

Deep Learning

Stéphanie Bricq
stephanie.bricq@u-bourgogne.fr

Université de Bourgogne

Plan

- 1 Introduction
- 2 Réseaux de neurones artificiels
- 3 Entraînement de Réseaux de neurones profonds
 - Pb de disparition et d'explosion des gradients
 - Réutiliser des couches préentraînées
 - Optimiseurs plus rapides
 - Régularisation
- 4 Réseaux de neurones convolutifs
 - Définition des couches
 - Architectures
- 5 Réseaux de neurones récurrents
- 6 Autres méthodes

Bibliographie

- [Cornuéjols18] Apprentissage artificiel : deep learning, concepts et algorithmes, A. Cornuéjols, L. Miclet, V. Barra, Paris, Eyrolles, 2018
- [Géron17] Hands-On Machine Learning with Scikit-Learn and TensorFlow : Concepts, Tools, and Techniques to Build Intelligent Systems, A. Géron, O'Reilly Media, Inc, USA, 2017
- [Goodfellow16] Deep Learning, I. Goodfellow, Y. Bengio and A. Courville, MIT Press, 2016

Apprentissage profond

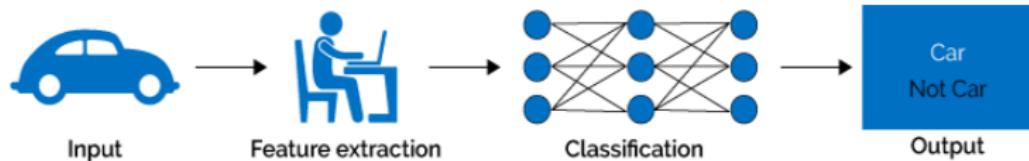
- peut être défini comme le développement d'algorithmes d'apprentissage automatique permettant d'apprendre de multiples niveaux de représentation et/ou d'abstraction de données

Apprentissage profond

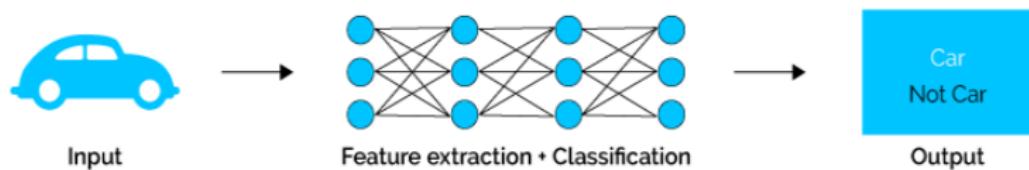
- peut être défini comme le développement d'algorithmes d'apprentissage automatique permettant d'apprendre de multiples niveaux de représentation et/ou d'abstraction de données
- nb tâches d'IA :
 - reconnaissance de la parole
 - traduction automatique
 - reconnaissance et suivi d'objets
 - ...
- améliore de manière sensible les résultats

Deep Learning | Machine Learning

Machine Learning



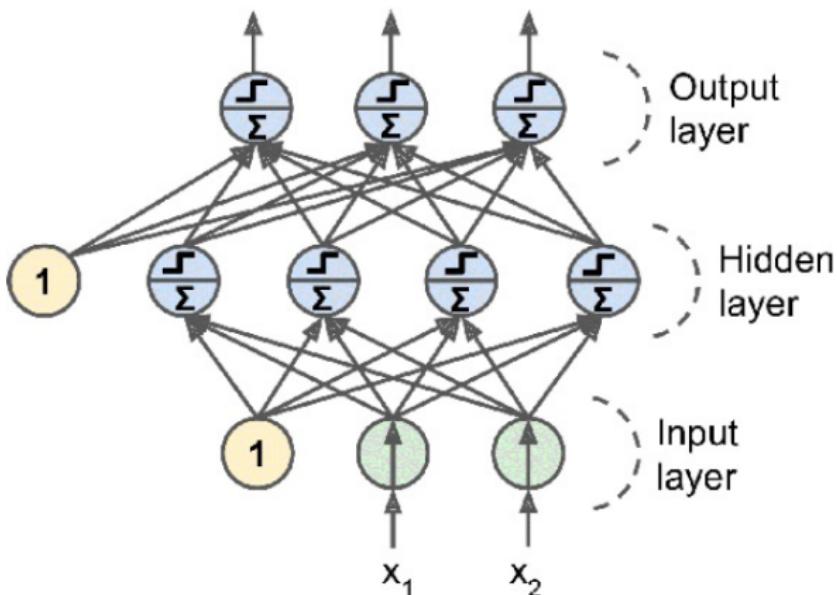
Deep Learning



Plan

- 1 Introduction
- 2 Réseaux de neurones artificiels
- 3 Entraînement de Réseaux de neurones profonds
 - Pb de disparition et d'explosion des gradients
 - Réutiliser des couches préentraînées
 - Optimiseurs plus rapides
 - Régularisation
- 4 Réseaux de neurones convolutifs
 - Définition des couches
 - Architectures
- 5 Réseaux de neurones récurrents
- 6 Autres méthodes

Rappel : Perceptron multicouche



PMC avec ReLU et softmax pour la classification [Géron17]

- Réseau de neurones artificiel avec 2 couches cachées ou plus \Rightarrow réseau de neurones profond (RNP) (Deep Neural Network (DNN))

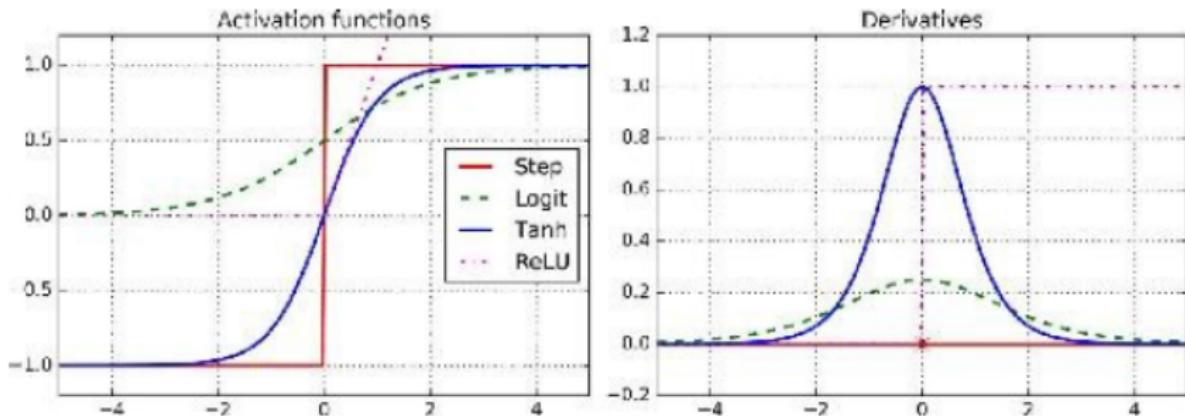
Algorithme d'entraînement : rétropropagation

- pour chaque instance d'entraînement,
 - l'algorithme commence par effectuer une prédiction (passe vers l'avant)
 - mesure l'erreur
 - traverse chaque couche en arrière pour mesurer la contribution à l'erreur de chaque connexion (passe vers l'arrière)
 - et termine en ajustant les poids de connexions pour réduire l'erreur (descente de gradient)

Fonctions d'activation

- Fonctions d'activation classiques

- fonction échelon remplacée par les suivantes
- fonction logistique $\sigma(z) = 1/(1 + \exp(-z))$
- fonction tangente hyperbolique $\tanh(z) = 2\sigma(2z) - 1$
- fonction ReLU : $\text{ReLU}(z) = \max(0, z)$

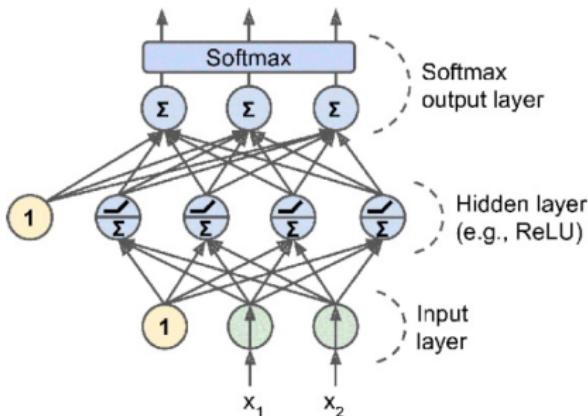


PMC

- PMC souvent employé pour la classification
 - cas classes exclusives, couche de sortie modifiée en remplaçant fonctions d'activation individuelles par 1 fonction partagée **softmax**
 - sortie de chaque neurone correspond à la proba estimée de la classe correspondante

PMC

- PMC souvent employé pour la classification
 - cas classes exclusives, couche de sortie modifiée en remplaçant fonctions d'activation individuelles par 1 fonction partagée **softmax**
 - sortie de chaque neurone correspond à la proba estimée de la classe correspondante
- Ex réseau de neurones non bouclé (*Feedforward Neural Network*)



PMC avec ReLU et softmax pour la classification [Géron17]

Fonction Softmax

- score softmax pour la classe k

$$s_k(x) = (\theta^{(k)})^T \cdot x$$

Fonction Softmax

- score softmax pour la classe k

$$s_k(x) = (\theta^{(k)})^T \cdot x$$

- fonction softmax

$$\hat{p}_k = \sigma(s(x))_k = \frac{\exp(s_k(x))}{\sum_{j=1}^K \exp(s_j(x))}$$

- K : nb de classes
- $s(x)$: vecteur contenant les scores de chaque classe pour l'observation x
- $\sigma(s(x))_k$: probabilité estimée que l'observation x appartienne à la classe k compte tenu des scores de chaque classe pour cette observation

