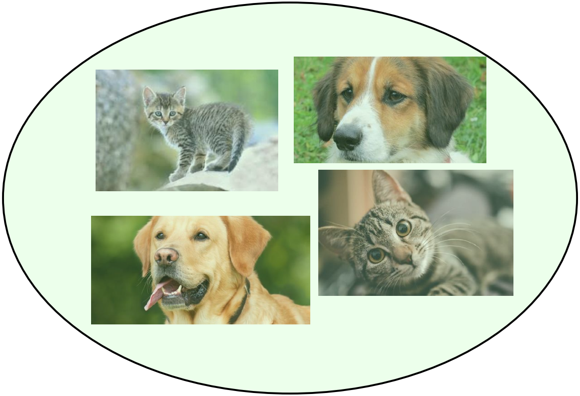
Introduction au deep learning

ENSTA 2024

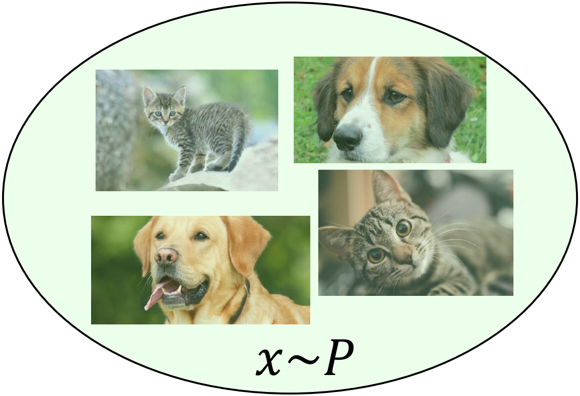
Adrien CHAN-HON-TONG
HDR ONERA

Rappel

Rappel

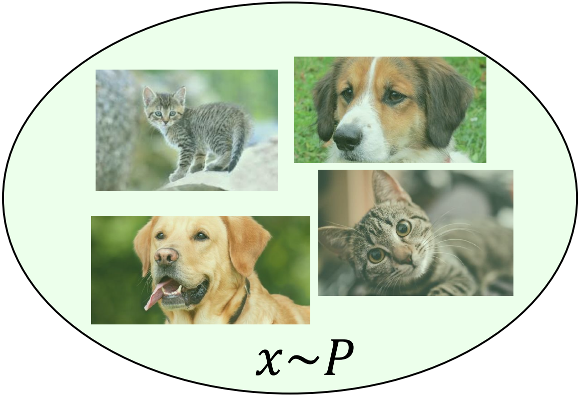


Rappel

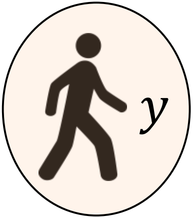


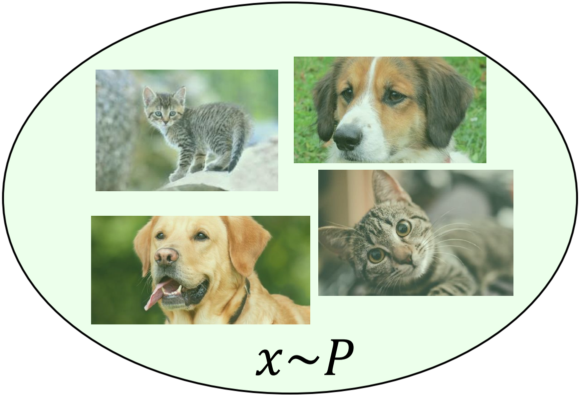
$x \sim P$

Rappel



Rappel

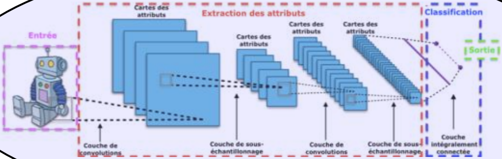




Rappel

y

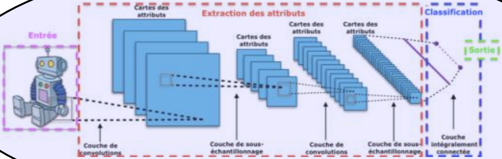
$x \sim P$



Rappel

y

$x \sim P$

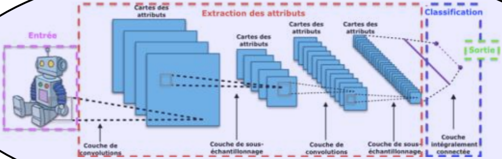


$\mathcal{F}$

Rappel

y

$x \sim P$



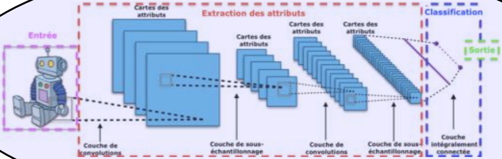
$\mathcal{F}$

NVIDIA®

Rappel

y

$x \sim P$



$\mathcal{F}$

$f \in \mathcal{F}$

NVIDIA

Rappel

Est-ce que

$$f(x) \approx y(x)$$

