Deep Learning

Sabeur Elkosantini
Email: Sabeur.Elkosantini@fsegn.u-carthage.tn
Website: http://sabeur.elkosantini.me

Types d'apprentissage

- Supervisé: Apprentissage avec des données labellisés

Exemple: classification des e-mails déjà étiquetés (spam ou non)

- Non supervisé: découvrez des modèles (des patterns) dans des données non étiquetées

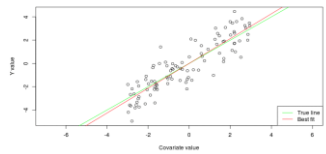
Exemple: regrouper des documents similaires en fonction du texte

- Apprentissage par renforcement: apprenez à agir en fonction des retours / récompenses

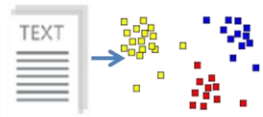
Exemple: apprendre à jouer à Go

Classification

Anomaly Detection
Sequence labeling
…



Regression



Clustering

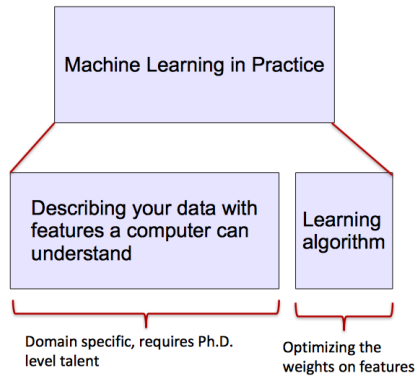
http://mbjoseph.github.io/2013/11/27/measure.html

ML vs. Deep Learning

- La plupart des méthodes d'apprentissage automatique (Machine Learning) fonctionnent bien en raison des représentations conçues par l'homme
- ML devient simplement l'optimisation des poids pour mieux faire une prédiction finale

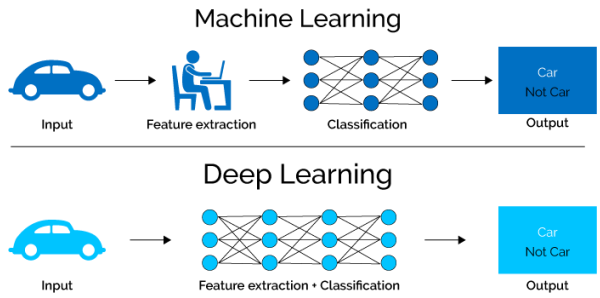
ML vs. Deep Learning

Most machine learning methods work well because of **human-designed representations** and **input features**
ML becomes just **optimizing weights** to best make a final prediction



What is Deep Learning (DL) ?

A machine learning subfield of learning **representations** of data. Exceptional effective at **learning patterns**.
Deep learning algorithms attempt to learn (multiple levels of) representation by using a **hierarchy of multiple layers**
If you provide the system **tons of information**, it begins to understand it and respond in useful ways.

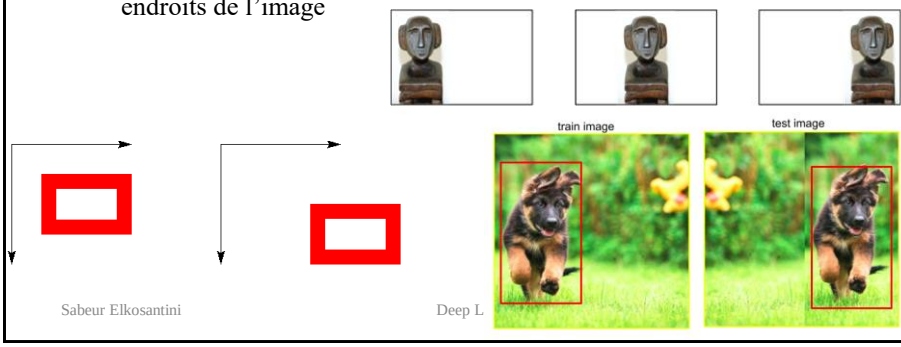


https://www.xenonstack.com/blog/static/public/uploads/media/machine-learning-vs-deep-learning.png

Réseaux de neurones convolutifs CNN

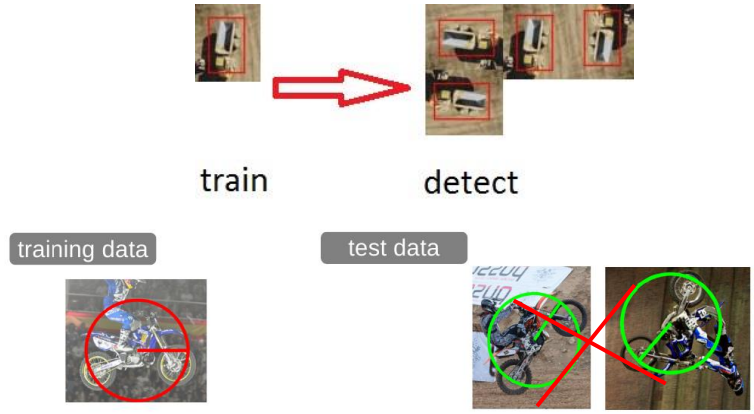
Motivation

- Motivations sur les images :
  - ✓ Corrélation locale des données
  - ✓ Détection de formes
  - ✓ Invariance par translation : les formes peuvent se trouver à différents endroits de l'image



Motivation

- Motivations sur les images :
  - ✓ Attention: pas d'invariance par rotation



Convolution

Filtre ou Kernel

Soit une image $I \in \mathbb{R}^{m \times n}$ et un filtre de convolution $F \in \mathbb{R}^{k \times k}$ de taille impaire $k = 2d + 1$ (avec $d \in \mathbb{N}^+$):

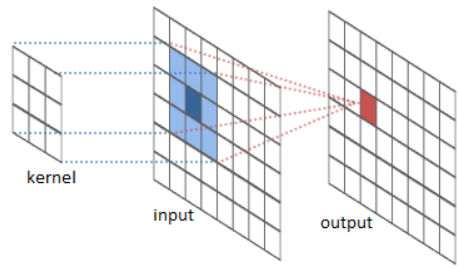
$$(I * F)[x, y] = \sum_{i=-d}^{+d} \sum_{j=-d}^{+d} I[x+i, y+j] \times F[i+d+1, j+d+1]$$

Exemple

$$\left(\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \right) [2, 2] = (i \cdot 1) + (h \cdot 2) + (g \cdot 3) + (f \cdot 4) + (e \cdot 5) + (d \cdot 6) + (c \cdot 7) + (b \cdot 8) + (a \cdot 9).$$

Convolution

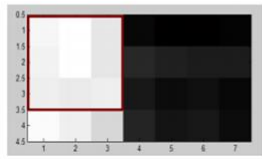
Filtre ou Kernel



Convolution

Filtre ou Kernel

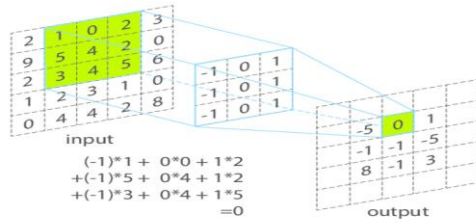
$$F = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$



207	210	195	63	57	56	59
204	212	197	82	76	74	75
202	198	202	72	65	67	63
209	201	187	78	69	71	64

$I * F$

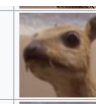
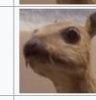
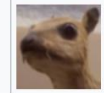
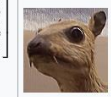
	-26	-533	-517	-28
	-29	-505	-513	-25



Motivation

- Effet de convolutions sur les images

Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Unsharp masking 5 × 5 Based on Gaussian blur with amount as 1 and threshold as 0 (with no image mask)	$\frac{-1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

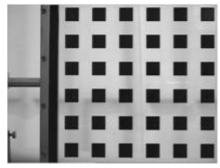
Motivation

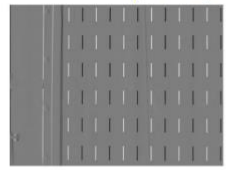
- Effet de convolutions sur les images

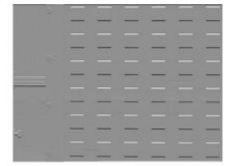
Détection bordure

Bordure verticale $\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$

Bordure horizontale $\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$



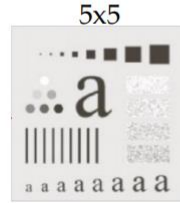




Flou

$\begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & 1 \\ 1 & 1 & 1 \end{bmatrix}$