Fonction Softmax

- score softmax pour la classe k

$$s_k(x) = (\theta^{(k)})^T \cdot x$$

- fonction softmax

$$\hat{p}_k = \sigma(s(x))_k = \frac{\exp(s_k(x))}{\sum_{j=1}^K \exp(s_j(x))}$$

- K : nb de classes
- $s(x)$: vecteur contenant les scores de chaque classe pour l'observation x
- $\sigma(s(x))_k$: probabilité estimée que l'observation x appartienne à la classe k compte tenu des scores de chaque classe pour cette observation
- prédiction du classifieur de régression softmax

$$\hat{y} = \operatorname{argmax}_k \sigma(s(x))_k = \operatorname{argmax}_k ((\theta^{(k)})^T \cdot x)$$

Choix de la fonction d'activation

- dans la plupart des cas, fonction d'activation ReLU (ou une variante) dans les couches cachées

Choix de la fonction d'activation

- dans la plupart des cas, fonction d'activation ReLU (ou une variante) dans les couches cachées
- pour la couche de sortie :
 - fonction d'activation softmax pour les tâches de classification quand les classes sont mutuellement exclusives

Choix de la fonction d'activation

- dans la plupart des cas, fonction d'activation ReLU (ou une variante) dans les couches cachées
- pour la couche de sortie :
 - fonction d'activation softmax pour les tâches de classification quand les classes sont mutuellement exclusives
 - fonction d'activation logistique quand elles ne le sont pas ou quand il n'y a que 2 classes

Choix de la fonction d'activation

- dans la plupart des cas, fonction d'activation ReLU (ou une variante) dans les couches cachées
- pour la couche de sortie :
 - fonction d'activation softmax pour les tâches de classification quand les classes sont mutuellement exclusives
 - fonction d'activation logistique quand elles ne le sont pas ou quand il n'y a que 2 classes
 - pour des tâches de régression, on peut ne choisir aucune fonction d'activation pour la couche de sortie

Réglage des hyperparamètres

- de nombreux hyperparamètres à ajuster
 - topologie du réseau (interconnexion des neurones)
 - nb de couches, nb neurones par couche
 - type de fonction d'activation, ...

Réglage des hyperparamètres

- de nombreux hyperparamètres à ajuster
 - topologie du réseau (interconnexion des neurones)
 - nb de couches, nb neurones par couche
 - type de fonction d'activation, ...
- comment connaître la combinaison d'hyperparamètres la mieux adaptée à une tâche ?
 - choix de qq valeurs pour chaque hyperparamètre, puis test de toutes les combinaisons possibles ⇒ recherche par quadrillage (*grid search*)
 - préférable d'utiliser une recherche aléatoire (*random search*)

Plan

- 1 Introduction
- 2 Réseaux de neurones artificiels
- 3 Entraînement de Réseaux de neurones profonds
 - Pb de disparition et d'explosion des gradients
 - Réutiliser des couches préentraînées
 - Optimiseurs plus rapides
 - Régularisation
- 4 Réseaux de neurones convolutifs
 - Définition des couches
 - Architectures
- 5 Réseaux de neurones récurrents
- 6 Autres méthodes

Réseau de neurones profond

- algo de rétropropagation
 - passe avant
 - puis passe arrière qui propage le gradient d'erreur de la couche de sortie vers la couche d'entrée

Réseau de neurones profond

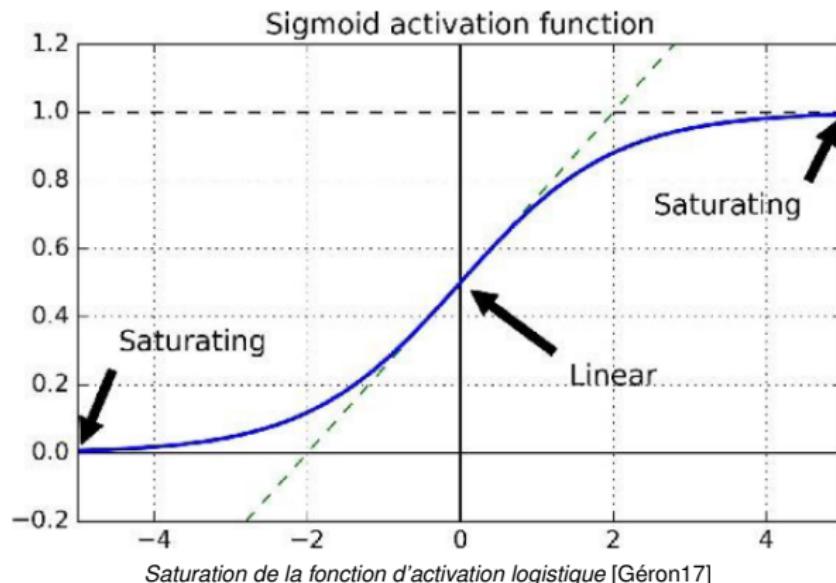
- algo de rétropropagation
 - passe avant
 - puis passe arrière qui propage le gradient d'erreur de la couche de sortie vers la couche d'entrée
 - quand l'algorithme progresse vers les couches inférieures, les gradients deviennent souvent de plus en petits
- ⇒ pb de disparition des gradients (*vanishing gradients*)

Réseau de neurones profond

- algo de rétropropagation
 - passe avant
 - puis passe arrière qui propage le gradient d'erreur de la couche de sortie vers la couche d'entrée
 - quand l'algorithme progresse vers les couches inférieures, les gradients deviennent souvent de plus en petits
- ⇒ pb de disparition des gradients (*vanishing gradients*)
- opposé peut se produire dans certains cas ⇒ pb d'explosion des gradients (*exploding gradients* (dans les réseaux de neurones récurrents))

Pb de disparition et d'explosion des gradients

- fonction d'activation logistique



Fonctions d'activation non saturantes

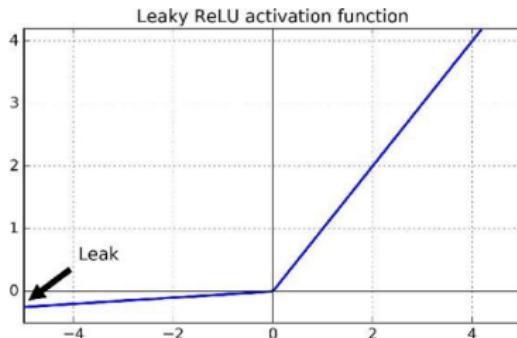
- pb de disparition/explosion des gradients en partie dus à un mauvais choix de la fonction d'activation

Fonctions d'activation non saturantes

- pb de disparition/explosion des gradients en partie dus à un mauvais choix de la fonction d'activation
- d'autres fonctions d'activation à la place de la sigmoïde \Rightarrow ReLU :
 - ne sature pas pour les valeurs positives
 - pb de mort des ReLU : certains neurones "meurent" au cours de l'entraînement

Fonctions d'activation non saturantes

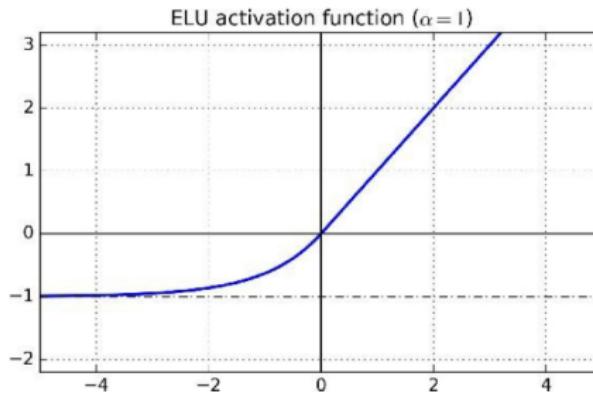
- pb de disparition/explosion des gradients en partie dus à un mauvais choix de la fonction d'activation
- d'autres fonctions d'activation à la place de la sigmoïde \Rightarrow ReLU :
 - ne sature pas pour les valeurs positives
 - pb de mort des ReLU : certains neurones "meurent" au cours de l'entraînement
- variante de ReLU : leaky ReLU $\text{LeakyReLU}_\alpha(z) = \max(\alpha z, z)$
 α : hyperparamètre définissant le niveau de "fuite" de la fonction, en général α fixé à 0.01



Fonctions d'activation non saturantes

- Autre variante : fonction d'activation ELU (*Exponential Linear Unit*)

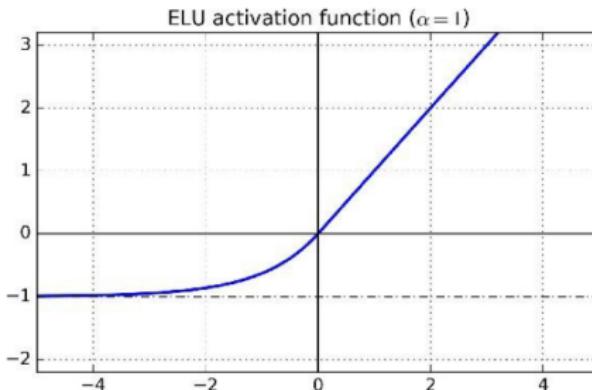
$$\text{ELU}_\alpha(z) = \begin{cases} \alpha(\exp(z) - 1) & \text{si } z < 0 \\ z & \text{si } z \geq 0 \end{cases}$$



Fonctions d'activation non saturantes

- Autre variante : fonction d'activation ELU (*Exponential Linear Unit*)

$$\text{ELU}_\alpha(z) = \begin{cases} \alpha(\exp(z) - 1) & \text{si } z < 0 \\ z & \text{si } z \geq 0 \end{cases}$$



- fonction d'activation SELU (*Scaled ELU*)

$$\text{SELU}(z) = 1.0507 \times \text{ELU}_{1.67326}(z)$$

Normalisation par lots

- *batch normalization (BN)* : pour traiter les pb de gradients
- technique :
 - ajout une opération dans le modèle qui centre sur 0 et normalise les entrées
 - puis mise à l'échelle et décalage du résultat en utilisant 2 nouveaux paramètres d'entrée