pour $\quad x \sim P \quad$?

Rappel

Est-ce que

$$f(x) \approx y(x)$$

pour $x \sim P$?

Si *f* n'est pas une bonne approximation sur les données d'apprentissage, c'est de l'underfiting.
Si *f* est une bonne approximation des données d'apprentissage mais pas de *P* c'est de l'overfiting.

Objectif du cours 1

Insister sur la différence entre architecture (i.e. famille de fonction) et fonction pour des MLP

La double descente ?

L'apprentissage en pratique

Objectif du cours 2

pytorch

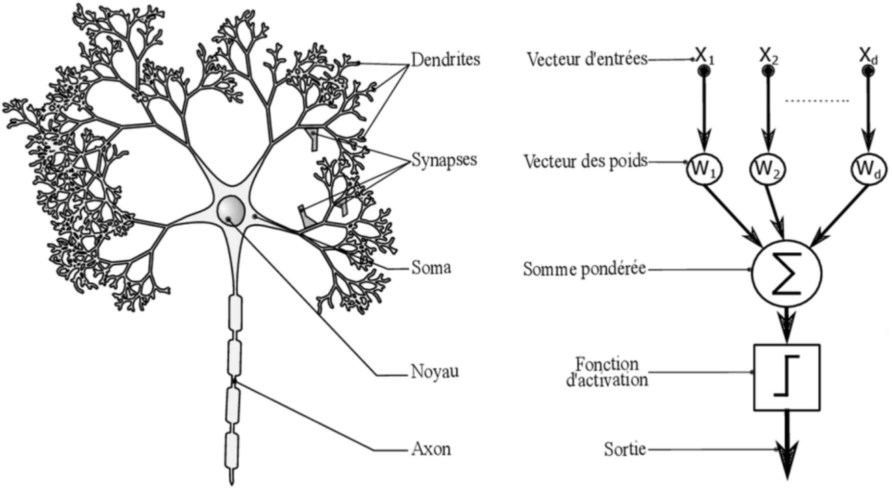
CNN

Transformer et perspective

PLAN

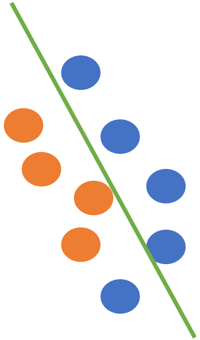
- COURS 1
  - **MLP**
  - Double descente ?
  - Apprentissage
- Cours 2
  - Pytorch
  - CNN
  - Transformer et perspective