5x5



15x15

Motivation

- Exemple de filtres

Augmenter le contraste

0	0	0	0	0
0	0	-1	0	0
0	-1	5	-1	0
0	0	-1	0	0
0	0	0	0	0

Flou

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

Amélioration des bords

	0	0	0	
	-1	1	0	
	0	0	0	

Détection des bords

	0	1	0	
	1	-4	1	
	0	1	0	

Repoussage

	-2	-1	0	
	-1	1	1	
	0	1	2	

Filtre de Sobel

	-1	0	1	
	-2	0	2	
	-1	0	0	

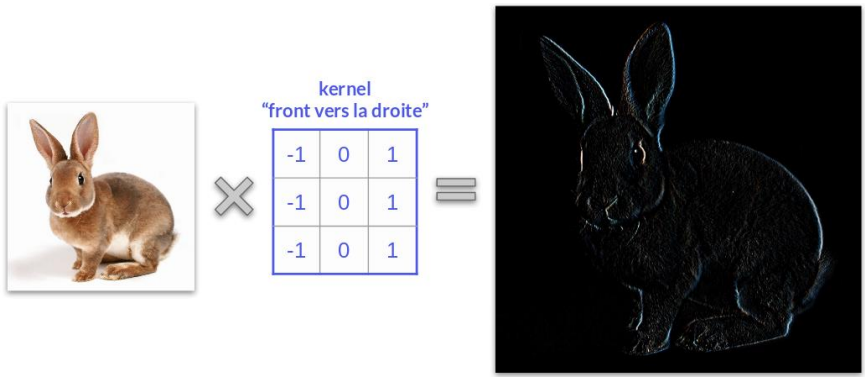
Motivation

- Effet de convolutions sur les images

On retrouve l'effet du filtrage passe-bas : un lissage global de l'image, avec floutage des contours, suppression des détails et lissage des textures.

Motivation

- Effet de convolutions sur les images



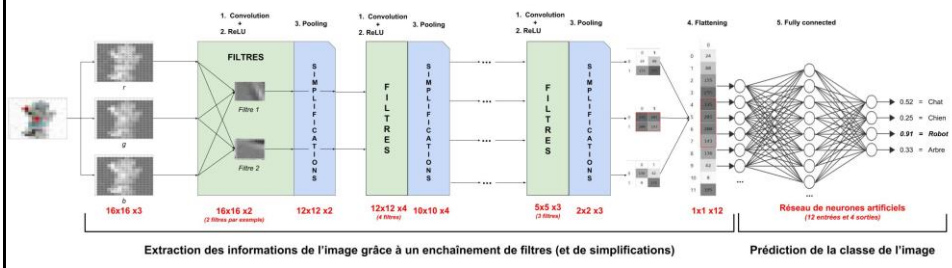
Réseaux de neurones convolutifs

- Architecture

  - Les couches:
    - Fully-Connected
    - Convolutive
    - Pooling
      - ✓ Max
      - ✓ Average et global average
      - ✓ Stochastic
      - ✓ Fractional

Réseaux de neurones convolutifs

- Architecture



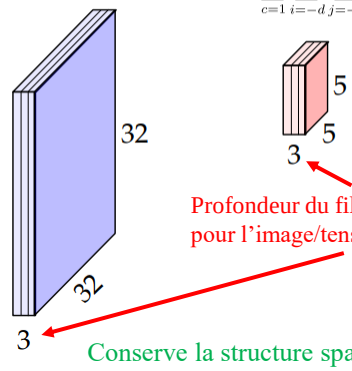
Réseaux de neurones convolutifs

- Couches convolutives

Une image couleur possède typiquement *C* = *3* canaux (RGB) :

Soit une image $I \in \mathbb{R}^{C \times m \times n}$ et un filtre de convolution $F \in \mathbb{R}^{C \times k \times k}$:

$$(I * F)[x, y] = \sum_{c=1}^{C} \sum_{i=-d}^{+d} \sum_{j=-d}^{+d} I[c, x+i, y+j] \times F[c, i+d+1, j+d+1]$$



Profondeur du filtre est TOUJOURS la même que pour l'image/tenseur en entrée de la couche

Conserve la structure spatiale/couleur de l'entrée

Réseaux de neurones convolutifs

- Couches convolutives

Calcul d'une étape de la convolution

Image

50	200	235	201
15	135	88	100
0	42	77	165
6	108	250	144

*

Convolution

1	2	3
4	5	6
7	8	9

Kernel = **3x3**
Stride = **1**
Padding = **0**

=

Résultat

3447	

Pour calculer, on fait :

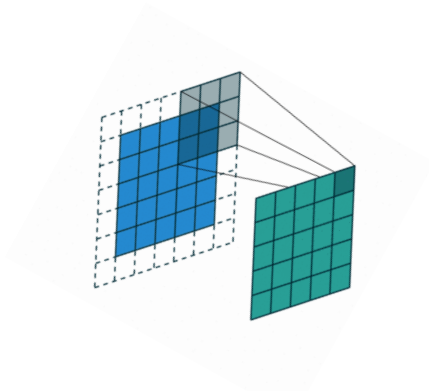
pixel1 de l'image **x** *pixel1 de la convolution* **+** *pixel2 de l'image* **x** *pixel2 de la convolution* **+** …

Ici, cela donne : 50*1 + 200*2 + 235*3 + 15*4 + 135*5 + 88*6 + 0*7 + 42*8 + 77*9 = 3447

Réseaux de neurones convolutifs

- Couches convolutives

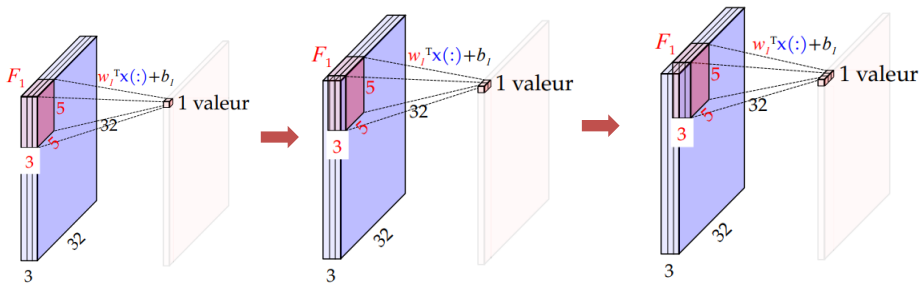
« Glisser » spatialement le filtre F sur l'image, en calculant produit scalaire à chaque endroit de x



Réseaux de neurones convolutifs

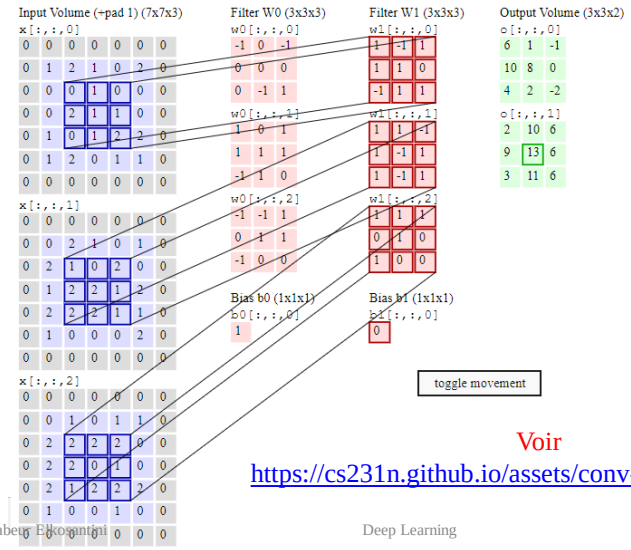
- Couches convolutives

« Glisser » spatialement le filtre F sur l'image, en calculant produit scalaire à chaque endroit de x



Réseaux de neurones convolutifs

- Couches convolutives



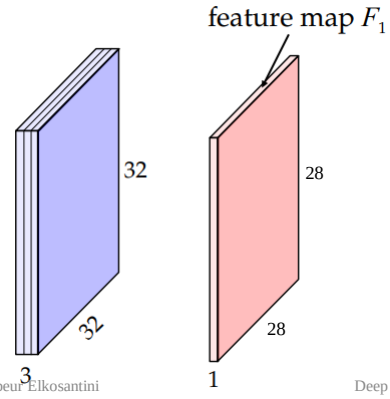
Voir
https://cs231n.github.io/assets/conv-demo/index.html

Réseaux de neurones convolutifs

- Couches convolutives

Sortie : feature map

« mesure » la présence du feature F_1 à cet endroit de l'image.

