apprentissage profond - deep learning

partie 2 : réseau convolutif

Adrien CHAN-HON-TONG
ONERA

Rappel

MLP

- $X = \mathbb{R}^D$ l'espace des points, P une distribution sur X
- y la fonction cible dans {-1,1}
- on approxime y par une fonction f (avec des poids w)
- erreur réelle $e_r = \int_X \mathbf{1}_-(y(x)f(x,w))P(x)\mathrm{d}x$
- $x_1, \ldots, x_N$ une base d'apprentissage tirée selon P
- $\chi_1, \ldots, \chi_M$ une base de test tirée selon P
- l'erreur d'apprentissage $e_a = \frac{1}{N}\sum_n \mathbf{1}_-(y(x_n)f(x_n,w))$
- l'erreur de test $e_t = \frac{1}{M}\sum_m \mathbf{1}_-(y(\chi_m)f(\chi_m,w)) \approx e_r$
- deep learning = choisir
  $f(x,w) = w_Q \times relu(w_{Q-1} \times relu(\ldots(relu(w_1 \times x))))$ et
  optimiser w pour avoir e_a faible
- c'est ce qui *marche le mieux aujourd'hui*

La vision par ordinateur

Est ce qu'une image est $x \in \mathbb{R}^D$?

$$1 \text{ image RGB de } 480\text{x}340 = 1 \text{ vecteur dans } [0, 255]^{163200}$$

Il est impossible et stupide de traiter les images comme des points !

Différence image vecteur

- Le traitement d'images repose sur les opérations de **convolution** et **pooling** pour passer d'une image à un vecteur.
- le traitement se termine par une classification de vecteurs.

Le neurone convolutif

- l'entrée est $u \in \mathbb{R}^{H \times W \times \phi}$ (ϕ image ou 1 image ϕ canaux)
- les poids du neurones convolutif sont $\theta \in \mathbb{R}^{(2\delta_H+1)\times(2\delta_W+1)\times\phi}$
- la sortie est une image $\theta \star u \in \mathbb{R}^{H \times W}$
- avec $(\theta \star u)_{h,w}$: $\sum\limits_{dh,dw,\rho \in [-\delta_H,\delta_H]\times[-\delta_H,\delta_W]\times[1,\phi]} x_{h+dh,w+dw,\rho} \times w_{dh,dw,\rho}$

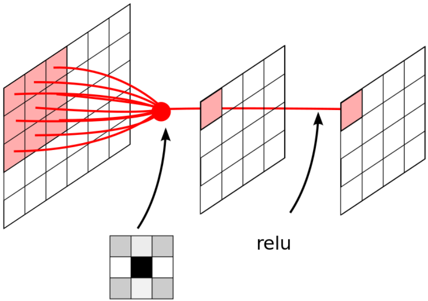
Le neurone convolutif c'est comme un neurone mais local !

La couche convolutive

- l'entrée est $u \in \mathbb{R}^{H \times W \times \phi}$ (ϕ images regroupées ou 1 image avec ϕ canaux)
- la couche contient ψ neurones convolutifs i.e. $\Theta \in \mathbb{R}^{(2\delta_H+1)\times(2\delta_W+1)\times\phi\times\psi}$
- la sortie est une image avec ψ canaux (ou ψ images regroupées) $\Theta \star u = (\Theta_1 \star u, ..., \Theta_\psi \star u) \in \mathbb{R}^{H \times W \times \psi}$

bref exactement comme la couche de neurones mais avec des neurones convolutifs

conv + relu



MLP vs ConvNet

MLP

$$f(x,w) = w_Q \times relu(w_{Q-1} \times relu(...(relu(w_1 \times x))))$$

ConvNet

$$f(x,w) = w_Q \star relu(w_{Q-1} \star relu(...(relu(w_1 \star x))))$$

On a réglé une partie du problème seulement car ça c'est la solution à un autre problème (segmentation : 1 décision par pixel)

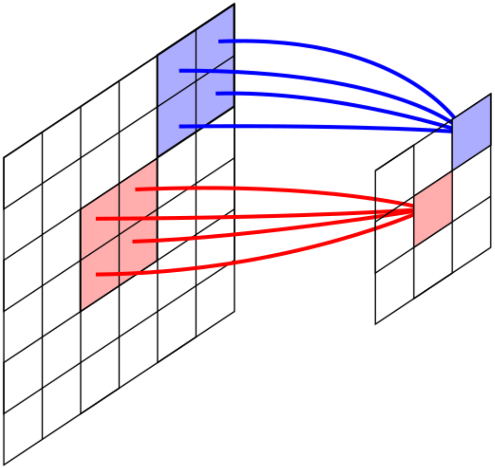
$$f(x,w) = w_{final} \times w_Q \star relu(w_{Q-1} \star relu(...(relu(w_1 \star x)))) \quad ?$$