Normalisation par lots

- *batch normalization (BN)* : pour traiter les pb de gradients
- technique :
 - ajout une opération dans le modèle qui centre sur 0 et normalise les entrées
 - puis mise à l'échelle et décalage du résultat en utilisant 2 nouveaux paramètres d'entrée
- en général : 1 couche BN à la sortie de chaque couche du réseau de neurones , souvent juste avant sa fonction d'activation
- la couche BN a besoin d'évaluer la moyenne et l'écart-type de ces entrées
- valeurs déterminées sur le mini-lot courant (petit sous-ensemble d'observations sélectionnées aléatoirement)

Normalisation par lots : Algorithme

$$\begin{aligned}\mu_B &= \frac{1}{m_B} \sum_{i=1}^{m_B} x^{(i)} & \sigma_B^2 &= \frac{1}{m_B} \sum_{i=1}^{m_B} (x^{(i)} - \mu_B)^2 \\ \hat{x}^{(i)} &= \frac{x^{(i)} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} & z^{(i)} &= \gamma \hat{x}^{(i)} + \beta\end{aligned}$$

- μ_B, σ_B : moyenne et écart-type empiriques
- m_B : nb d'instances dans le mini-lot
- $x^{(i)}$: entrée de la couche BN considérée
- $\hat{x}^{(i)}$: entrée centrée sur 0 et normalisée
- γ : paramètre de mise à l'échelle pour la couche
- β : paramètre de décalage pour la couche
- ϵ : terme de lissage (en général 10^{-3})
- $z^{(i)}$: sortie de l'opération de normalisation

Ecrêtage de gradients

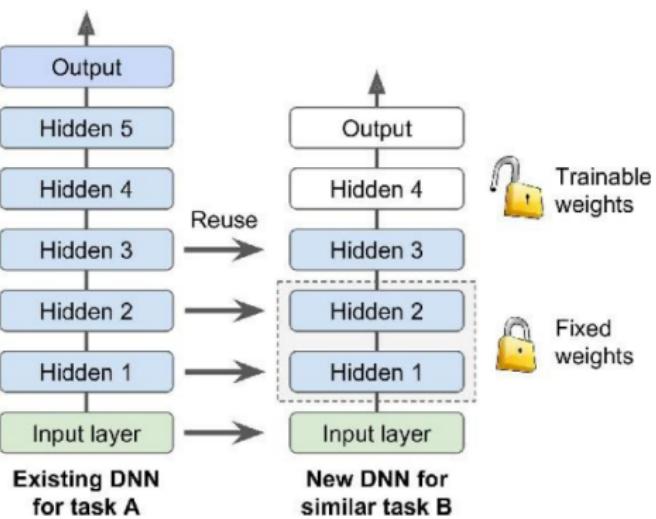
- technique pour minimiser le problème de l'explosion des gradients
⇒ écrêtage de gradient (*gradient clipping*)
- écrêter les gradients pendant la rétropropagation pour qu'ils ne dépassent jamais un certain seuil

Plan

- 1 Introduction
- 2 Réseaux de neurones artificiels
- 3 Entraînement de Réseaux de neurones profonds
 - Pb de disparition et d'explosion des gradients
 - Réutiliser des couches préentraînées
 - Optimiseurs plus rapides
 - Régularisation
- 4 Réseaux de neurones convolutifs
 - Définition des couches
 - Architectures
- 5 Réseaux de neurones récurrents
- 6 Autres méthodes

Transfer Learning

- Réutilisation de couches pré-entraînées
⇒ transfert d'apprentissage (*transfer learning*)
 - accélère l'entraînement
 - permet d'obtenir de bonnes performances avec des jeux de données d'entraînement assez petits



Réutilisation de couches pré-entraînées [Géron17]

Optimiseurs

- Quand le RNP est très grand, l'entraînement peut être très lent
- Pour accélérer l'entraînement
 - bonne stratégie d'initialisation des poids de connexion
 - choix d'une bonne fonction d'activation
 - normalisation par lots
 - réutilisation d'un réseau pré-entraîné

Optimiseurs

- Quand le RNP est très grand, l'entraînement peut être très lent
- Pour accélérer l'entraînement
 - bonne stratégie d'initialisation des poids de connexion
 - choix d'une bonne fonction d'activation
 - normalisation par lots
 - réutilisation d'un réseau pré-entraîné
- Autre piste : utiliser un optimiseur + rapide que l'optimiseur de descente de gradient ordinaire
 - optimisation d'inertie
 - gradient accéléré de Nesterov
 - AdaGrad
 - RMSProp
 - optimisation Adam

Régularisation

- enjeu : construire des algos ayant une bonne capacité de généralisation
- régularisation
- nombreuses méthodes
 - régularisation de la fonction de coût
 - dropout
 - partage de paramètres

Techniques de régularisation

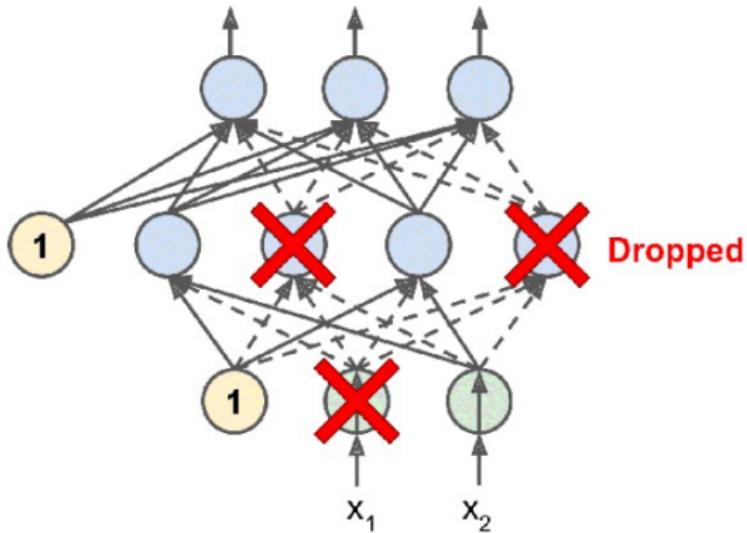
- arrêt prématué :
 - interrompre l'entraînement lorsque ses performances sur le jeu de validation commencent à baisser

Techniques de régularisation

- arrêt prématué :
 - interrompre l'entraînement lorsque ses performances sur le jeu de validation commencent à baisser
- régularisation l_1 , l_2
 - pour contraindre les poids des connexions d'un réseau de neurones
 - régression Lasso : ajout d'un terme de régularisation égal à $\alpha \sum_{i=1}^n \theta_i^2$ à la fonction de coût
 - régression ridge : ajout d'un terme de régularisation égal à $\alpha \sum_{i=1}^n |\theta_i|$ à la fonction de coût

Techniques de régularisation

- Dropout : technique la plus répandue



Régularisation par dropout [Géron2017]

- hyperparamètre p : taux d'extinction (*dropout rate* (souvent 50%))
- après l'entraînement, il faut multiplier les poids des connexions d'entrée par la probabilité de conservation (*keep probability*) $1 - p$

Techniques de régularisation

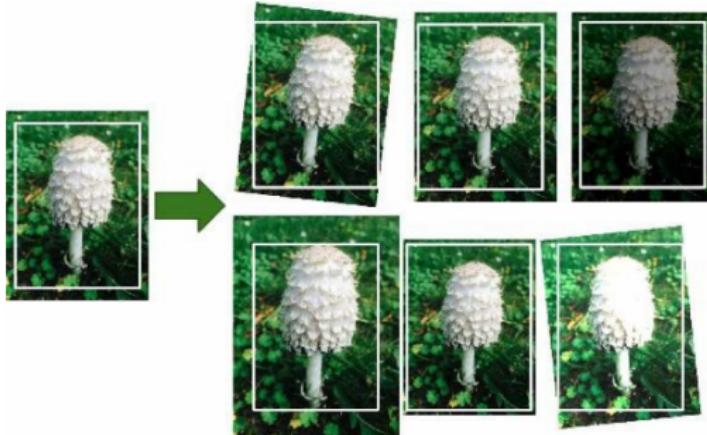
- inconvénient de Dropout : perturbe les propriétés d'auto-normalisation de la fonction d'activation SELU

Techniques de régularisation

- inconvénient de Dropout : perturbe les propriétés d'auto-normalisation de la fonction d'activation SELU
- ⇒ variante : *alpha dropout* compatible avec SELU
- comme dropout : choix aléatoire des neurones à éteindre à chaque itération
 - diffère sur la façon de calculer la sortie des neurones éteints et allumés de façon à préserver la moyenne et la variance des sorties de chaque couche du réseau de neurones pendant l'entraînement

Techniques de régularisation

- Augmentation de données : utilisation des instances d'entraînement existantes pour en générer de nouvelles pour grossir artificiellement la taille du jeu d'entraînement
- permet de réduire le surajustement
- génération d'instances réalistes



Génération de nouvelles instances d'entraînement à partir de celles existantes [Géron2017]

Plan

- 1 Introduction
- 2 Réseaux de neurones artificiels
- 3 Entraînement de Réseaux de neurones profonds
 - Pb de disparition et d'explosion des gradients
 - Réutiliser des couches préentraînées
 - Optimiseurs plus rapides
 - Régularisation
- 4 Réseaux de neurones convolutifs
 - Définition des couches
 - Architectures
- 5 Réseaux de neurones récurrents
- 6 Autres méthodes

Réseaux convolutifs CNN

CNN : Convolutional Neural Network

- un type de réseau de neurones artificiels acyclique à propagation avant
- empilage multicouche de perceptrons dont le but est de prétraiter de petites quantités d'informations
- apparus suite à l'étude du cortex visuel du cerveau et sont utilisés dans la reconnaissance d'images depuis les années 1980

Réseaux convolutifs CNN

CNN : Convolutional Neural Network

- un type de réseau de neurones artificiels acyclique à propagation avant
- empilage multicouche de perceptrons dont le but est de prétraiter de petites quantités d'informations
- apparus suite à l'étude du cortex visuel du cerveau et sont utilisés dans la reconnaissance d'images depuis les années 1980
- Applications
 - reconnaissance d'images et vidéos
 - systèmes de classification automatique de vidéos
 - reconnaissance vocale
 - traitement automatique du langage naturel

