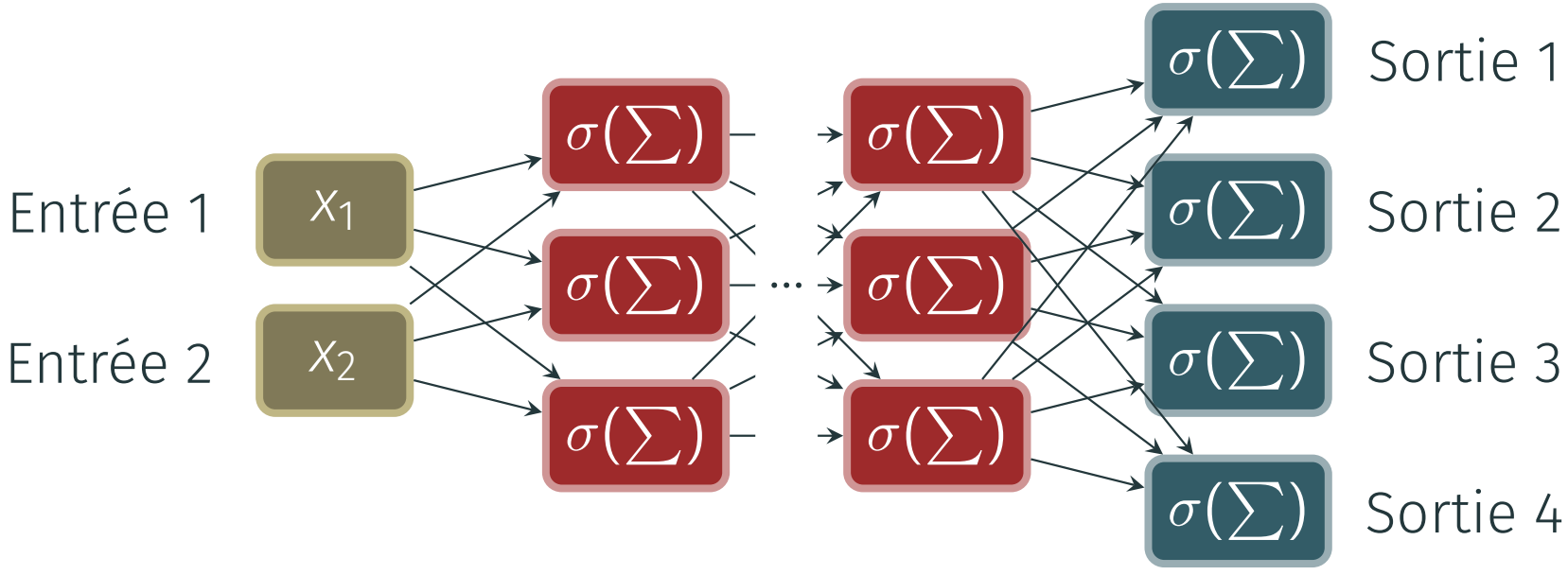
Réseau sans couche cachée



Réseau avec une couche cachée

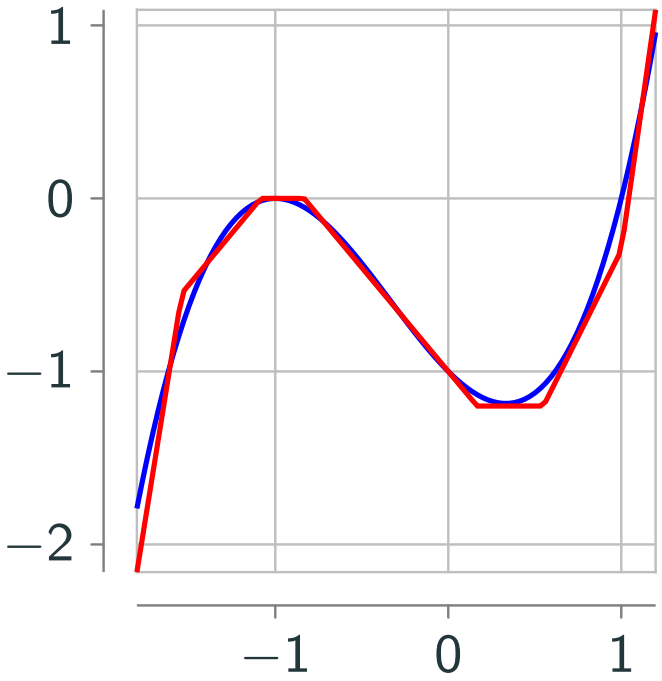


Réseau profond



Un potentiel infini

Kurt Hornik, 1991 : Théorème d'approximation universelle



$-n_1 - n_2 - n_3 + n_4 + n_5 + n_6$

$x^3 + x^2 - x - 1$

Modélisation matricielle — Échantillon

Représentable sous forme de vecteur à d colonnes correspondant à d caractéristiques :

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \dots & x_d \end{bmatrix}$$

Voire même de matrice dans le cas d'un batch (groupe d'échantillons) :

$$\mathbf{X} = \begin{bmatrix} X_{1,1} & X_{1,2} & \dots & X_{1,d} \\ X_{2,1} & X_{2,2} & \dots & X_{2,d} \end{bmatrix}$$

Modélisation matricielle — Poids

Représentables sous forme de matrice de poids et de vecteur de biais :

$$\mathbf{W} = \begin{bmatrix} W_{1,1} & W_{1,2} & \dots & W_{1,n} \\ W_{2,1} & W_{2,2} & \dots & W_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ W_{d,1} & W_{d,2} & \dots & W_{d,n} \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} b_1 & b_2 & \dots & b_n \end{bmatrix}$$

Modélisation matricielle complète

$$O = \sigma(\mathbf{X}.\mathbf{W} + \mathbf{b})$$

$$= \sigma\left(\begin{bmatrix} X_{1,1} & X_{1,2} & \dots & X_{1,d} \\ X_{2,1} & X_{2,2} & \dots & X_{2,d} \end{bmatrix} \begin{bmatrix} W_{1,1} & W_{1,2} & \dots & W_{1,n} \\ W_{2,1} & W_{2,2} & \dots & W_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ W_{d,1} & W_{d,2} & \dots & W_{d,n} \end{bmatrix} + \begin{bmatrix} b_1 & b_2 & \dots & b_n \end{bmatrix}\right)$$

$$= \begin{bmatrix} O_{1,1} & O_{1,2} & \dots & O_{1,n} \\ O_{2,1} & O_{2,2} & \dots & O_{2,n} \end{bmatrix}$$

où :

- **X** Données en entrée de dimension d
- **W & b** Paramètres à trouver des n neurones de notre modèle
- $\boldsymbol{\sigma}$ Fonction d'activation
- **O** Sortie du réseau

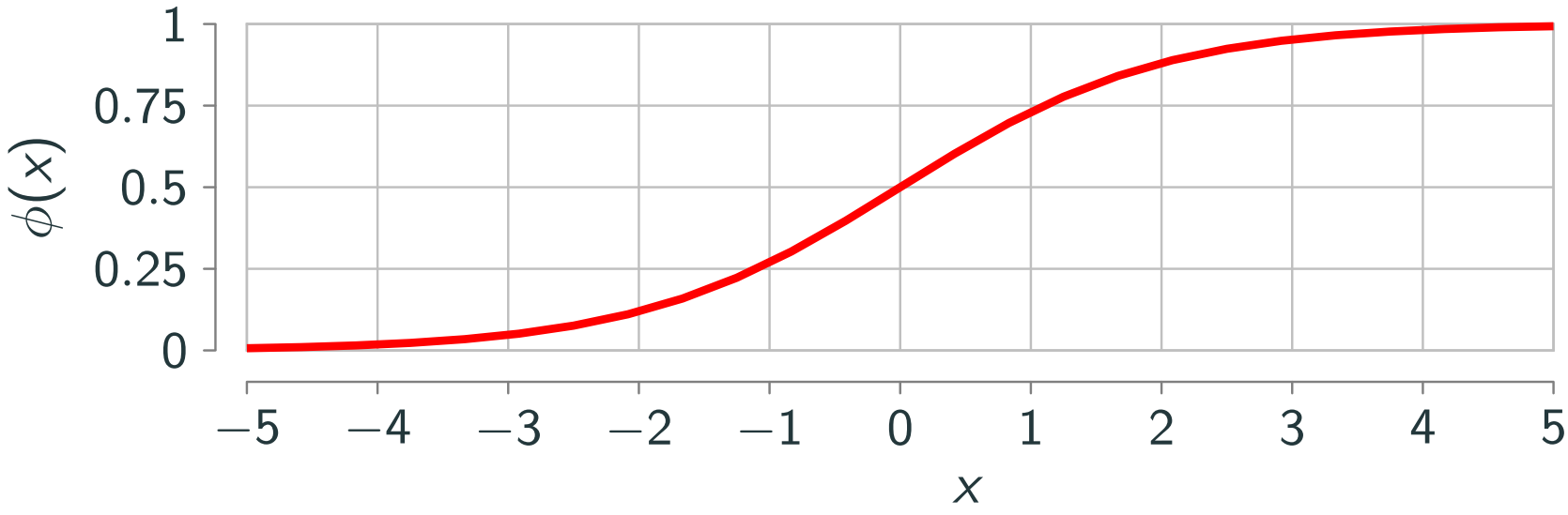
Fonctions d'activation — Critères de choix

- Propriétés mathématiques (conservation du gradient)
- Propriétés d'apprentissage (éviter la création de poids morts)
- Rapidité de calcul
- Intervalle de sortie pour la dernière couche

Fonctions d'activation — Les plus classiques

- Sigmoïde
- Tanh
- Softmax
- ReLU
- ...