Réseau de neurones



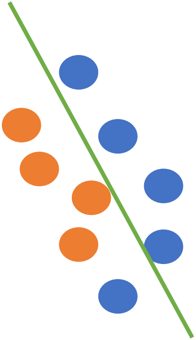
La classification

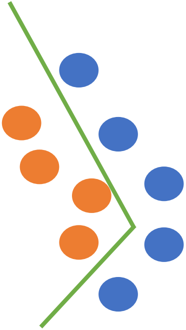
1 neurone
ne peut être que linéaire



La classification

1 neurone
ne peut être que linéaire

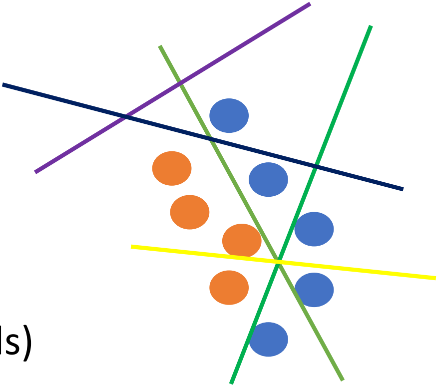




4 neurones + activation
peut être une fonction **non** linéaire

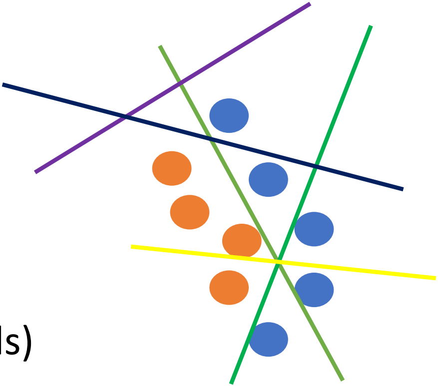
La classification

1 neurone
peut coder TOUTES
ces fonctions
(via la valeur des poids)



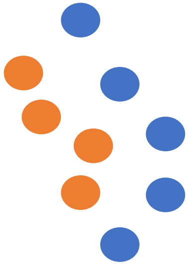
La classification

1 neurone
peut coder TOUTES
ces fonctions
(via la valeur des poids)



Le SVM en est 1 seule fonction
(optimisation des poids)

La classification



4 neurones + activation
=> On ne peut même pas dessiner l’ensemble des fonctions qu’on peut obtenir via les valeurs des poids !

Il va falloir en choisir 1

MLP

Notebook illustration

+

https://playground.tensorflow.org

MLP

1 neurone : $f(x) = w^T x + b = \sum_{d=1}^{D} w_d x_d + b$

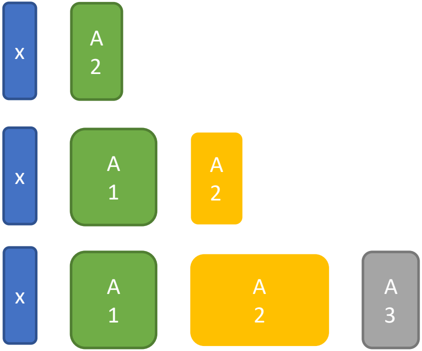
1 couche de 2 neurones + 1 neurone :

$$f(x) = w_3^T \begin{pmatrix} \phi(w_1^T x + b_1) \\ \phi(w_2^T x + b_2) \end{pmatrix} + b_3$$

1 couche de 2 neurones
+ 1 couche de 3 neurones
+ 1 neurone :

$$f(x) = w_6^T \begin{pmatrix} \phi(w_3^T \begin{pmatrix} \phi(w_1^T x + b_1) \\ \phi(w_2^T x + b_2) \end{pmatrix} + b_3) \\ \phi(w_4^T \begin{pmatrix} \phi(w_1^T x + b_1) \\ \phi(w_2^T x + b_2) \end{pmatrix} + b_4) \\ \phi(w_5^T \begin{pmatrix} \phi(w_1^T x + b_1) \\ \phi(w_2^T x + b_2) \end{pmatrix} + b_5) \end{pmatrix} + b_6$$

MLP



1 neurone : $f(x) = Ax + b$

1 couche de 2 neurones + 1 neurone :

$$f(x) = A_2\big(\phi(A_1 x + b_1)\big) + b_2$$

1 couche de 2 neurones
+ 1 couche de 3 neurones
+ 1 neurone :

$$f(x) = A_3\big(\phi(A_2(\phi(A_1 x + b_1)) + b_2)\big) + b_3$$

ϕ d'un vecteur c'est le vecteur des ϕ

MLP

la famille de MLP à activation ϕ et à Q couches de taille $i_1, \ldots, i_Q$
est la famille des fonctions qui peuvent s'écrire

$$f(x) = A_Q\left(\phi\big(A_{Q-1}(\ldots\phi\big(A_2(\phi(A_1x + b_1)) + b_2\big)\ldots) + b_{q-1}\big)\right) + b_q$$

Avec $A_q \in \mathbb{R}^{i_q \times i_{q-1}}$ et $b_q \in \mathbb{R}^{i_q}$

Convention i0 correspond à la dimension des entrées « x »

L'activation la plus classique est : $relu(t) = \max(t, 0)$

MLP

Notebook illustration

+

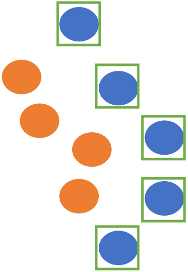
https://playground.tensorflow.org

PLAN

- COURS 1
  - MLP
  - **Double descente ?**
  - Apprentissage
- Cours 2
  - Pytorch
  - CNN
  - Transformer et perspective

