Réseaux de Neurones Artificiels

Machine Learning

Cédric RICHARD
Université Côte d'Azur

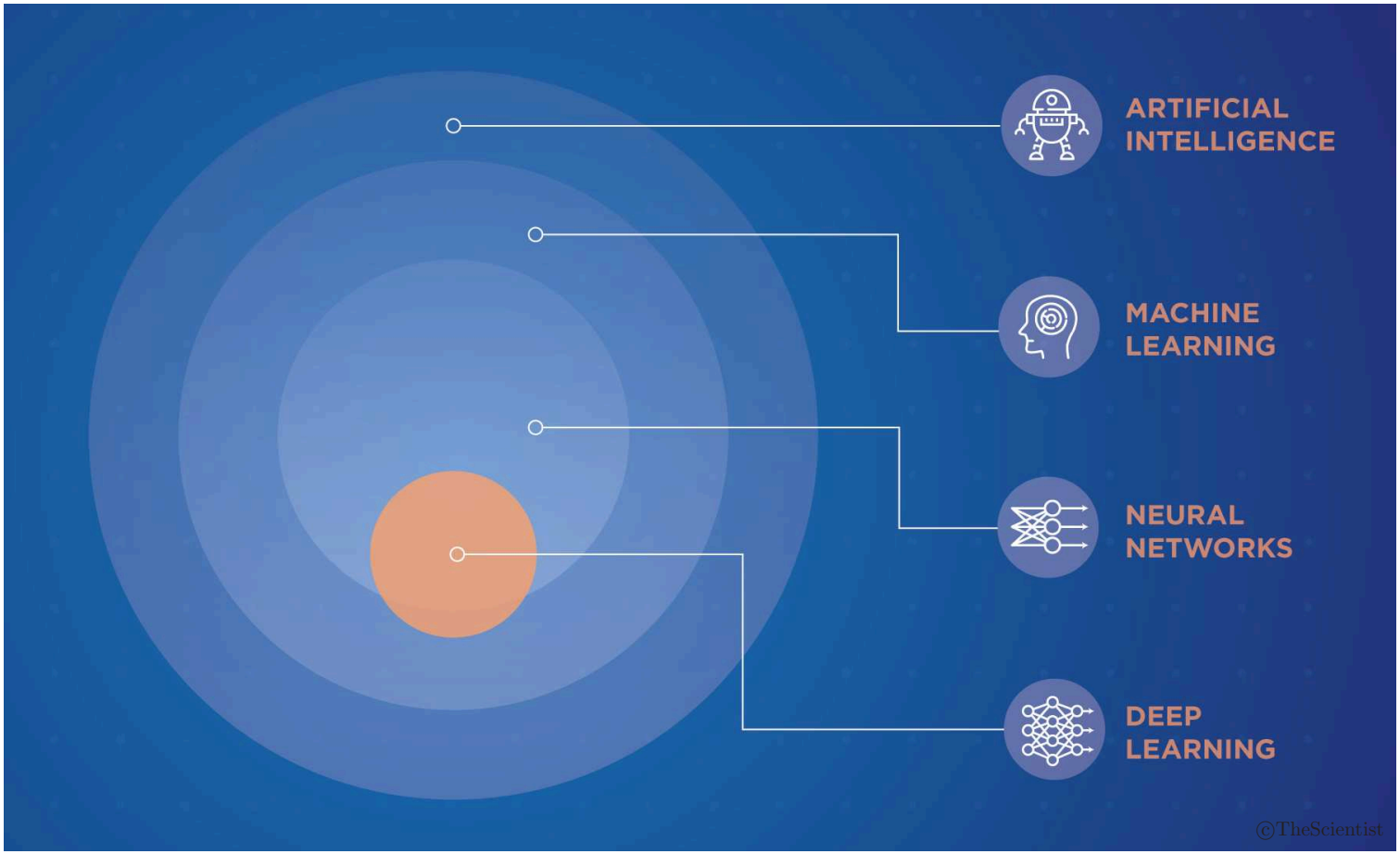
Réseaux de neurones artificiels

L'ère de l'IA

DEEP LEARNING
REINFORCEMENT LEARNING

Réseaux de neurones artificiels

L'ère de l'IA



Réseaux de neurones artificiels

Apprentissage profond en quelques mots

Branche du Machine Learning

- ▷ datant des années 40
- ▷ ayant survécu à deux (longs) hivers

Renaissance spectaculaire en 2006-2012 grâce à la disponibilité

- ▷ de grandes puissances de calcul (GPU)
- ▷ d'importantes quantités de données (internet)
- ▷ de nouvelles architectures profondes et d'algorithmes plus performants

Apprend des représentations complexes :

- ▷ à partir de données
- ▷ par des séquences de transformations non-linéaires
- ▷ agnostique

Réseaux de neurones artificiels

Applications clés

Reconnaissance des formes

- ▷ reconnaissance optique de caractères (OCR)
- ▷ reconnaissance automatique de la parole (ASR)
- ▷ traitement du langage naturel (NLP)
- ▷ compréhension d'images

Prédiction, décision et contrôle

- ▷ véhicules autonomes
- ▷ robotique
- ▷ systèmes complexes

Traitement de l'information :

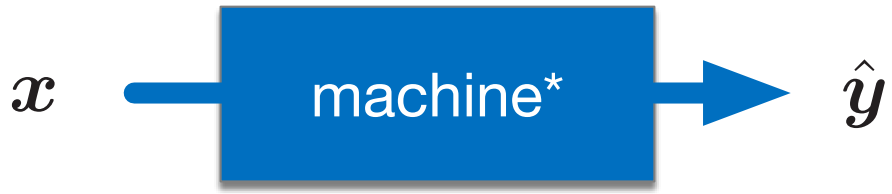
- ▷ amélioration du rapport signal sur bruit
- ▷ hors ligne ou en ligne

Réseaux de neurones artificiels

Machine Learning

A partir de données d'apprentissage, sans information a priori

- ▷ apprentissage en vue de tâches de prédiction (usuel. supervisé)
- ▷ apprentissage en vue d'extraire des structures (éventuel. non-supervisé)



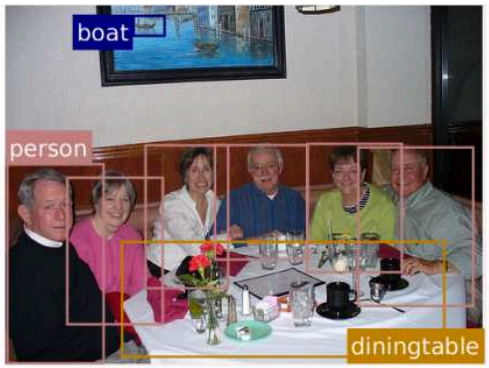
données symboliques
données quantitatives

données symboliques
données quantitatives

* programme avec paramètres ou/et représentation interne

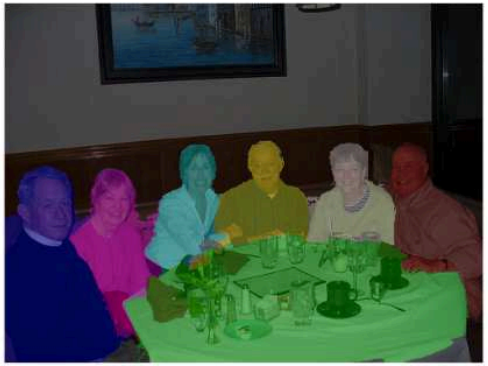
Réseaux de neurones artificiels

Exemples en traitement d'images



object detection

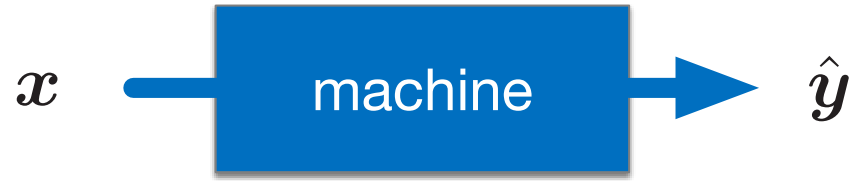

object segmentation


instance segmentation

- ▷ détection d'objet : catégoriser des objects et les repérer avec une boîte
- ▷ segmentation d'objet : identification des pixels appartenant à un objet
- ▷ segmentation d'instance : différencier les objets d'une même classe

Réseaux de neurones artificiels

Exemples en traitement d'images

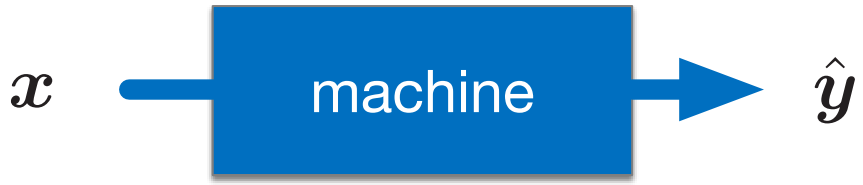


image

carte de profondeur

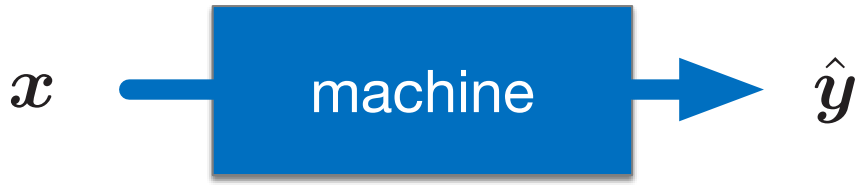
Réseaux de neurones artificiels

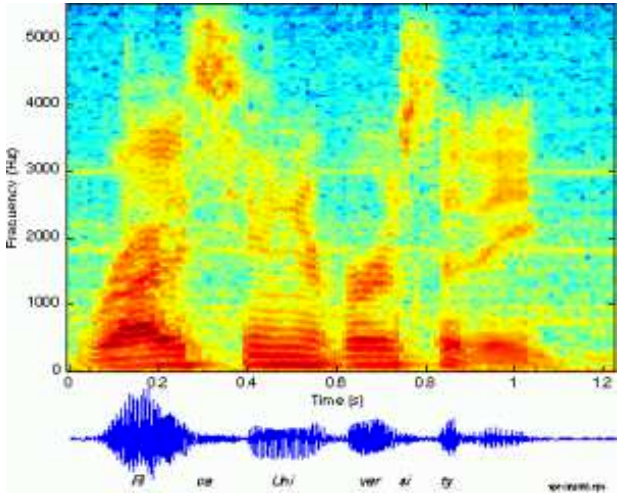
Exemples en traitement d'images



Réseaux de neurones artificiels

Exemples en traitement audio

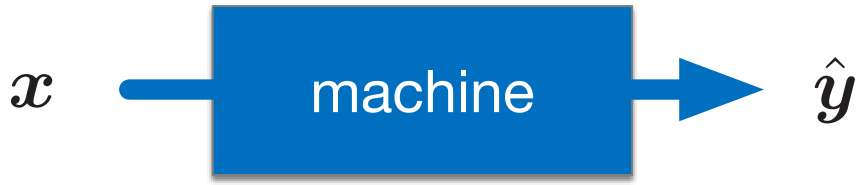




- classification audio
- relevé d'empreintes audio
- marquage automatique
- segmentation audio
- séparation de sources audio
- suivi du rythme
- recommandation musicale
- recherche de signaux audio
- transcription de musique
- detection d'apparition

Réseaux de neurones artificiels

Traitement automatique du langage




```
Mourante, elle vous sera comptée comme le froment.  Sombre comme une
figure de femme, portant une lance et une épée.  Maintenues par la
peur, il prit l'usage des sondes, ce fut elle encore qui le soutint,
pendant plusieurs années, mais peu honorable.  Enchanté est-il d'entrer
en scène.  Nouvelles promesses de l'autre un cadenas.
```

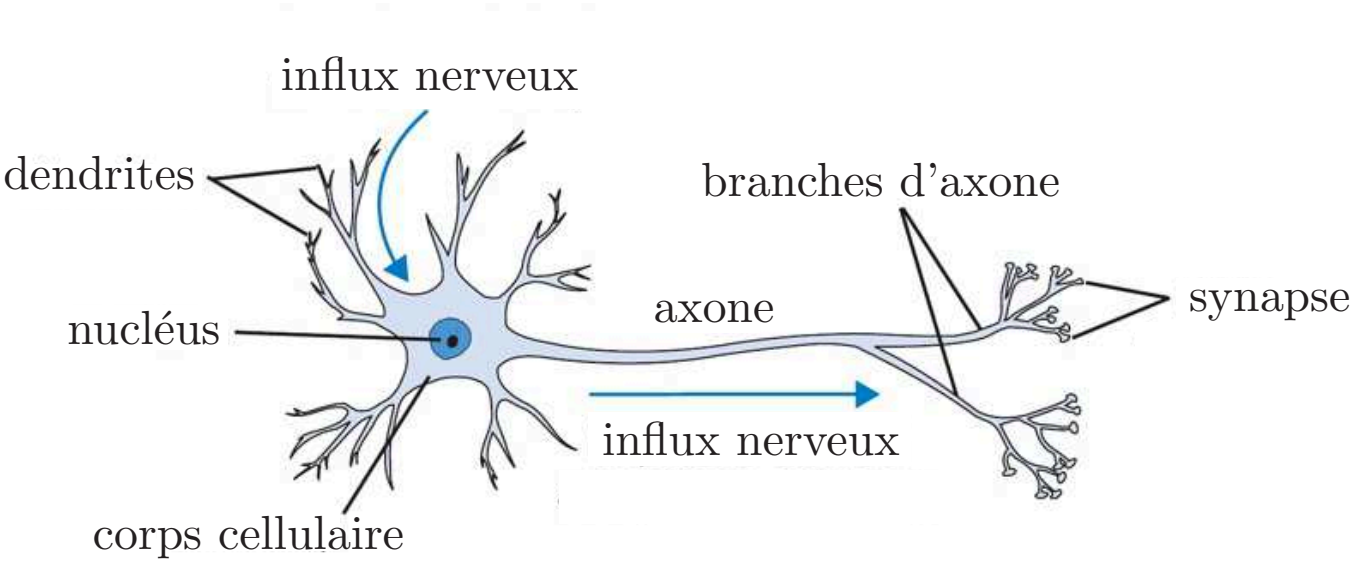
- classification de texte
- modélisation linguistique
- reconnaissance de la parole
- génération de légendes
- traduction automatique
- synthèses de documents
- systèmes de question-réponse