Pooling

- l'entrée est $u \in \mathbb{R}^{H \times W \times \phi}$
- la sortie est une image $pool(u) \in \mathbb{R}^{\frac{H}{2} \times \frac{W}{2} \times \phi}$
- avec $pool(u)_{h,w,\rho} = \max_{dh,dw \in [0,1]} x_{2h+dh,2w+dw,\rho}$

- le pooling est un type d'activation avec dimension spatiale
- elle permet de diminuer la taille des objets
- elle donne de l'invariance au petites déformations

Pooling



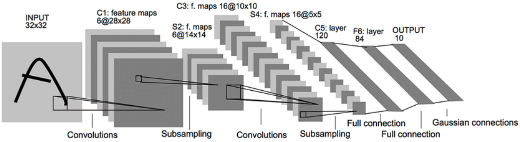
Convolution, Pooling, Stride, Padding

Classiquement : la convolution maintient la taille et le pooling la divise par 2. Cependant, la bonne notion est la notion de **stride** : de combien de case déplace-t-on la fenêtre (1 pour garder la taille et 2 pour diviser par 2) indépendamment du fait que ce soit une convolution ou un pooling.

Attention Dans la convolution, il y a un donc un effet de bord pour $h < \delta_H$ et $h > H - \delta_H$! Une solution à ce problème consiste à considérer au $x = 0$ en dehors de l'emprise initiale. Informatiquement, cela revient à travailler sur une image où on a rajouter δ_H lignes et δ_W colonnes de part et d'autre. On parle de padding qui peut en réalité être régler car cet effet de bord peut être utile.

au final : si l'entrée est $c \times h \times w$ elle devient $c' \times \frac{h+2*pad_h-kernel_h}{stride_h} + 1 \times \frac{w+2*pad_w-kernel_w}{stride_w} + 1$ en passant dans une convolution $c' \times c \times kernel_h \times kernel_w$ avec un padding de pad_h, pad_w et un stride de $stride_h, stride_w$ (idem pooling).

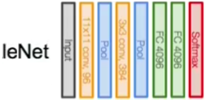
Lenet


INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections

Lenet

Layer		Feature Map	Size	Kernel Size	Stride
Input	Image	1	32x32	-	-
1	Convolution	6	28x28	5x5	1
2	Average Pooling	6	14x14	2x2	2
3	Convolution	16	10x10	5x5	1
4	Average Pooling	16	5x5	2x2	2
5	Convolution	120	1x1	5x5	1
6	FC	-	84	-	-
Output	FC	-	10	-	-

Lenet


leNet
Input
11x11 conv, 96
Pool
3x3 conv, 384
Pool
FC 4096
FC 4096
Softmax

Lenet vs AlexNet vs VGG

MLP vs ConvNet

MLP

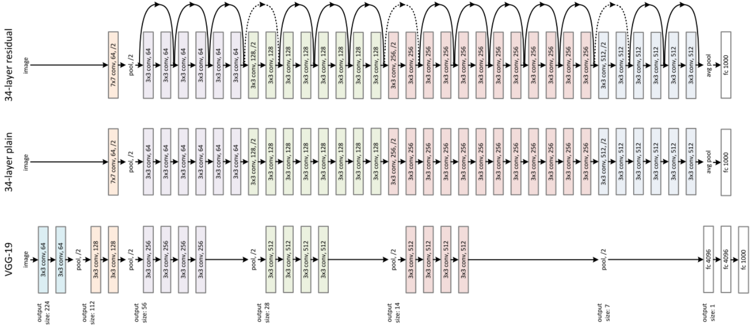
$$f(x, w) = w_Q \times relu(w_{Q-1} \times relu(...(relu(w_1 \times x))))$$

ConvNet

$$f(x, w) = w_Q \times activation(w_{Q-1} \star activation(...(activation(w_1 \star x))))$$

les activations pouvant être des relu et/ou des pooling, les convolutions pouvant avoir ou non des strides et/ou du padding

VGG19 vs ResNet34



j'afficherai pas ResNet100

MLP vs ConvNet

MLP

$$f(x,w) = w_Q \times relu(w_{Q-1} \times relu(...(relu(w_1 \times x))))$$

ConvNet

$f(x,w) =$ un réseau sans cycle qui mélange des convolutions du pooling et des relu

Bilan

ConvNet

- X l'espace des **images**
- P, y, f, w
- $e_t = \frac{1}{M}\sum_m \mathbf{1}_-(y(\chi_m)f(\chi_m, w)) \approx \int_X \mathbf{1}_-(y(x)f(x, w))P(x)\mathrm{d}x$
- $x_1, ..., x_N$ une base d'apprentissage tirée selon P
- l'erreur d'apprentissage $e_a = \frac{1}{N}\sum_n \mathbf{1}_-(y(x_n)f(x_n, w))$
- deep learning = choisir
  $f(x, w) = w_Q \times poolrelu(w_{Q-1} \star poolrelu(...(poolrelu(w_1 \star x))))$
  ou plus généralement un truc qui mélange convolutions, pooling et relu
  et optimiser w pour avoir e_a faible
- c'est ce qui *marche le mieux aujourd'hui*

Questions ouvertes

- comment on optimise w ?