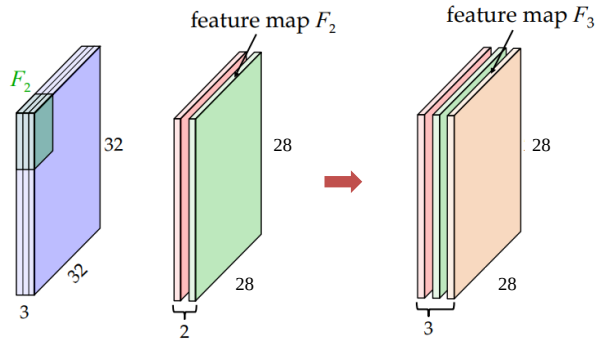


Réseaux de neurones convolutifs

- Couches convolutives

Sortie : feature map

Banque de filtres convolutifs

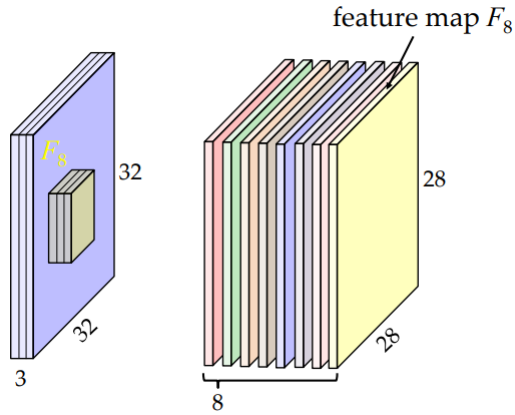


Réseaux de neurones convolutifs

- Couches convolutives

Sortie : feature map

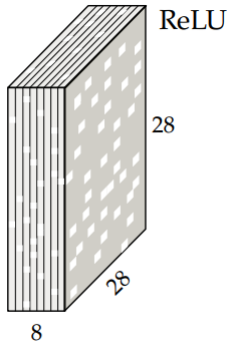
Banque de filtres convolutifs



Réseaux de neurones convolutifs

- ReLU

Applique la fonction d'activation sur chaque entrée, individuellement



- Augmenter la non linéarité dans l'image (la linéarité est quasi absente dans les images, exemple les coins)
- ReLU est une fonction brise (une partie de) la linéarité en supprimant une partie des valeurs (toutes celles négatives)
- En supprimant une partie des données, ReLU permet également d'accélérer les calculs

Réseaux de neurones convolutifs

- ReLU

Applique la fonction d'activation sur chaque entrée, individuellement

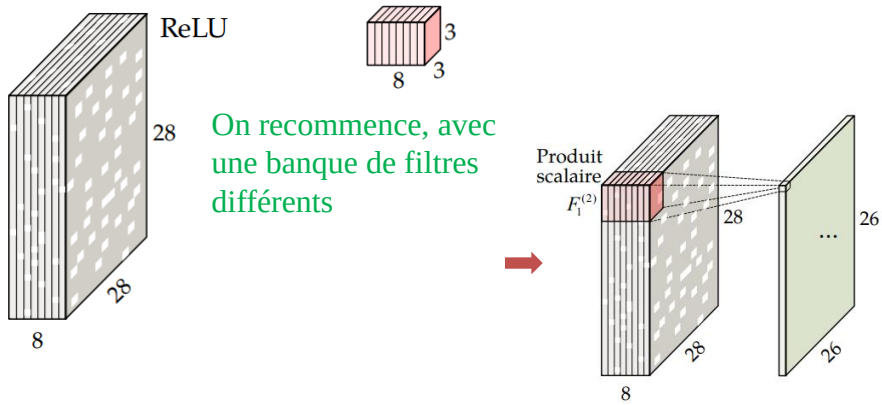
Avant relification

Après relification

Réseaux de neurones convolutifs

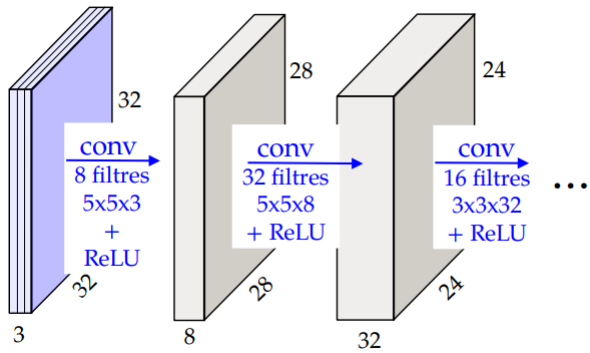
- ReLU

Applique la fonction d'activation sur chaque entrée, individuellement



Réseaux de neurones convolutifs

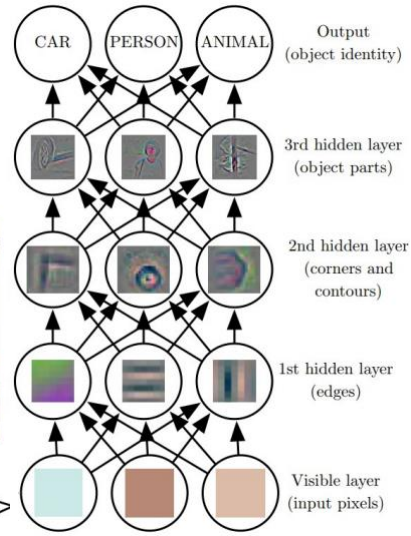
- Convolution + ReLU



Réseaux de neurones convolutifs

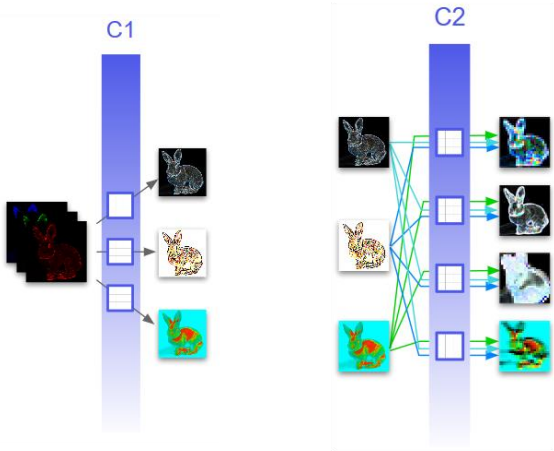
- Hiérarchie de filtres

Perme d'établir des liens entre des pixels de plus en plus éloignés



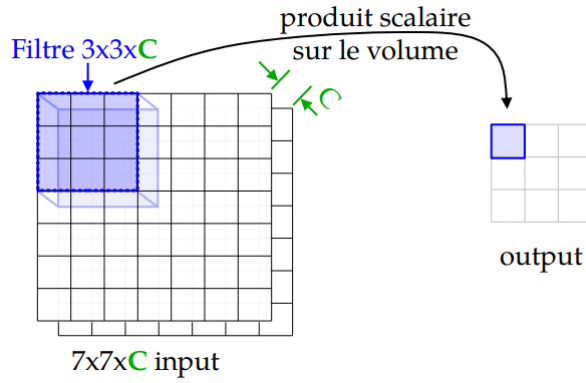
Réseaux de neurones convolutifs

- Hiérarchie de filtres



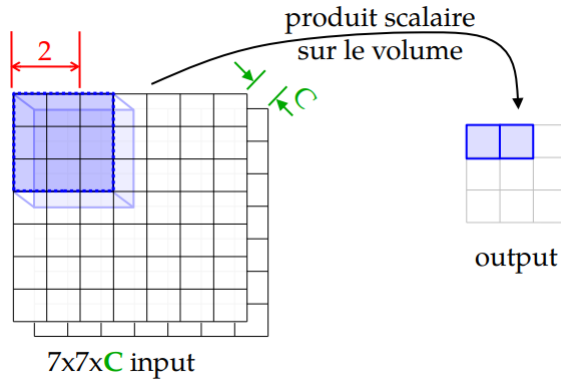
Réseaux de neurones convolutifs

- Pas du filtre (« stride »)



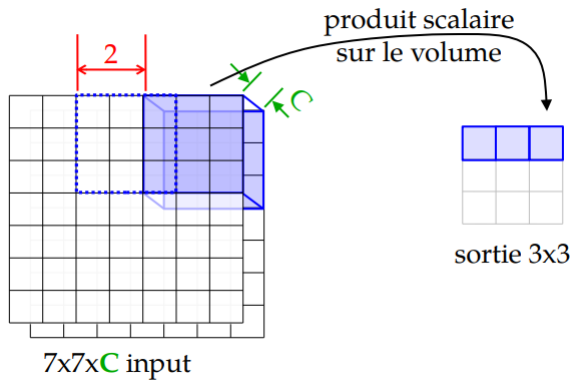
Réseaux de neurones convolutifs

- Pas du filtre (« stride »)