Réseaux de neurones artificiels

Modèle

De nombreuses méthodes de Machine Learning sont inspirées de la biologie

Notre cerveau dispose de 10^{11} neurones, chacun est connecté à 10^4 neurones



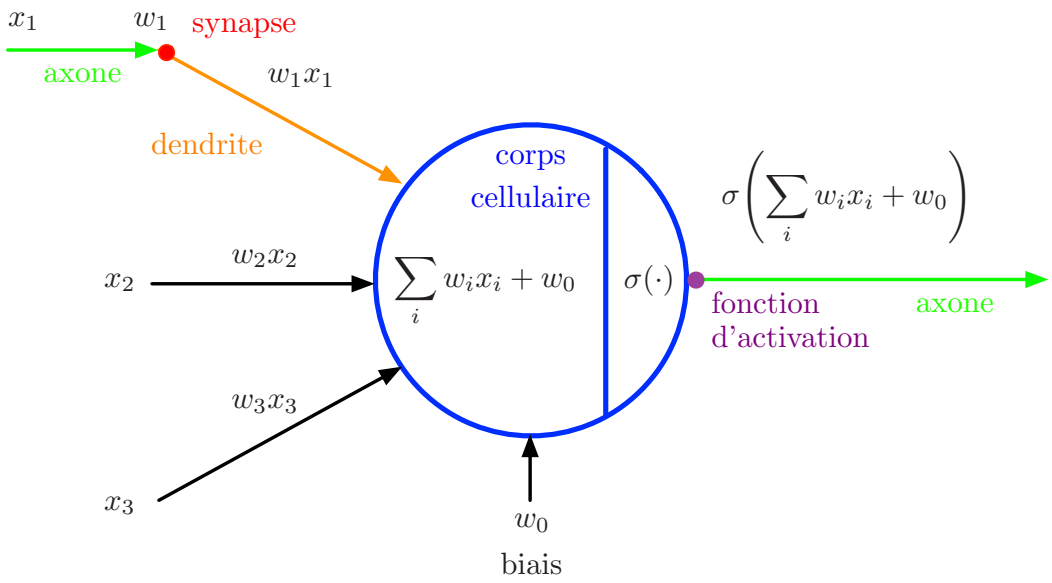
Réseaux de neurones artificiels

Modèle de neurone

Chaque neurone artificiel calcule une sortie à partir de ses entrées

Les opérations mises en œuvre sont

- ▷ combinaison linéaire des entrées
- ▷ application d'une fonction d'activation

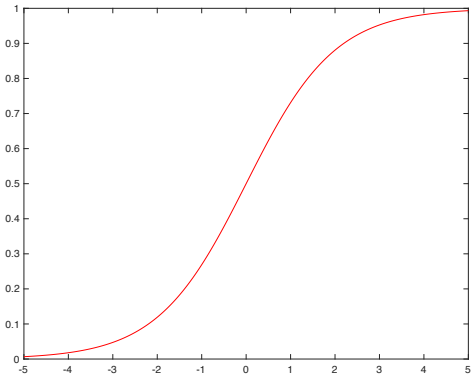


Réseaux de neurones artificiels

Fonctions d'activation

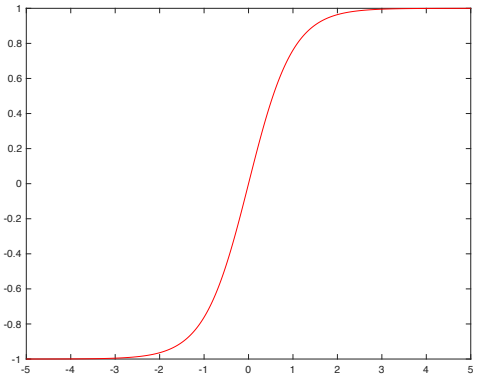
Les fonctions d'activation principalement utilisées sont :

sigmoïde



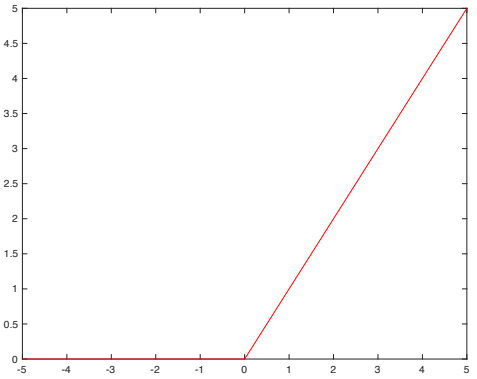
$$\sigma(a) = \frac{1}{1+\exp(-a)}$$

tangente hyperbolique



$$\sigma(a) = \frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)}$$

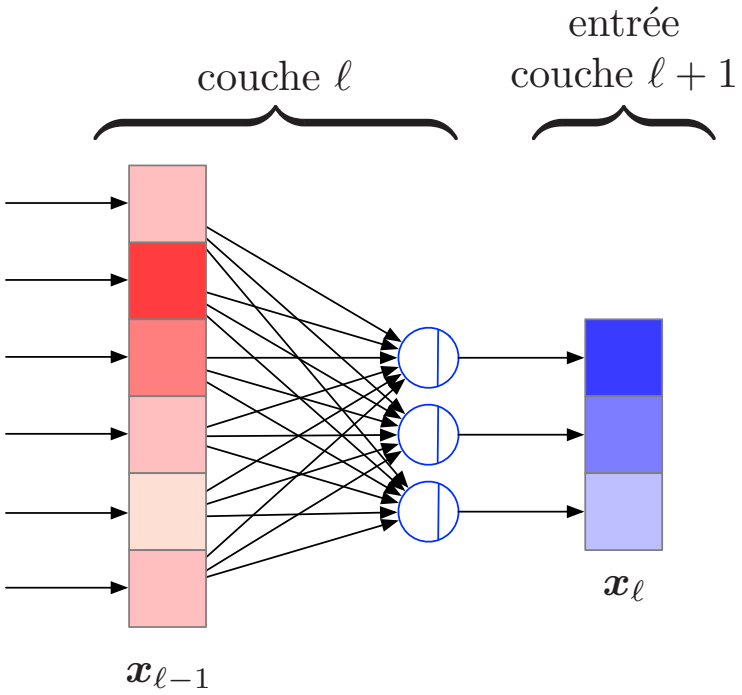
rectified linear unit (ReLU)



$$\sigma(a) = \max\{0, a\}$$

Réseaux de neurones artificiels

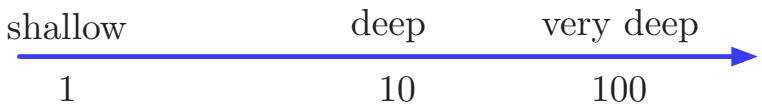
Architecture neuronale

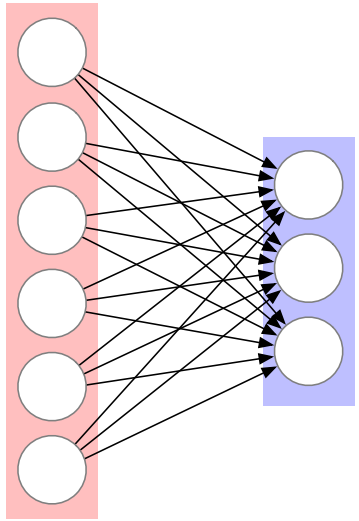


$$\boldsymbol{x}_\ell = \sigma(W_\ell\, \boldsymbol{x}_{\ell-1} + \boldsymbol{w}_{\ell 0})$$

Réseaux de neurones artificiels

Représentation simplifié





couche $\ell - 1$ couche ℓ

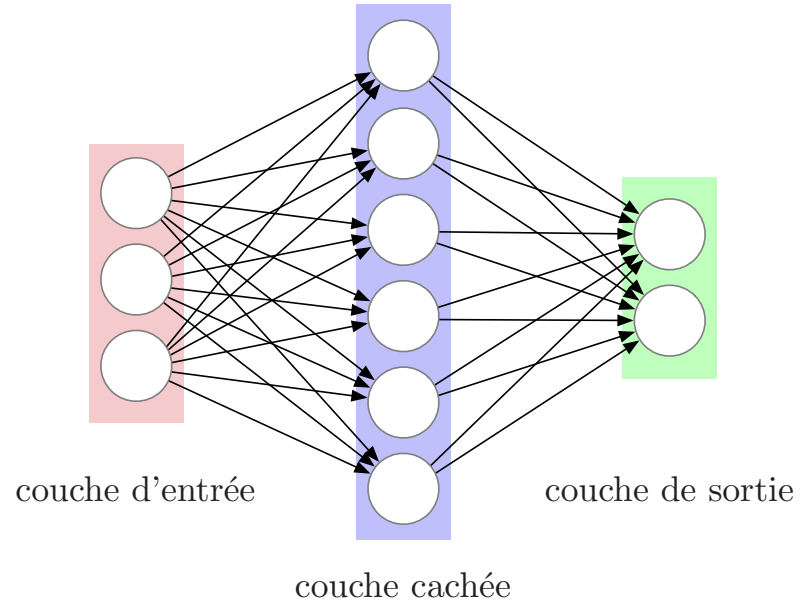
Réseaux de neurones artificiels

Perceptron multicouche

Réseau de neurones à 2 couches (par convention) :

- ▷ une couche cachée à 6 neurones
- ▷ une couche de sortie à 2 neurones

La couche d'entrée n'est pas comptée



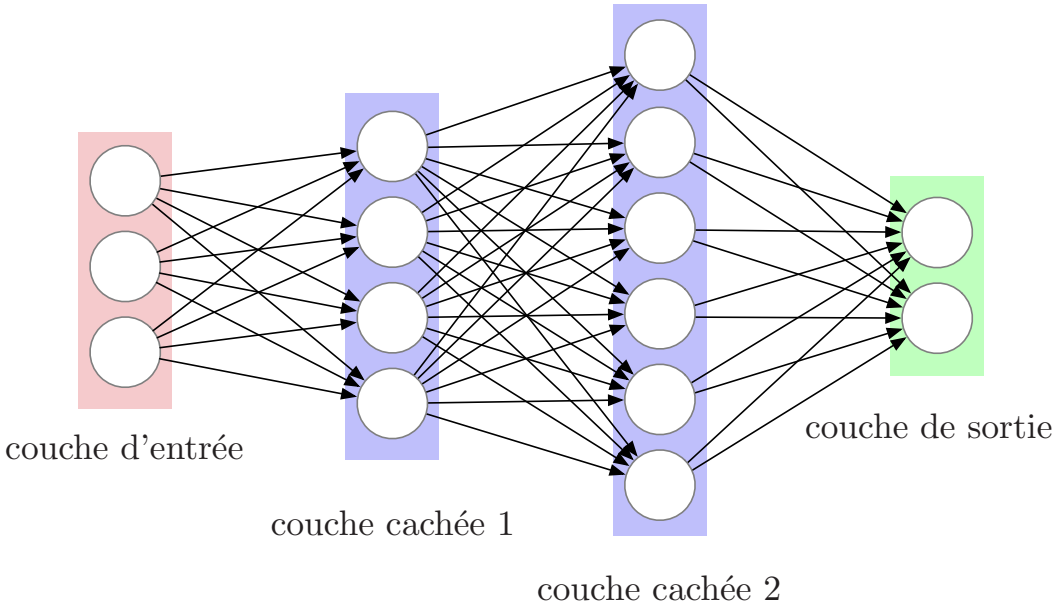
Réseaux de neurones artificiels

Perceptron multicouche

Réseau de neurones à L couches (par convention) :

- ▷ $(L-1)$ couches cachées
- ▷ une couche de sortie

La couche d'entrée n'est pas comptée



Réseaux de neurones artificiels

Perceptron multicouche

La couche d'entrée est le point d'entrée des données
Elle ne dispose pas de fonction d'activation

Chaque couche cachée ℓ est caractérisée par

- ▷ une matrice W_ℓ de poids à estimer, de taille $\dim(\boldsymbol{x}_\ell) \times \dim(\boldsymbol{x}_{\ell-1})$
  notation : nbre de lignes $\times$ nbre de colonnes
- ▷ une fonction d'activation $\sigma(\cdot)$
  notation : $\sigma(\boldsymbol{z})$ s'entend par $\sigma(z_i)$ appliqué à chaque composante z_i de $\boldsymbol{z}$

La couche de sortie dispose éventuellement d'une fonction d'activation $\phi(\cdot)$, différente de $\sigma(\cdot)$, qui dépend du problème à résoudre (régression, classification)

Une fonction objectif à optimiser pour déterminer les matrices W_ℓ de poids

Réseaux de neurones artificiels

Formulation du perceptron multicouche à L couches

▷ **couche d'entrée** : la donnée $\boldsymbol{x}$ est présentée

$$\boldsymbol{x}_0 = \boldsymbol{x}$$

▷ pour chaque **couche cachée** $\ell = 1, \ldots, L-1$

$$\boldsymbol{z}_\ell = W_\ell \boldsymbol{x}_{\ell-1} + w_{\ell 0}$$
$$\boldsymbol{x}_\ell = \sigma(\boldsymbol{z}_\ell)$$

▷ **couche de sortie** : la sortie $\hat{\boldsymbol{y}}$ est obtenue

$$\boldsymbol{z}_L = W_L \boldsymbol{x}_{L-1} + w_{L0}$$
$$\hat{\boldsymbol{y}} = \phi(\boldsymbol{z}_L)$$

Réseaux de neurones artificiels

Théorème d'approximation universelle

On s'intéresse aux réseaux de neurones feedforward à 1 couche cachée, c'est-à-dire, aux fonctions $f : \mathbb{R}^D \to \mathbb{R}$ de la forme

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i \, \sigma(\boldsymbol{w}_i^\top \boldsymbol{x} + w_{i0}) \tag{1}$$

avec $\sigma : \mathbb{R} \to \mathbb{R}$ une fonction d'activation

Cybenko (1989)
Soit $\sigma : \mathbb{R} \to \mathbb{R}$ continue et sigmoïdale ($\lim_{-\infty} \sigma = 0$ et $\lim_{+\infty} \sigma = 1$)
L'ensemble des réseaux de neurones (**??**) est dense dans $\mathcal{C}([0,1]^D, \mathbb{R})$

Hornik (1991)
Soit $K \subset \mathbb{R}^D$ un compact. Soit $\sigma : \mathbb{R} \to \mathbb{R}$ continue, bornée et non-constante
L'ensemble des réseaux de neurones (**??**) est dense dans $\mathcal{C}(K, \mathbb{R})$

Réseaux de neurones artificiels

Fonction objectif

Soit $\hat{\boldsymbol{y}} = f(\boldsymbol{x}, \boldsymbol{\theta})$ la sortie du perceptron où :

$\boldsymbol{x}$: entrée du perceptron

$\boldsymbol{\theta}$: ensemble de tous les paramètres $W_1, \ldots, W_L$ à estimer

Fonction objectif à minimiser :

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{n=1}^{N} J\big(f(\boldsymbol{x}_n, \boldsymbol{\theta}), \boldsymbol{y}_n\big)$$

où

$\mathcal{A} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^{N}$: base d'apprentissage

$J(\hat{\boldsymbol{y}}_n, \boldsymbol{y}_n)$: fonction de perte (loss function) pour la paire $(\hat{\boldsymbol{y}}_n, \boldsymbol{y}_n)$

Réseaux de neurones artificiels

Régression

Fonction de perte

Comme pour la régression linéaire, on utilise généralement l'écart quadratique :

$$\boxed{J(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \|\hat{\boldsymbol{y}} - \boldsymbol{y}\|^2}$$

Couche de sortie

Comme fonction d'activation $\phi(\cdot)$, on utilise généralement l'application identité :

$$\boxed{\phi(\boldsymbol{x}) = \boldsymbol{x}}$$

Ceci signifie qu'aucune non-linéarité n'est appliquée en sortie, après la dernière combinaison linéaire $W_L \boldsymbol{x}_{L-1}$

Réseaux de neurones artificiels

Classification à C classes $\omega_1, \ldots, \omega_C$

Fonction de perte

Comme pour la régression logistique, on utilise généralement l'entropie croisée :

$$\boxed{J(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\sum_{i=1}^{C} y_i \log \hat{y}_i}$$

où y_i (resp., $\hat{y}_i$) désigne la i^{e} composante de $\boldsymbol{y}$ (resp., $\hat{\boldsymbol{y}}$)

Couche de sortie

Puisque l'entropie croisée opère sur des distributions de probabilité, la fonction d'activation $\phi(\cdot)$ de la couche de sortie doit faire en sorte que $\boldsymbol{y}$ en soit une

La fonction d'activation $\phi(\cdot)$ est généralement la fonction *softmax* multidim. :

$$\boxed{\phi_i(\boldsymbol{x}) = \frac{\exp x_i}{\sum_{j=1}^{C} \exp x_j}, \quad \forall i = 1, \ldots, C}$$

Codage des étiquettes

En accord avec les choix précédents, pour chaque donnée $\boldsymbol{x}$ de l'ensemble d'apprentissage, on définit son étiquette $\boldsymbol{y} \in \mathbb{R}^C$ par la fonction indicatrice :

$$\boxed{y_i = 1 \text{ si } \boldsymbol{x} \in \omega_i, \text{ et } y_j = 0 \ \forall j \neq i}$$

Avec ce codage, la fonction de perte (entropie croisée) se ramène à

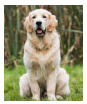
$$\boxed{J(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\log \hat{y}_i} \quad \text{pour} \quad \boldsymbol{x} \in \omega_i$$

où $\hat{y}_i$ désigne la i^{e} composante de $\hat{\boldsymbol{y}}$

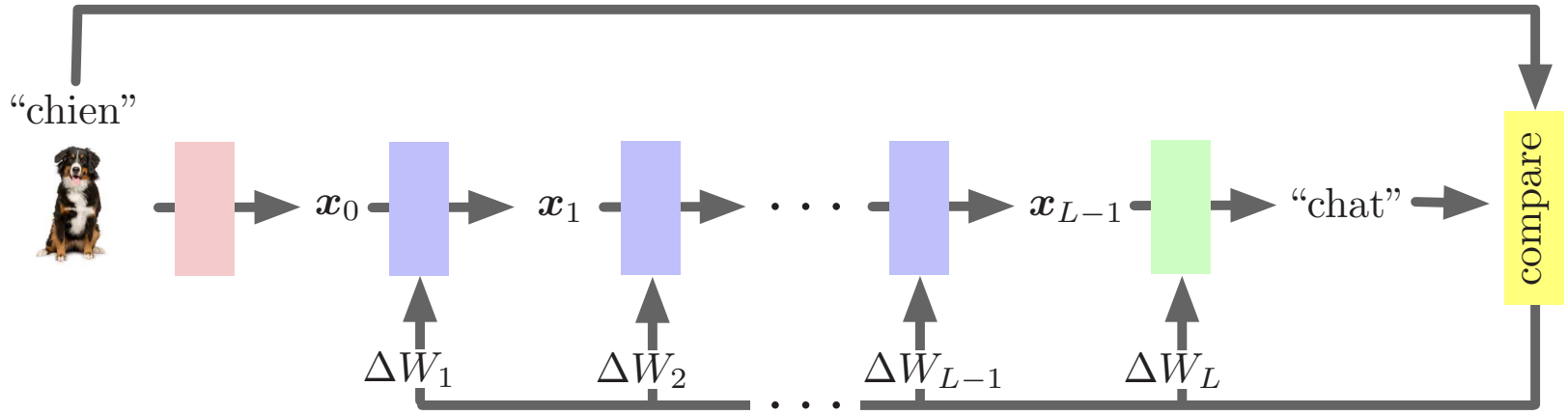
Réseaux de neurones artificiels

Apprentissage supervisé

Données d'apprentissage : base d'exemples annotés (chien, chat, ...)