Fonctions d'activation — Sigmoïde



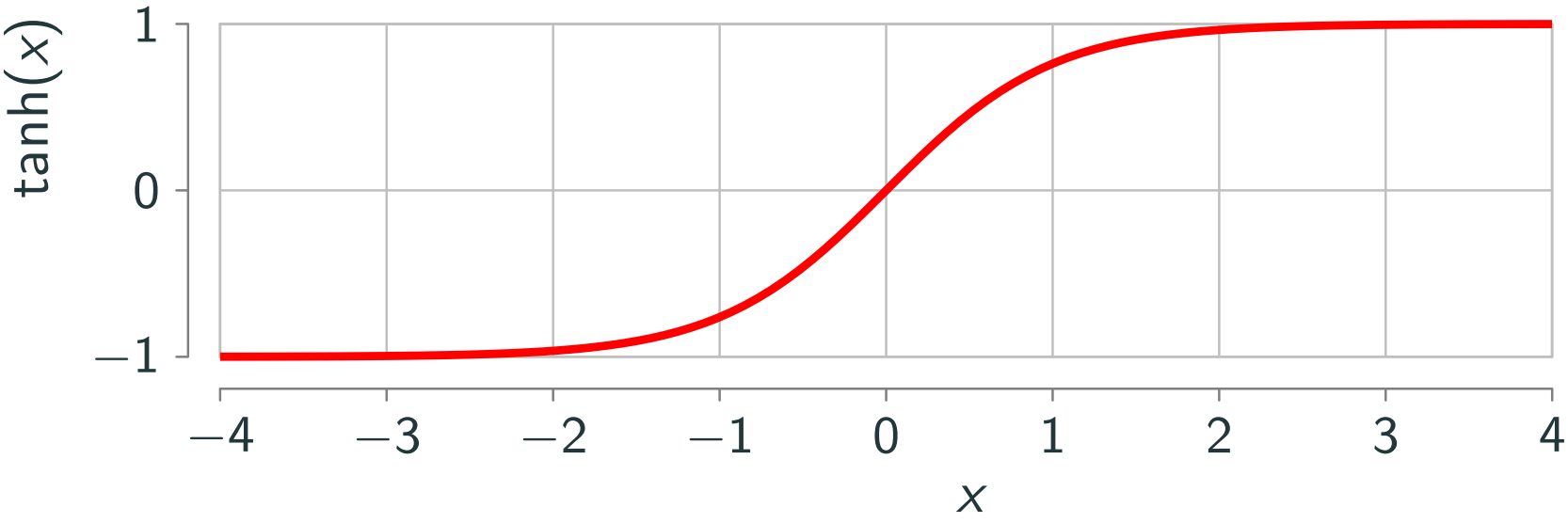
Définition

$$\phi(x) = \frac{1}{1 + e^{-x}}$$

Dérivée

$$\phi'(x) = \phi(x)(1 - \phi(x))$$

Fonctions d'activation — Tangente hyperbolique



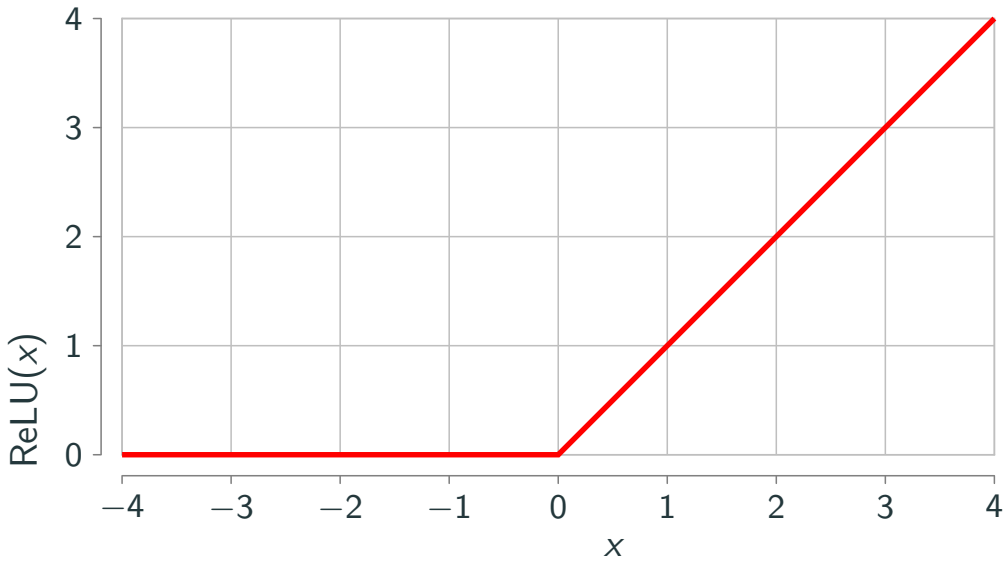
Définition

$$\tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

Dérivée

$$\tanh'(x) = 1 - \tanh^2(x)$$

Fonctions d'activation — ReLU



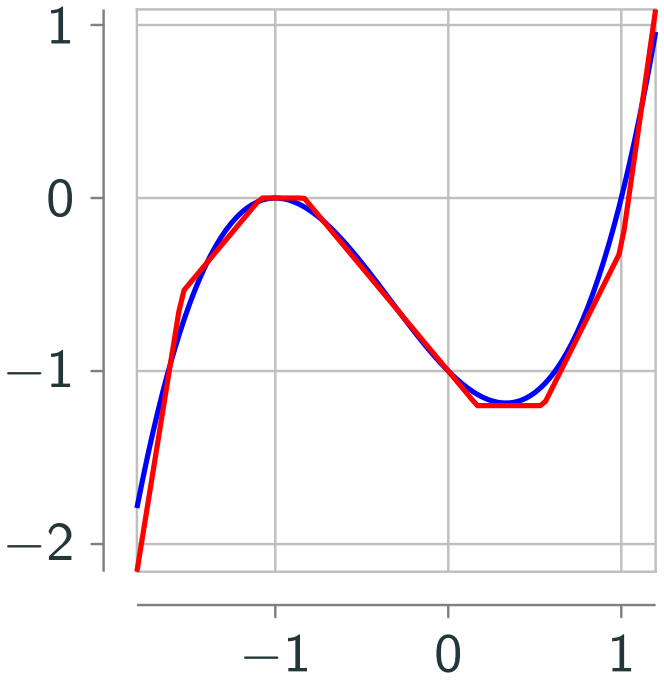
Définition

$$\mathrm{ReLU}(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$$

Dérivée

$$\mathrm{ReLU}'(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases}$$

Fonctions d'activation — Approximation d'une fonction



$-n_1 - n_2 - n_3 + n_4 + n_5 + n_6$

$x^3 + x^2 - x - 1$

$n_1 = \text{ReLU}(-5x - 7.7)$

$n_2 = \text{ReLU}(-1.2x - 1.3)$

$n_3 = \text{ReLU}(1.2x - 1)$

$n_4 = \text{ReLU}(1.2x - 0.2)$

$n_5 = \text{ReLU}(2x - 1.1)$

$n_6 = \text{ReLU}(5x - 5)$

Fonctions d'activation — Softmax

Définition

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{k=1}^{n} \exp(x_k)}$$

Propriété

$$\sum_{i=1}^{n} \text{softmax}(x_i) = 1$$

Gradient

$$\frac{\partial \, \text{softmax}(x_i)}{\partial x_j} = \begin{cases} \text{softmax}(x_i)(1 - \text{softmax}(x_j)) & i = j \\ -\text{softmax}(x_i)\,\text{softmax}(x_j) & i \neq j \end{cases}$$

Réseaux de neurones

Descente de gradient

Principe

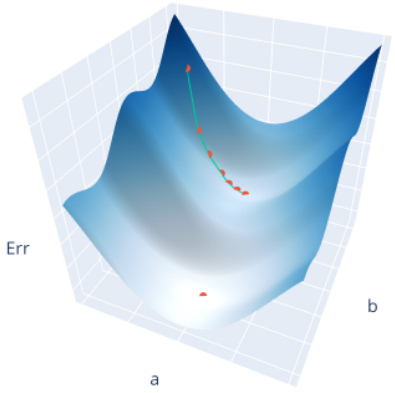
Calcul du gradient de l'erreur par rapport aux paramètres:

$$\frac{\partial Err}{\partial w_i}$$

Mise à jour :

$$w_i' = w_i - \gamma * grad$$

où : $0 < \gamma < 1$ (learning rate)



Surface de l'erreur en fonction des paramètres a et b, F.-M. Giraud & H. Mougard, CC-BY-SA-4.0.

Algorithme

1. Initialisation aléatoire du modèle
2. Tant qu'aucun critère d'arrêt n'est satisfait :
   - Selection aléatoire d'un **batch** de données
   - **Forward** : Passe avant du **batch** dans le modèle
   - Calcul de l'erreur par rapport aux sorties attendues
   - **Backward** : Rétropropagation du gradient de l'erreur en fonction des paramètres dans le modèle (mise à jour du modèle)
   - Calcul des critères d'arrêt

Exemple de dérivation — Régression linéaire

$$E_\Omega = \frac{1}{2n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

$$E_\Omega = \frac{1}{2n}\sum_{i=1}^{n}(\hat{y}_i - (ax_i + b))^2$$

...

$$\frac{\partial E_\Omega}{\partial a} = \frac{1}{n}\sum_{i=1}^{n}(ax_i + b - \hat{y}_i)x_i$$

$$\frac{\partial E_\Omega}{\partial b} = \frac{1}{n}\sum_{i=1}^{n}(ax_i + b - \hat{y}_i)$$

$$\boxed{y = ax + b}$$

$$\boxed{U^{2\,\prime} = 2U' \times U}$$

Mise à jour

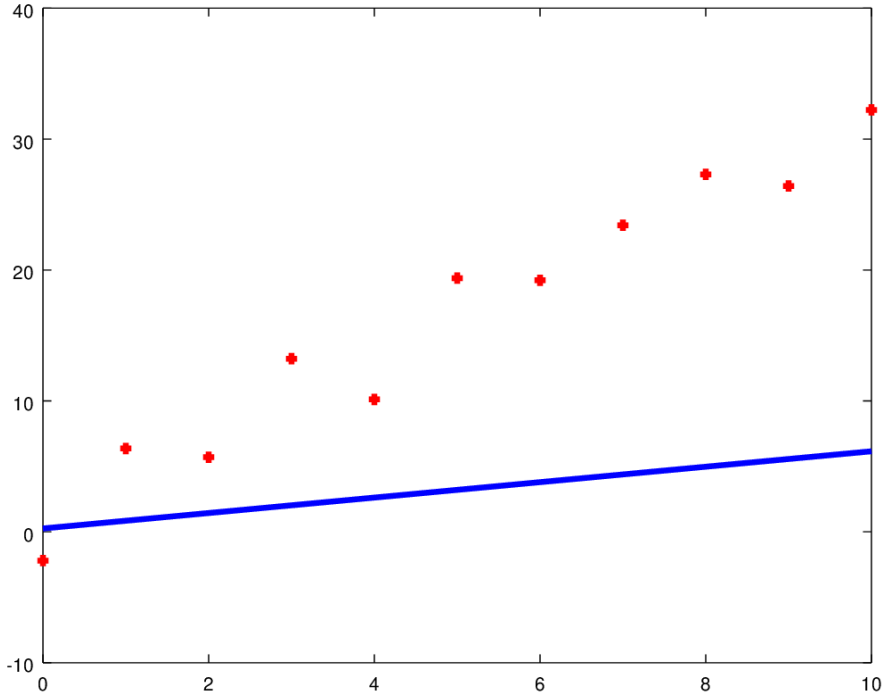
$$a \leftarrow a - \gamma\frac{\partial E_\Omega}{\partial a}$$
$$b \leftarrow b - \gamma\frac{\partial E_\Omega}{\partial b}$$

où $0 < \gamma < 1$ (pas d'apprentissage)

Exemple

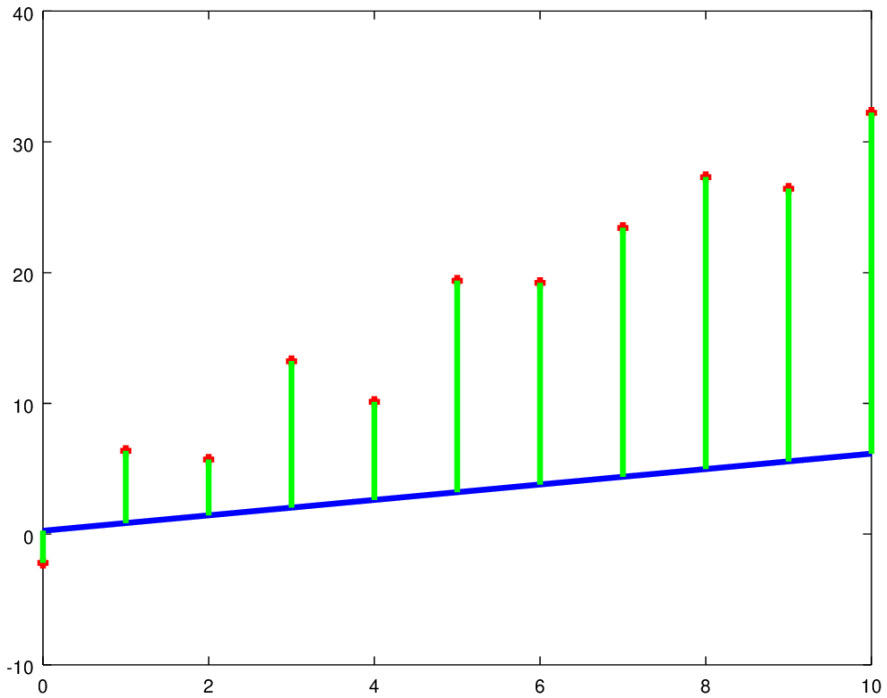
Initialisation au hasard ($\gamma = 0.01$)