Réseaux de neurones convolutifs

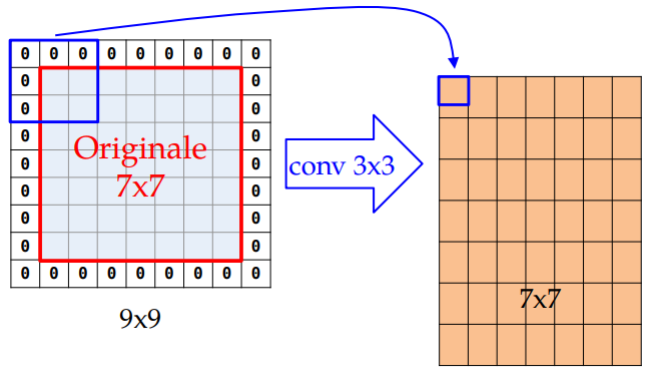
- Pas du filtre (« stride »)



Réseaux de neurones convolutifs

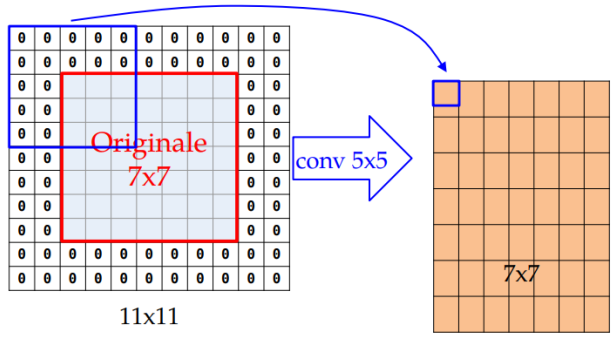
- Ajout de zéros (« zero padding »)

Le **padding** consiste simplement à ajouter des 0 tout autour d'une matrice (image) pour en augmenter la taille.



Réseaux de neurones convolutifs

- Ajout de zéros (« zero padding »)



Pour filtre 3x3, padding de largeur 1
Pour filtre 5x5, padding de largeur 2
Pour filtre 7x7, padding de largeur 3

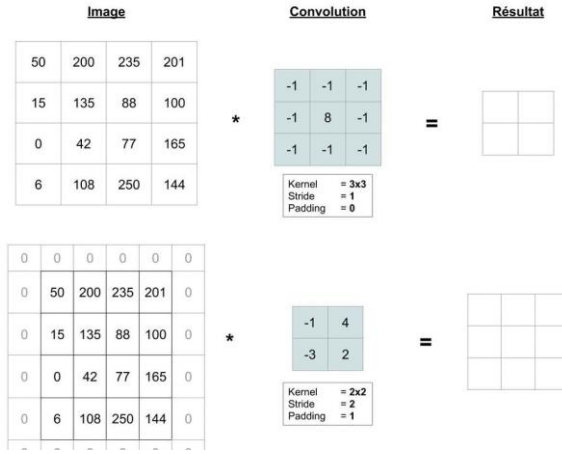
Réseaux de neurones convolutifs

- Remarque

- Padding (ajout de zero) peut agrandir la taille de l'image
- Le "stride" peut réduire la resolution de la sortie des convolutions
- Padding et stride permettent d'ajouter la dimensionalité des données (images ou son)

Réseaux de neurones convolutifs

- Calcul de deux convolutions sur deux images, avec des kernels, strides et paddings différents

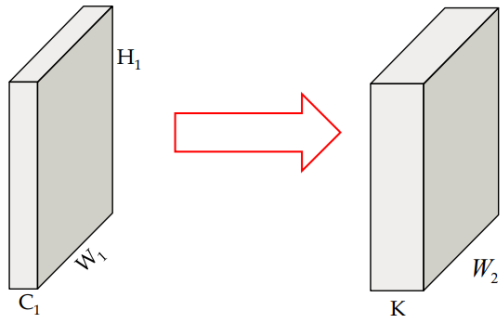


Réseaux de neurones convolutifs

- Tailles entrée et sortie

K : nombre de filtres
F : taille du filtre (FxF)
S : pas (stride)
P : quantité de 0-padding

Valeurs typiques
K : puissances de 2
F=3, S=1, P=1
F=5, S=1, P=2
F=5, S=2, P=autant que nécessaire
F=1, S=1, P=0



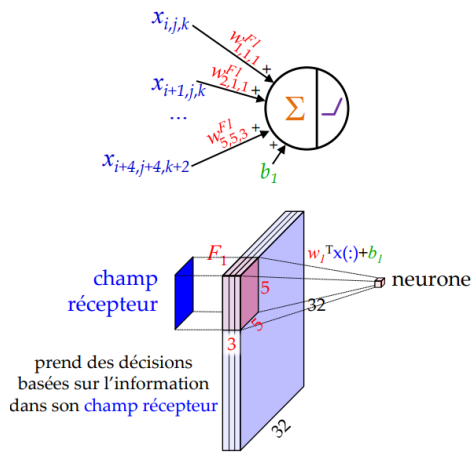
$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

paramètres : $F \cdot F \cdot C_1 \cdot K$ poids + K biais

Réseaux de neurones convolutifs

- Assimiler la couche convolution + ReLU à une neurone



Réseaux de neurones convolutifs

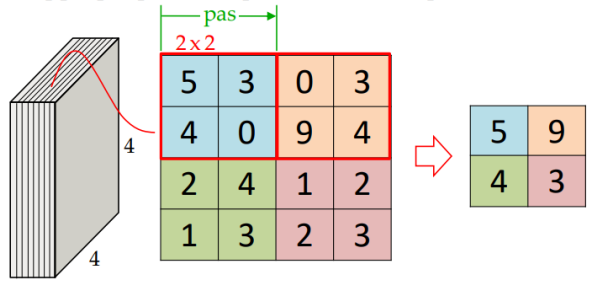
- Pooling

  - Opération simple qui consiste à remplacer un carré de pixels par une valeur unique
    - ✓ L'image diminue en taille et se retrouve simplifiée (lissée).
  - Il existe plusieurs types de pooling :
    - ✓ Le « max pooling » revient à prendre la valeur maximale de la sélection. C'est le type le plus utilisé car il est rapide à calculer (immédiat), et permet de simplifier efficacement l'image
    - ✓ Le « mean pooling »: on calcule la somme de toutes les valeurs et on divise par le nombre de valeurs. On obtient ainsi une valeur intermédiaire pour représenter ce lot de pixels
    - ✓ Le « sum pooling », c'est la moyenne sans avoir divisé par le nombre de valeurs (on ne calcule que leur somme)

Réseaux de neurones convolutifs

- Pooling
  - Max pooling

• Appliqué pour chaque tranche, indépendamment



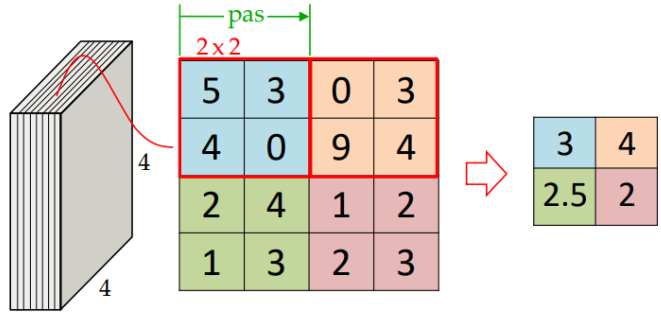
Doit spécifier :
- taille
- pas

N'ajoute aucun paramètre entrainable

Réseaux de neurones convolutifs

- Pooling
  - Mean (ou average) pooling

• On fait la moyenne sur chaque fenêtre

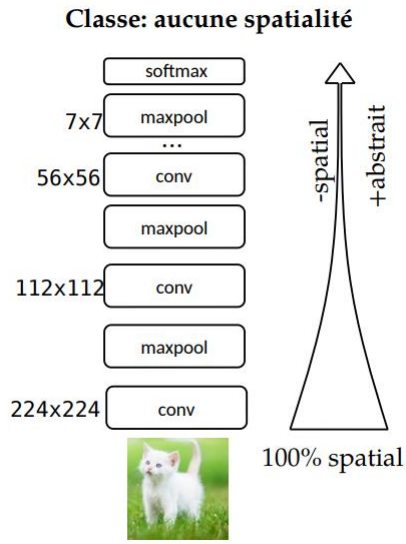


Doit spécifier :
- taille
- pas

Réseaux de neurones convolutifs

- Pooling

- Réduit le nombre de paramètre
- Confère une certaine invariance aux transformations géométriques (translation, échelle)