( , "chien") ( , "chien") ( , "chat") ( , "ara") . . .

Apprentissage : choix des matrices W_ℓ par minimisation d'une fonction objectif



Réseaux de neurones artificiels

Apprentissage par rétropropagation du gradient

Fonction objectif à minimiser :

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{n=1}^{N} J\big(f(\boldsymbol{x}_n, \boldsymbol{\theta}), \boldsymbol{y}_n\big)$$

Afin de résoudre ce problème, une méthode de descente de gradient stochastique doit être mise en œuvre car il n'existe pas de solution analytique.

L'algorithme de rétropropagation du gradient (backprop), de Rumelhart (1988), joue un rôle central en Deep Learning.

Il permet d'estimer le gradient de la fonction objectif en tout point $\boldsymbol{\theta}$.

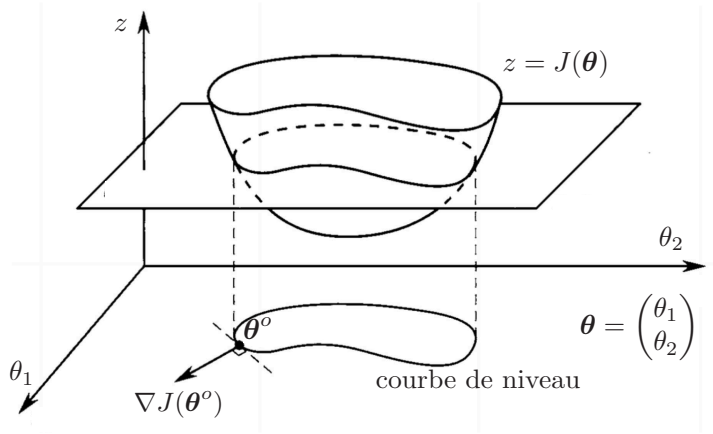
Réseaux de neurones artificiels

Gradient

Le gradient d'une fonction réelle $J(\boldsymbol{\theta})$ de D variables réelles, différentiable en un point $\boldsymbol{\theta}^o$, caractérise la variabilité de la fonction au voisinage de ce point

Dans un système de coordonnées cartésiennes, il est défini par

$$\boxed{\nabla J = \left(\frac{\partial J}{\partial \theta_1}, \ldots, \frac{\partial J}{\partial \theta_D} \right)^\top}$$



Réseaux de neurones artificiels

Gradient

Par la formule de Taylor-Young au point $\boldsymbol{\theta}^o$, on a l'approximation linéaire suivante

$$J(\boldsymbol{\theta}^o + \boldsymbol{h}) = J(\boldsymbol{\theta}^o) + \boldsymbol{h}^\top \nabla J(\boldsymbol{\theta}^o) + o(\|\boldsymbol{h}\|)$$

Donc, l'accroissement de $J(\boldsymbol{\theta})$ en $\boldsymbol{\theta}^o$ dans la direction $\boldsymbol{h}$ avec $\|\boldsymbol{h}\| = 1$ vérifie

$$\boldsymbol{h}^\top \nabla J(\boldsymbol{\theta}^o) \leq \|\boldsymbol{h}\| \cdot \|\nabla J(\boldsymbol{\theta}^o)\| = \|\nabla J(\boldsymbol{\theta}^o)\|$$

En posant $\boldsymbol{h} = \nabla J(\boldsymbol{\theta}^o)/\|\nabla J(\boldsymbol{\theta}^o)\|$, on obtient

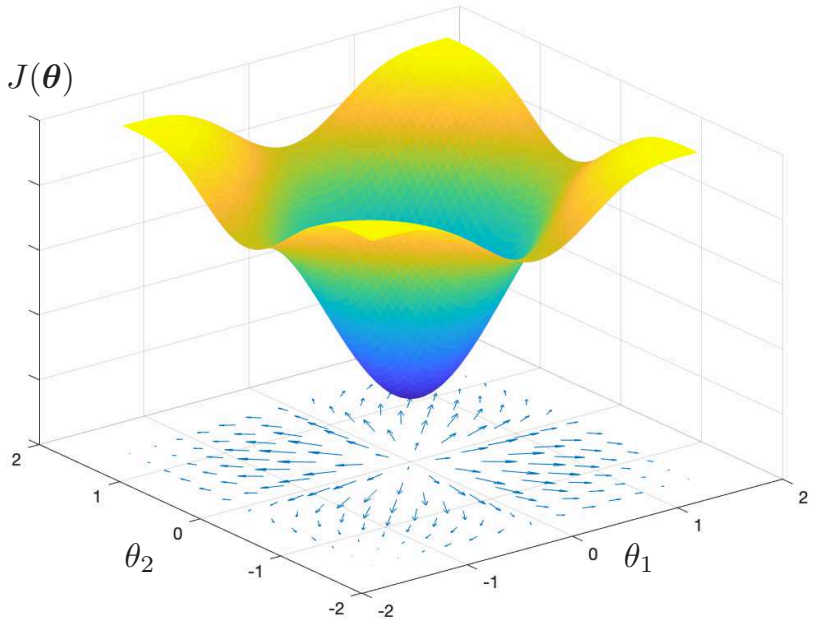
$$\boldsymbol{h}^\top \nabla J(\boldsymbol{\theta}^o) = \|\nabla J(\boldsymbol{\theta}^o)\|$$

On en déduit que $\nabla J(\boldsymbol{\theta}^o)$ est la direction de plus grande pente montante
De même on montre que $-\nabla J(\boldsymbol{\theta}^o)$ est celle de plus grande pente descendante

Réseaux de neurones artificiels

Gradient

Dans un repère orthonormé, $\nabla J(\boldsymbol{\theta}^o)$ pointe dans la direction où la fonction $J(\boldsymbol{\theta})$ croît le plus rapidement en $\boldsymbol{\theta}^o$, et $\|\nabla J(\boldsymbol{\theta}^o)\|$ est son coefficient d'accroissement



Réseaux de neurones artificiels

Descente de gradient

Soit $\boldsymbol{\theta}^o$ un point initial, et $\mu > 0$ un pas d'apprentissage

Par la formule de Taylor-Young en $\boldsymbol{\theta} = \boldsymbol{\theta}^o - \mu \nabla J(\boldsymbol{\theta}^o)$, on a

$$J(\boldsymbol{\theta}) - J(\boldsymbol{\theta}^o) = -\mu \|\nabla J(\boldsymbol{\theta}^o)\|^2 + o(\alpha)$$

En conséquence, si $\nabla J(\boldsymbol{\theta}^o) \neq \mathbf{0}$ et μ suffisamment petit, alors

$$J(\boldsymbol{\theta}) < J(\boldsymbol{\theta}^o)$$

ce qui signifie que $\boldsymbol{\theta}$ est une amélioration par rapport à $\boldsymbol{\theta}^o$ au sens du critère $J(\boldsymbol{\theta})$ lorsque l'on souhaite minimiser celui-ci

Réseaux de neurones artificiels

Algorithme de descente du gradient

Soit un problème de la forme

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

où $J(\boldsymbol{\theta})$ est une fonction objectif différentiable.

Si la condition d'optimalité $\nabla J(\boldsymbol{\theta}) = \mathbf{0}$ n'admet pas de solution analytique, il est nécessaire de recourir à une méthode numérique.

Algorithme du gradient :
Choisir un point initial $\boldsymbol{\theta}^{(0)}$, un seuil ϵ, et un pas d'apprentissage $\mu > 0$
Itérer les étapes suivantes à partir de $k = 0$

1. Calculer $\nabla J(\boldsymbol{\theta}^{(k)})$
2. Test d'arrêt (exemple) : si $\|\nabla J(\boldsymbol{\theta}^{(k)})\| < \epsilon$, arrêt
3. Nouvel itéré : $\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \mu \nabla J(\boldsymbol{\theta}^{(k)})$

Réseaux de neurones artificiels

Tests d'arrêt

La condition nécessaire d'optimalité $\|\nabla J(\boldsymbol{\theta}^{(k)})\| = \mathbf{0}$ n'a pas d'intérêt pratique

En pratique, on préconise

- ▷ Condition sur le gradient : $\|\nabla J(\boldsymbol{\theta}^{(k)})\| < \epsilon$
- ▷ Conditions sur la fonction objectif : $|J(\boldsymbol{\theta}^{(k)}) - J(\boldsymbol{\theta}^{(k-1)})| < \varepsilon$
  ou encore
  $$\frac{|J(\boldsymbol{\theta}^{(k)}) - J(\boldsymbol{\theta}^{(k-1)})|}{J(\boldsymbol{\theta}^{(k-1)})} < \varepsilon$$
- ▷ Conditions sur les itérés : $\|\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}^{(k-1)}\| < \varepsilon$
  ou encore
  $$\frac{\|\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}^{(k-1)}\|}{\|\boldsymbol{\theta}^{(k-1)}\|} < \varepsilon$$
- ▷ On peut remplacer les dénominateurs "$\times$" ci-dessus par $\max\{1, \times\}$

Réseaux de neurones artificiels

Choix du pas d'apprentissage μ

Choix d'un petit pas μ

- ▷ avantage : convergence assurée, le long de la ligne de plus grande pente
- ▷ inconvénient : convergence lente

Choix d'un grand pas μ

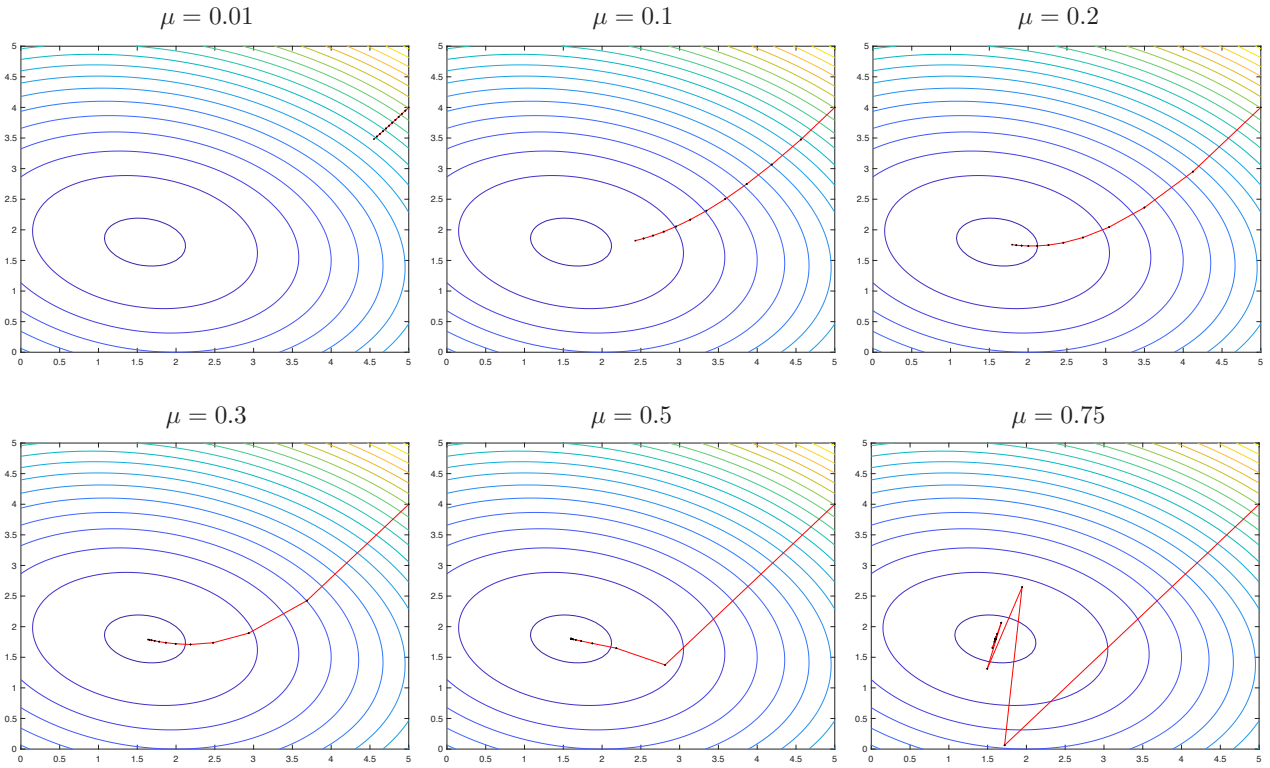
- ▷ avantage : convergence rapide, en peu d'itérations
- ▷ inconvénient : risque important de divergence de l'algorithme

En pratique

- ▷ pas fixe lorsque la fonction objectif est L-Lipschitz ($\mu < \frac{1}{L}$)
- ▷ pas variable par recherche linéaire