- a = 0.58 ($\hat{a} = 3.0$)
- b = 0.25 ($\hat{b} = 0.5$)



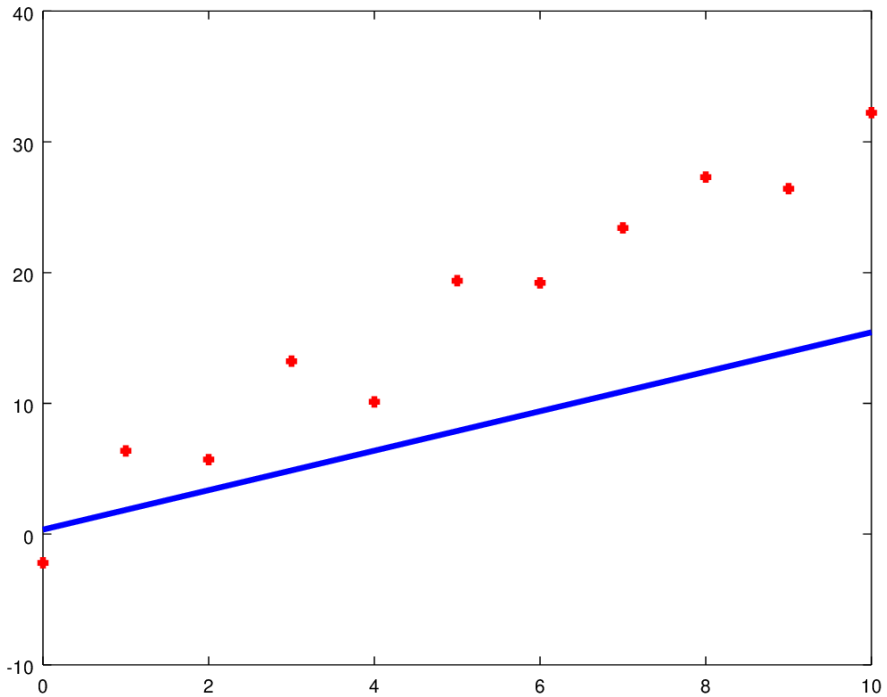
Exemple

- a = 0.58 ($\hat{a} = 3.0$)
- b = 0.25 ($\hat{b} = 0.5$)



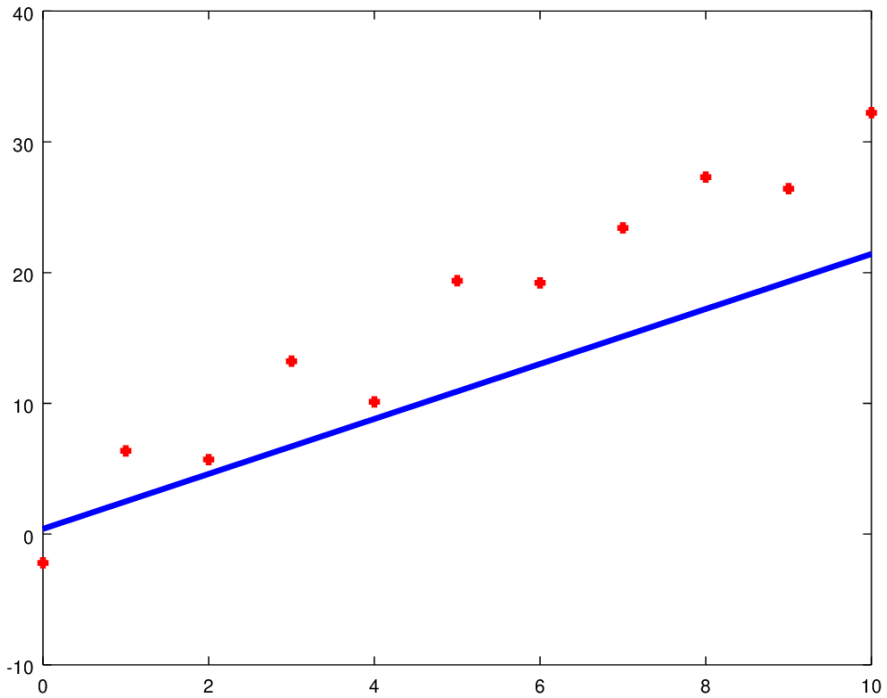
Exemple

- a = 1.50 ($\hat{a} = 3.0$)
- b = 0.35 ($\hat{b} = 0.5$)



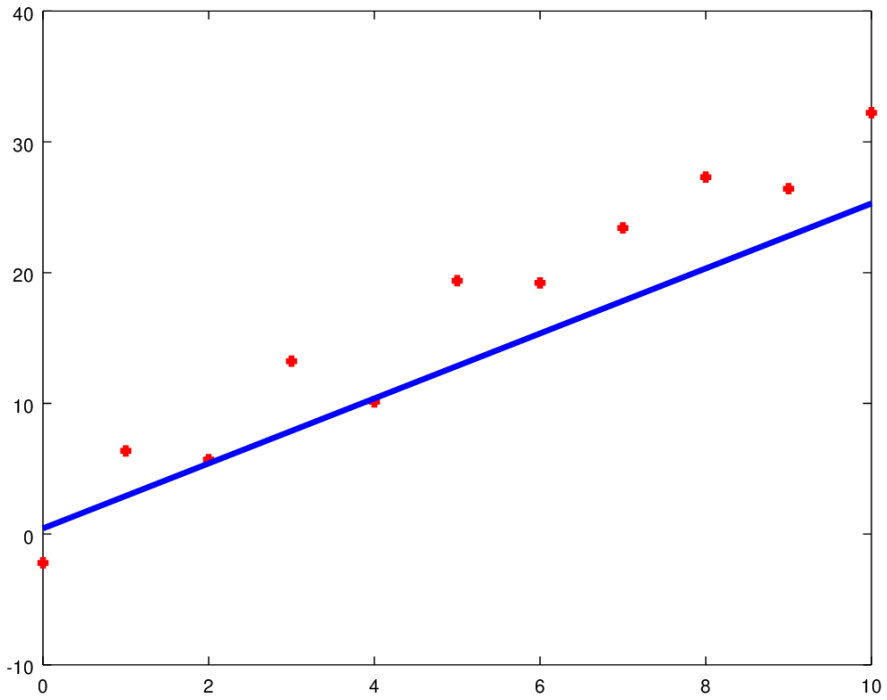
Exemple

- a = 2.10 ($\hat{a} = 3.0$)
- b = 0.40 ($\hat{b} = 0.5$)



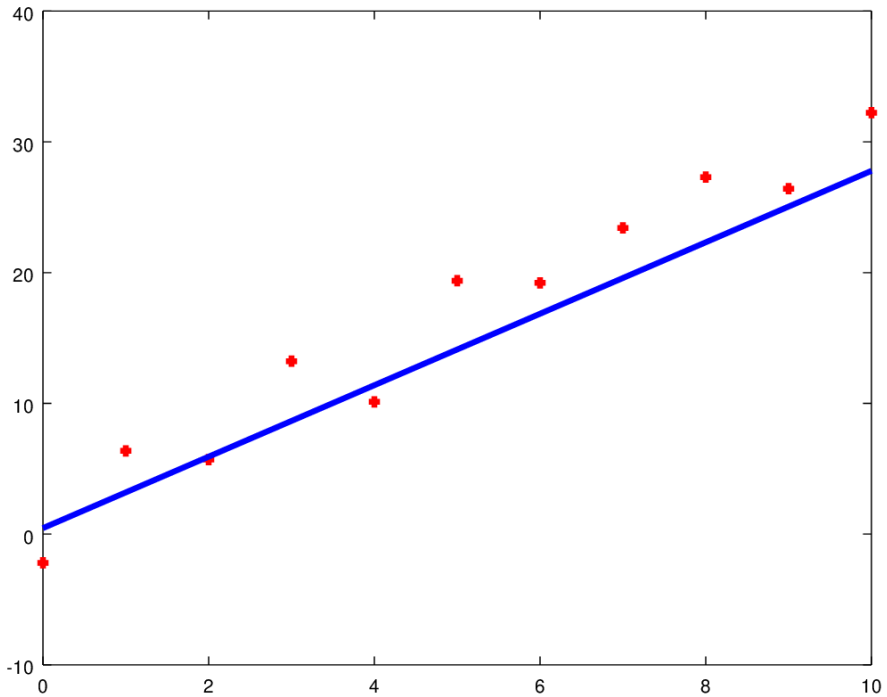
Exemple

- a = 2.48 ($\hat{a} = 3.0$)
- b = 0.43 ($\hat{b} = 0.5$)



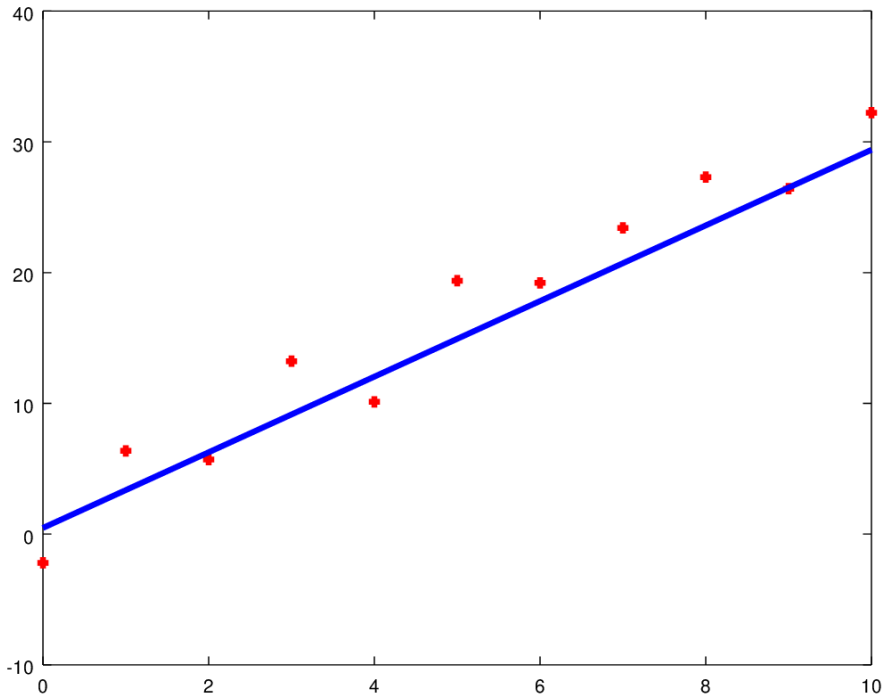
Exemple

- a = 2.73 ($\hat{a} = 3.0$)
- b = 0.46 ($\hat{b} = 0.5$)