Théorème d’universalité

Avec 100 neurones,
on peut faire



Théorème d'universalité

$$\forall x \in \mathbb{R}, |x| = relu(x) + relu(-x)$$

Théorème d'universalité

$$\forall x \in \mathbb{R}, |x| = relu(x) + relu(-x)$$
$$\forall x \in \mathbb{R}^D, ||x||_1 = \mathbf{1}^T \left(relu(Ix) + relu(-Ix)\right)$$

Théorème d'universalité

$$\forall x \in \mathbb{R}, |x| = relu(x) + relu(-x)$$
$$\forall x \in \mathbb{R}^D, ||x||_1 = \mathbf{1}^T\left(relu(Ix) + relu(-Ix)\right)$$
$$\forall x, u \in \mathbb{R}^D, ||x - u||_1 = \mathbf{1}^T\left(relu(Ix - u) + relu(-Ix + u)\right)$$

Théorème d'universalité

$$\forall x \in \mathbb{R}, |x| = relu(x) + relu(-x)$$
$$\forall x \in \mathbb{R}^D, ||x||_1 = \mathbf{1}^T\left(relu(Ix) + relu(-Ix)\right)$$
$$\forall x, u \in \mathbb{R}^D, ||x-u||_1 = \mathbf{1}^T\left(relu(Ix-u) + relu(-Ix+u)\right)$$
$$\forall x, u \in \mathbb{R}^D, relu(1 - ||x-u||_1)$$
$$= relu\left(1 - \mathbf{1}^T\left(relu(Ix-u) + relu(-Ix+u)\right)\right)$$

Théorème d'universalité

$$\forall x \in \mathbb{R}, |x| = relu(x) + relu(-x)$$
$$\forall x \in \mathbb{R}^D, ||x||_1 = \mathbf{1}^T\left(relu(Ix) + relu(-Ix)\right)$$
$$\forall x, u \in \mathbb{R}^D, ||x-u||_1 = \mathbf{1}^T\left(relu(Ix-u) + relu(-Ix+u)\right)$$
$$\forall x, u \in \mathbb{R}^D, relu(1 - ||x-u||_1)$$

$$= relu\left(1 - \mathbf{1}^T\left(relu(Ix-u) + relu(-Ix+u)\right)\right)$$

$$\forall x, u, v \in \mathbb{R}^D, \begin{pmatrix} relu(1-||x-u||_1) \\ relu(1-||x-v||_1) \end{pmatrix}$$

$$= relu\left(\mathbf{1} - \begin{pmatrix} 1 & \dots & 1 & 0 & \dots & 0 \\ 0 & \dots & 0 & 1 & \dots & 1 \end{pmatrix} relu\begin{pmatrix} Ix-u \\ -Ix+u \\ Ix-v \\ -Ix+v \end{pmatrix}\right)$$

$$= \left[\mathbf{1} - \begin{pmatrix} 1 & \dots & 1 & 0 & \dots & 0 \\ 0 & \dots & 0 & 1 & \dots & 1 \end{pmatrix} \begin{bmatrix} Ix-u \\ -Ix+u \\ Ix-v \\ -Ix+v \end{bmatrix}_+\right]_+$$

Théorème d'universalité

Soit la matrice $W \in \mathbb{R}^{N\times(N\times 2D)}$ tel que $\forall n \in \{1,...,N\}, i \in \{1,...,N\times 2D\}$,

$$W_{n,i} = \begin{cases} 1 & ssi \quad 2nD \leq i < 2nD + 2D \\ 0 & sinon \end{cases}$$

alors, $\forall x_1,...,x_N \in \mathbb{R}^D$

$$\left[\mathbf{1} - \begin{pmatrix} ||x - x_1||_1 \\ ||x - x_2||_1 \\ ... \\ ||x - x_N||_1 \end{pmatrix}\right]_+ = \left[\mathbf{1} - W \begin{bmatrix} Ix - x_1 \\ -Ix + x_1 \\ Ix - x_2 \\ -Ix + x_2 \\ ... \\ Ix - x_N \\ -Ix + x_N \end{bmatrix}_+ \right]_+$$