Architecture du cortex visuel

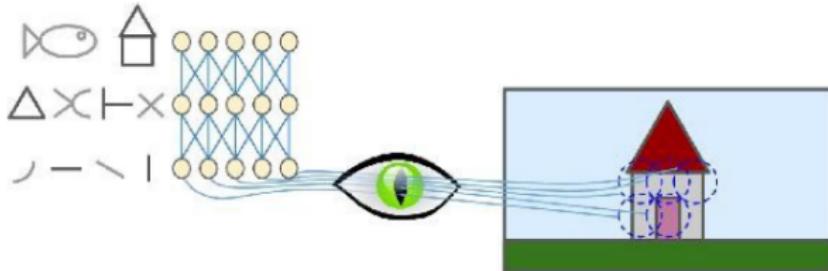
- de nombreux neurones du cortex visuel ont un petit champ récepteur local ⇒ réagissent uniquement à un stimulus visuel se trouvant dans une région limitée du champ visuel
- champs récepteurs des différents neurones peuvent se chevaucher : couvrent l'intégralité du champ visuel

Architecture du cortex visuel

- de nombreux neurones du cortex visuel ont un petit champ récepteur local ⇒ réagissent uniquement à un stimulus visuel se trouvant dans une région limitée du champ visuel
- champs récepteurs des différents neurones peuvent se chevaucher : couvrent l'intégralité du champ visuel
- certains neurones réagissent uniquement aux images de lignes horizontales, d'autres réagissent aux lignes ayant d'autres orientations

Architecture du cortex visuel

- de nombreux neurones du cortex visuel ont un petit champ récepteur local \Rightarrow réagissent uniquement à un stimulus visuel se trouvant dans une région limitée du champ visuel
- champs récepteurs des différents neurones peuvent se chevaucher : couvrent l'intégralité du champ visuel
- certains neurones réagissent uniquement aux images de lignes horizontales, d'autres réagissent aux lignes ayant d'autres orientations
- certaines neurones ont des champs récepteurs plus larges et réagissent à des motifs plus complexes, correspondant à des combinaisons de motifs de plus bas niveau

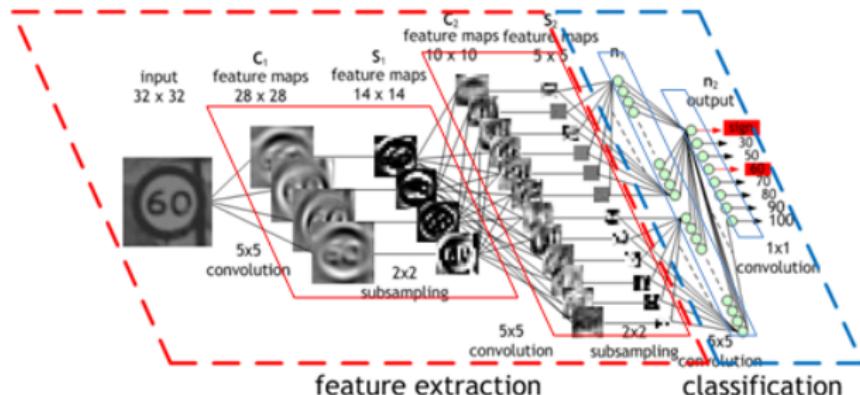


Champs récepteurs locaux dans le cortex visuel [Géron2017]

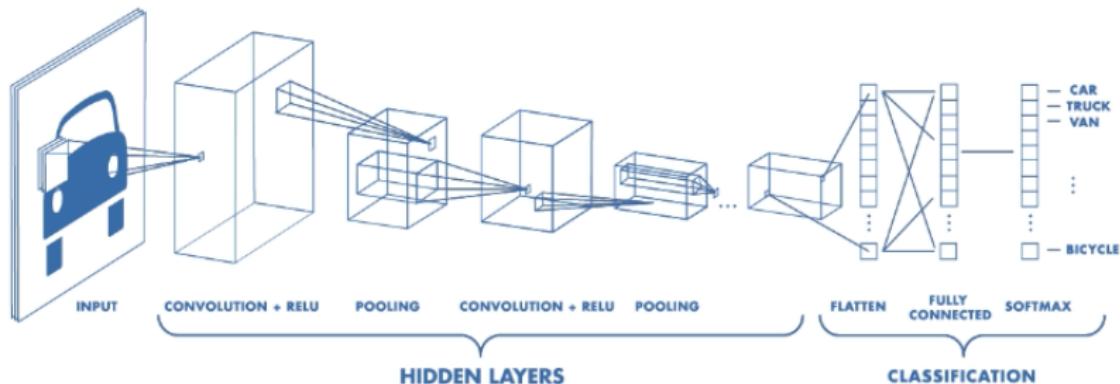
Réseaux convolutifs CNN

CNN : Convolutional Neural Network

- se compose de 2 types de neurones agencés en couches
- cas du traitement de données de type image
 - **neurones de traitement** qui traitent une portion limitée de l'image au travers d'une fonction de convolution
 - **neurones d'agrégation totale ou partielle (*pooling*)** de mise en commun des sorties



Réseaux convolutifs CNN



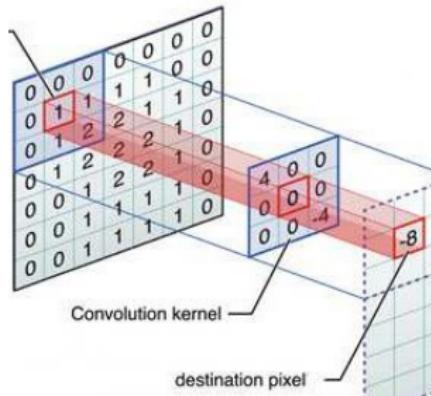
Source : <https://www.mathworks.com/>

Convolution discrète

- la convolution discrète de I par le filtre K est donnée par

$$(I * K)_{r,s} = \sum_{u=-h_1}^{h_1} \sum_{v=-h_2}^{h_2} K_{u,v} I_{r+u, s+v}$$

où K est donné par $K = \begin{pmatrix} K_{-h_1, -h_2} & \cdots & K_{-h_1, h_2} \\ \vdots & K_{0,0} & \vdots \\ K_{h_1, -h_2} & \cdots & K_{h_1, h_2} \end{pmatrix}$



Définition des couches

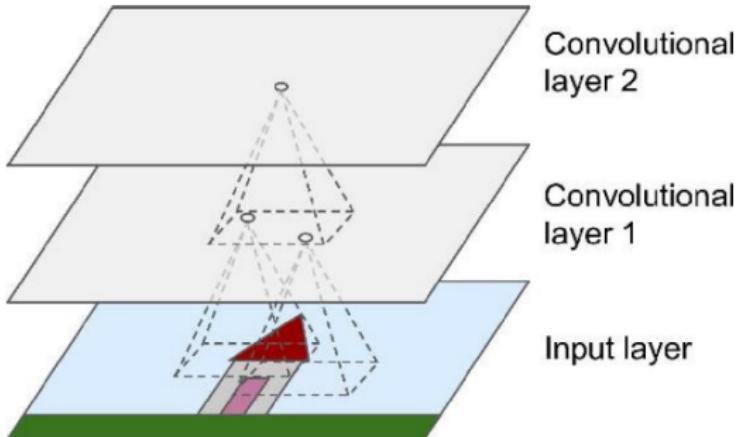
- différents types de couches utilisées dans les réseaux convolutifs
 - couche de convolution
 - couche non linéaire
 - couches de normalisation
 - couche d'agrégation et de sous-échantillonnage
 - couche complètement connectée

Définition des couches

- différents types de couches utilisées dans les réseaux convolutifs
 - couche de convolution
 - couche non linéaire
 - couches de normalisation
 - couche d'agrégation et de sous-échantillonnage
 - couche complètement connectée
- l'assemblage de ces couches permet de construire des architectures complexes pour la classification ou la régression

Couche de convolution

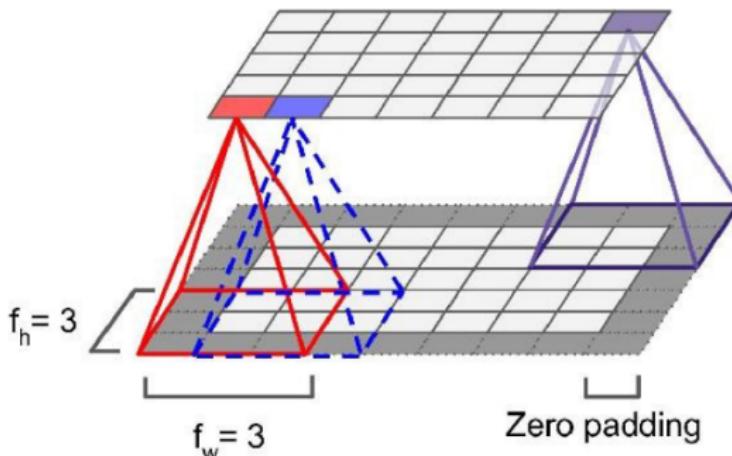
- permet au réseau de se focaliser sur des caractéristiques de bas niveau dans la 1ère couche cachée
- puis de les assembler en caractéristiques de plus haut niveau dans la couche cachée suivante



Couches d'un CNN avec des champs récepteurs locaux rectangulaires [Géron17]

Couche de convolution

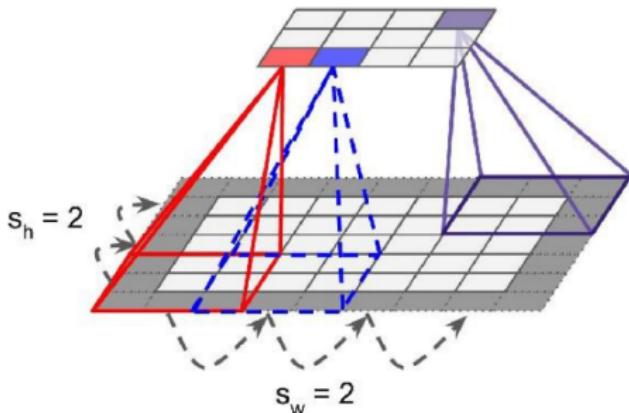
- neurone situé en (i, j) d'une couche donnée est connecté aux sorties des neurones de la couche précédente situés aux lignes i à $i + f_h - 1$ et aux colonnes j à $j + f_w - 1$
- pour qu'une couche ait la même hauteur et largeur que la couche précédente \Rightarrow zero-padding