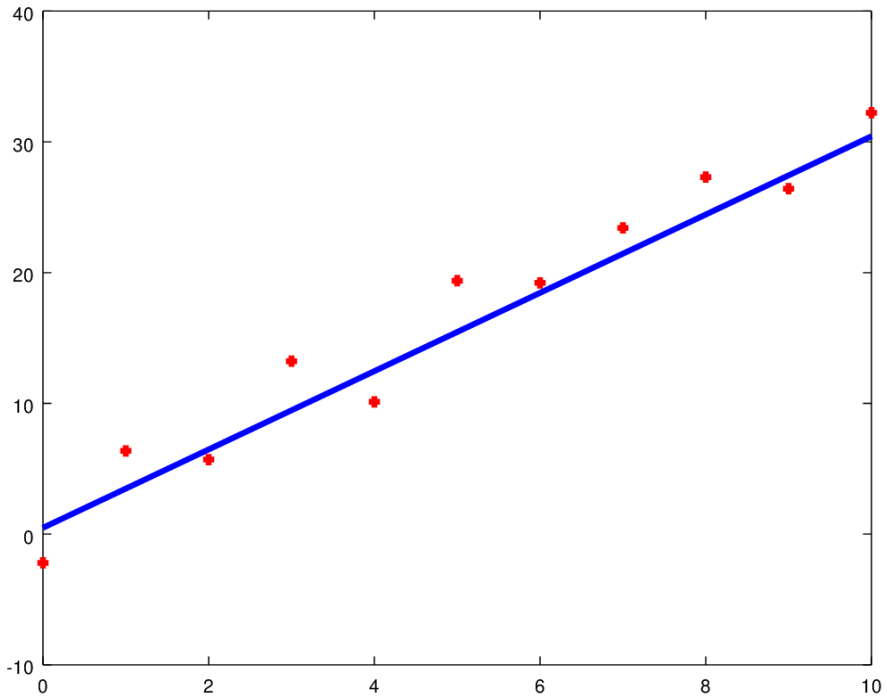
Exemple

- a = 2.89 ($\hat{a} = 3.0$)
- b = 0.47 ($\hat{b} = 0.5$)



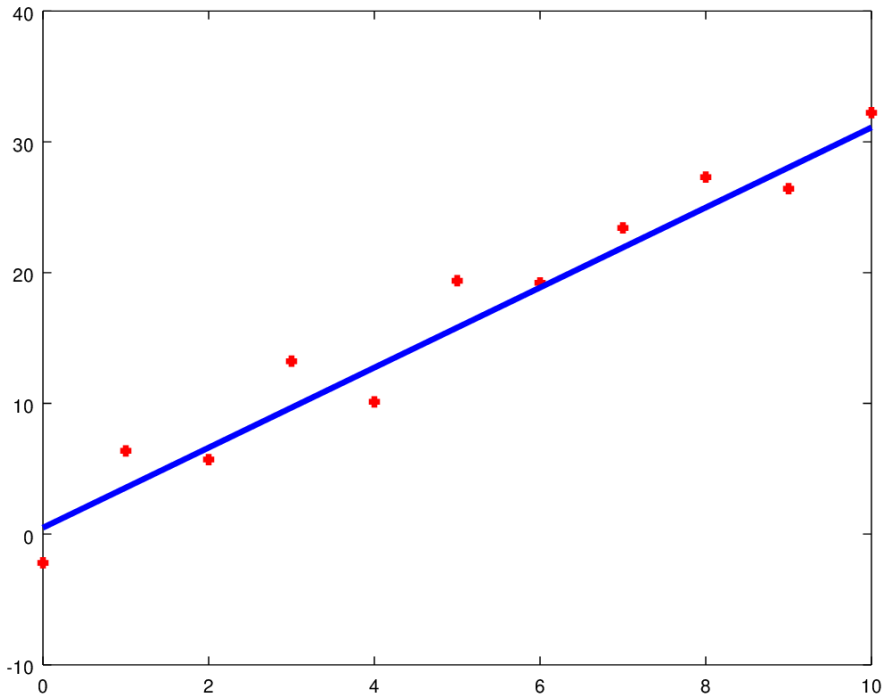
Exemple

- a = 2.99 ($\hat{a} = 3.0$)
- b = 0.48 ($\hat{b} = 0.5$)



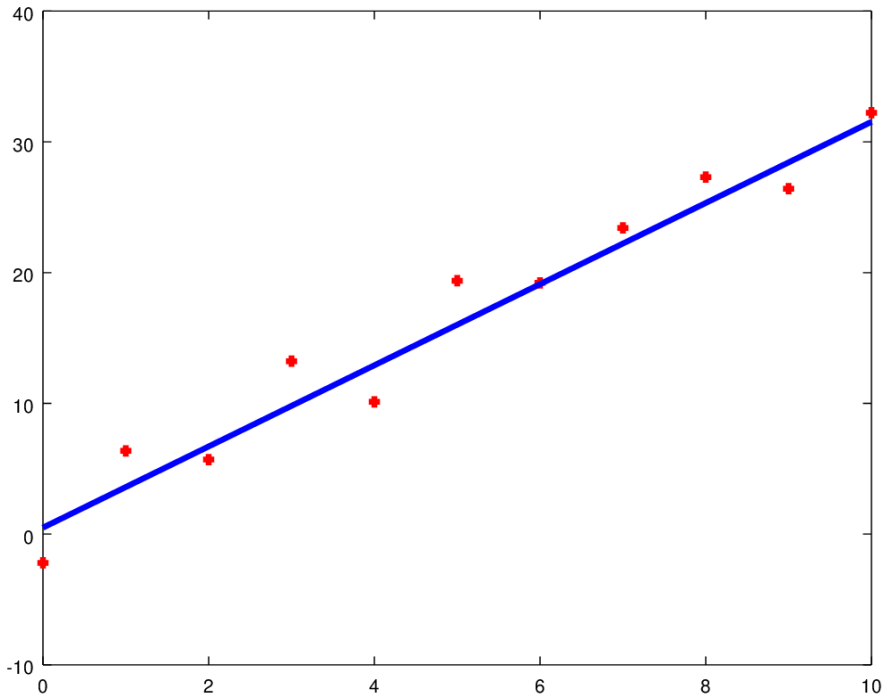
Exemple

- a = 3.06 ($\hat{a} = 3.0$)
- b = 0.49 ($\hat{b} = 0.5$)



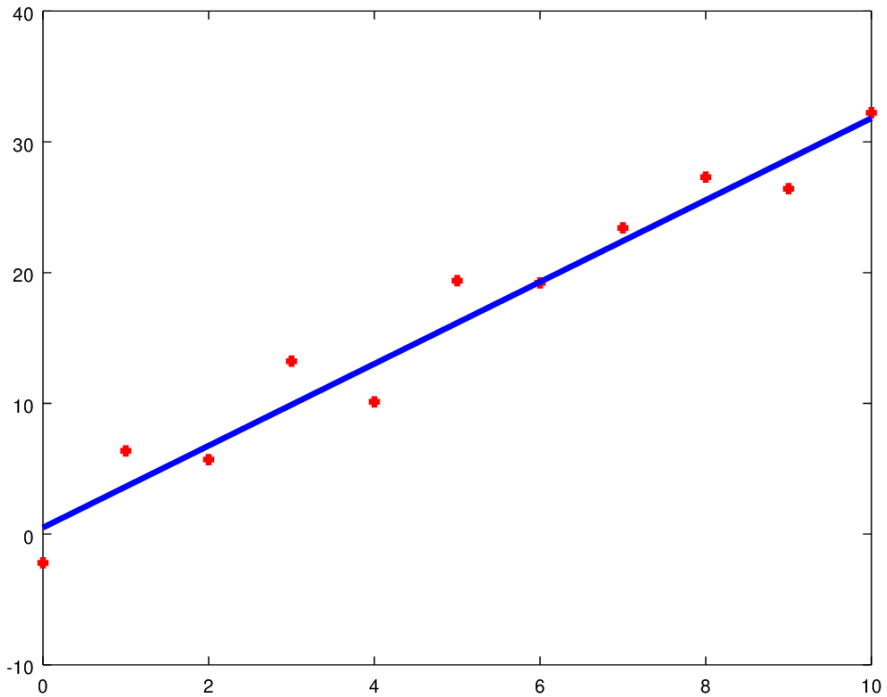
Exemple

- a = 3.10 ($\hat{a} = 3.0$)
- b = 0.49 ($\hat{b} = 0.5$)



Exemple

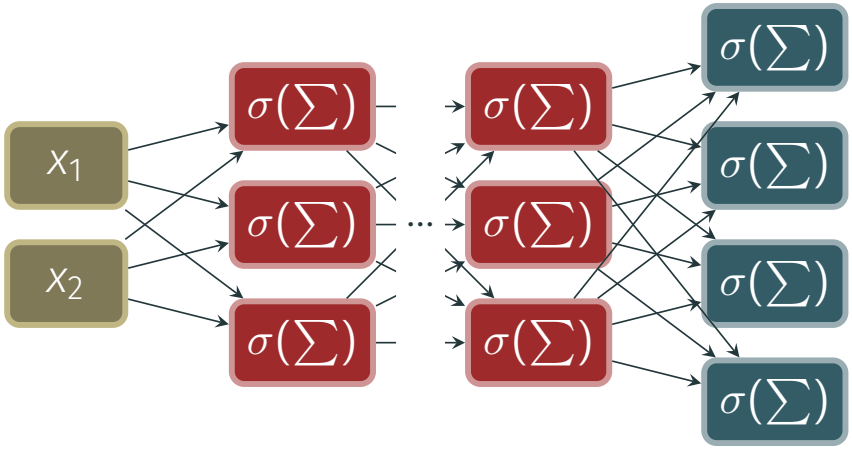
- a = 3.13 ($\hat{a} = 3.0$)
- b = 0.50 ($\hat{b} = 0.5$)



Réseaux de neurones

Optimisation des hyper-paramètres

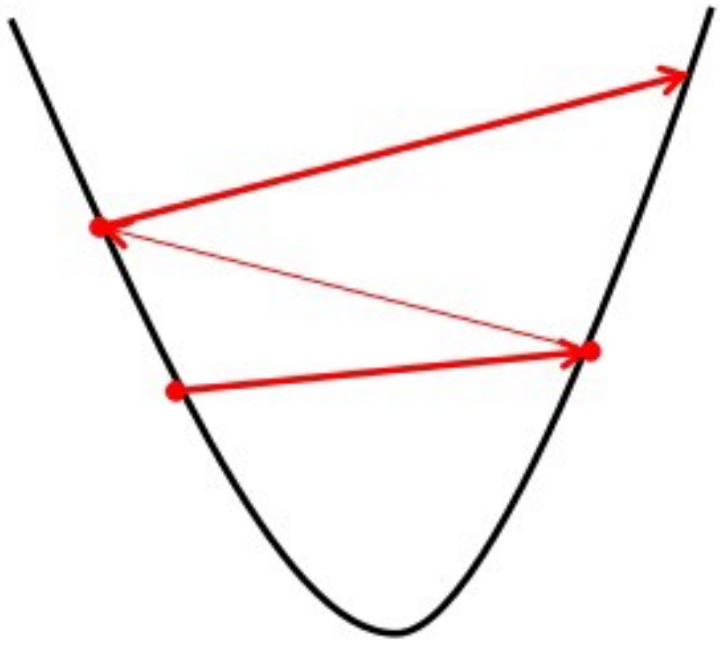
Qu'est-ce qu'un hyper-paramètre ?