Réseaux de neurones artificiels

Choix du pas d'apprentissage μ

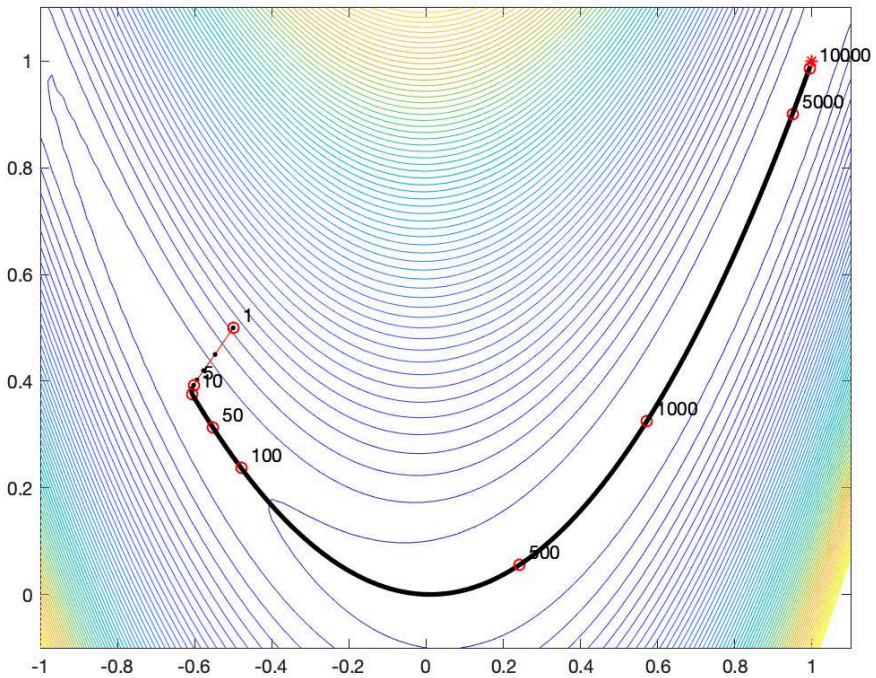


$\text{iter}_{\max} = 10$

Réseaux de neurones artificiels

Choix du pas d'apprentissage μ

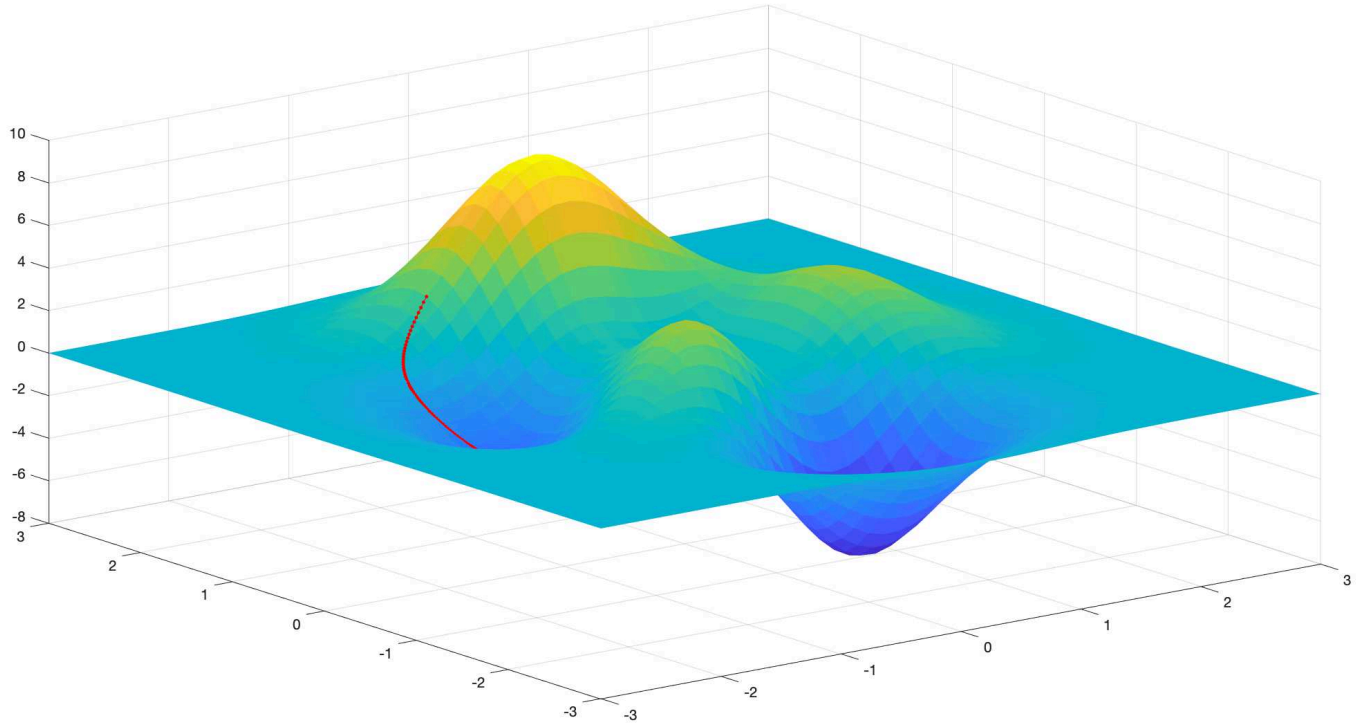
$\text{iter}_{\max} = 10\text{K},\ \mu = 10^{-3}$



Cas pathologique : fonction de Rosenbrock

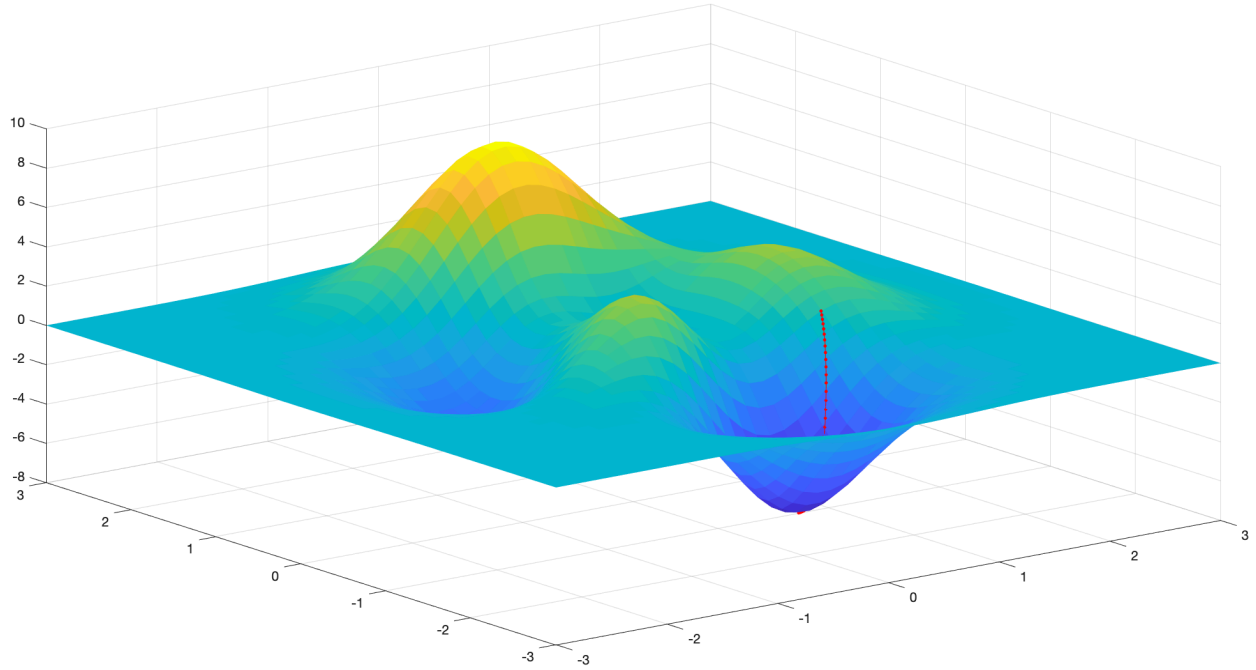
Réseaux de neurones artificiels

Algorithme de descente du gradient : optimums locaux



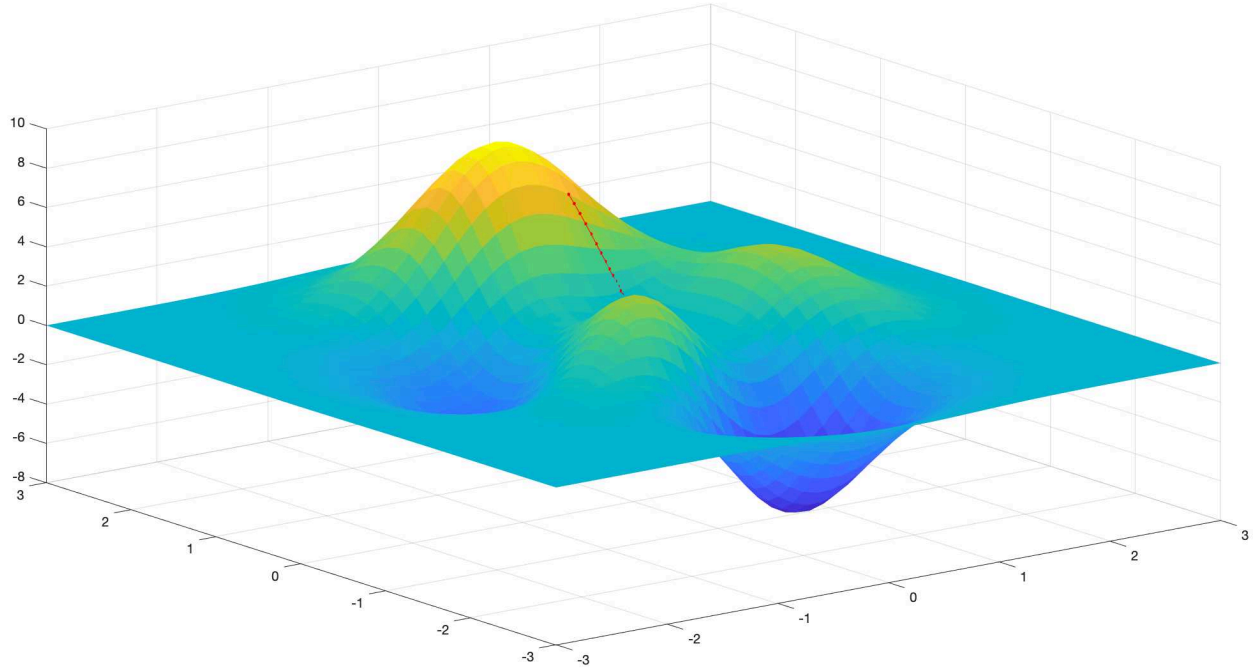
Réseaux de neurones artificiels

Algorithme de descente du gradient : optimums locaux



Réseaux de neurones artificiels

Algorithme de descente du gradient : optimums locaux



Réseaux de neurones artificiels

Apprentissage de réseau de neurones

Descente de gradient : suit la direction de plus grande pente en chaque point

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \frac{\mu}{N} \sum_{n=1}^{N} \nabla_{\boldsymbol{\theta}} J\big(f(\boldsymbol{x}_n, \boldsymbol{\theta}^{(k)}), \boldsymbol{y}_n\big)$$

Descente de gradient stochastique (SGD : stochastic gradient descent) : estime le gradient à partir de la donnée d'apprentissage courante

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \mu \nabla_{\boldsymbol{\theta}} J\big(f(\boldsymbol{x}_{n_k}, \boldsymbol{\theta}^{(k)}), \boldsymbol{y}_{n_k}\big)$$

Descente de gradient stochastique mini-batch : estime le gradient à partir d'un sous-ensemble $\mathcal{B}_k$ de données d'apprentissage

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \frac{\mu}{|\mathcal{B}_k|} \sum_{n \in \mathcal{B}_k} \nabla_{\boldsymbol{\theta}} J\big(f(\boldsymbol{x}_n, \boldsymbol{\theta}^{(k)}), \boldsymbol{y}_n\big)$$

Réseaux de neurones artificiels

Algorithme général

Époque e : passe complète de l'ensemble d'apprentissage
Pas d'apprentissage μ_e : réduit progressivement, e.g., à chaque nouvelle époque e
Mini-batch SGD : traitement d'un sous-ensemble d'apprentissage $\mathcal{B}_k$ à la fois

Réseaux de neurones artificiels

Algorithme général

Data: base d'apprentissage $\mathcal{A}$

Résultat: poids du réseaux de neurone $\boldsymbol{\theta}$

initialisation aléatoire des poids du réseau (petites valeurs);

pour $e = 1$ **à** E **faire**

mettre à jour μ_e;

mélanger la base d'apprentissage;

diviser en sous-ensemble $\mathcal{B}_k$ (mini-batches);

pour $k = 1$ **à** K **faire**

mettre à jour les poids :

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \frac{\mu_e}{|\mathcal{B}_k|} \sum_{n \in \mathcal{B}_k} \nabla_{\boldsymbol{\theta}} J\big(f(\boldsymbol{x}_n, \boldsymbol{\theta}^{(k)}), \boldsymbol{y}_n\big)$$

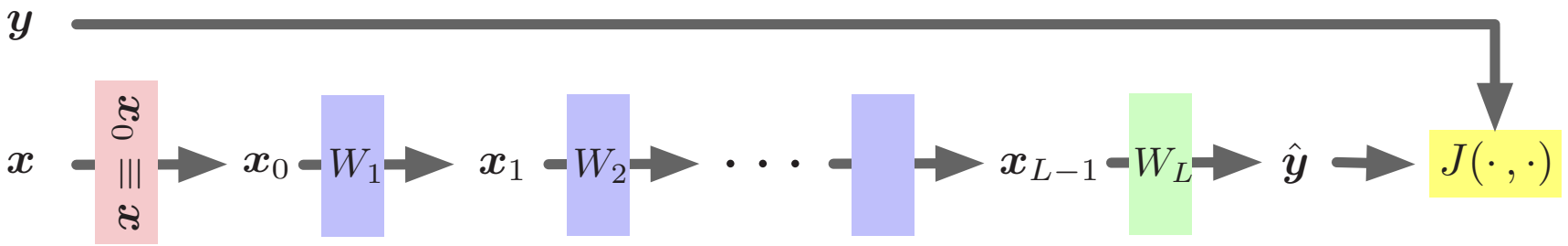
fin

fin

Réseaux de neurones artificiels

Principe de la rétropropagation du gradient

▷ Définition séquentielle de la fonction d'apprentissage



$$\hat{\boldsymbol{y}} = f_L \circ f_{L-1} \circ \ldots \circ f_1(\boldsymbol{x}) = f(\boldsymbol{x}; \underbrace{W_1, \ldots, W_L}_{\boldsymbol{\theta}})$$

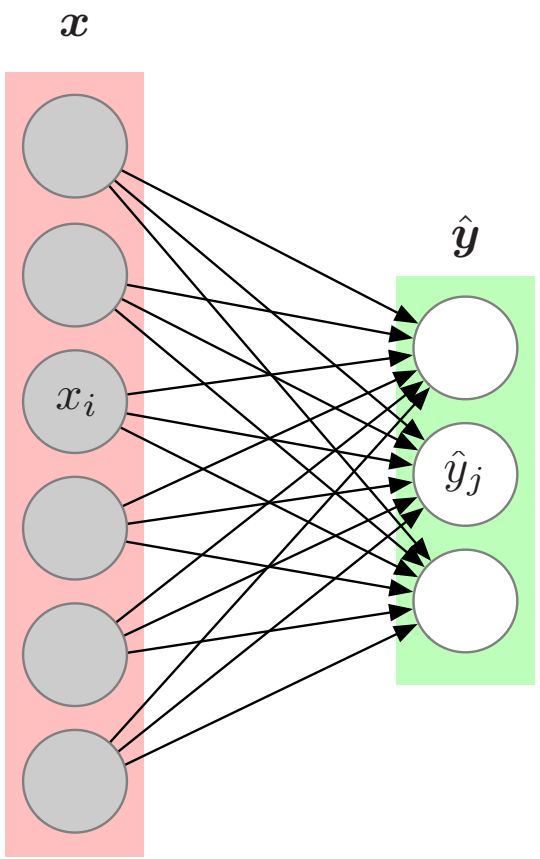
▷ Dépendance à rebours (backward) des paramètres

$$\hat{\boldsymbol{y}} = \mathrm{Fun}(\boldsymbol{x}_{\ell-1}, W_\ell, W_{\ell+1}, \ldots, W_L), \quad \ell = 1, \ldots, L-1$$

▷ Rétropropagation du gradient : calcul à rebours du gradient $W_L \longrightarrow W_1$