Connexions entre les couches et marges de zéros [Géron17]

Couche de convolution

- possibilité de connecter une couche d'entrée large à une couche plus petite en espaçant les champs récepteurs
- espace = pas (*stride*) (valeur peut être \neq dans les 2 dims)
- neurone en (i, j) d'une couche : connecté aux sorties des neurones de la couche précédente situés aux lignes $i \times s_h$ à $i \times s_h + f_h - 1$ et aux colonnes $j \times s_w$ à $j \times s_w + f_w - 1$

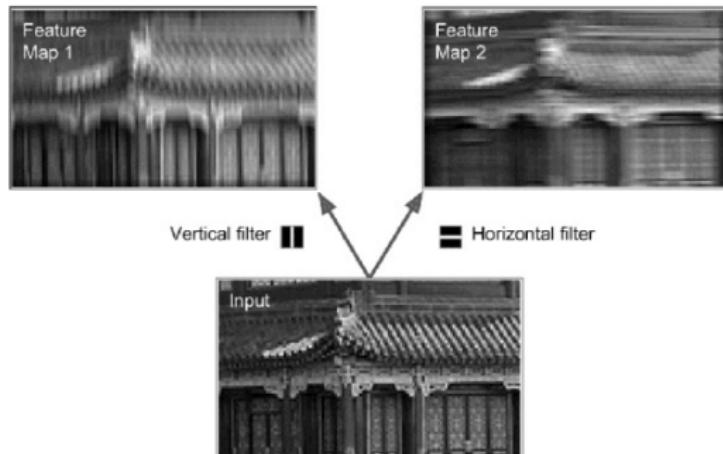


Diminution de la dimensionnalité grâce à un pas de 2 [Géron17]

Couche de convolution

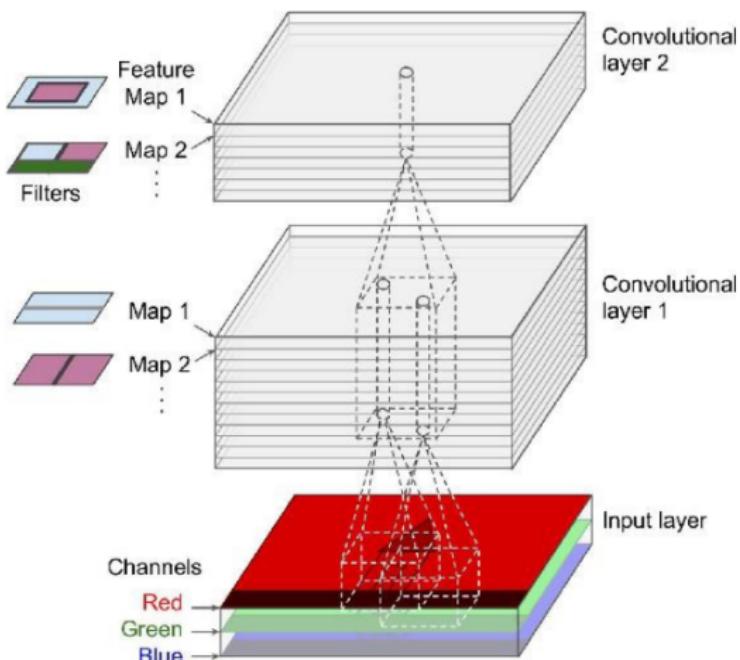
Filtre

- Filtres ou noyaux de convolution
- carte de caractéristiques



Application de 2 filtres différents pour obtenir 2 cartes de caractéristiques [Géron17]

Couche de convolution : Empiler plusieurs cartes de caractéristiques



Couches de convolution avec plusieurs cartes de caractéristiques et images à 3 canaux de couleur [Géron17]

Sortie d'un neurone dans une couche de convolution

$$z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_{n'}-1} x_{i',j',k'} \cdot w_{u,v,k',k}$$

avec

$$\begin{cases} i' &= i \times s_h + u \\ j' &= j \times s_w + v \end{cases}$$

- $z_{i,j,k}$: sortie du neurone situé en ligne i et colonne j dans la carte de caractéristiques k de la couche de convolution (couche l)
- s_h et s_w : pas vertical et horizontal
- f_h et f_w : hauteur et largeur du champ récepteur
- f_n : nb de caractéristiques de la couche précédente (couche $l - 1$)
- $x_{i',j',k'}$: sortie du neurone situé dans la couche $l - 1$, ligne i' , colonne j' , carte de caractéristiques (ou canal) k'
- b_k : terme constant de la carte de caractéristiques k (dans la couche l)
- $w_{u,v,k',k}$: poids de la connexion entre tout neurone de la carte k de la couche l et son entrée située ligne u , colonne v dans la carte k'

Couche de convolution

Besoins en mémoire

- les couches de convolution ont besoin d'une grande quantité de RAM

Couche de convolution

Besoins en mémoire

- les couches de convolution ont besoin d'une grande quantité de RAM
- si l'entraînement échoue à cause d'un manque de mémoire, plusieurs options
 - réduire la taille du mini-lot
 - diminuer la dimensionnalité en appliquant un pas plus important ou en retirant qq couches
 - passer à des nombres à virgule flottante sur 16 bits (au lieu de 32)
 - distribuer le CNN sur plusieurs processeurs

Couche de pooling

- sous-échantillonnage (*pooling*) des cartes obtenues par les couches précédentes pour réduire
 - la charge de calcul
 - l'utilisation de la mémoire
 - nb de paramètres

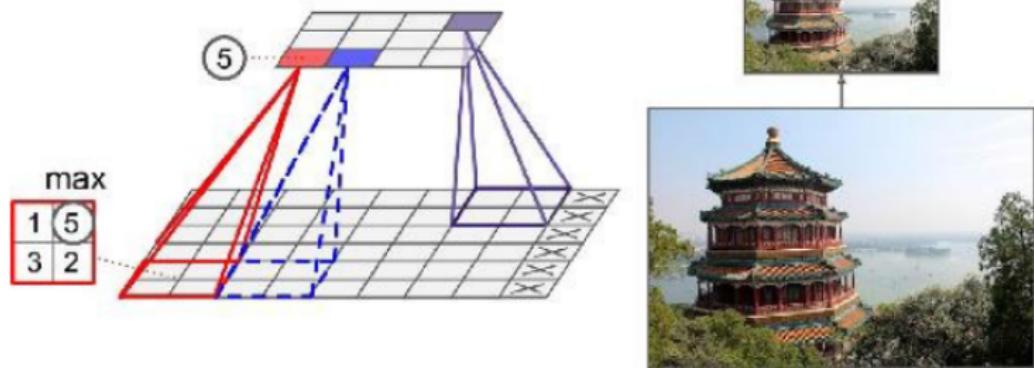
Couche de pooling

- sous-échantillonnage (*pooling*) des cartes obtenues par les couches précédentes pour réduire
 - la charge de calcul
 - l'utilisation de la mémoire
 - nb de paramètres
- on doit définir sa taille, le pas et le type de remplissage
- généralement 2 types d'agrégation
 - moyenne : on utilise un filtre K_B de taille $(2h_1 + 1) \times (2h_2 + 1)$ défini par

$$(K_B)_{r,s} = \frac{1}{(2h_1 + 1) \times (2h_2 + 1)}$$

- maximum : valeur max de la fenêtre est retenue

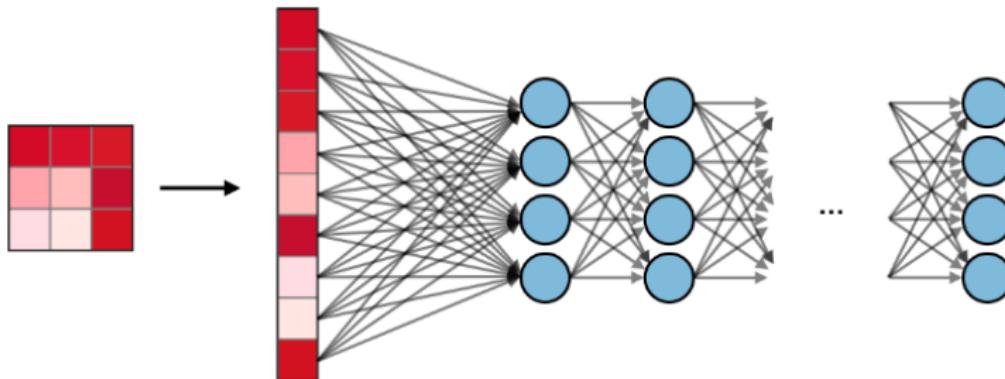
Couche de pooling



Couche de max-pooling [Geron17]