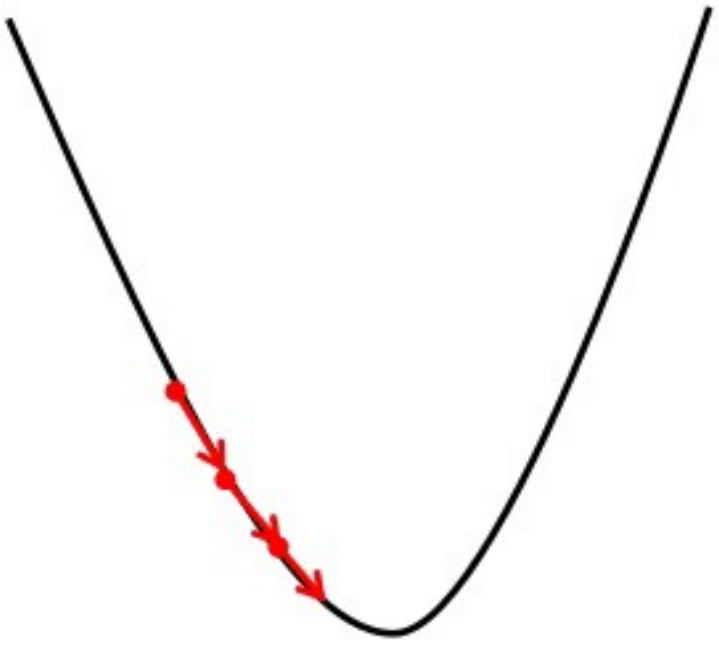
- Learning rate
- Taille de batch
- Nombre de couches
- Taille des couches

Learning rate

trop grand



trop petit



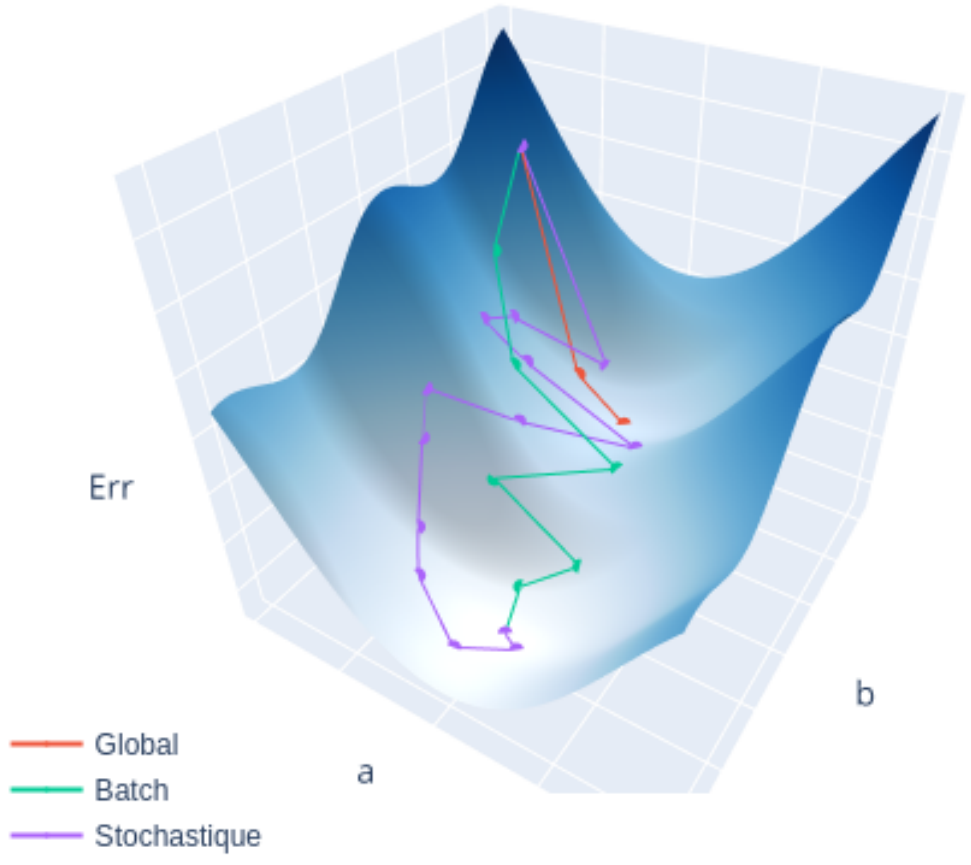
Learning rate

Utilisation d'une échelle logarithmique (dans un premier temps) :

```
lr_grid = [ 0.1 0.001 0.0001 0.00001]
for lr in lr_grid:
    train = tf.train.GradientDescentOptimizer(lr).minimize(loss)
    ...
```

Taille du batch

Usuellement, des puissances de 2, pour optimiser les ressources GPU



Effet de la taille du batch, F.-M. Giraud & H. Mougard, CC-BY-SA-4.0.

Taille des couches

Usuellement, des puissances de 2, pour optimiser les ressources GPU.
Même intuition que pour le nombre de couches.

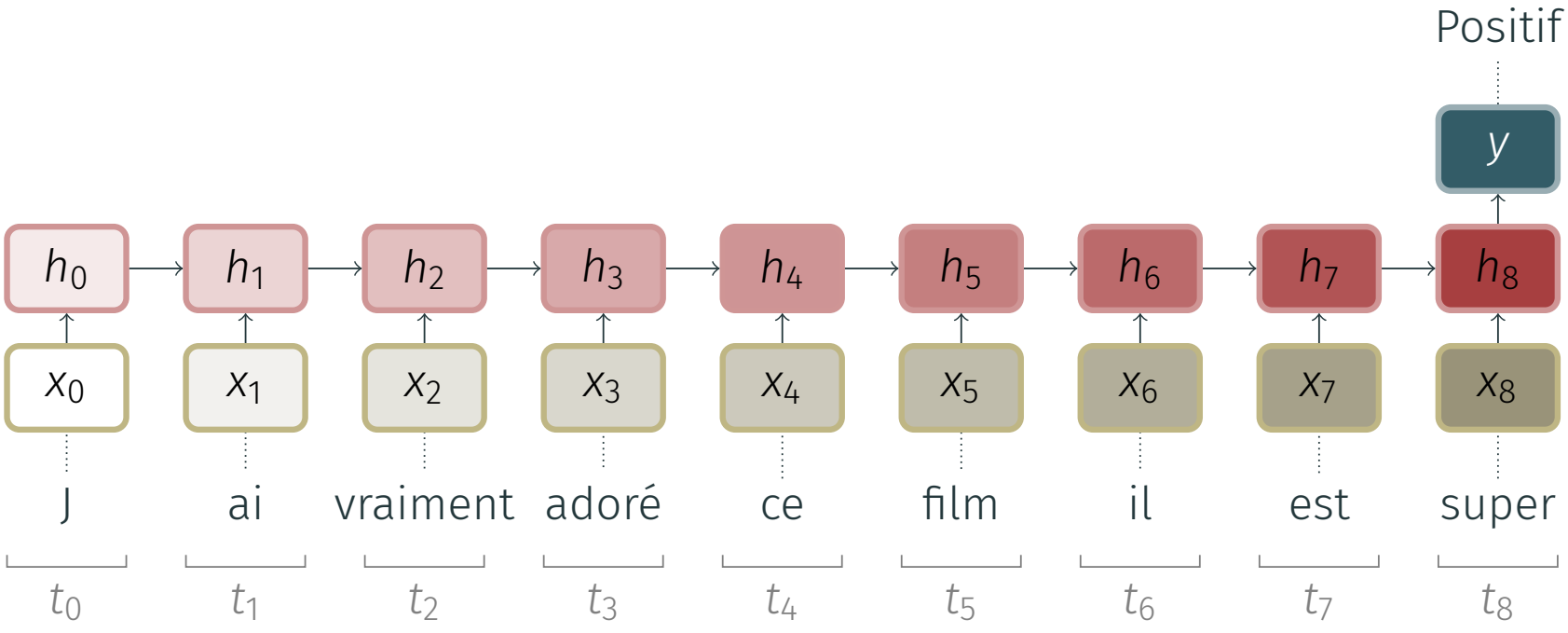
Nombre de couches

```
while (perf > perf_precedente):
    perf_precedente = perf
    ajouter_une_couche(model)
    apprendre(model,data_train)
    perf = calcul_perf(model, data_validation)
```

Réseaux de neurones

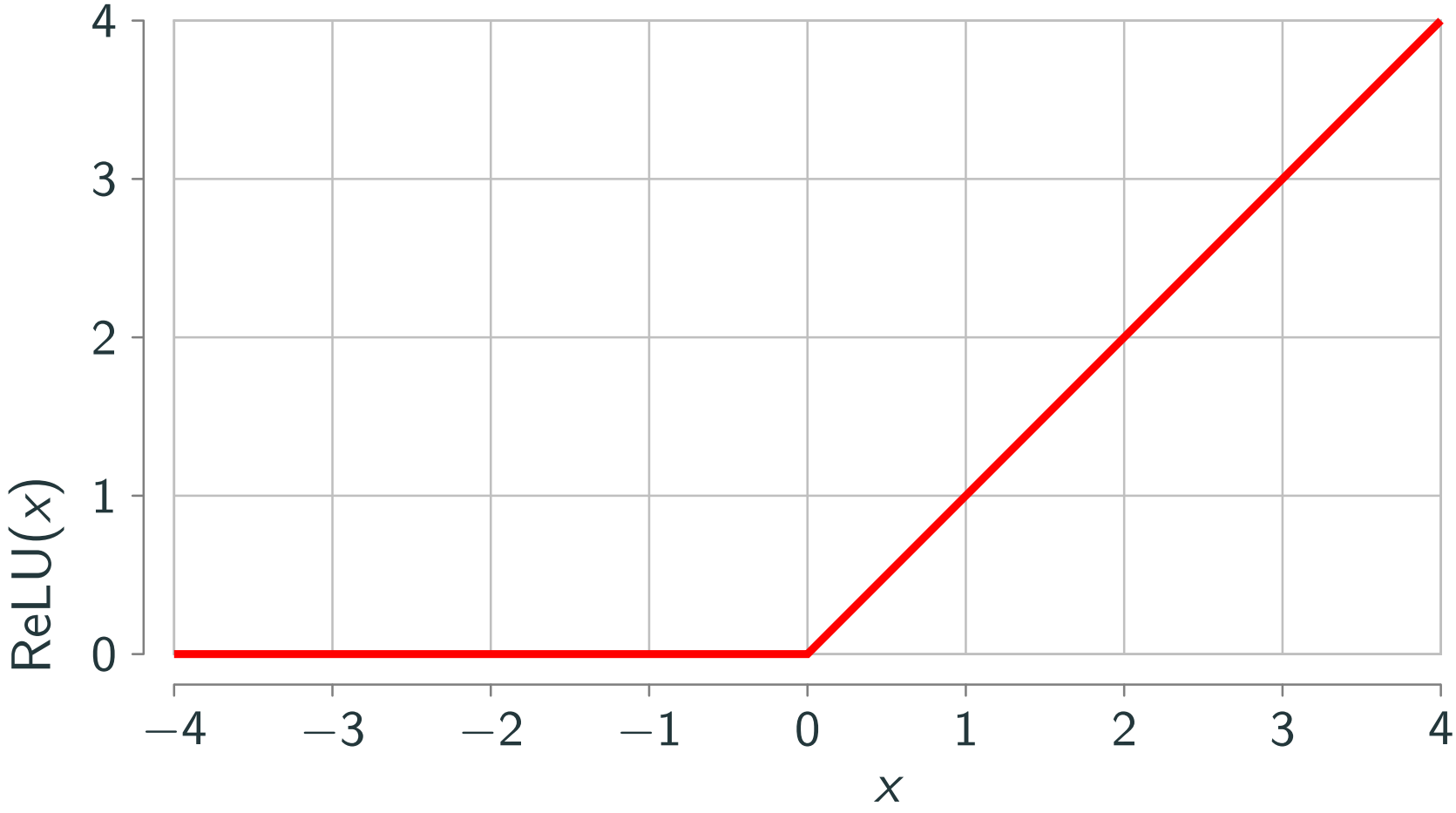
Optimisation de réseaux profonds

Disparition du gradient (ou son explosion)



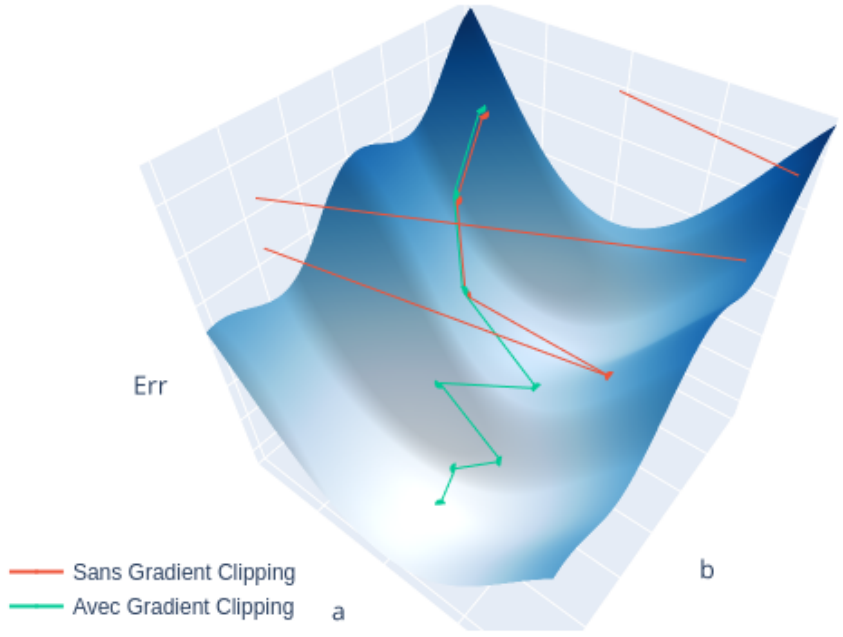
Disparition des gradients

Vanishing Gradient ⇒ utilisation de ReLu plutôt que les sigmoïdes



Explosion des gradients

Exploding Gradient ⇒ Gradient clipping



Effet de l'écrêtage de gradient , F.-M. Giraud & H. Mougard, CC-BY-SA-4.0.