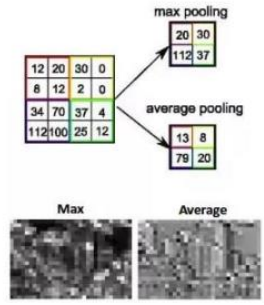
Réseaux de neurones convolutifs

- Pooling

- Max, ou mean ou sum ?
- « sum » fait augmenter la valeur des pixels de l'image. Elle aura tendance à creuser les écarts entre régions de l'image
  - ➔ Mettre en avant certaines caractéristiques.
- « Mean » permet de rester borné,
  - ➔ seules les features moins marquées ressortiront.
  - ➔ A tendance à lisser les features
- « Max » a tendance à retenir les caractéristiques (features) les plus marquées et simples de la sélection de pixels, comme par exemple une arête verticale.
  - ➔ détecte la présence d'un feature dans une région

Réseaux de neurones convolutifs

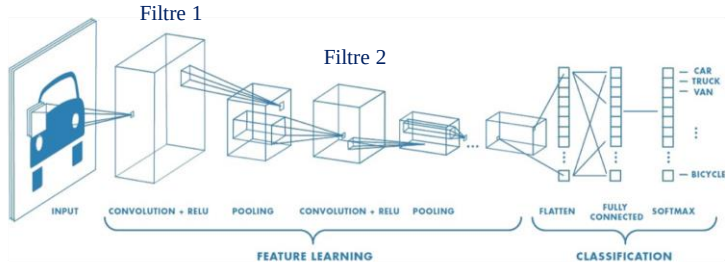
- Pooling
  - Max, ou mean ou sum ?



Réseaux de neurones convolutifs

- Pooling

Et après ?



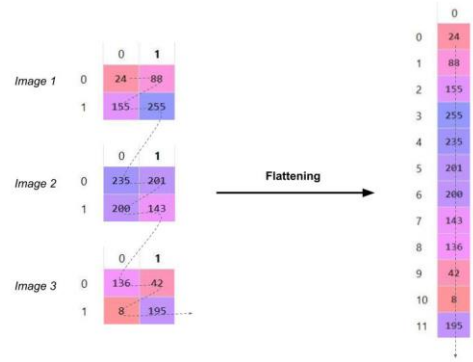
Réseaux de neurones convolutifs

- Le flattening (ou mise à plat ou Vectorisation)

  - Consiste simplement à **mettre bout à bout** toutes les images

Le réseau de neurones prend simplement en entrée un vecteur

Elle détruit toute relations spatiales et les canaux de couleurs

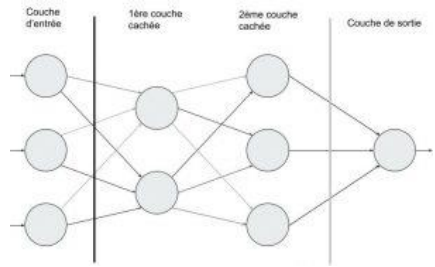


Réseaux de neurones convolutifs

- Le réseau « fully connected »

  - La couche sortie utilise la fonction d'activation softmax

La sortie envoie des probabilités pour chaque classe de prédiction



  - Un réseau à deux couches: celle d'entrée et celle de sortie

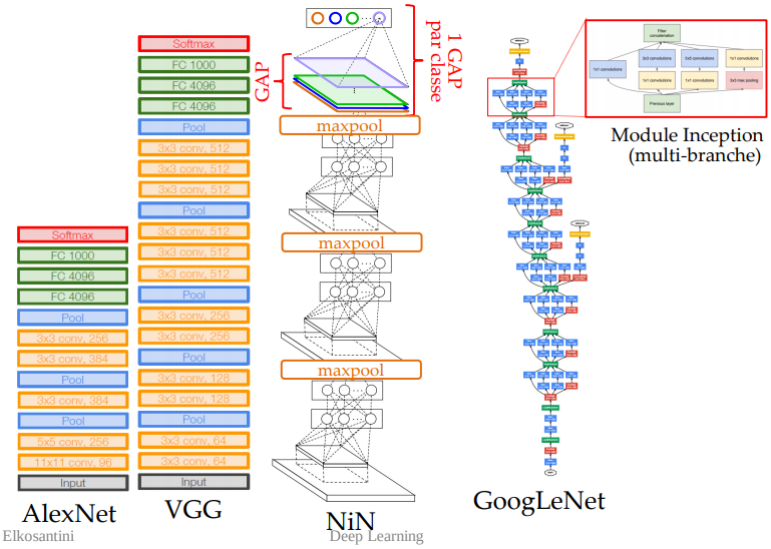
Réseaux de neurones convolutifs

- Architecture

❑ Il existe plusieurs architecture CNN:
  - ✓ LeNet (le plus simple, pour découvrir les CNN),
  - ✓ AlexNet (lancé en 2012),
  - ✓ ZFNet (amélioration d'AlexNet tombée dans l'oubli),
  - ✓ GoogLeNet (parent du célèbre Inception, un algorithme de reconnaissance d'image avec d'autres subtilités, au même titre que YOLO, MobileNet, etc…),
  - ✓ VGGNet (encore très utilisé et puissant),
  - ✓ ResNet (pareil, même s'il s'éloigne d'un CNN traditionnel)

Réseaux de neurones convolutifs

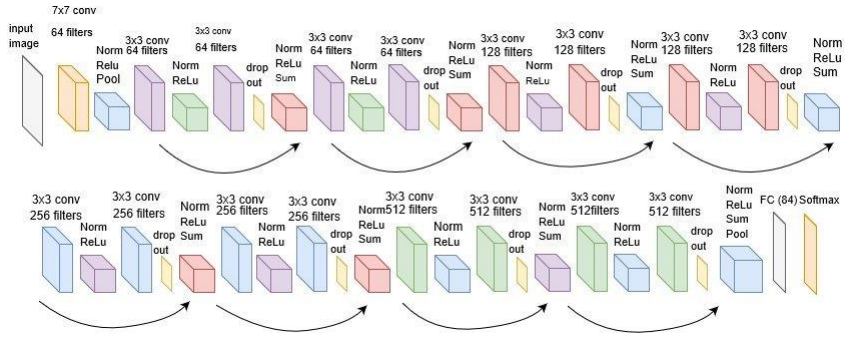
- Architecture



Réseaux de neurones convolutifs

- Architecture

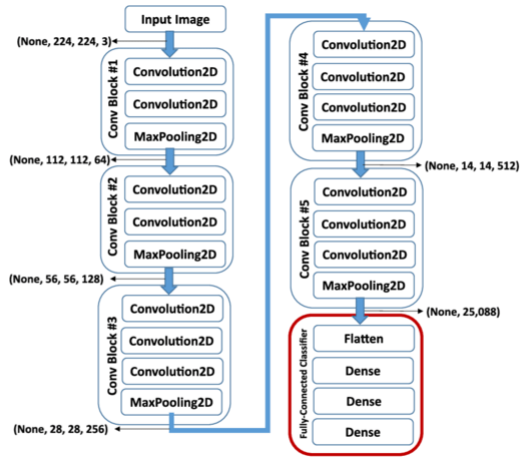
ResNet



Réseaux de neurones convolutifs

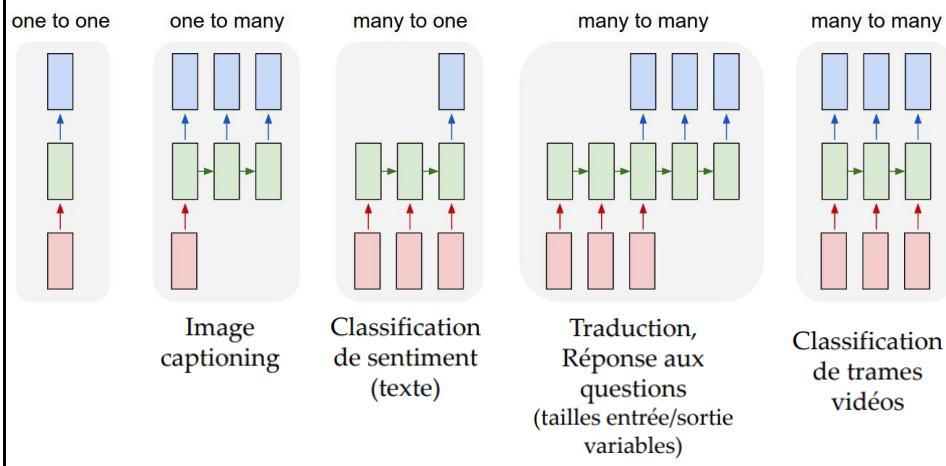
- Architecture

VGG 16



Réseaux de neurones récurrents
RNN

Réseaux de neurones récurrents



Réseaux de neurones récurrents

❑ Pourquoi les RNN ?