Réseaux de neurones artificiels

Calcul du gradient : cas d'un réseau à 1 couche

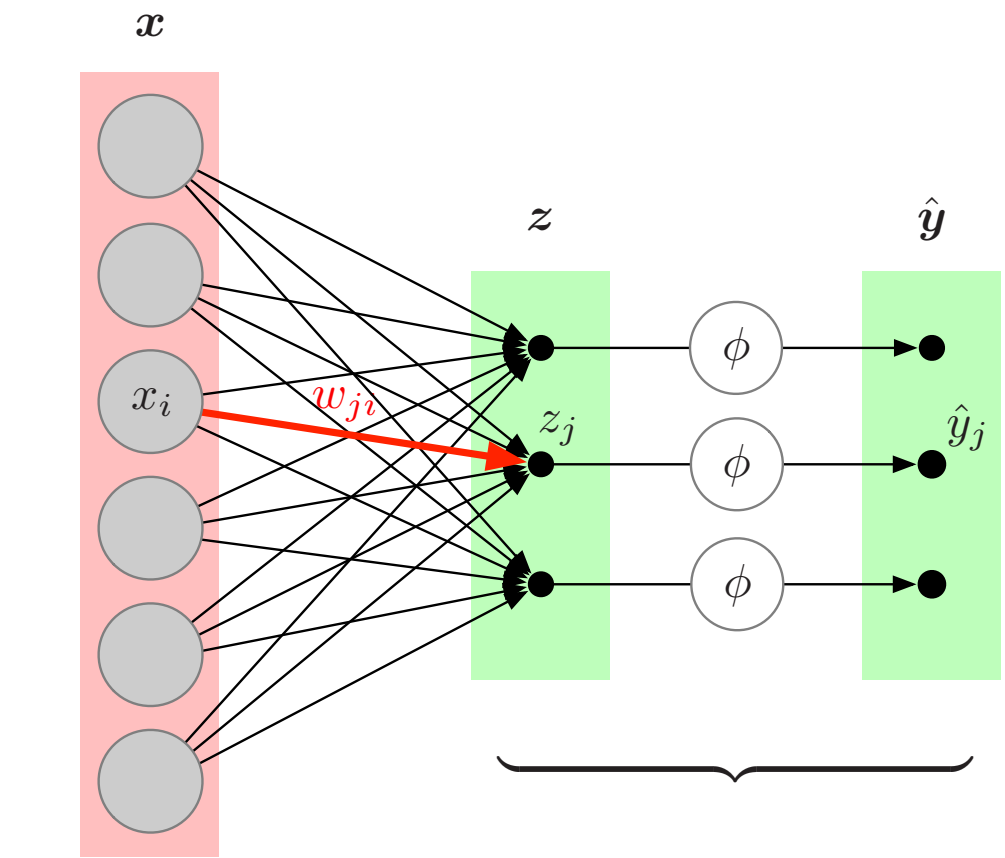


couche d'entrée couche de sortie

$$\begin{cases} \boldsymbol{z} = W\boldsymbol{x} \\ \hat{\boldsymbol{y}} = \phi(\boldsymbol{z}) \end{cases}$$

Réseaux de neurones artificiels

Calcul du gradient : cas d'un réseau à 1 couche

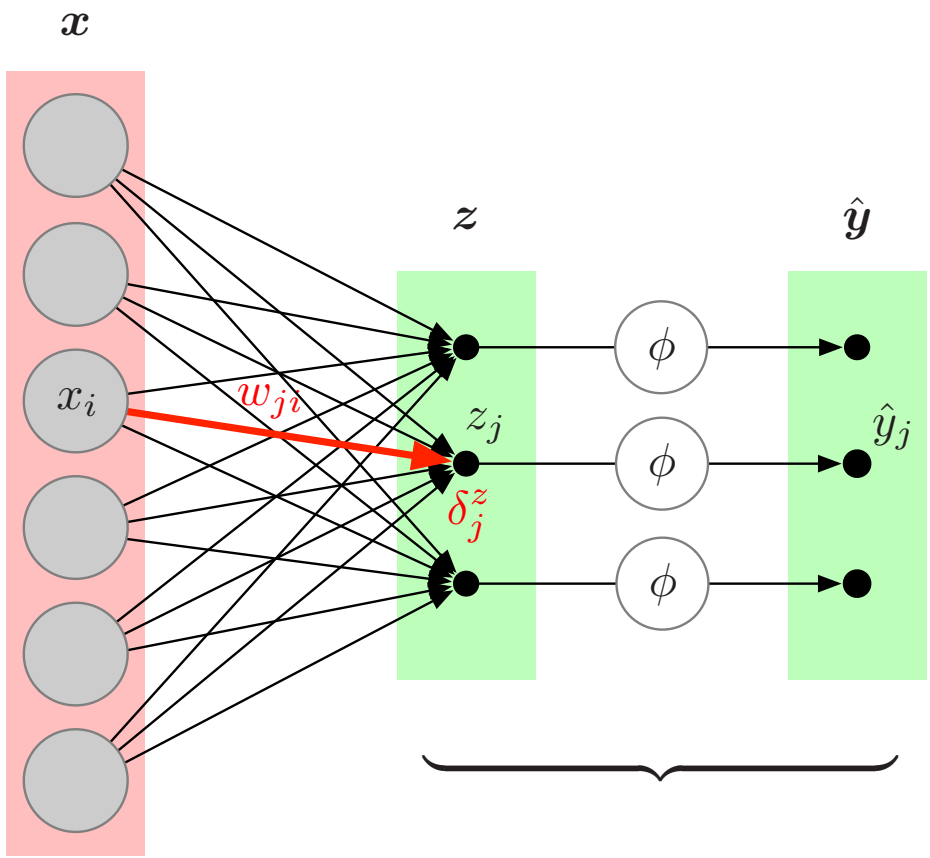


$$\begin{cases} \boldsymbol{z} = W\boldsymbol{x} \\ \hat{\boldsymbol{y}} = \phi(\boldsymbol{z}) \end{cases}$$

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{ji}}$$

Réseaux de neurones artificiels

Calcul du gradient : cas d'un réseau à 1 couche

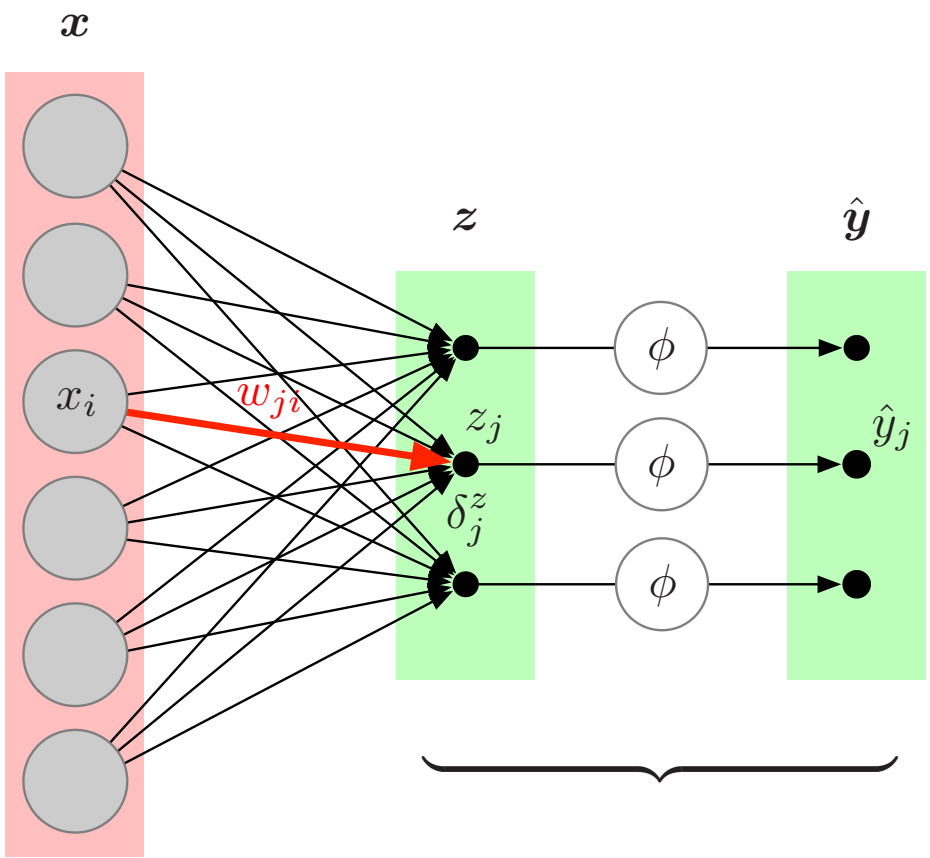


$$\begin{cases} \boldsymbol{z} = W\boldsymbol{x} \\ \hat{\boldsymbol{y}} = \phi(\boldsymbol{z}) \end{cases}$$

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{ji}} = \delta_j^z \cdot \frac{\partial z_j}{\partial w_{ji}}$$

Réseaux de neurones artificiels

Calcul du gradient : cas d'un réseau à 1 couche



$$\begin{cases} \boldsymbol{z} = W\boldsymbol{x} \\ \hat{\boldsymbol{y}} = \phi(\boldsymbol{z}) \end{cases}$$

$$\frac{\partial J}{\partial w_{ji}} = \delta_j^z \cdot \frac{\partial z_j}{\partial w_{ji}} = \delta_j^z \cdot x_i$$

Le résultat précédent, $\frac{\partial J}{\partial w_{ji}} = \delta_j^z \, x_i$, permet de réécrire le gradient recherché sous forme matricielle :

$$\boxed{\nabla_W J = \boldsymbol{\delta}^z \boldsymbol{x}^\top}$$

Réseaux de neurones artificiels

Calcul de δ_j^z

Dans le cas général : $\hat{y}_j = \phi(\boldsymbol{z})$

$$\boxed{\delta_j^z \triangleq \frac{\partial J}{\partial z_j} = \sum_p \frac{\partial J}{\partial \hat{y}_p} \cdot \frac{\partial \hat{y}_p}{\partial z_j}}$$

Voir l'exemple de la fonction *softmax* plus loin

En pratique

On trouve surtout $\hat{y}_j = \phi(z_j)$, en particulier pour les couches cachées. Alors :

$$\delta_j^z = \frac{\partial J}{\partial \hat{y}_j} \cdot \frac{\partial \hat{y}_j}{\partial z_j}$$

car $\frac{\partial \hat{y}_p}{\partial \hat{z}_j} = 0$ si $p \neq j$. En notant $\odot$ le produit terme à terme, on écrit

$$\boxed{\boldsymbol{\delta}^z = \phi'(\boldsymbol{z}) \odot \frac{\partial J}{\partial \hat{\boldsymbol{y}}}}$$

Réseaux de neurones artificiels à 1 couche

Application à la régression

Soit une donnée d'apprentissage $(\boldsymbol{x}, \boldsymbol{y})$

Fonction de perte

La fonction de perte pour $(\boldsymbol{x}, \boldsymbol{y})$, i.e., l'écart quadratique, s'écrit :

$$J(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{2}\|\hat{\boldsymbol{y}} - \boldsymbol{y}\|^2$$

En conséquence

$$\frac{\partial J}{\partial \hat{\boldsymbol{y}}} = \hat{\boldsymbol{y}} - \boldsymbol{y}$$

Fonction d'activation

La fonction d'activation de sortie ϕ, i.e., la fonction *identité* $\phi(\boldsymbol{z}) = \boldsymbol{z}$, conduit à :

$$\phi'(\boldsymbol{z}) = \mathbf{1}$$

En combinant les résultats précédents, on obtient :

$$\boldsymbol{\delta}^z = \hat{\boldsymbol{y}} - \boldsymbol{y}$$

Conclusion

Étant donné un élément de la base d'apprentissage $(\boldsymbol{x}, \boldsymbol{y})$, on en déduit que le gradient de J par rapport à W pour le couple entrée-sortie $(\boldsymbol{x}, \boldsymbol{y})$ est donné par

$$\boxed{\nabla_W J = (\hat{\boldsymbol{y}} - \boldsymbol{y})\, \boldsymbol{x}^\top}$$

Réseaux de neurones artificiels à 1 couche

Application à la régression

Data: base d'apprentissage $\mathcal{A}$

Résultat: poids W de la couche

initialisation aléatoire de W (petites valeurs);

pour $e = 1$ **à** E **faire**

mettre à jour μ_e;

mélanger la base d'apprentissage ; diviser en K mini-batches $\mathcal{B}_k$;

calculer la sortie estimée $\hat{\boldsymbol{y}}_n$ de chaque donnée $(\boldsymbol{x}_n, \boldsymbol{y}_n)$ de $\mathcal{B}_k$;

pour $k = 1$ **à** K **faire**

mettre à jour les poids :

$$W^{(k+1)} = W^{(k)} - \frac{\mu_e}{|\mathcal{B}_k|} \sum_{n \in \mathcal{B}_k} (\hat{\boldsymbol{y}}_n - \boldsymbol{y}_n)\, \boldsymbol{x}_n^\top$$

fin

fin

Réseaux de neurones artificiels à 1 couche

Application à la classification à C classes $\omega_1, \ldots, \omega_C$

Soit une donnée d'apprentissage $\boldsymbol{x}_0 \in \omega_{i_0}$

Fonction de perte

La fonction de perte pour $\boldsymbol{x}_0$, i.e., l'entropie croisée, s'écrit :

$$J(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\log \hat{y}_{i_0}$$

En conséquence

$$\frac{\partial J}{\partial \hat{y}_p} = -\frac{1}{\hat{y}_{i_0}} \text{ si } p = i_0, \text{ sinon } \frac{\partial J}{\partial \hat{y}_p} = 0$$

Fonction d'activation

La fonction d'activation de sortie ϕ, i.e., la fonction *softmax*, conduit à :

$$\frac{\partial \hat{y}_p}{\partial z_j} = \hat{y}_p(1 - \hat{y}_p) \text{ si } p = j, \text{ sinon } \frac{\partial \hat{y}_p}{\partial z_j} = -\hat{y}_p \hat{y}_j$$

Parce que $\delta_j^z = \sum_p \frac{\partial J}{\partial \hat{y}_p} \cdot \frac{\partial \hat{y}_p}{\partial z_j}$, en combinant les résultats précédents, on obtient :

$$\delta_j^z = \hat{y}_j - 1 \text{ si } j = i_0, \text{ sinon } \delta_j^z = \hat{y}_j$$

Puisque $y_j = 1$ si $j = i_0$, sinon $y_j = 0$, on peut écrire $\delta_j^z = \hat{y}_j - y_j$, soit

$$\boldsymbol{\delta}^z = \hat{\boldsymbol{y}} - \boldsymbol{y}$$

Conclusion

Étant donné un élément de la base d'apprentissage $\boldsymbol{x}_0 \in \omega_{i_0}$, on en déduit que le gradient de J par rapport à W pour le couple entrée-sortie $(\boldsymbol{x}, \boldsymbol{y})$ est donné par

$$\boxed{\nabla_W J = (\hat{\boldsymbol{y}} - \boldsymbol{y})\, \boldsymbol{x}^\top}$$

Réseaux de neurones artificiels à 1 couche

Application à la classification à C classes $\omega_1, \ldots, \omega_C$

Data: base d'apprentissage $\mathcal{A}$

Résultat: poids W de la couche

initialisation aléatoire de W (petites valeurs);

pour $e = 1$ **à** E **faire**

mettre à jour μ_e;

mélanger la base d'apprentissage ; diviser en K mini-batches $\mathcal{B}_k$;

calculer la sortie estimée $\hat{\boldsymbol{y}}_n$ de chaque donnée $(\boldsymbol{x}_n, \boldsymbol{y}_n)$ de $\mathcal{B}_k$;

pour $k = 1$ **à** K **faire**

mettre à jour les poids :