Dans le cadre des réseaux réccurents ⇒ initialisation avec des matrices orthogonales

Algorithme d'optimisation

Optimisateur plus rapide que SGD

- Adaptative Gradient Algorithm (AdaGrad)
- Root Mean Square Propagation (RMSProp)
- Adaptative Moment Estimation (Adam) ⇐
- ... AdaBound (2019) ?

Adaptative Gradient Algorithm (AdaGrad)

Calcul d'un learning rate adapté à chaque itération, pour chaque paramètre du modèle.

Root Mean Square Propagation (RMSProp)

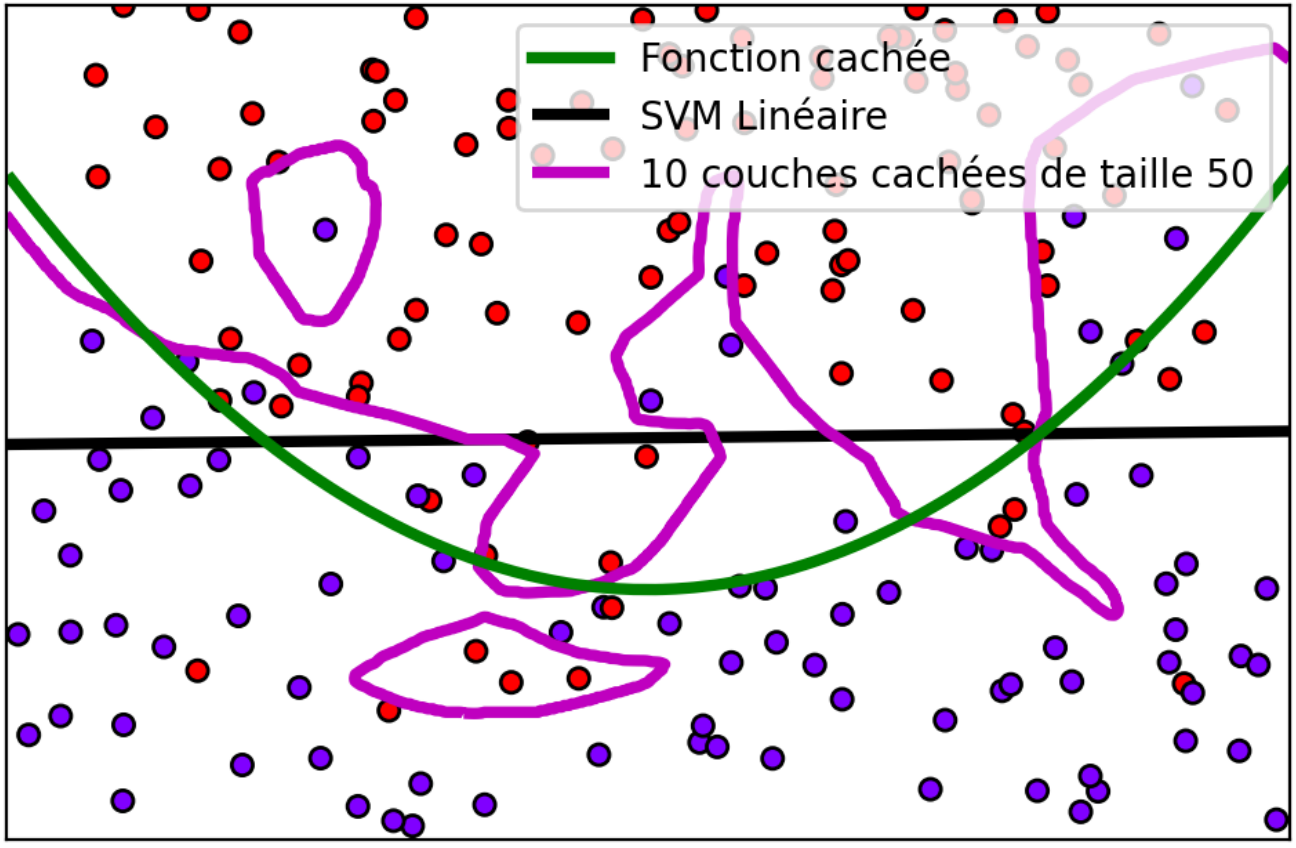
Utilisation d'inertie dans le gradient appliqué :

$$Grad_t = Grad_t + Grad_{t-1}$$

Réseaux de neurones

Régularisation

Sur-apprentissage (et sous-apprentissage)



Mise en évidence du surapprentissage, F.-M. Giraud & H. Mougard, CC-BY-SA-4.0.

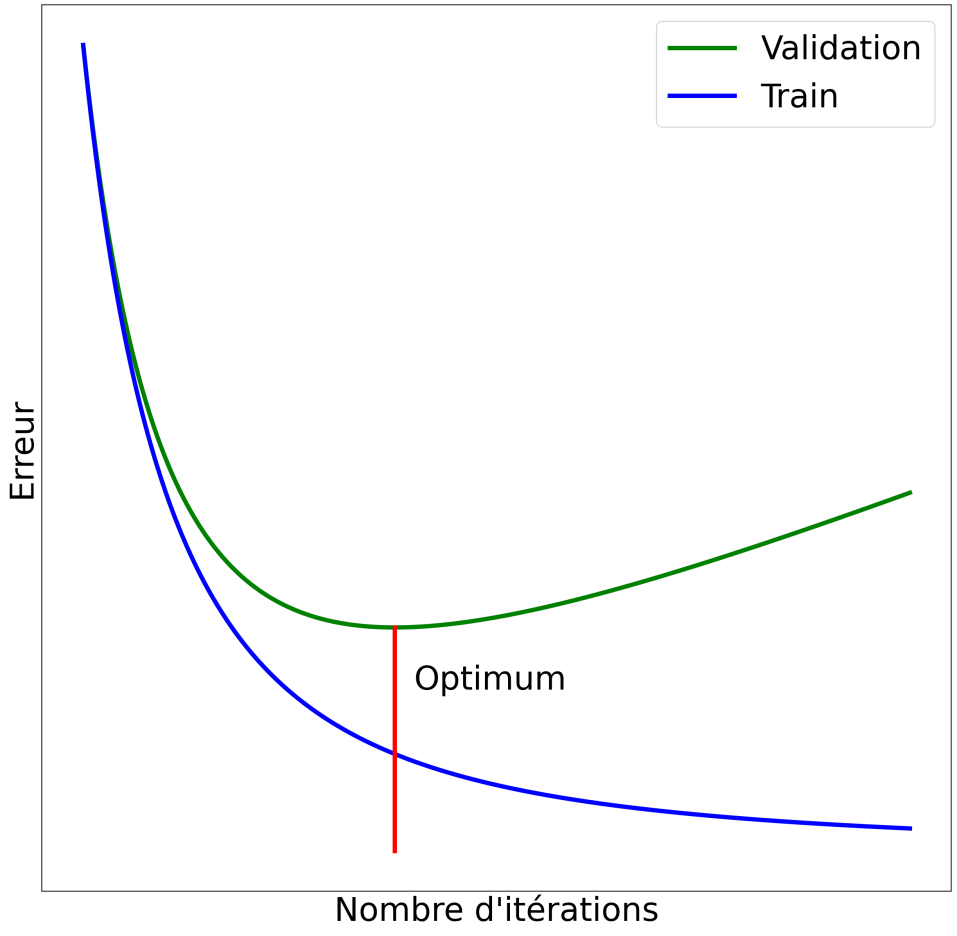
Par la pénalisation de l'utilisation des paramètres

Coût supplémentaire pour l'utilisation des paramètres dans la fonction de perte :