Théorème d'universalité

Soit la matrice $W \in \mathbb{R}^{N\times(N\times 2D)}$ tel que $\forall n \in \{1,...,N\}, i \in \{1,...,N\times 2D\}$,

$$W_{n,i} = \begin{cases} 1 & ssi \quad 2nD \leq i < 2nD + 2D \\ 0 & sinon \end{cases}$$

alors, $\forall x_1, ..., x_N \in \mathbb{R}^D$

$$\left[\mathbf{1} - \begin{pmatrix} ||x - x_1||_1 \\ ||x - x_2||_1 \\ ... \\ ||x - x_N||_1 \end{pmatrix}\right]_+ = \left[\mathbf{1} - W \begin{bmatrix} Ix - x_1 \\ -Ix + x_1 \\ Ix - x_2 \\ -Ix + x_2 \\ ... \\ Ix - x_N \\ -Ix + x_N \end{bmatrix}_+\right]_+$$

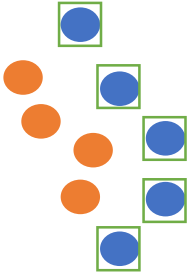
$\forall x_1, ..., x_N \in \mathbb{Z}^D, y_1, ..., y_N \in \{-1, 1\}$ la fonction

$$f(x) = \begin{pmatrix} y_1 \\ y_2 \\ ... \\ y_N \end{pmatrix}^T \left[\mathbf{1} - W \begin{bmatrix} Ix - x_1 \\ -Ix + x_1 \\ Ix - x_2 \\ -Ix + x_2 \\ ... \\ Ix - x_N \\ -Ix + x_N \end{bmatrix}_+\right]_+$$

vérifie $\forall n \in \{1, ..., N\},\ y_n f(x_n) > 0$.

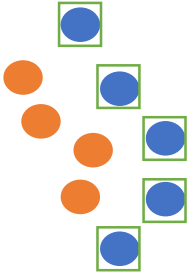
Théorème d'universalité

Avec 100 neurones,
on peut faire ça



Théorème d’universalité

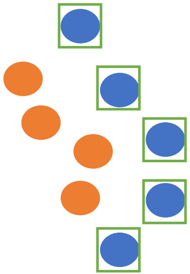
Avec 100 neurones,
on peut faire ça



→ Sauf que c’est la pire chose possible

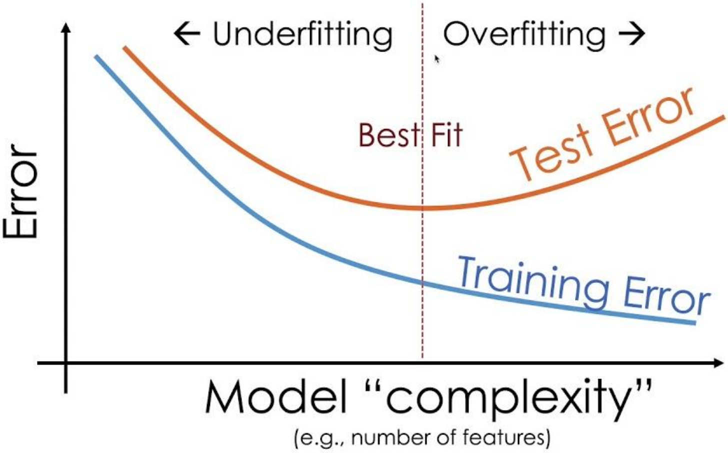
Théorème d’universalité

Avec 100 neurones,
on peut faire ça

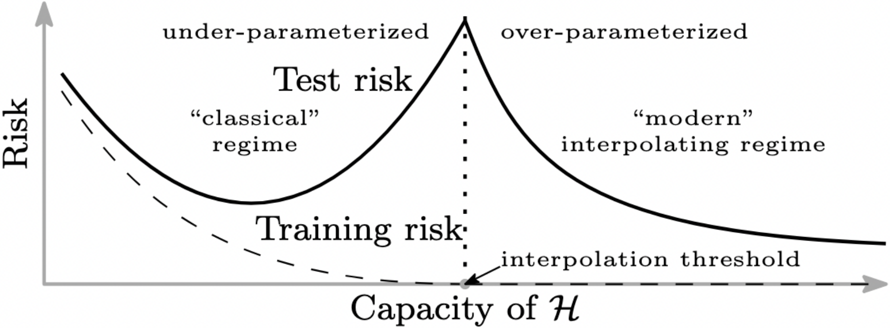


- Sauf que c’est la pire chose possible
- Il se trouve que ça arrive pas ????