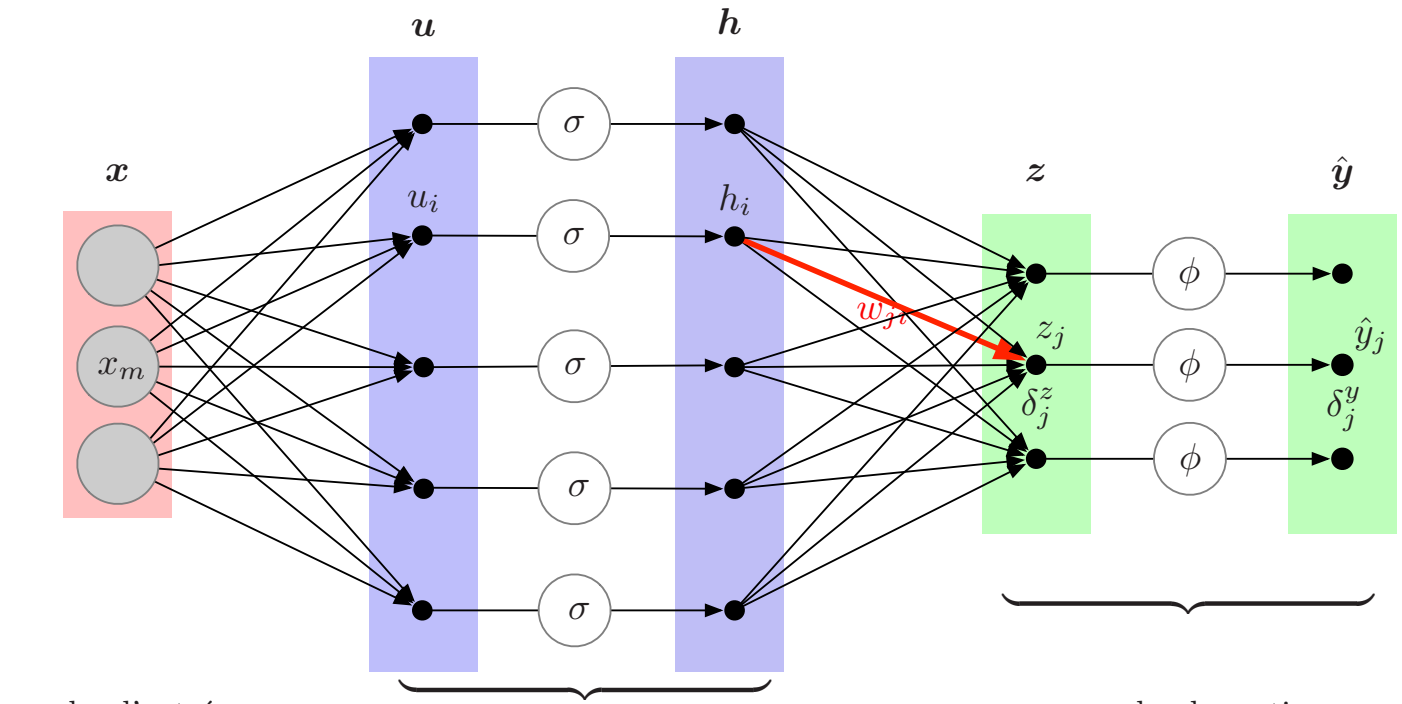
$$W^{(k+1)} = W^{(k)} - \frac{\mu_e}{|\mathcal{B}_k|} \sum_{n \in \mathcal{B}_k} (\hat{\boldsymbol{y}}_n - \boldsymbol{y}_n)\, \boldsymbol{x}_n^\top$$

fin

fin

Réseaux de neurones artificiels

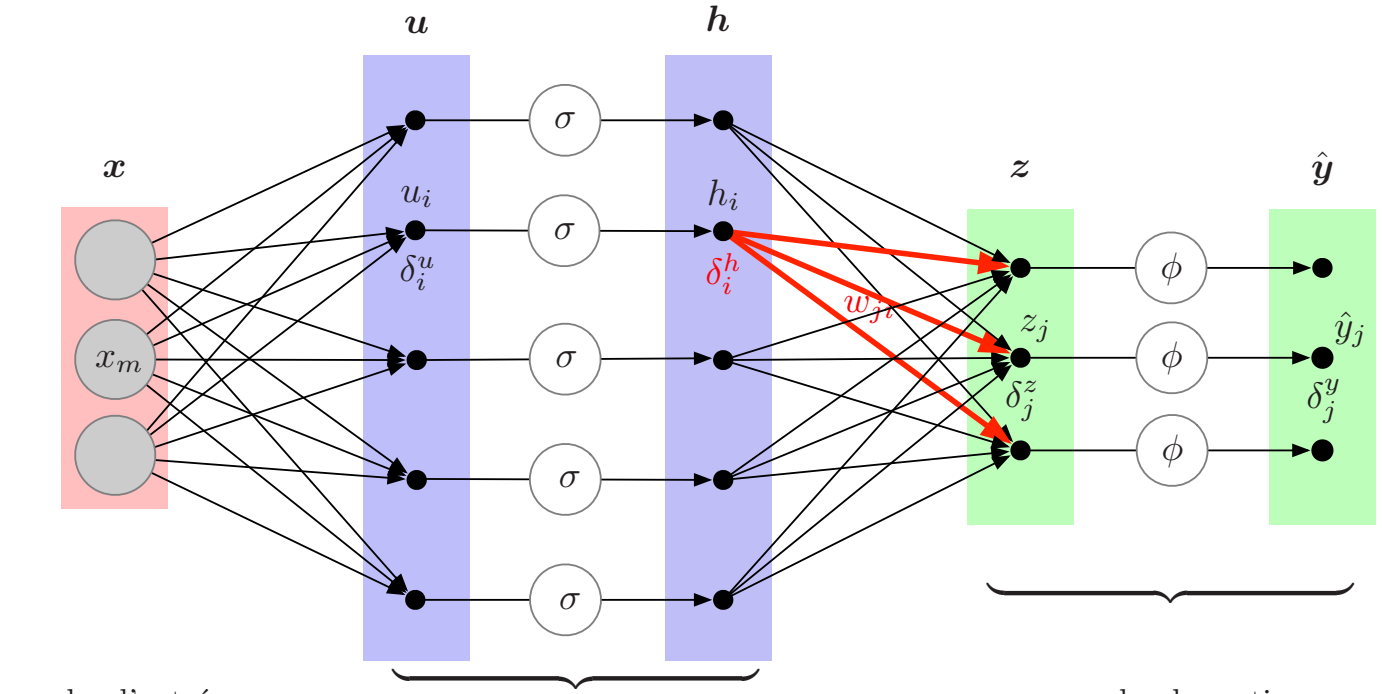
Calcul du gradient : cas d'un réseau à 2 couches



$$\frac{\partial J}{\partial w_{ji}} = \delta_j^z \cdot h_i$$

Réseaux de neurones artificiels

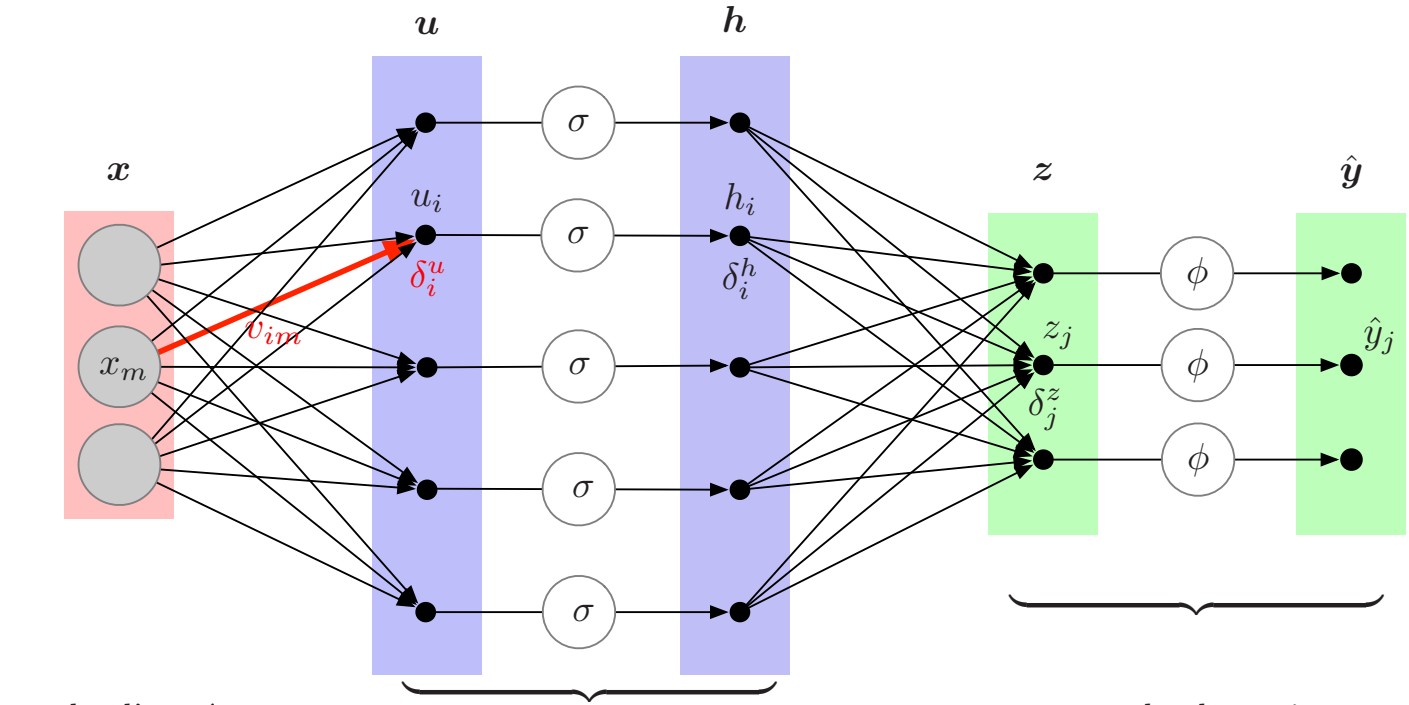
Calcul du gradient : cas d'un réseau à 2 couches



$$\frac{\partial J}{\partial h_i} = \sum_j \frac{\partial J}{\partial z_j} \cdot \frac{z_j}{\partial h_i} = \sum_j \delta_j^z \cdot w_{ji} \triangleq \delta_i^h$$

Réseaux de neurones artificiels

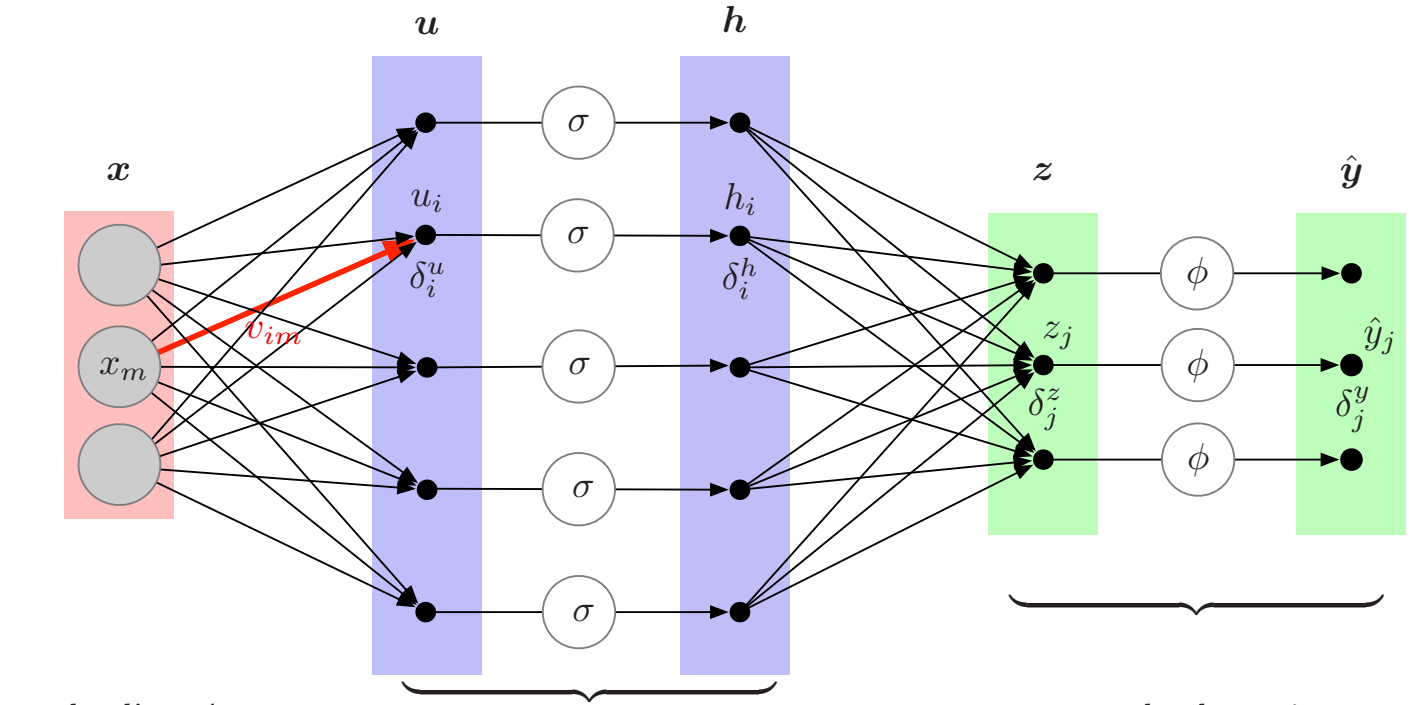
Calcul du gradient : cas d'un réseau à 2 couches



$$\frac{\partial J}{\partial v_{im}} = \frac{\partial J}{\partial h_i} \cdot \frac{\partial h_i}{\partial u_i} \cdot \frac{\partial u_i}{\partial v_{im}} = \delta_i^h \cdot \frac{\partial h_i}{\partial u_i} \cdot \frac{\partial u_i}{\partial v_{im}} = \delta_i^u \cdot \frac{\partial u_i}{\partial v_{im}}$$

Réseaux de neurones artificiels

Calcul du gradient : cas d'un réseau à 2 couches



$$\frac{\partial J}{\partial v_{im}} = \delta_i^u \cdot \frac{\partial u_i}{\partial v_{im}} = \delta_i^u \cdot x_m$$

Réseaux de neurones artificiels

Calcul du gradient : cas d'un réseau à 2 couches

Les résultats précédents permettent d'écrire le gradient recherché par rapport aux poids V de la couche cachée sous forme matricielle :

$$\boxed{\begin{array}{l}\nabla_V J = \boldsymbol{\delta}^u \boldsymbol{x}^\top \\ \\ \boldsymbol{\delta}^u = \sigma'(\boldsymbol{u}) \odot \dfrac{\partial J}{\partial \boldsymbol{h}}\end{array}}$$

Les 2 expressions ci-dessus sont identiques à celles d'un réseau à 1 couche

Seul le calcul de $\frac{\partial J}{\partial \boldsymbol{h}}$ change, en faisant intervenir le couplage avec la couche suivante via $\boldsymbol{\delta}^z$ ainsi

$$\boxed{\frac{\partial J}{\partial \boldsymbol{h}} \triangleq \boldsymbol{\delta}^h = U^\top \boldsymbol{\delta}^z}$$

En observant que $\boldsymbol{u}$ tient le même rôle que $\boldsymbol{z}$, et que la sortie $\boldsymbol{h}$ tient le rôle d'entrée pour la couche suivante, on est en mesure de généraliser l'algorithme à L couches

Calcul du gradient pour chaque couche $W_{\ell=1,\dots,L}$ **en** $(\boldsymbol{x}_n, \boldsymbol{y}_n)$

calculer la sortie estimée $\hat{\boldsymbol{y}}_n$ au point $(\boldsymbol{x}_n, \boldsymbol{y}_n)$ souhaité;