- Traiter des données séquentielles

  Image X vs. $\{x^{(1)}, x^{(2)}, \cdots, x^{(\tau)}\}$

  – séries temporelles
  – séquences de pixels
  – séquences de mots
- Souvent de longueur ***variable***
- Pas clair d'avance où l'information pertinente est située

I went to Nepal in 2009

In 2009, I went to Nepal

Réseaux de neurones récurrents

❑ Les réseaux de neurones récurrents reposent sur deux principes :

✓ le premier principe est celui de la fenêtre glissante qui permet de traiter des signaux de taille variable ;

✓ le second principe est l'utilisation de connexions récurrentes qui permettent d'analyser la partie passée du signal.

Réseaux de neurones récurrents

Exemple textuel :

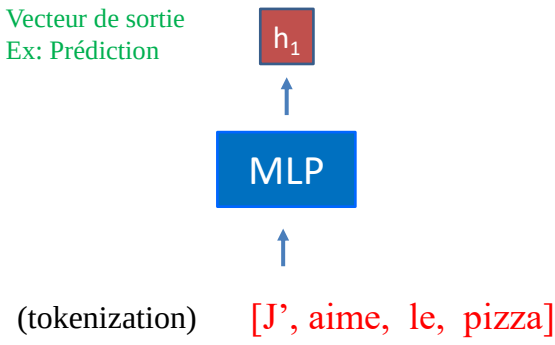
- Prédire la phrase suivante :

J'aime le pizza

Réseaux de neurones récurrents

Exemple textuel :

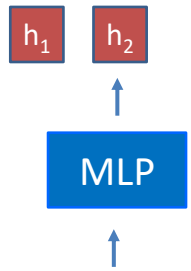
Vecteur de sortie
Ex: Prédiction



Réseaux de neurones récurrents

Exemple textuel :

Vecteur de sortie
Ex: Prédiction

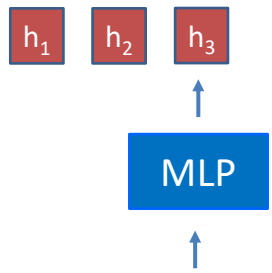


(tokenization) [J’, aime, le, pizza]

Réseaux de neurones récurrents

Exemple textuel :

Vecteur de sortie
Ex: Prédiction

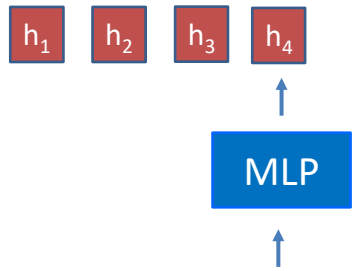


(tokenization) [J’, aime, le, pizza]

Réseaux de neurones récurrents

Exemple textuel :

Vecteur de sortie
Ex: Prédiction



(tokenization) [J’, aime, le, pizza]

Réseaux de neurones récurrents

Exemple textuel :

Vecteur de sortie
Ex: Prédiction

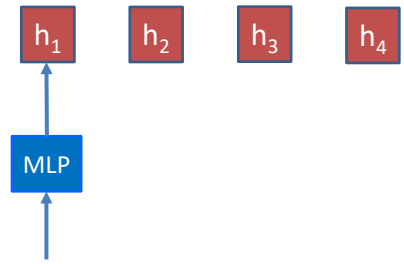
Pas de relation entre les h_i

(tokenization) [J’, aime, le, pizza]

Réseaux de neurones récurrents

Exemple textuel : Les réseaux de neurones récurrents

Vecteur de sortie
Ex: Prédiction

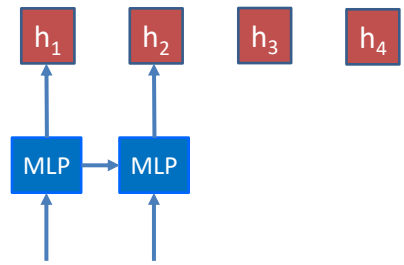


(tokenization) [J’, aime, le, pizza]

Réseaux de neurones récurrents

Exemple textuel : Les réseaux de neurones récurrents

Vecteur de sortie
Ex: Prédiction

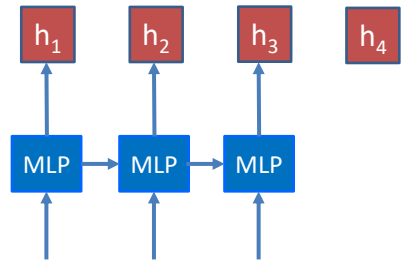


(tokenization) [J’, aime, le, pizza]

Réseaux de neurones récurrents

Exemple textuel : Les réseaux de neurones récurrents

Vecteur de sortie
Ex: Prédiction

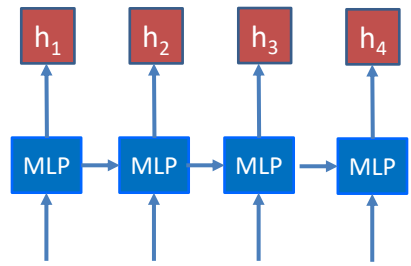


(tokenization) [J’, aime, le, pizza]

Réseaux de neurones récurrents

Exemple textuel : Les réseaux de neurones récurrents

Vecteur de sortie
Ex: Prédiction

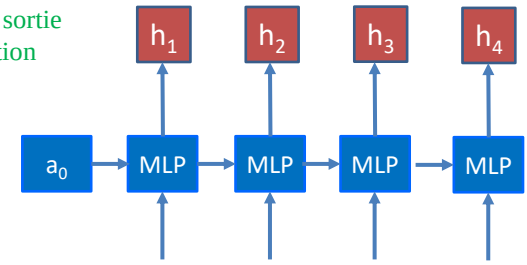


(tokenization) [J’, aime, le, pizza]

Réseaux de neurones récurrents

Exemple textuel : Les réseaux de neurones récurrents

Vecteur de sortie
Ex: Prédiction



(tokenization) [J’, aime, le, pizza]

Réseaux de neurones récurrents

Exemple textuel : Les réseaux de neurones récurrents

Vecteur de sortie
Ex: Prédiction

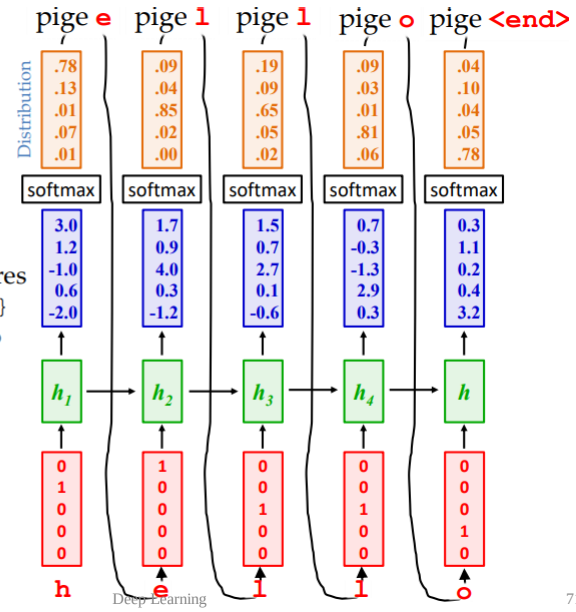
Maintenant h_4 pourra contenir de l'information sur toute la séquence

(tokenization) [J’, aime, le, pizza]

Réseaux de neurones récurrents

Exemple 2

- Réseau entraîné à prédire des caractères {e,h,l,o,<end>}
- Entraîné sur hello

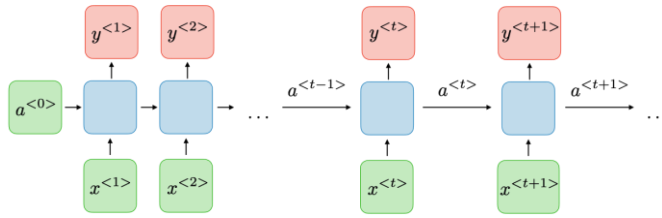


Réseaux de neurones récurrents

1. Être capable de traiter les séquences de longueur variable
2. Garder la trace des dépendances à long terme
3. Conserver l'information sur l'ordre
4. Partager les paramètres le long de la séquence

Réseaux de neurones récurrents

Architecture



À l'instant t, l'activation $a^{<t>}$ et la sortie $y^{<t>}$ sont de la forme suivante :

$$\boxed{a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)} \quad \text{et} \quad \boxed{y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)}$$

où $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ sont des coefficients indépendants du temps et où g_1, g_2 sont des fonctions d'activation.