$$\text{perte} = \text{perte} + \lambda \sum ||w||^2$$

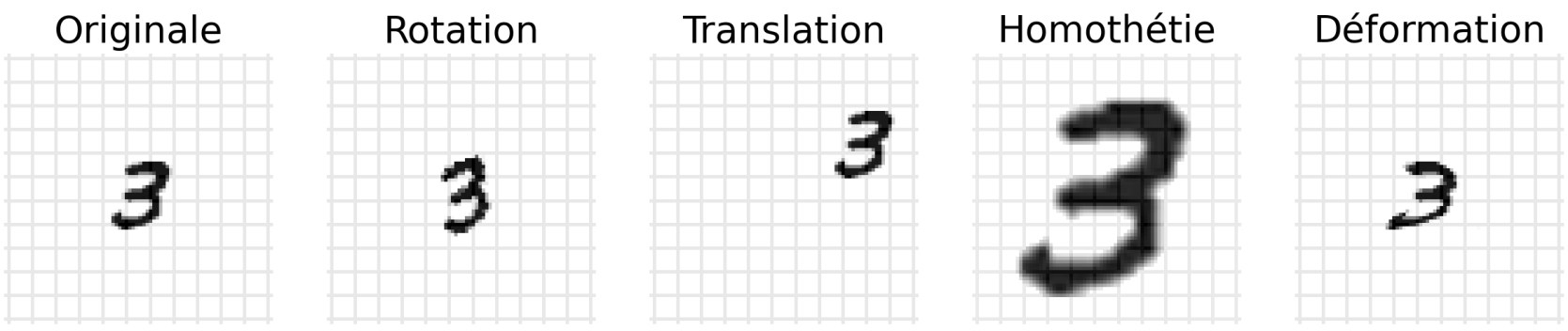
où λ est un hyperparamètre

Par early stopping



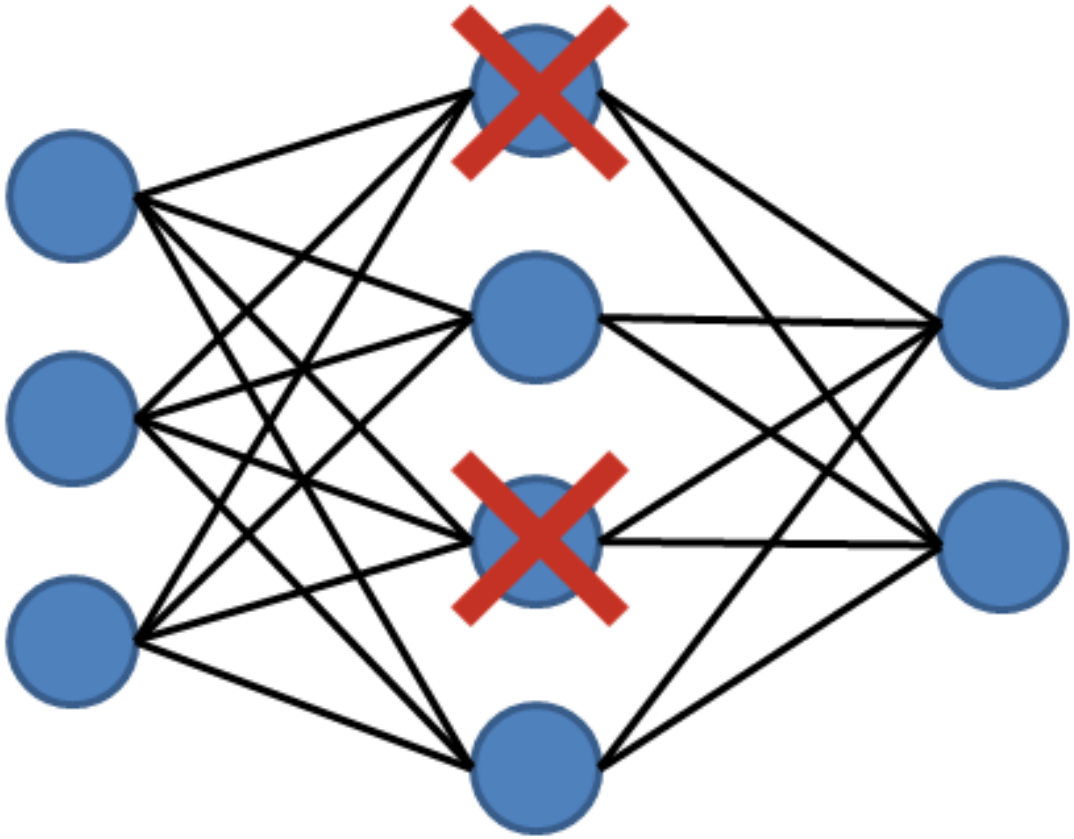
Effet du nombre d'itération sur le surraprentissage des réseaux de neurones, F.-M. Giraud & H. Mougard, CC-BY-SA-4.0.

Par augmentation/bruitage des données

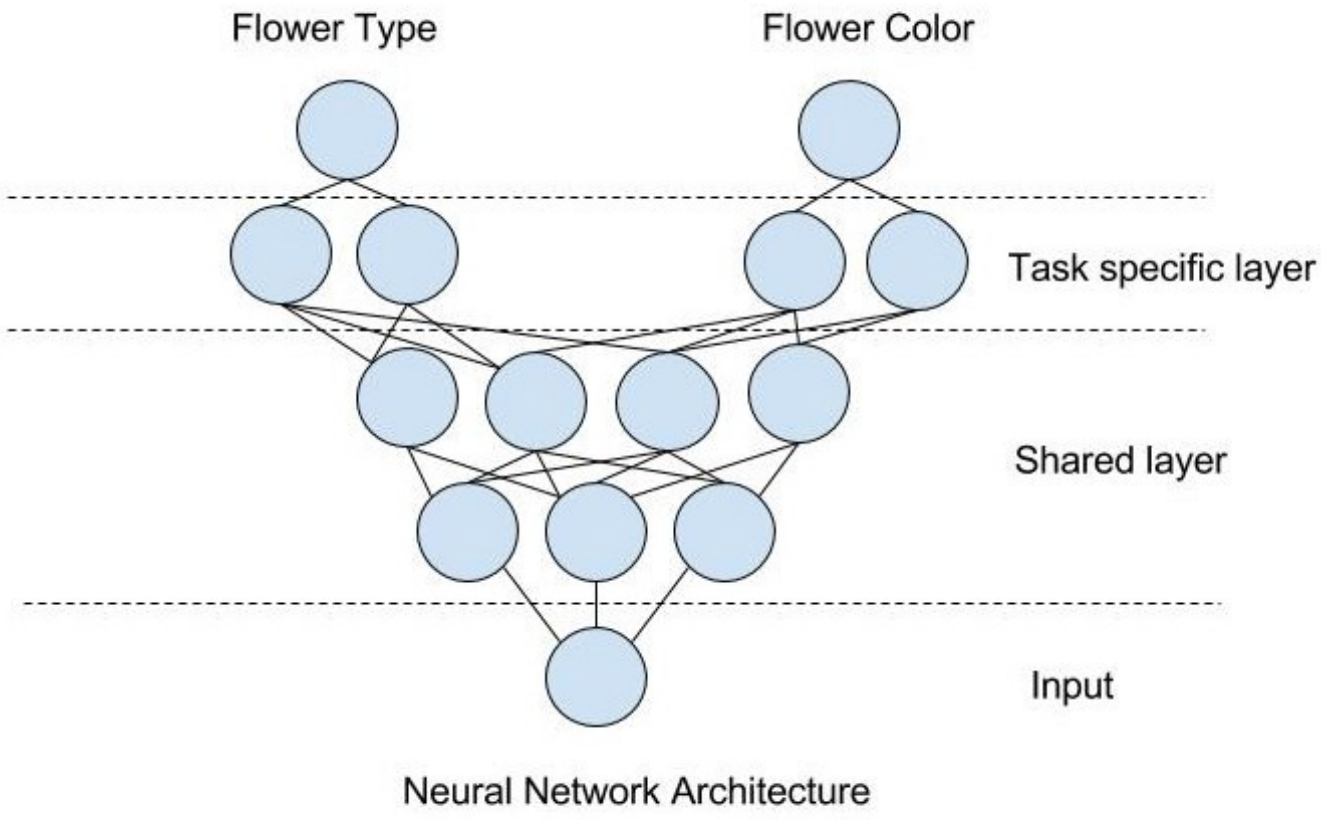


Exemples d'augmentation de données, F.-M. Giraud & H. Mougard, CC-BY-SA-4.0.

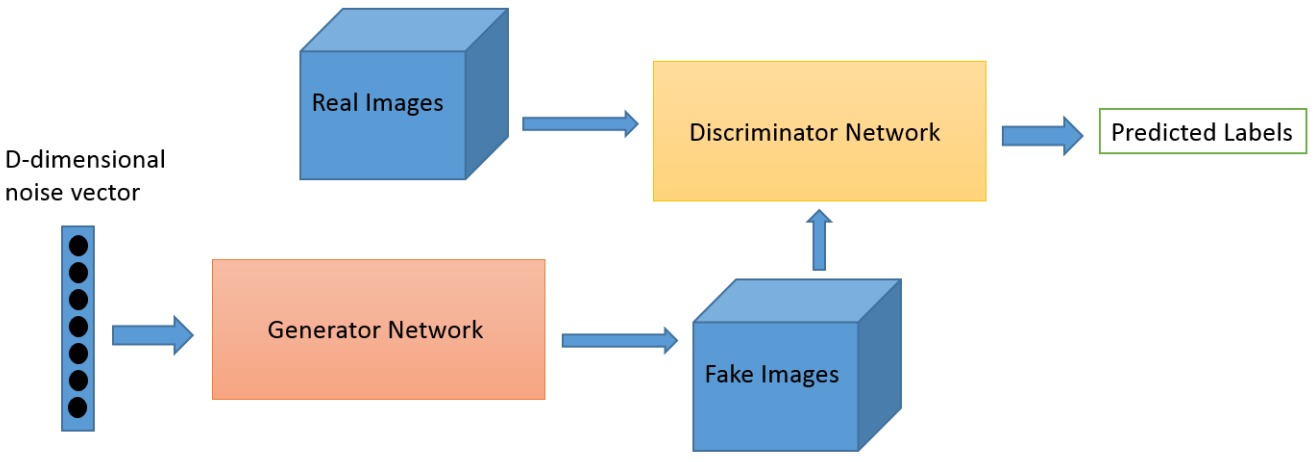
Par dropout



En entraînant sur plusieurs tâches



En opposant des réseaux de neurones



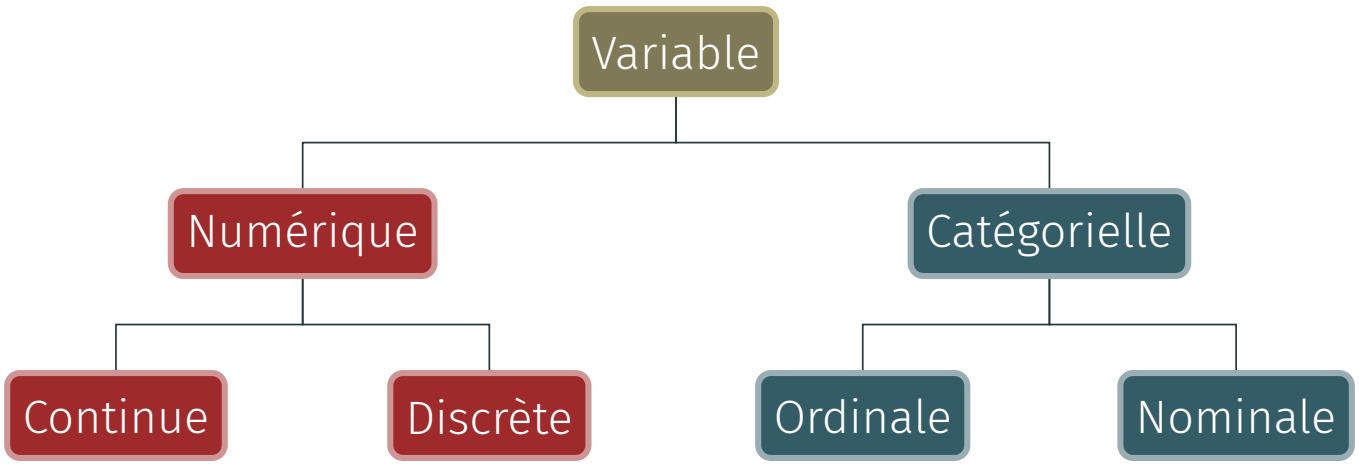
Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, J.-Y. Zhu et al., arXiv.

Démonstration

Visualisation d'un réseau simple

Avez-vous des questions ?

Classification



Classification

A few samples from the MNIST test dataset, Josef Steppan, CC-BY-SA-4.0.

Classification


001.ak47


002.american-flag


003.backpack


004.baseball-bat


005.baseball-glove


006.basketball-hoop


007.bat


008.bathtub


009.bear


010.beer-mug


011.billiards


012.binoculars

Caltech-256 Object Category Dataset, G. Griffin et al., Caltech.

Classification

≈ Distance entre la sortie et la cible ?

Sortie :

0.00	0.10	0.40	0.00	0.00	0.20	0.10	0.00	0.20	0.00

Cible :

0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Classification

Critère de l'erreur absolue (MAE) = 0.12

Sortie :

0.00	0.10	0.40	0.00	0.00	0.20	0.10	0.00	0.20	0.00

Cible :

0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Classification

Critère de l'erreur absolue (MAE) = 0.12

Sortie :

0.12	0.12	0.88	0.12	0.12	0.12	0.12	0.12	0.12	0.12

Cible :

0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Classification

$\Rightarrow$ entropie croisée entre la sortie et la cible :

$$\mathrm{H}(p, q) = -\sum_{x} p(x) \log q(x)$$

- minimale quand sortie = cible
- prend en compte la distribution de la sortie

Principaux outils

Principaux outils

Introduction

Intérêt de l'outillage

Le ML est à la croisée des mathématiques et de l'**ingénierie** :

- Gérer des expérimentations (modèles, données)
- Stocker des quantités importantes de données
- Pouvoir calculer en parallèle
- Mettre en production
- ...

$\rightarrow$ Il est très important de s'outiller !