$\boldsymbol{\delta}_L^x = \dfrac{\partial J}{\partial \hat{\boldsymbol{y}}}$;

pour $\ell = L$ **à** 1 **faire**

$$\boldsymbol{\delta}_\ell^z = \sigma'(\boldsymbol{z}_\ell) \odot \boldsymbol{\delta}_\ell^x$$

$$\boldsymbol{\delta}_{\ell-1}^x = W_\ell^\top \boldsymbol{\delta}_\ell^z$$

$$\nabla_{W_\ell} J = \boldsymbol{\delta}_\ell^z \, \boldsymbol{x}_{\ell-1}^\top$$

fin

Remarque : changer $\sigma(\cdot)$ par $\phi(\cdot)$ pour la couche $\ell = L$

Réseaux de neurones artificiels

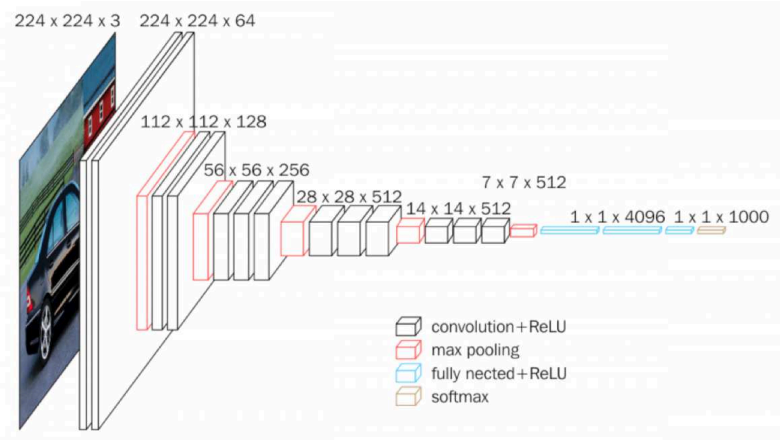
Dérivées usuelles

nom	**fonction**	**dérivée**
sigmoïde	$\sigma(z) = \frac{1}{1+\exp(-z)}$	$\sigma(z) \cdot (1 - \sigma(z))$
tanh	$\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$	$\frac{1}{\cosh^2(z)}$
ReLU	$\mathrm{ReLU}(z) = \max(0, z)$	1 si $z > 0$, sinon 0

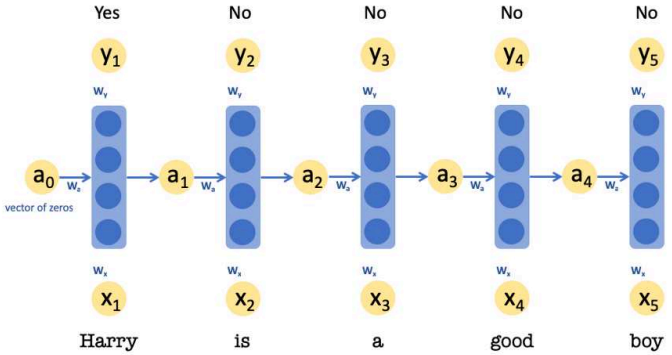
Réseaux de neurones artificiels

Entrées et/ou sorties structurées

données 2D : réseaux convolutifs



séries : réseaux récurrents



Constat

Pour une image RGB de taille $256 \times 256 \times 3$, chaque neurone de la première couche cachée connecté à tous les pixels comporterait $2 \cdot 10^5$ poids synaptiques à estimer.

Pour une couche cachée comportant autant de neurones que de pixels, le nombre total de poids s'élève à : $4 \cdot 10^{10}$

Élements de solution

- ▷ Prise en compte la corrélation spatiale des images
- ▷ Invariance en translation

Réseaux de neurones artificiels

Réseaux de neurones convolutifs

CNN : convolutional neural network

MLP : multilayer perceptron (vu précédemment)

Traits distinctifs des CNN par rapport aux MLP

- ▷ Utilisation de neurones opérant localement pour en limiter le nombre
- ▷ Partage des poids des neurones (convolution)
- ▷ Invariance en translation par regroupement (pooling)

Réseaux de neurones artificiels

Couches constitutives

Une architecture de réseau de neurones convolutifs est formée par un empilement de couches de traitement :

- ▷ la couche de convolution (CONV)
- ▷ la couche d'activation (ReLU)
- ▷ la couche de pooling (POOL)
- ▷ la couche entièrement connectée (FC), type perceptron
- ▷ la couche de sortie

Réseaux de neurones artificiels

Couche de convolution

Le volume de la couche de convolution est caractérisée par :

- ▷ le nombre de noyaux de convolution, qui définit le nombre *d'activation maps*
- ▷ la taille du champ récepteur de chaque noyau de convolution
- ▷ le pas de chevauchement des champs récepteurs
- ▷ la marge à 0, ou zero-padding, en frontière du volume d'entrée

Réseaux de neurones artificiels

Couche de convolution : noyau ou filtre

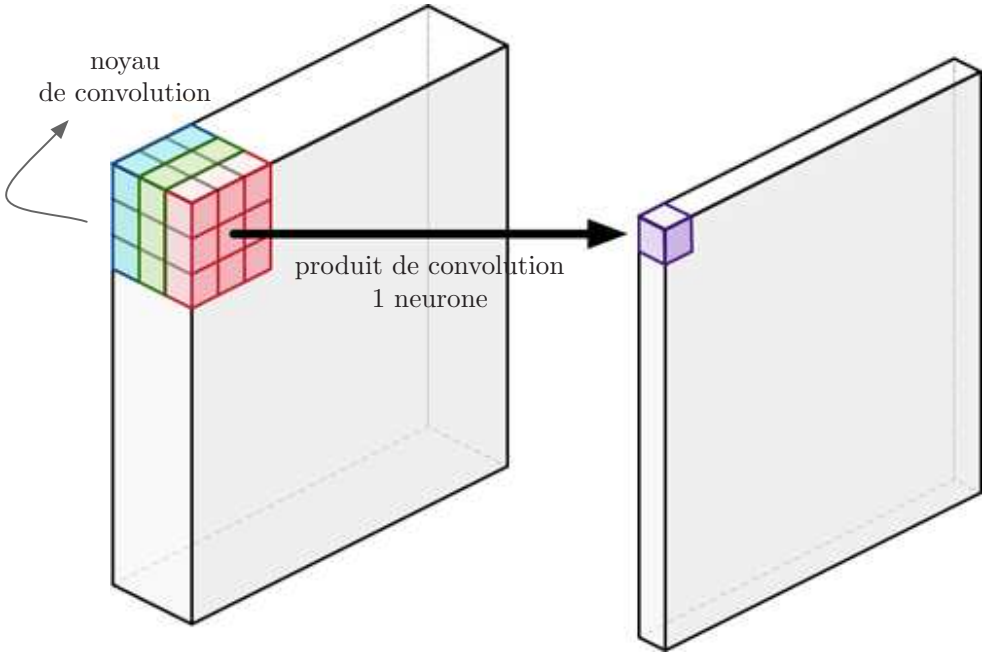
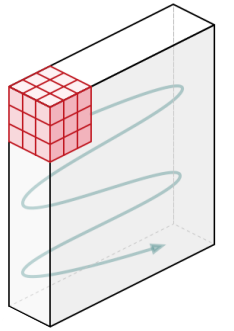


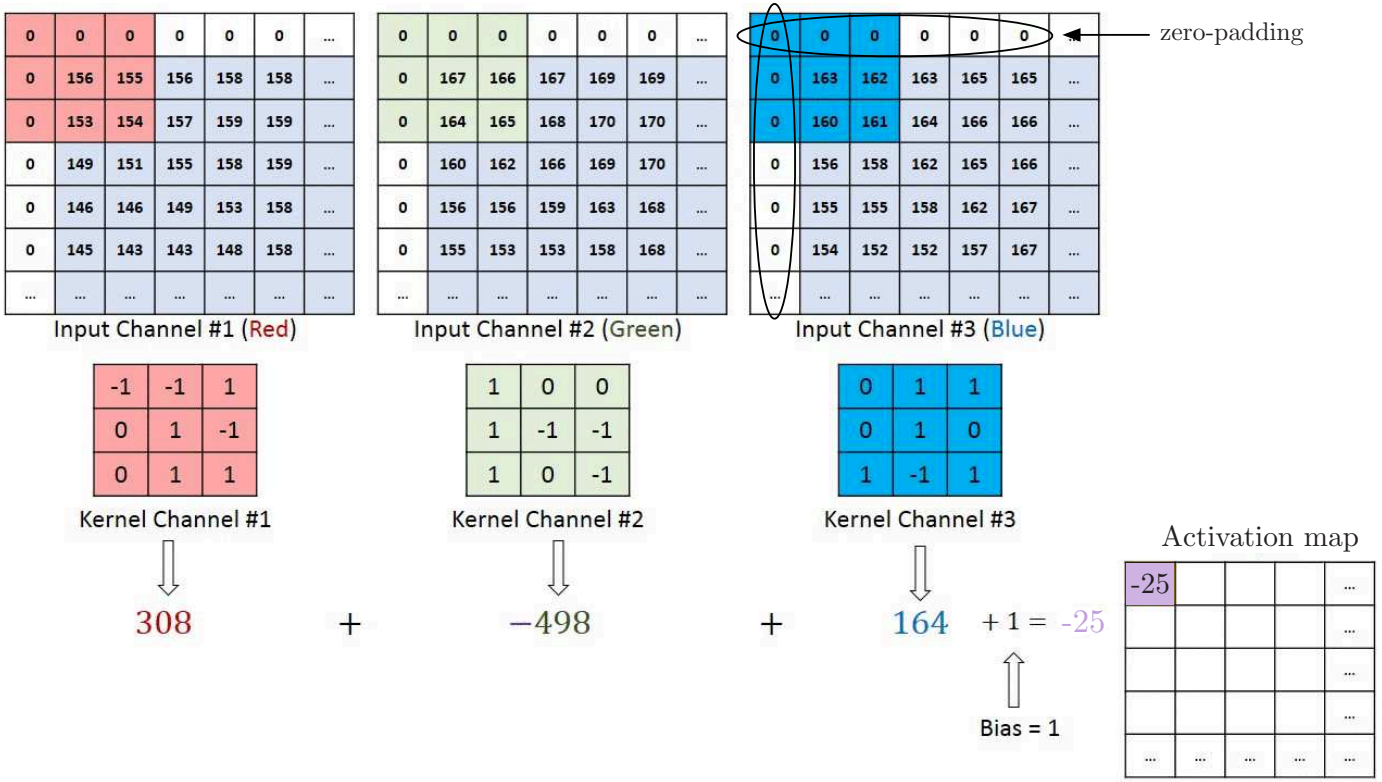
image d'entrée (RGB)
volume d'entrée

activation map
1 couche du volume de sortie



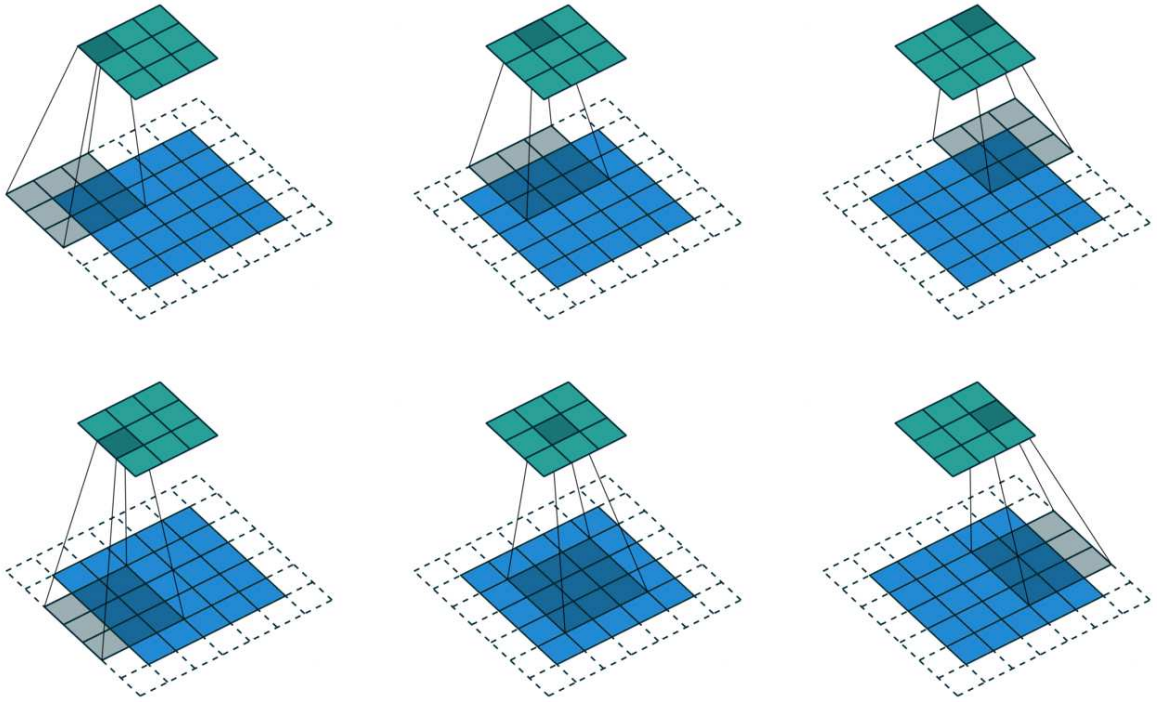
Réseaux de neurones artificiels

Couche de convolution : produit de convolution



Réseaux de neurones artificiels

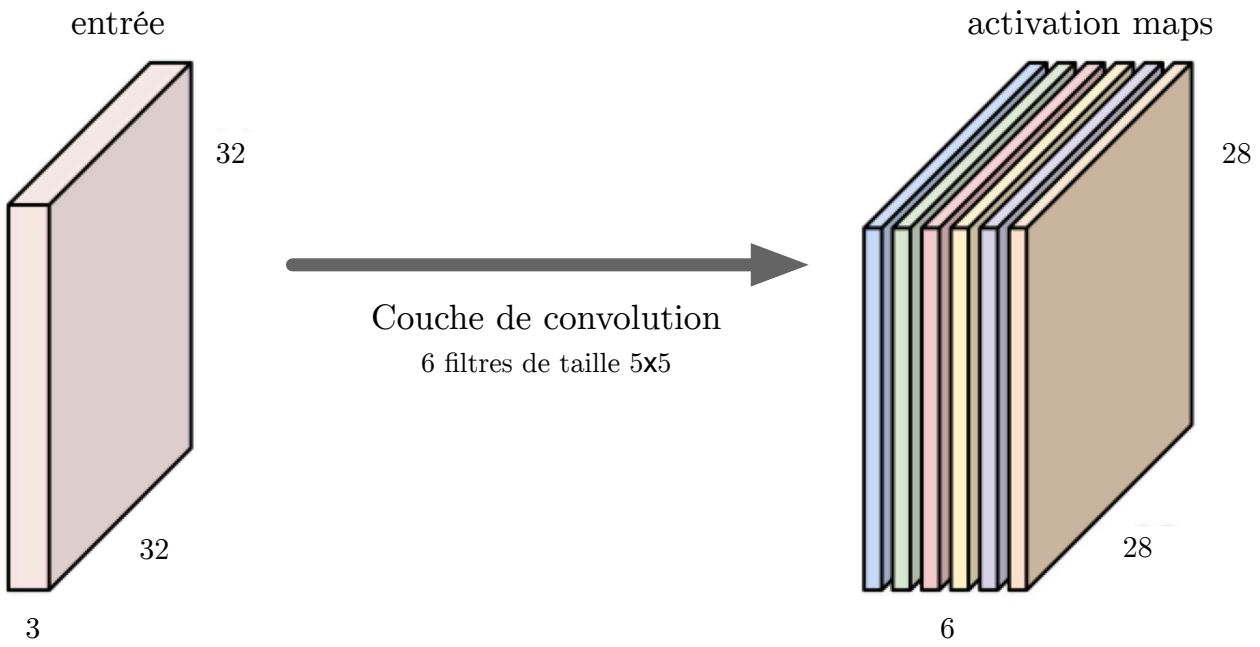
Couche de convolution : pas de chevauchement



pas de chevauchement = 1

Réseaux de neurones artificiels

Couche de convolution : synthèse



Réseaux de neurones artificiels

Couche de pooling

La couche de pooling (mise en commun) a pour objectif de :

- ▷ réduire la taille des représentations
- ▷ créer une forme d'invariance en translation