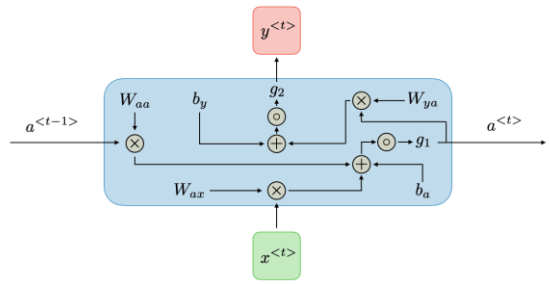
Réseaux de neurones récurrents

Architecture

À l'instant t, l'activation $a^{<t>}$ et la sortie $y^{<t>}$ sont de la forme suivante :

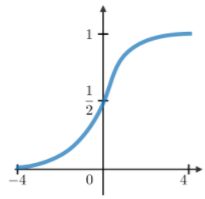
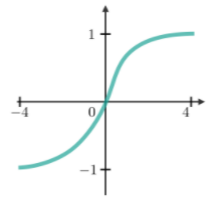
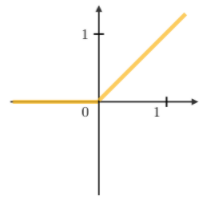
$$\boxed{a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)} \quad \text{et} \quad \boxed{y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)}$$

où $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ sont des coefficients indépendants du temps et où g_1, g_2 sont des fonctions d'activation.



Réseaux de neurones récurrents

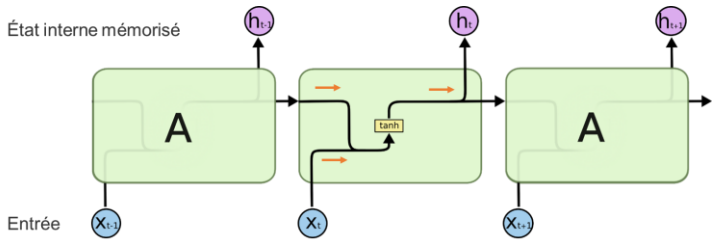
❑ Les fonctions d'activations utilisées dans RNN

Sigmoïde	Tanh	RELU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$
		

Réseaux de neurones récurrents

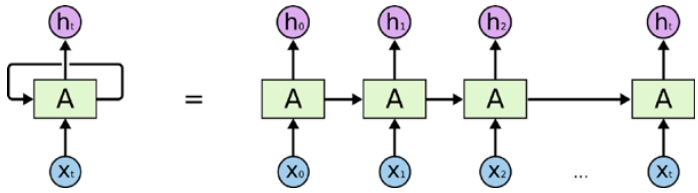
❑ Les fonctions d'activations utilisées dans RNN: tanh

✓ La mémoire du réseau est $h_t = tanh(W \cdot [h_{t-1}, x_t] + b)$



Réseaux de neurones récurrents

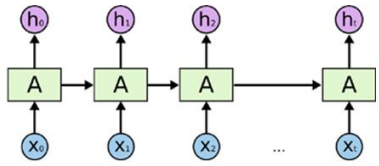
- Une connexion récurrente permet de réinjecter la sortie d'une couche en entrée de celle-ci.

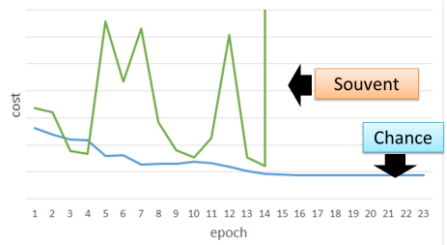


- Avantage
  - ✓ Possibilité de prendre en compte des entrées de toute taille
  - ✓ La taille du modèle n'augmente pas avec la taille de l'entrée
  - ✓ Les calculs prennent en compte les informations antérieures
  - ✓ Les coefficients sont indépendants du temps

Réseaux de neurones récurrents

- Inconvénient:
  - ✓ Les RNR n'apprennent pas facilement

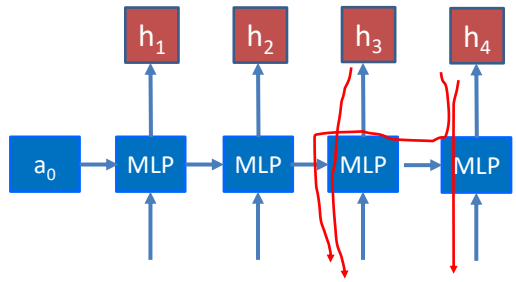




  - ✓ Le temps de calcul est long
  - ✓ Difficulté d'accéder à des informations d'un passé lointain

Réseaux de neurones récurrents

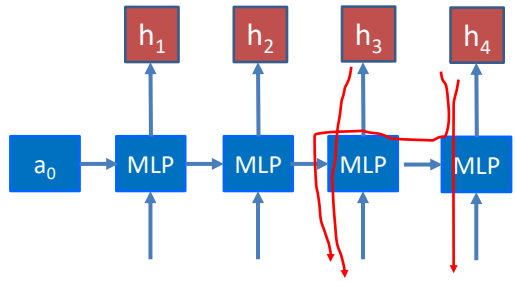
❑ Problème de vanishing (disparition)et exploiding (explosion)



$$
\begin{aligned}
h_t &= W f(h_{t-1}) + W^{(hx)} x_{[t]} \\
\hat{y}_t &= W^{(S)} f(h_t)
\end{aligned}
$$

Réseaux de neurones récurrents

❑ Problème de vanishing (disparition)et exploiding (explosion)



$$
\begin{aligned}
h_t &= W f(h_{t-1}) + W^{(hx)} x_{[t]} \\
\hat{y}_t &= W^{(S)} f(h_t)
\end{aligned}
$$

L'erreur totale est la somme des erreurs aux instants t

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

Réseaux de neurones récurrents

❑ Problème de vanishing (disparition) et exploiding (explosion)

L'erreur totale est la somme des erreurs aux instants t

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

chain rule of differentiation $$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \boxed{\frac{\partial h_t}{\partial h_k}} \frac{\partial h_k}{\partial W}$$

$$\boxed{\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}}$$

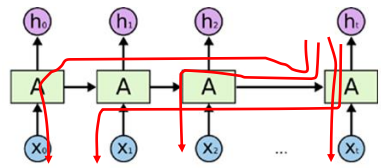
Problème de vanishing (disparition) et exploiding (explosion)

Réseaux de neurones récurrents

❑ Problème de vanishing (disparition) et exploiding (explosion)

$$\boxed{\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}}$$

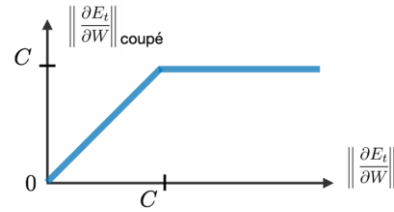
Problème de vanishing (disparition) et exploiding (explosion)



Réseaux de neurones récurrents

❑ Problème de vanishing (disparition) et exploiding (explosion)

✓ Pour « exploding problem » : Coupure de gradient (Gradient Clipping)

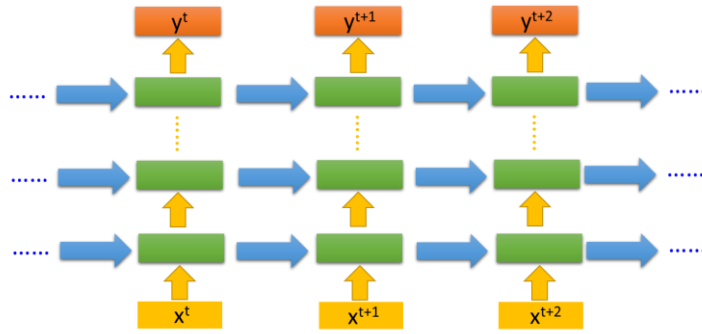


✓ Pour « vanishing problem » : utilisation de quelques portes (voir LSTM)

Réseaux de neurones récurrents

❑ Autres architectures RNN

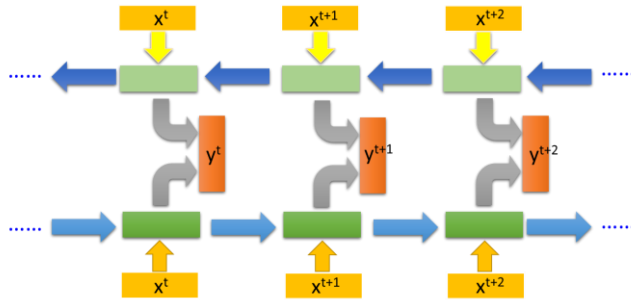
RNN profonds (DRNN)