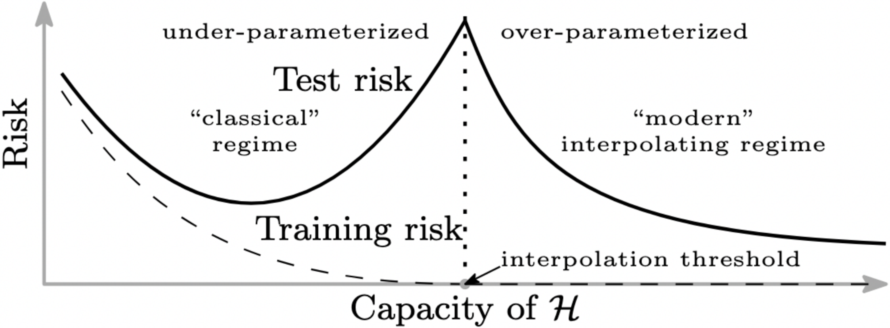
Ancien paradigme



Ce qu'on observe



Ce qu'on observe

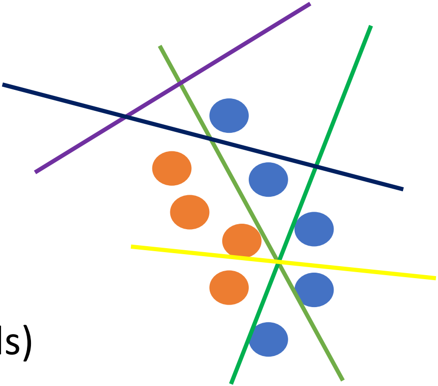


On pourrait penser que limiter le nombre de neurones est le moyen de limiter le sur-apprentissage.
En pratique, c'est plus compliqué.

PLAN

- COURS 1
  - MLP
  - Double descente ?
  - **Apprentissage**
- Cours 2
  - Pytorch
  - CNN
  - Transformer et perspective

Rappel



1 neurone
peut coder TOUTES
ces fonctions
(via la valeur des poids)

Le SVM en est 1 seule fonction
(optimisation des poids)

https://playground.tensorflow.org

Comment apprendre un MLP en pratique ?

- Descente de gradient
- Descente de gradient stochastique
- Fonction de perte
- Calcul du gradient

Descente de gradient

$$J(u+\varepsilon) \approx J(u) + [\nabla_u J]^T \varepsilon + \varepsilon o(\varepsilon)$$

Descente de gradient

$$J(u+\varepsilon) \approx J(u) + [\nabla_u J]^T \varepsilon + \varepsilon o(\varepsilon)$$

$$\begin{gathered} J(u - \lambda \nabla_u J) \\ \approx\ J(u) - \lambda [\nabla_u J]^T [\nabla_u J] + \lambda o(\lambda) \\ \approx\ J(u) - \|\nabla_u J\|^2 \times \lambda + \lambda o(\lambda) \end{gathered}$$

Descente de gradient

$$J(u+\varepsilon) \approx J(u) + [\nabla_u J]^T \varepsilon + \varepsilon o(\varepsilon)$$

$$\begin{gathered} J(u - \lambda \nabla_u J) \\ \approx\ J(u) - \lambda [\nabla_u J]^T [\nabla_u J] + \lambda o(\lambda) \\ \approx\ J(u) - \|\nabla_u J\|^2 \times \lambda + \lambda o(\lambda) \end{gathered}$$

$$\begin{gathered} \nabla_u J \neq 0 \Longrightarrow \exists \lambda > 0, \\ J(u - \lambda \nabla_u J) < J(u) \end{gathered}$$

Descente de gradient

- Initialiser u
- Si $\|\nabla_u J\| < \delta$, sortir
- Sinon
  - Initialiser λ
  - Si $\mathrm{J}(\mathrm{u} - \lambda \nabla_u J) < \mathrm{J}(\mathrm{u})$, alors $\mathrm{u} = \mathrm{u} - \lambda \nabla_u J$
  - Sinon réessayer avec $\lambda = \frac{\lambda}{2}$

Descente de gradient

- Initialiser u
- Si $\|\nabla_u J\| < \delta$, sortir
- Sinon
  - Initialiser λ
  - Si $\mathrm{J}(\mathrm{u} - \lambda \nabla_u J) < \mathrm{J}(\mathrm{u})$, alors $\mathrm{u} = \mathrm{u} - \lambda \nabla_u J$
  - Sinon réessayer avec $\lambda = \frac{\lambda}{2}$