Couche complètement connectée

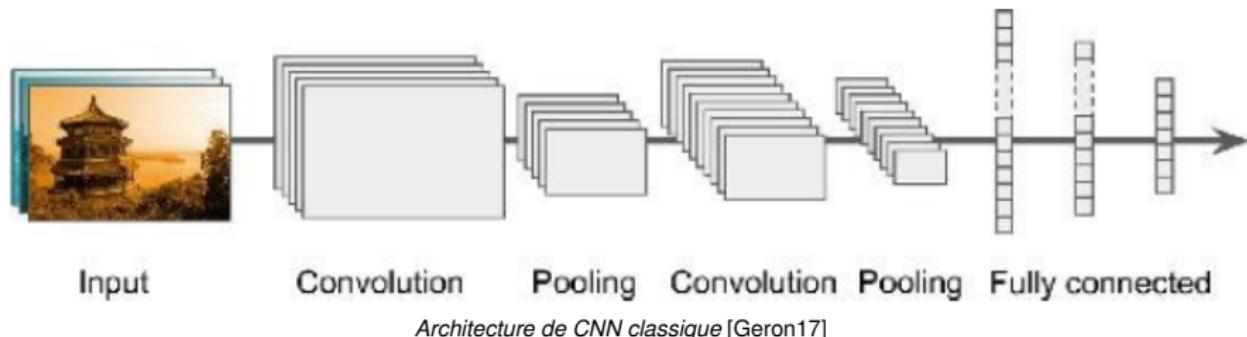
- *fully connected layer*
- la ou les couches complètement connectées sont utilisées en bout de réseau pour des tâches de classification
- neurones dans une couche entièrement connectée ont des connexions vers toutes les sorties de la couche précédente



Plan

- 1 Introduction
- 2 Réseaux de neurones artificiels
- 3 Entraînement de Réseaux de neurones profonds
 - Pb de disparition et d'explosion des gradients
 - Réutiliser des couches préentraînées
 - Optimiseurs plus rapides
 - Régularisation
- 4 Réseaux de neurones convolutifs
 - Définition des couches
 - Architectures
- 5 Réseaux de neurones récurrents
- 6 Autres méthodes

Architectures de CNN



- ex de réseaux ayant prouvé leur efficacité lors de compétitions sur une base de données ImageNet : plus de 14 millions d'images, classées en plus de 20 000 catégories (avions, voitures, chats, ...)
- compétition ILSVRC (ImageNet Large Scale Visual Recognition Challenge)

Architecture : LeNet

- 1er modèle de CNN proposé par LeCun et al [LeCun90]
- dédié à la classification de chiffres manuscrits
- composé d'une alternance de couches de convolution et de pooling, précédant des couches complètement connectées
- famille de 5 réseaux : LeNet à LeNet-5

Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully Connected	–	10	–	–	RBF
F6	Fully Connected	–	84	–	–	tanh
C5	Convolution	120	1×1	5×5	1	tanh
S4	Avg Pooling	16	5×5	2×2	2	tanh
C3	Convolution	16	10×10	5×5	1	tanh
S2	Avg Pooling	6	14×14	2×2	2	tanh
C1	Convolution	6	28×28	5×5	1	tanh
In	Input	1	32×32	–	–	–

LeNet-5 [Geron17]

Architecture : AlexNet

- a remporté le challenge ILSVRC en 2012
- développé par Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton
- ressemble à LeNet-5, mais + large et + profonde
- 1ère architecture à empiler des couches de convolution les unes au dessus des autres sans intercaler de couche de pooling

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully Connected	—	1,000	—	—	—	Softmax
F9	Fully Connected	—	4,096	—	—	—	ReLU
F8	Fully Connected	—	4,096	—	—	—	ReLU
C7	Convolution	256	13 × 13	3 × 3	1	SAME	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
S4	Max Pooling	256	13 × 13	3 × 3	2	VALID	—
C3	Convolution	256	27 × 27	5 × 5	1	SAME	ReLU
S2	Max Pooling	96	27 × 27	3 × 3	2	VALID	—
C1	Convolution	96	55 × 55	11 × 11	4	SAME	ReLU
In	Input	3 (RGB)	224 × 224	—	—	—	—

AlexNet [Geron17]

Architecture : VGG

- réseaux VGG (Visual Geometry Group, université d'Oxford)
- finaliste en 2014
- améliorations par rapport à AlexNet en remplaçant les grands filtres par plusieurs filtres de taille du noyau 3×3 , l'un après l'autre.
- différentes versions

Architecture : GoogLeNet

- gagnant 2014
- Christian Szegedy et al de Google Research
- réseau plus profond que les précédents
- présence de sous-réseaux nommés module Inception permettant à GoogLeNet d'utiliser les paramètres de manière plus efficace que dans les architectures précédentes

Architecture : ResNet

- gagnant 2015
- réseau résiduel *residual network* développé par Kaiming He et al.
- très profond : 152 couches
- utilise des connexions de saut (*skip connections*)
 - le signal fourni à une couche est également ajouté à la couche qui se trouve un peu plus haut dans la pile

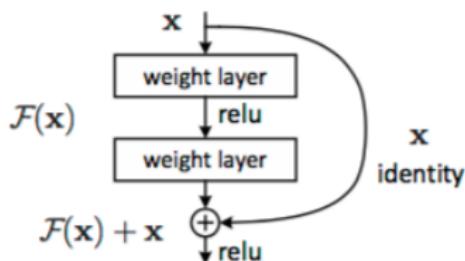


Figure Residual learning: a building block.

Plan

- 1 Introduction
- 2 Réseaux de neurones artificiels
- 3 Entraînement de Réseaux de neurones profonds
 - Pb de disparition et d'explosion des gradients
 - Réutiliser des couches préentraînées
 - Optimiseurs plus rapides
 - Régularisation
- 4 Réseaux de neurones convolutifs
 - Définition des couches
 - Architectures
- 5 Réseaux de neurones récurrents
- 6 Autres méthodes

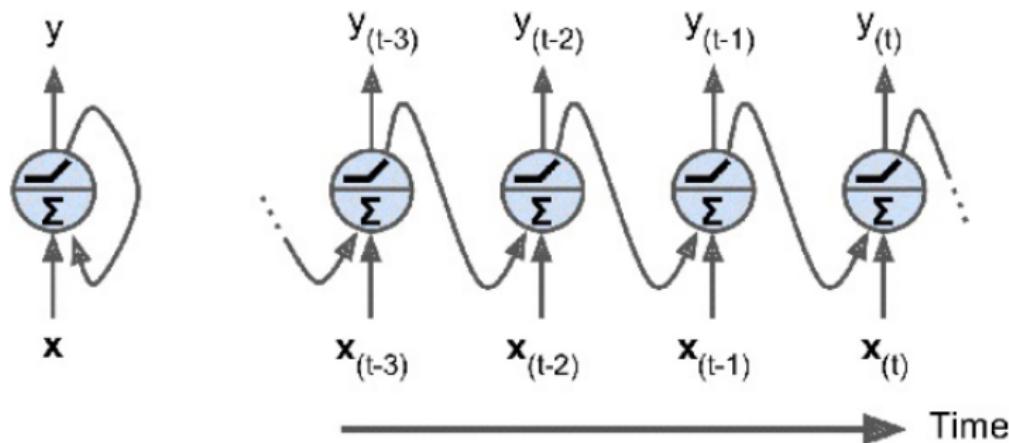
Réseaux récurrents RNN

RNN : Recurrent Neural Network

- capables d'analyser des séries chronologiques
- peuvent travailler sur des séries de longueur quelconque
- très utiles pour le traitement automatique du langage naturel : systèmes de traduction automatique, reconnaissance automatique de la parole, ...
- réseau de neurones à propagation avant sur lesquels une prise en compte du temps est introduite
- certaines connexions reviennent en arrière dans le réseau

Neurones récurrents

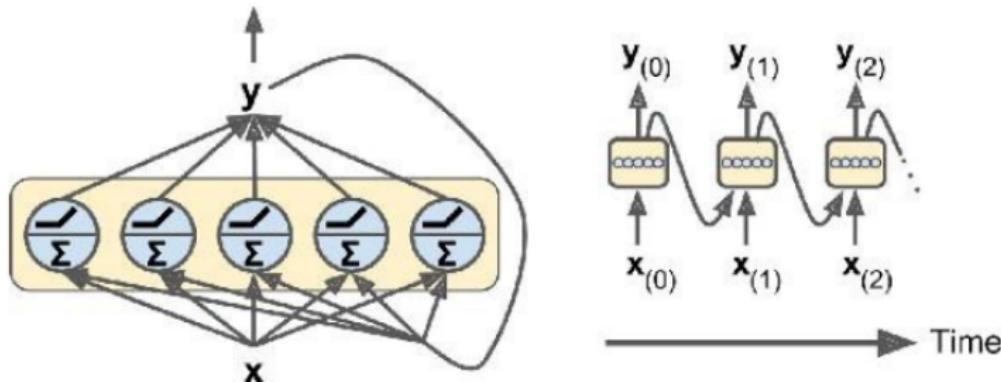
- cas le plus simple : un seul neurone qui reçoit des entrées, produit une sortie et se renvoie celle-ci
- A chaque étape temporelle t (trame), le neurone récurrent reçoit
 - le vecteur d'entrées $\mathbf{x}_{(t)}$
 - et sa propre sortie produite à l'étape temporelle précédente $y_{(t-1)}$



Un neurone récurrent (à gauche), déplié dans le temps (à droite) [Géron17]

Couche de neurones récurrents

- chaque neurone récurrent possède 2 jeux de poids :
 - w_x pour les entrées $x_{(t)}$
 - w_y pour les sorties de l'étape temporelle précédente $y_{(t-1)}$



Couche de neurones récurrents (à gauche), déplié dans le temps (à droite) [Géron17]