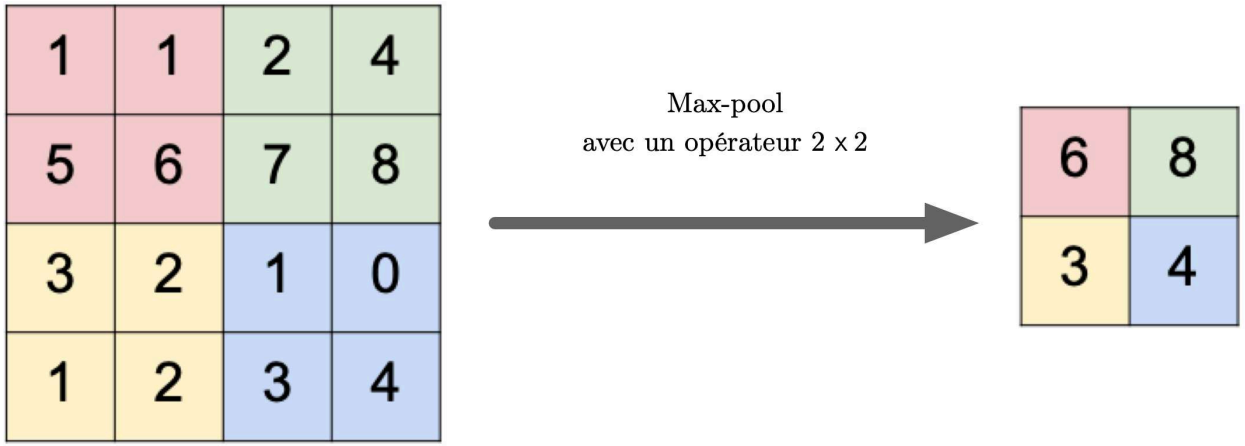
Généralement, l'opérateur de pooling repose sur :

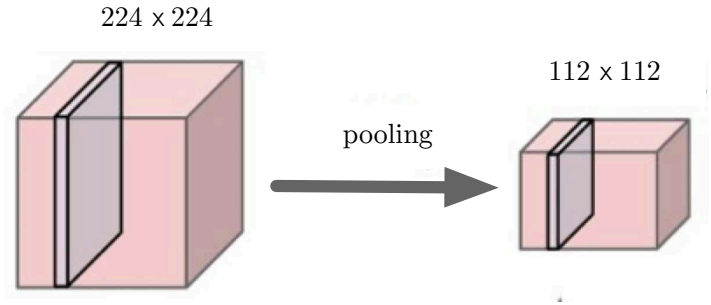
- ▷ un masque de taille 2×2 ou 4×4
- ▷ une fonction max (max-pool), parfois sur la moyenne (average-pool)

L'opérateur de pooling opère indépendamment sur chaque *activation map*

Réseaux de neurones artificiels

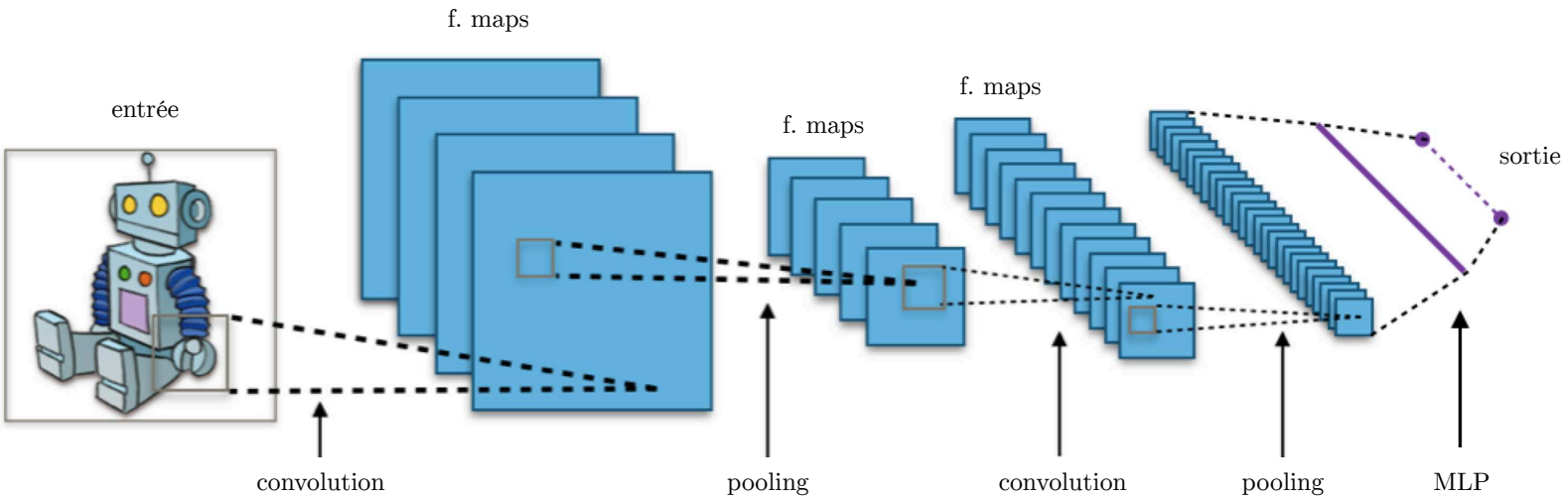
Couche de pooling





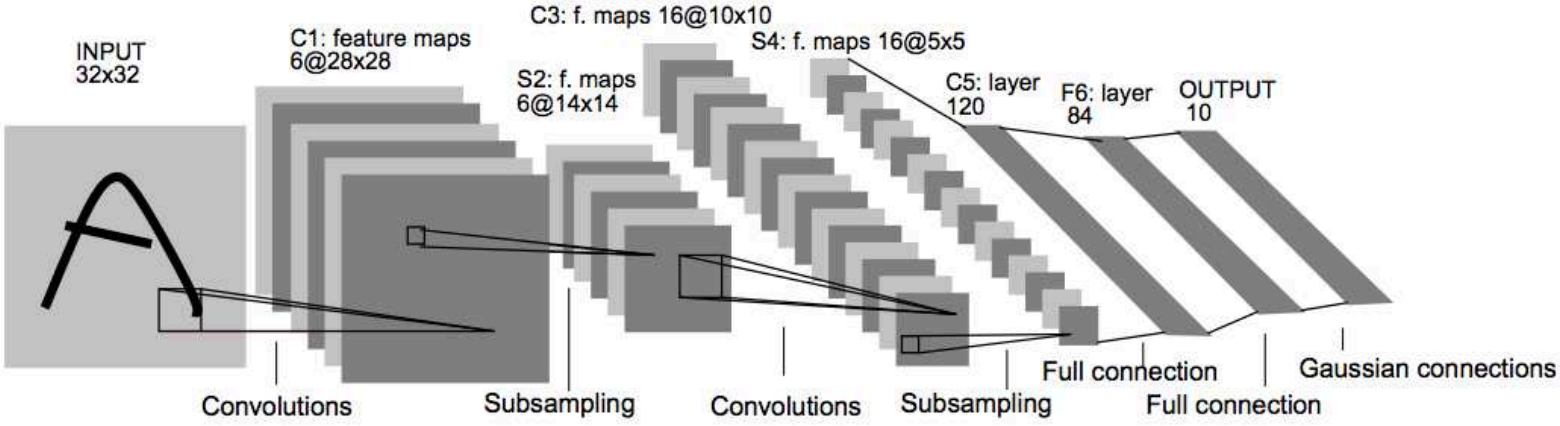
Réseaux de neurones artificiels

Réseaux de neurones convolutifs : récapitulatif



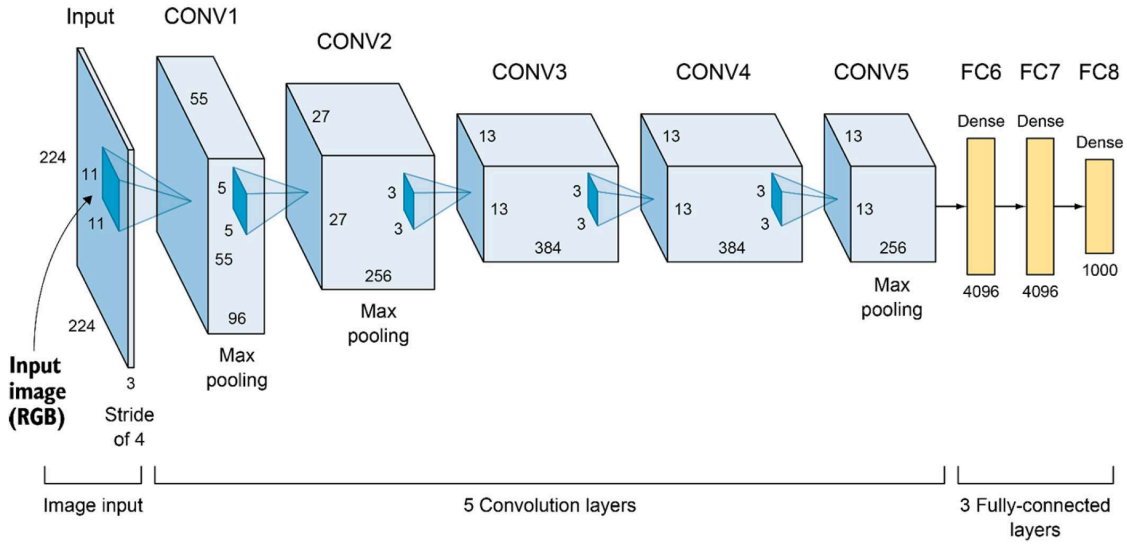
Réseaux de neurones artificiels

Réseaux de neurones convolutifs : LeNet5 (Le Cun, 1998)



Réseaux de neurones artificiels

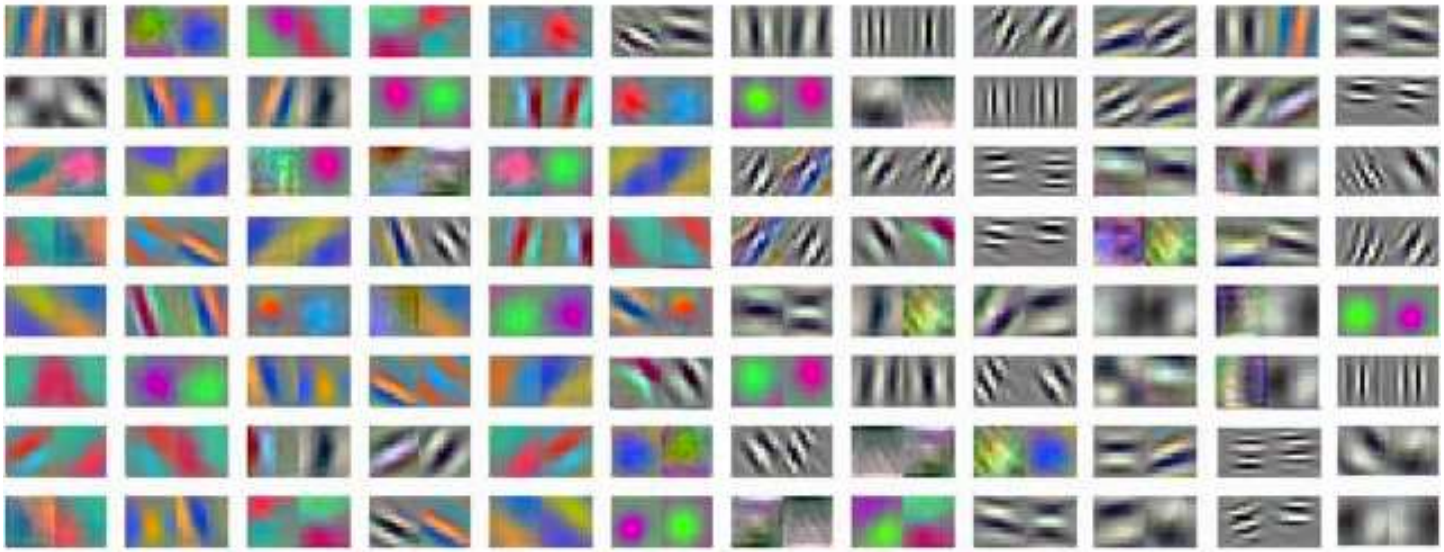
Réseaux de neurones convolutifs : AlexNet (Krizhevsky *et al.*, 2012)



2012 :
1.2MM images (app.)
50M images (val.)
150M images (test)
1M catégories

Réseaux de neurones artificiels

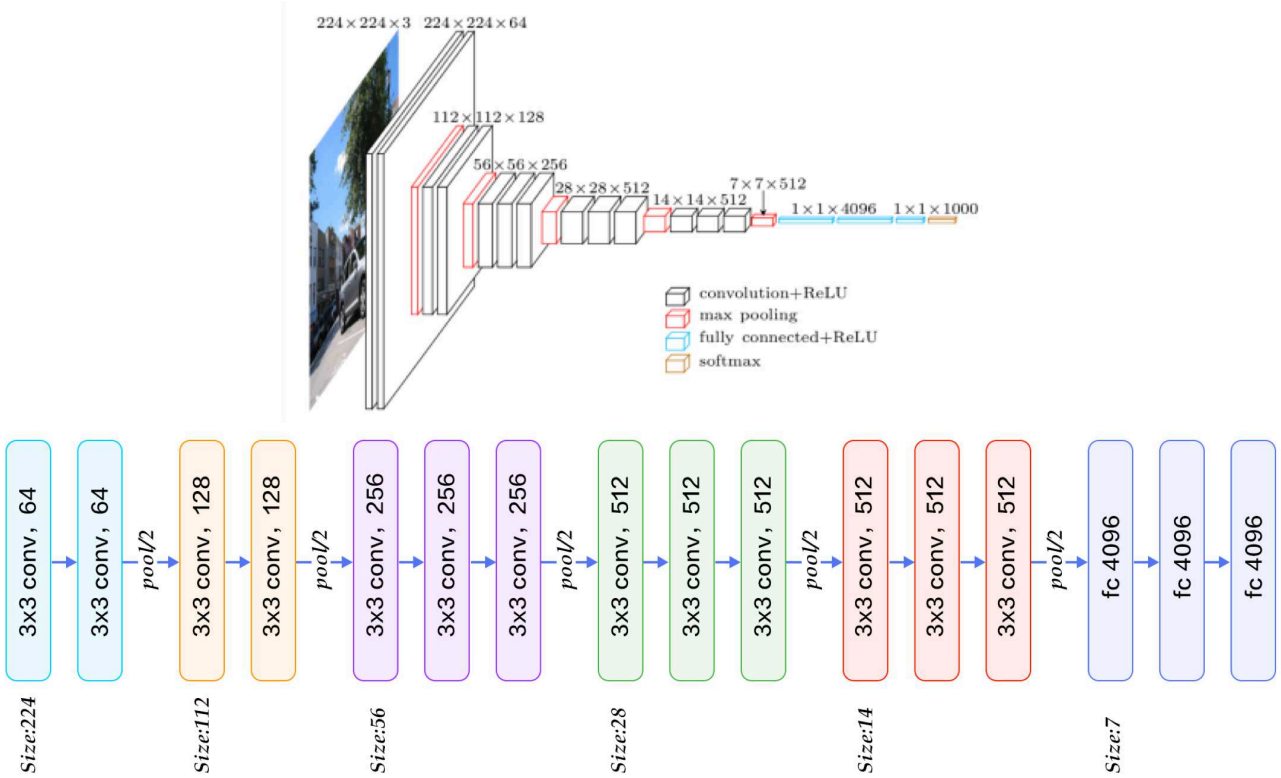
Réseaux de neurones convolutifs : AlexNet (Krizhevsky *et al.*, 2012)



AlexNet Conv1 filters

Réseaux de neurones artificiels

Réseaux de neurones convolutifs : VGG (Simonyan *et al.*, 2014)



Réseaux de neurones artificiels

Réseaux de neurones convolutifs : ImageNet