⇒ Converge vers un point u tel que $\|\nabla_u J\| < \delta$

Si la fonction est bornée

Descente de gradient

$$u = A^{-1}b$$

$$\min_{u} ||Au - b||^2$$

1. $u = 0$
2. Si $||2A^T(Au - b)|| < 10^{-6}$, sortir
3. $u = u - \lambda A^T(Au - b)$, GOTO2

Descente de gradient

$$u = A^{-1}b$$

$$\min_{\mathrm{u}} \|Au - b\|^2$$

$O(N^{\gamma}L), \qquad \gamma \approx 2.34$

1. $u = 0$
2. Si $\|2A^T(Au - b)\| < 10^{-6}$, sortir
3. $u = u - \lambda A^T(Au - b)$, GOTO2

$O(N^2 2^L L)$

Descente de gradient **stochastique**

$$J(u) = \sum_{k=1}^{K} j_k(u)$$

Si je tire *k* uniformément dans {1,…,*K*}, et que $g = j_k(u)$

$$\mathbf{E}[\nabla g] = \frac{1}{\mathrm{K}} \sum_{k=1}^{K} \nabla j_k = \nabla J$$

Descente de gradient **stochastique**

$$J(u) = \sum_{k=1}^{K} j_k(u)$$

Si je tire *k* uniformément dans {1,…,*K*}, et que $g = j_k(u)$

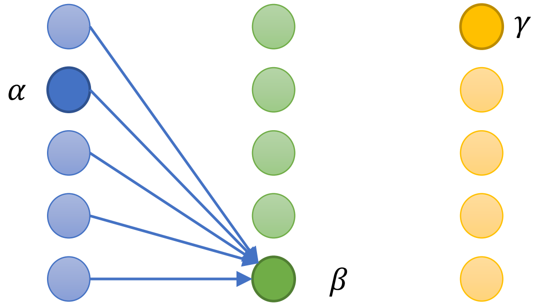
$$\mathbf{E}[\nabla g] = \frac{1}{\mathrm{K}} \sum_{k=1}^{K} \nabla j_k = \nabla J$$

➡ Si *K* est très très grand, il n'y a pas d'alternative à chercher une approximation probabiliste du gradient

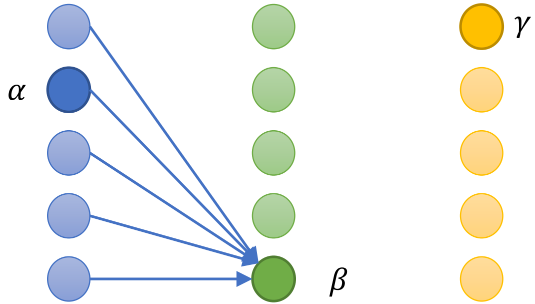
Descente de gradient **stochastique** pour un MLP

- Choisir une fonction de perte *J*
- Initialiser aléatoirement les poids *w*
- Faire un certain nombre de fois
  - Sélectionner un paquet de données *X*
  - Comparer *f(X)* et *y(X)* à travers *J*
  - Calculer le gradient correspondant *dw*
  - *w = w – lr * dw*

Calcul du gradient ?

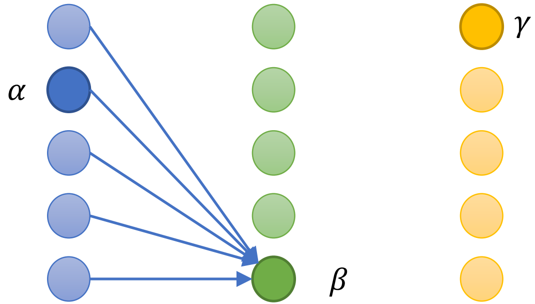


Calcul du gradient ?



$$\beta = w_{\{\beta,\alpha\}}\alpha + \cdots$$
$$\gamma = w_{\{\gamma,\beta\}}\beta + \cdots$$