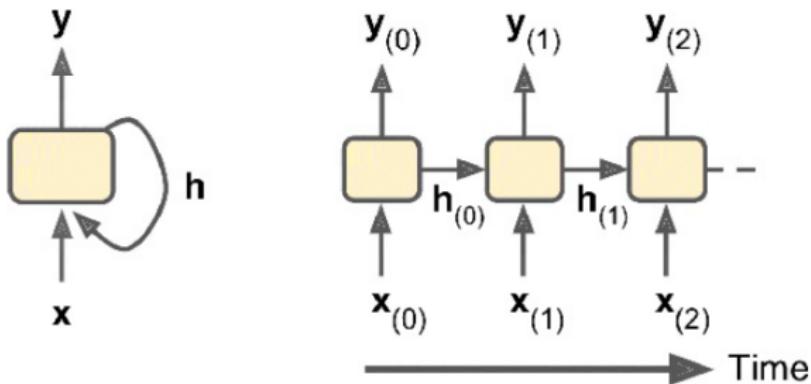
Couche de neurones récurrents

- sortie d'une couche de neurones récurrents à l'étape temporelle t pour une seule instance : $\mathbf{y}_{(t)} = \phi(\mathbf{W}_x^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_y^T \cdot \mathbf{y}_{(t-1)} + \mathbf{b})$
- sortie d'une couche de neurones récurrents pour toutes les instances d'un mini-lot : $\mathbf{Y}_{(t)} = \phi(\mathbf{X}_{(t)} \cdot \mathbf{W}_x + \mathbf{Y}_{(t-1)} \cdot \mathbf{W}_y + \mathbf{b})$
 - $\mathbf{Y}_{(t)}$: matrice $m \times n_{\text{neurones}}$ contenant les sorties de la couche à l'étape temporelle t pour chaque instance du mini-lot (m nb d'instance du mini-lot, n_{neurones} nb de neurones)
 - $\mathbf{X}_{(t)}$: matrice $m \times n_{\text{entrées}}$ contenant les entrées de toutes les instances ($n_{\text{entrées}}$ nb de caractéristiques d'entrée)
 - \mathbf{W}_x : matrice $n_{\text{entrées}} \times n_{\text{neurones}}$ contenant les poids des connexions des entrées pour l'étape temporelle courante
 - \mathbf{W}_y : matrice $n_{\text{neurones}} \times n_{\text{neurones}}$ contenant les poids des connexions des sorties pour l'étape temporelle précédente
 - \mathbf{b} vecteur de taille n_{neurones} contenant le terme constant de chaque neurone

Cellules de mémoire

- sortie d'un neurone récurrent à t est une fonction de toutes les entrées des étapes temporelles précédentes \Rightarrow mémoire
- cellule de mémoire (ou cellule) : partie d'un réseau de neurones qui conserve un état entre plusieurs étapes temporelles
- $\mathbf{h}_{(t)}$: état d'une cellule à l'étape t

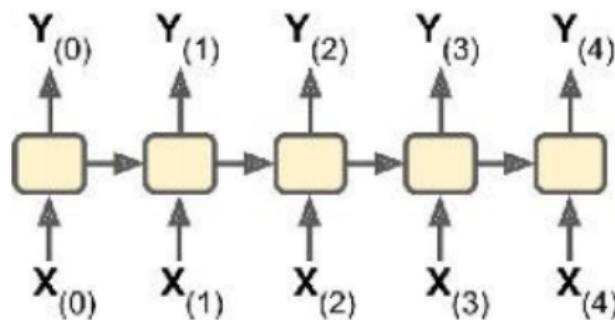
$$\mathbf{h}_{(t)} = f(\mathbf{h}_{(t-1)}, \mathbf{x}_{(t)})$$



l'état caché d'une cellule et sa sortie peuvent être différents [Géron17]

Séries en entrée et en sortie

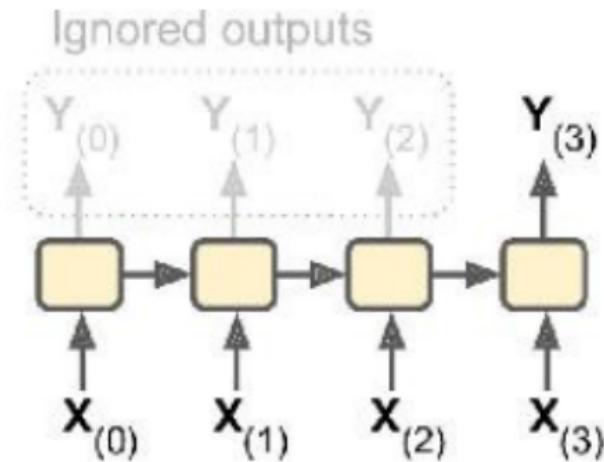
- RNN peut simultanément prendre une série d'entrées et produire une série de sorties
 - par ex pour prédire des séries chronologiques



Série vers série [Géron17]

Séries en entrée et en sortie

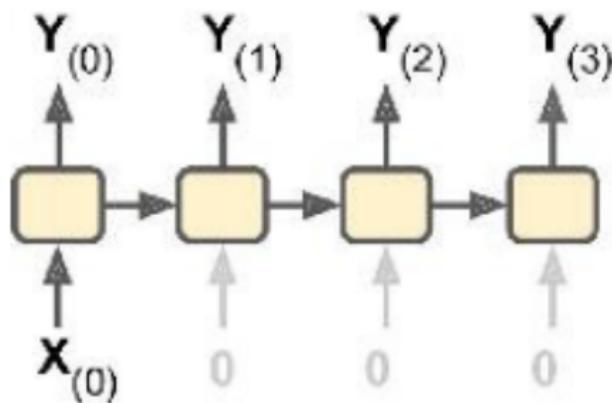
- réseau série vers vecteur : on fournit une série d'entrées et on ignore les sorties sauf la dernière
 - par ex : entrée suite de mots correspondant aux critiques d'un film
 - sortie : note d'appréciation



Série vers vecteur [Géron17]

Séries en entrée et en sortie

- réseau vecteur vers série : 1 seule entrée lors de la 1ère étape , et une sortie en série
 - par ex : entrée : image
 - sortie : légende pour cette image

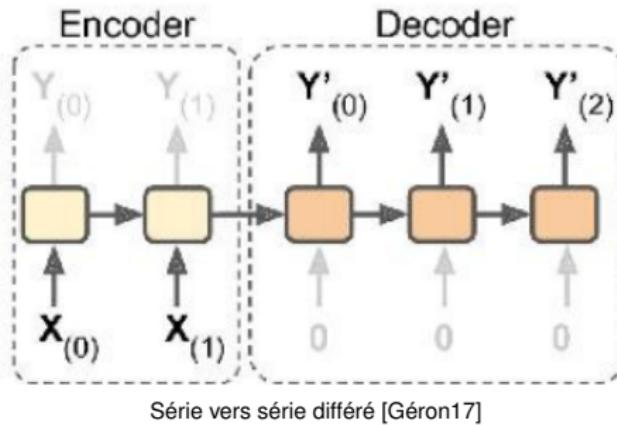


Vecteur vers série [Géron17]

Séries en entrée et en sortie

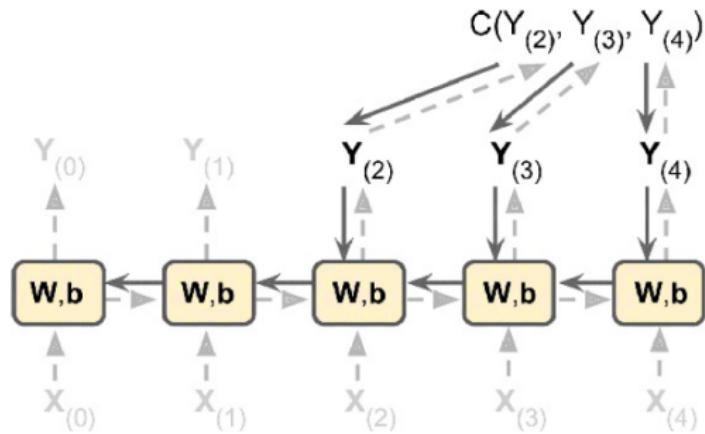
- encodeur-décodeur

- encodeur : réseau série vers vecteur
- item suivi d'un décodeur : réseau vecteur vers série
- ex : traduction d'une phrase d'une langue vers une autre



Entraînement des RNN

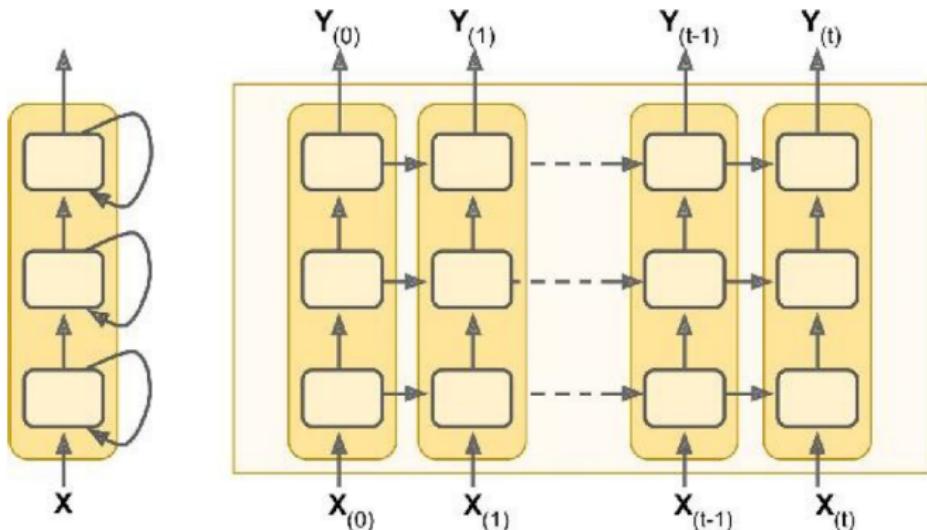
- on déplie le réseau dans le temps, puis on utilise une rétropropagation classique
- ⇒ rétropropagation dans le temps (*back-propagation through time*)



Rétropropagation dans le temps [Géron17]

RNN profond

- plusieurs couches de cellules de mémoire empilées
⇒ RNN profond



RNN profond (à gauche), déplié dans le temps (à droite) [Géron17]

RNN profond

- risque d'aboutir au surajustement du jeu d'entraînement
 - ⇒ régularisation dropout
- difficultés d'entraînement sur de longues séries
 - peut être confronté au problème de disparition/explosion des gradients
 - solutions vues précédemment peuvent être employées mais pas suffisantes
 - autre solution : *rétropropagation tronquée dans le temps* : on déplie le RNN sur un nombre limité d'étapes temporelles pendant l'entraînement
- Autre problème : la mémoire des entrées disparaît progressivement
 - ⇒ utilisation d'autres types de cellules : cellules avec une mémoire à long terme