Principaux outils

Machine Learning & Deep Learning

Librairies fondamentales

Utilisées dans tous les frameworks Python :

- pandas
- NumPy

Machine Learning tabulaire

- scikit-learn
- XGBoost

Deep Learning général

- TensorFlow
- Keras
- PyTorch
- Apache MXNet

Deep Learning pour le texte

- AllenNLP
- spaCy

Deep Learning pour les images

Incontournable : OpenCV

Deep Learning pour les graphes

- DGL
- Graph Nets

Principaux outils

Environnement logiciel

Introduction

Contrôler son environnement logiciel pour :

- Rendre son environnement de développement reproductible
- Contrôler les dépendences pour la prod
- Déployer son environnement facilement sur différents clouds

`virtualenv`

`virtualenv` permet d'isoler un environnement Python :

- N'intéragit pas avec l'environnement système
- Permet la cohabitation de plusieurs environnements incompatibles
- Plus rapide et natif que les solutions basées sur les conteneurs
- Copie d'une distribution Python de base + customisation

Solutions basées sur `virtualenv`

pip + setup.py Définition traditionnelle d'une librarie Python

pip + requirements.txt Liste de dépendences

pip + pip-compile Lister les dépendences puis les geler pour la stabilité

Pipenv Définition moderne d'une **application** Python (pas librairie)

poetry Définition moderne d'application ou librairie

Solutions basées sur les conteneurs

Docker : conteneurs qui embarquent un système d'exploitation en plus de l'environnement Python :

- Permet de contrôler les librairies natives en plus des librairies Python
- Plus grande robustesse si le logiciel s'exécute sur plusieurs OS
- Plus lourd à mettre en place que `virtualenv`
- Plus adapté pour la prod (déploiement facile par Kubernetes)

Principaux outils

Ingénierie

Outils d'aide à l'ingénierie

Buts :

- Collaborer
- Contrôler le source code**, les modèles & les données**
- Déployer facilement

Solutions

Open source :

- mlflow
- dvc
- Kubeflow

Propriétaires :

- Neptune
- Weights & Biases
- comet

Principaux outils

Big Data

Outils pour le Big Data

Buts :

- Stocker efficacement les données et modèles
- Pouvoir traiter la masse importante de données
- Exprimer les algorithmes de ML/DL de manière distribuée

Solutions cloud

- Cloud AWS
- Google Cloud Platform
- Microsoft Azure

Solutions cloud — Intérêts

- Tous les services sont intégrés
- Puissance de calcul ajustable
- APIs intéressantes accessibles (reconnaissance d'images, de texte, ...)

Solutions cloud — Problèmes

- Coût important
- Vendor lock-in
- Confidentialité des données

Libraries Big Data

Librairies de calcul & stockage distribués :

- Apache Spark (+ MLlib)
- Dask (+ Dask-ML)
- Apache Hadoop (récemment de plus en plus délaissé pour Spark)

Librairies de déploiement distribué :

- Kubernetes
- Terraform

Principaux outils

APIs

Introduction

Chaque grand acteur a son API :

- Permet de prototyper très rapidement
- Bonne intégration au reste des plate-formes
- Potentiellement cher

Types d'API proposées

- Traitement images, vidéo
- Traitement de texte, analyse de sentiment, traduction, détection d'entités, ...
- Speech to text, text to speech
- Chatbots
- AutoML, inférence sur séries temporelles, recommendations, ...
- Prévention de fraude

Principaux outils

S’informer

Trouver des papiers scientifiques

- Google Scholar
- Semantic Scholar
- arXiv
- arXiv Sanity Preserver

Rester à jour

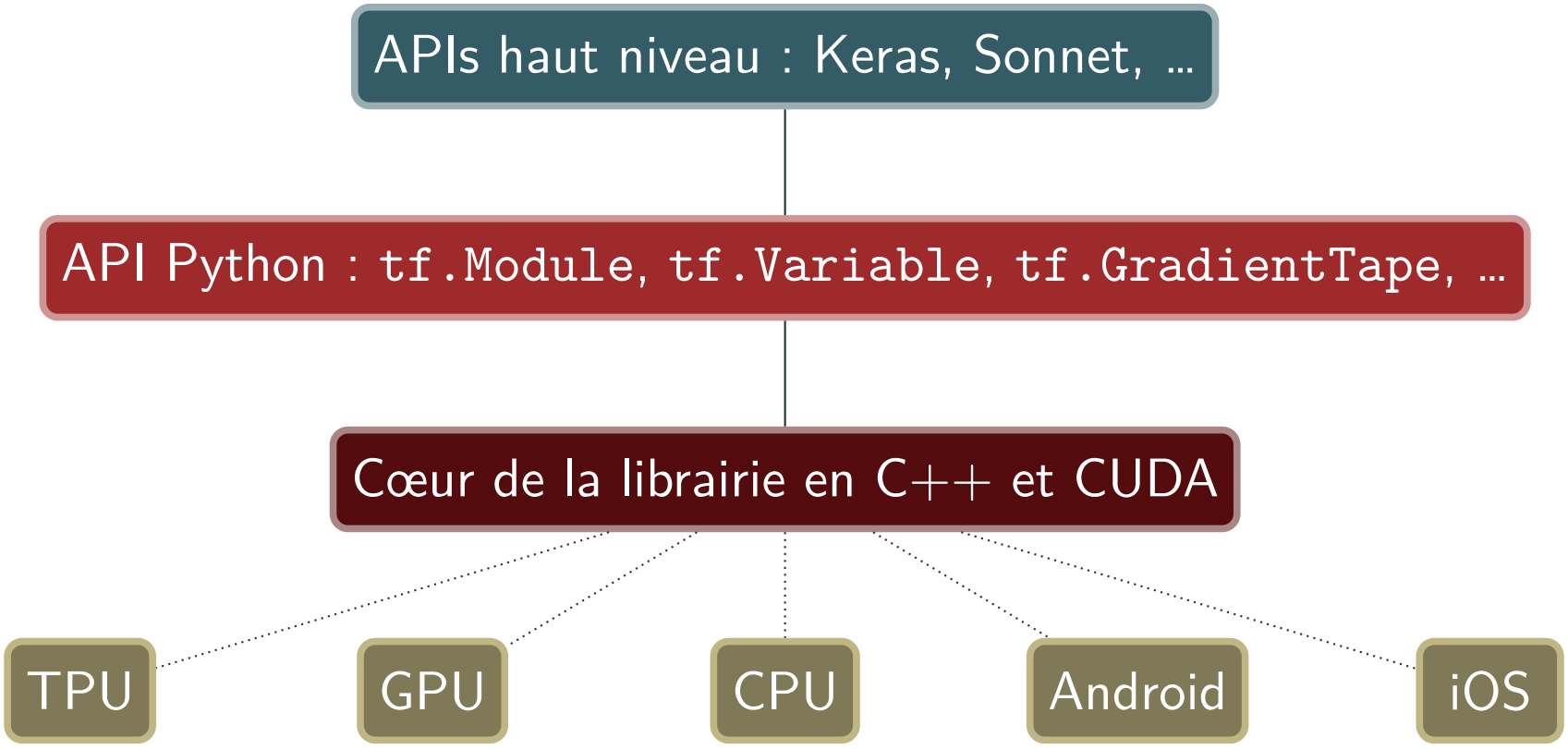
Bien configurer des alertes si nécessaire

Avez-vous des questions ?

TensorFlow 2

Introduction

Structure de l'API



Deux styles

Mode graphe : on définit un graphe de calcul qu'on exécute ensuite.

Mode « eager »: on définit des fonctions python qui opèrent directement sur des valeurs.

TensorFlow 1 vs TensorFlow 2

Principales différences

TF1

- Graphes de calcul explicites par défaut
- API de haut niveau Keras indépendante

TF2

- Graphes de calcul implicites par défaut
- API de haut niveau Keras intégrée à TF

Tenseurs — Objets au cœur de TensorFlow

Tableaux multidimensionnels, utilisés pour :

- Poids des réseaux
- Données

`tf.Tensor` ≈ `numpy.ndarray` dans l'univers TensorFlow.

Tenseurs — Création

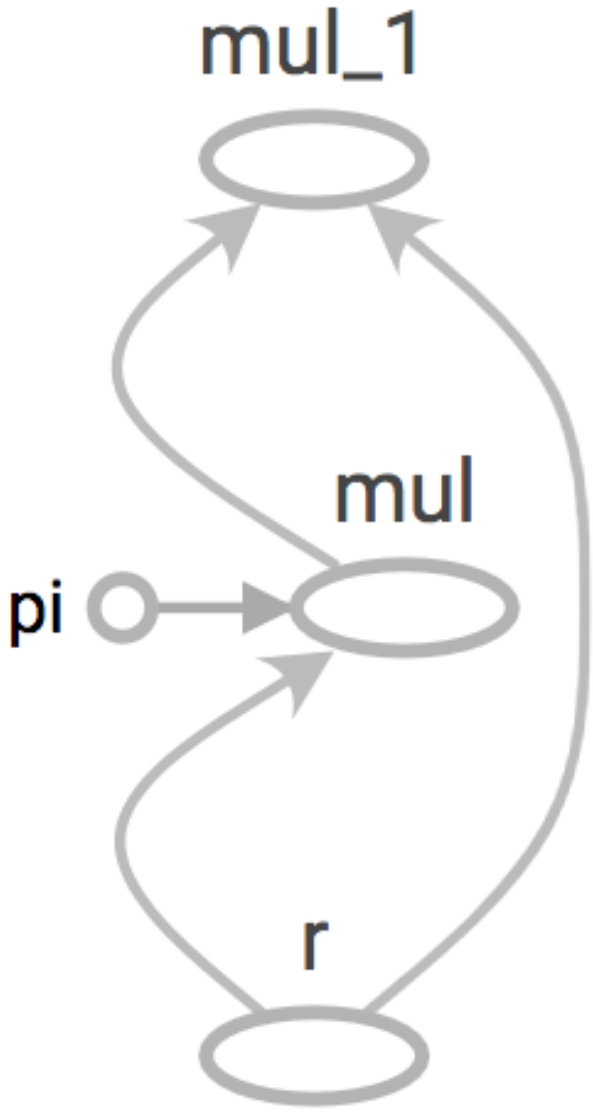
- Directement à partir de tableaux NumPy ou de listes
- Échantillonnés aléatoirement
- Appel à une fonction tensorflow avec un tableau NumPy
- Pipeline `tf.data`

`tf.Variable`

Surcouche mutable de `tf.Tensor` pour les poids des modèles.

Graphe de calcul

$$\pi * r^2 \quad \rightarrow$$



Graphe de calcul

Conversion du mode « eager » au mode graphe par `tf.function`.
Permet :

- L'optimisation du graphe des opérations
- Le déploiement vers Android, iOS, TPU, GPU, ...
- De faire tourner le réseau sans interpréteur Python !

Différenciation automatique

Calcul automatique des gradients pour faciliter la rétropropagation des gradients pendant la descente de gradient.

`tf.GradientTape` enregistre les opérations effectuées sur une variable : permet l'autodiff.

Organisation du code

`tf.Module` regroupe du code d'une même couche ou unité logique.

Un modèle = une combinaison de `tf.Module`s.

API principale :

- Accès aux variables du modèle par `model.variables`
- Accès aux variables entraînables du modèle par `model.trainable_variables`
- Sauvegarde des poids par `tf.train.Checkpoint`
- Chargement des poids par `tf.train.Checkpoint.restore`
- Sauvegarde du modèle complet par `tf.saved_model.save`
- Chargement du modèle complet par `tf.saved_model.load`

Avez-vous des questions ?

Travaux Pratiques — Réseau de neurones avec TensorFlow

Instructions

Réseaux de neurones avec une API Tensorflow de Base