Réseaux de neurones récurrents

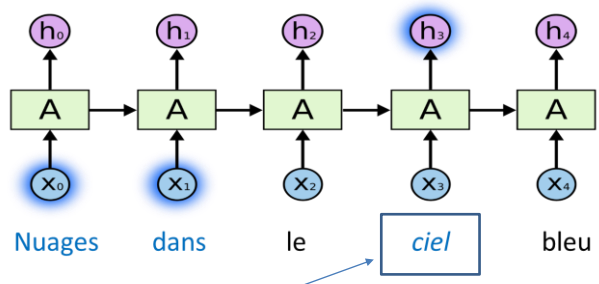
- Autres architectures RNN

RNN bi-directionnelle (BRNN)



Long Short Term Memory
LSTM

LSTM

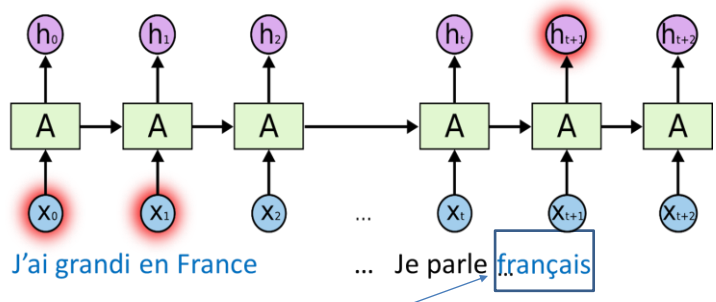


https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Mot à prédire

Le contexte est proche du mot à prédire; peu d'itérations les séparent

LSTM



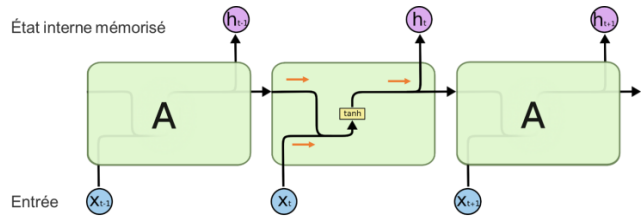
Mot à prédire

Le contexte est loin du mot à prédire; beaucoup d'itérations les séparent!
=> problème de gradient possible

LSTM

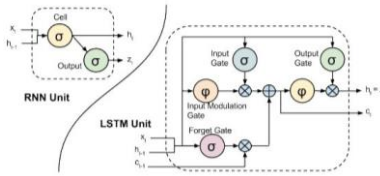
Il faut empêcher le gradient de disparaître…

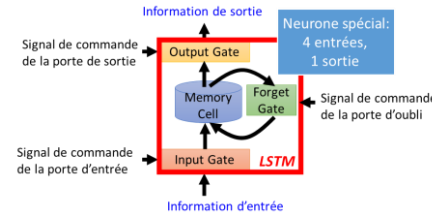
- Normalement, la mémoire du réseau est $h_t = tanh(W \cdot [h_{t-1}, x_t] + b)$ et implique un seul niveau de traitement, créant le risque du gradient évanescent.



LSTM

- Ajoute une mémoire de contexte qui affecte le flux d'information et son traitement (cell state)
- Trois portes décident ce qu'une cellule doit oublier de son passé, retenir pour son état courant, et rendre disponible pour le futur





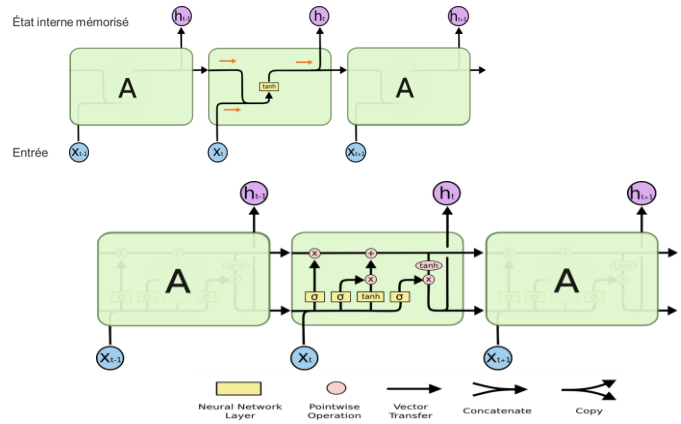
Les signaux de de commande viennent typiquement de perceptrons

LSTM

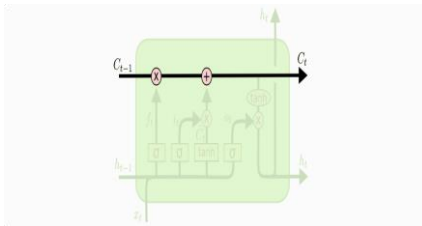
- La récurrence dans une cellule LSTM implique 4 niveaux de traitement

RNN

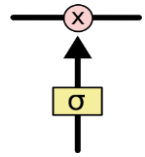
LSTM



LSTM



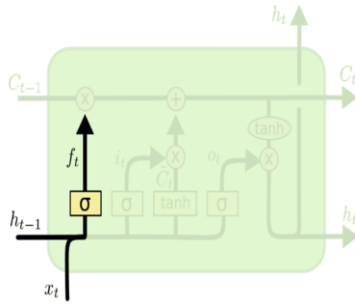
Une cellule STM agit comme un convoyeur d'information contrôlé à l'aide de portes configurables (par entraînement)



Chaque porte est un réseau en soi (perceptron)

LSTM

Porte d'oubli



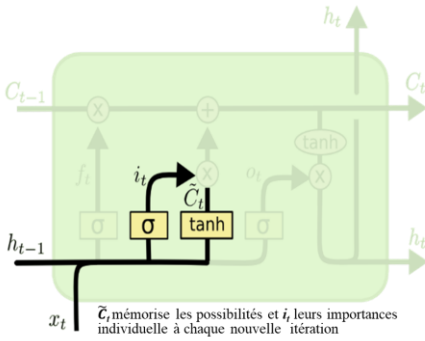
Décide la contribution du contexte précédent à la sortie du réseau

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Ex.: dans la prédiction des mots d'une nouvelle phrase, il n'est pas utile de garder le genre du sujet dans la phrase précédente

LSTM

Porte d'entrée et circuit de mémoire



Décide la contribution de la mémoire ($\tilde{C}_t$ est l'équivalent de h_t dans un RNN standard) à la sortie du réseau

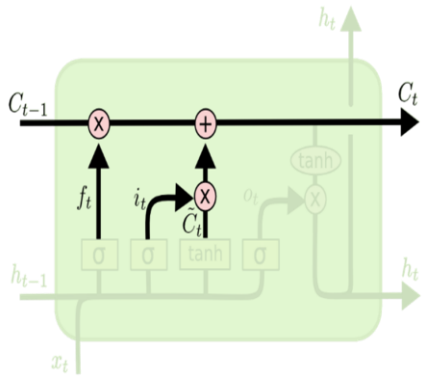
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Ex.: dans la prédiction des mots d'une nouvelle phrase, il est utile de remplacer le genre du sujet de la phrase précédente par le nouveau.

LSTM

Mise à jour de la cellule



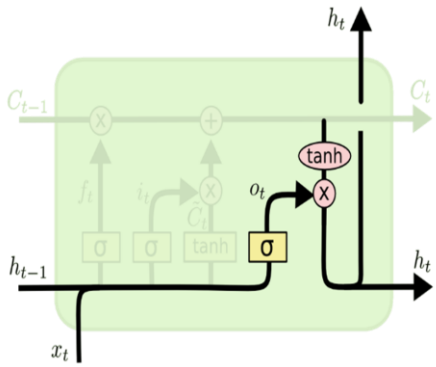
On combine ce qui a été retenu de l'état précédent et ce qui est utilisé de la mémoire courante

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- Agit comme une nouvelle entrée au réseau adaptée au contexte

LSTM

Porte de sortie



Décide quoi rendre public du nouveau contexte

$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Ex.: dans la prédiction des mots d'une nouvelle phrase, il peut être utile de retenir la pluralité du sujet à la vue d'un verbe, ce qui peut simplifier la conjugaison après.

Travail à faire

- ✓ Transformers Neural Network (http://jalammar.github.io/illustrated-transformer)
- ✓ L'architecture AlexNet
- ✓ Gated recurrent unit
- ✓ Comment choisir les hyperparamètres des modèles ?

Lecture

- ✓ Reconnaissance d'image par CNN de « With Draw » de google : https://quickdraw.withgoogle.com/
- ✓ Exemple d'application de CNN pour la classification d'image : https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html
- ✓ Cours RNN : https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-recurrents
- ✓ LSTM (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)
- ✓ LTSM: https://blog.octo.com/les-reseaux-de-neurones-recurrents-des-rnn-simples-aux-lstm/