Plan

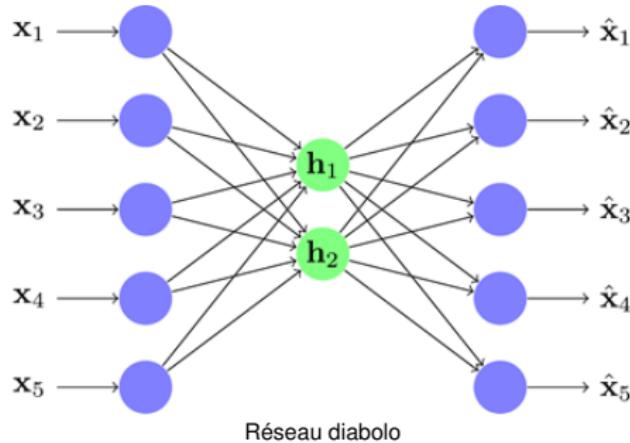
- 1 Introduction
- 2 Réseaux de neurones artificiels
- 3 Entraînement de Réseaux de neurones profonds
 - Pb de disparition et d'explosion des gradients
 - Réutiliser des couches préentraînées
 - Optimiseurs plus rapides
 - Régularisation
- 4 Réseaux de neurones convolutifs
 - Définition des couches
 - Architectures
- 5 Réseaux de neurones récurrents
- 6 Autres méthodes

Autoencodeurs

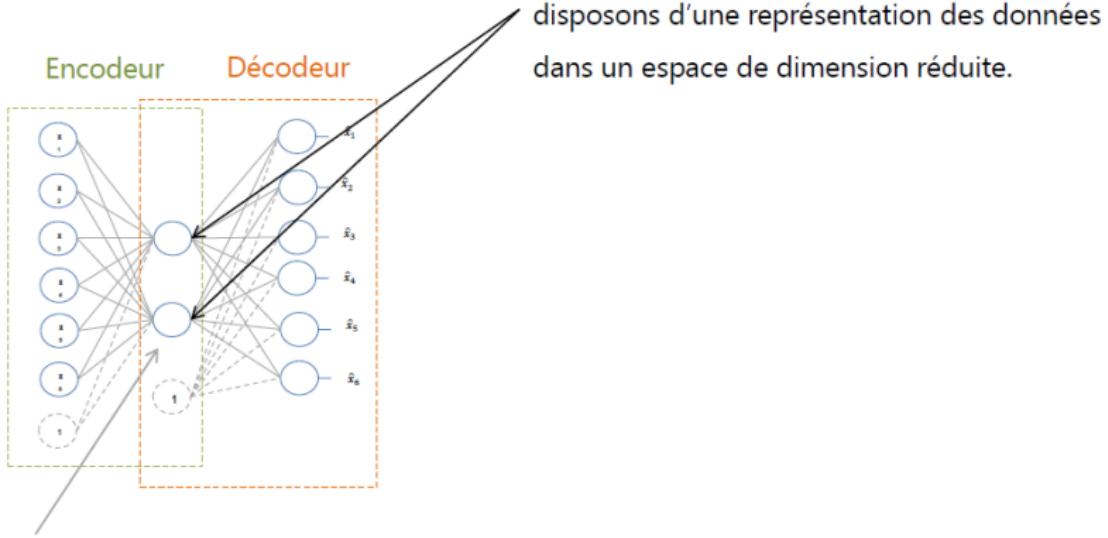
- réseaux de neurones artificiels capables d'apprendre des représentations efficaces des données d'entrée (codages) sans aucune supervision
- ces codages ont souvent une dimension plus faible que les données d'entrée
- Autoencodeurs : détecteurs puissants de caractéristiques
- peuvent être utilisés pour le préentraînement non supervisé de réseaux de neurones profonds
- certains autoencodeurs sont capables de générer de nouvelles données originales ressemblant aux données d'entraînement : modèle génératif

Autoencodeurs

- Architecture semblable à un PMC mais le nb de neurones de la couche de sortie doit être égal au nb d'entrées
- 2 parties : encodeur / décodeur



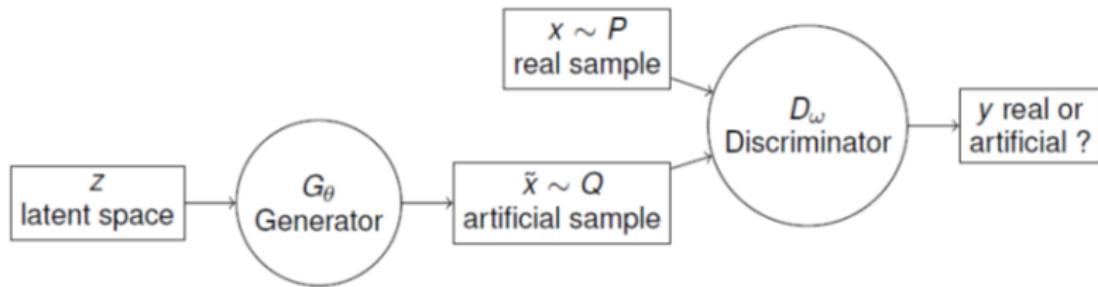
Autoencodeurs



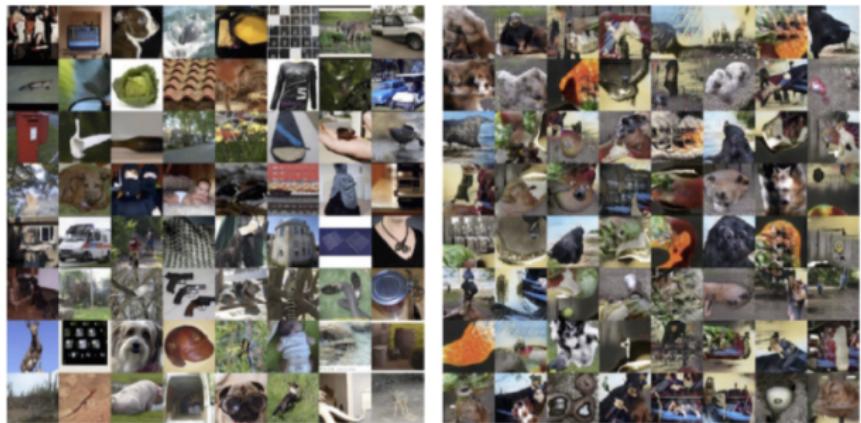
« code » qui permet de compresser (avec perte) l'information portée par les données, d'où le terme « auto-encodeur »

En sortie de la couche centrale, nous disposons d'une représentation des données dans un espace de dimension réduite.

Generative Adversarial Network (GAN)



Generative Adversarial Network (GAN)



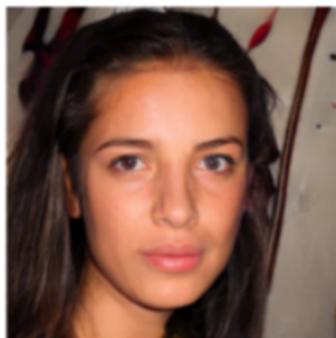
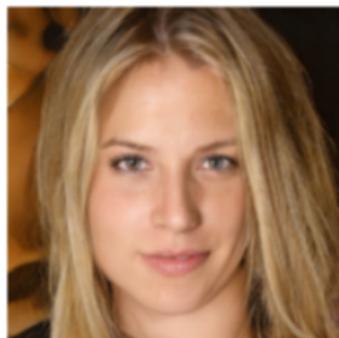
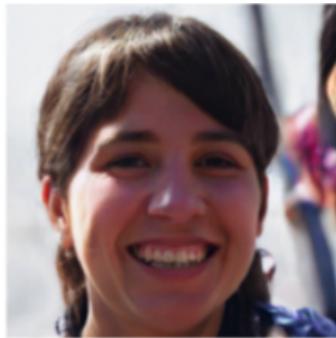
Training Data

Samples

Figure: GAN on ImageNet

Ian Goodfellow. "NIPS 2016 tutorial: Generative adversarial networks". In: *arXiv preprint arXiv:1701.00160* (2016) [_](#) [_](#)

Generative Adversarial Network (GAN)



<https://www.thispersondoesnotexist.com/>

Generative Adversarial Network (GAN)

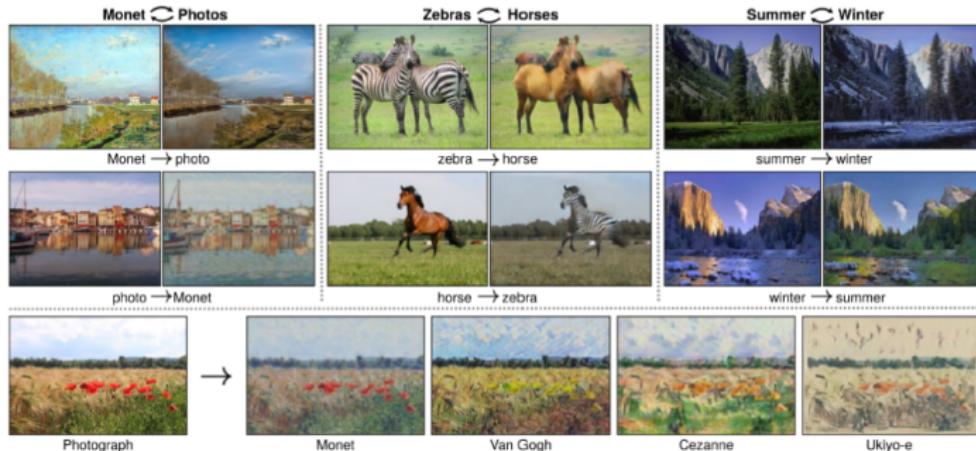


Figure: CycleGAN

Jun-Yan Zhu et al. "Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017 IEEE International Conference on Computer Vision (ICCV). Venice: IEEE, Oct. 2017, pp. 2242–2251. ISBN: 978-1-5386-1032-9. DOI: 10.1109/ICCV.2017.244. URL: <http://ieeexplore.ieee.org/document/8237506/> (visited on 02/21/2019)