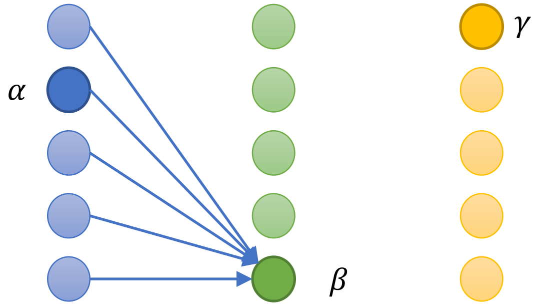
Calcul du gradient ?



$$\beta = w_{\{\beta,\alpha\}}\alpha + \cdots \qquad \gamma\,(\alpha, \ldots) = \gamma\,(\beta(\alpha, \ldots), \ldots)$$

$$\gamma = w_{\{\gamma,\beta\}}\beta + \cdots$$

Calcul du gradient ?



$$\beta = w_{\{\beta,\alpha\}}\alpha + \cdots$$
$$\gamma = w_{\{\gamma,\beta\}}\beta + \cdots$$

$$\gamma\,(\alpha, \ldots) = \gamma\,(\beta(\alpha, \ldots), \ldots)$$

$$\frac{\partial\gamma}{\partial\alpha} = \frac{\partial\gamma}{\partial\beta}\frac{\partial\beta}{\partial\alpha} + \cdots$$

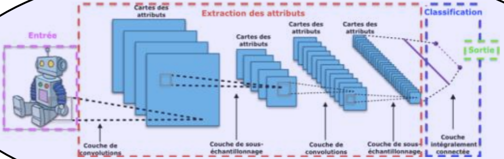
Calcul du gradient

Et en pratique à la fin c’est pytorch qui le calcul pour vous !

Rappel

y

$x \sim P$



$\mathcal{F}$

$f \in \mathcal{F}$

NVIDIA

Pourquoi une fonction de perte ?

Ce qu’on veut c’est trouver w tel que

$$f_w(x) \approx y(x)$$

sur la base d’apprentissage.

Pourquoi une fonction de perte ?

Ce qu'on veut c'est trouver w tel que

$$f_w(x) \approx y(x)$$

sur la base d'apprentissage.

⇒ 1 erreur c'est χ tel que $y(\chi)f_w(\chi) \leq 0$

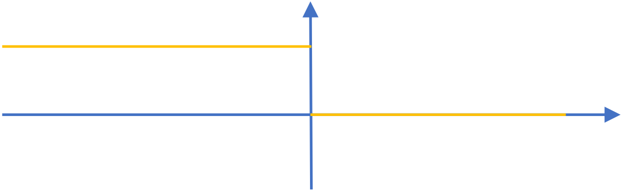
Pourquoi une fonction de perte ?

Ce qu'on veut c'est trouver w tel que

$$f_w(x) \approx y(x)$$

sur la base d'apprentissage.

1 erreur c'est χ tel que $y(\chi)f_w(\chi) \leq 0$



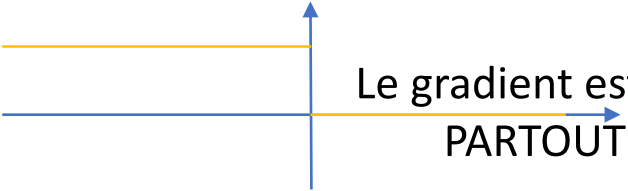
Pourquoi une fonction de perte ?

Ce qu'on veut c'est trouver w tel que

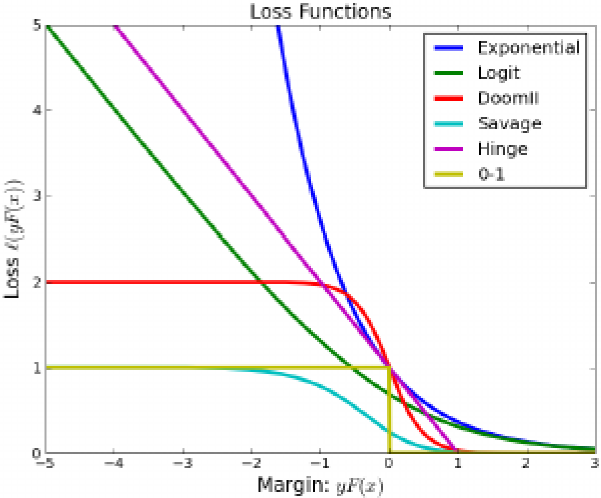
$$f_w(x) \approx y(x)$$

sur la base d'apprentissage.

1 erreur c'est χ tel que $y(\chi)f_w(\chi) \leq 0$

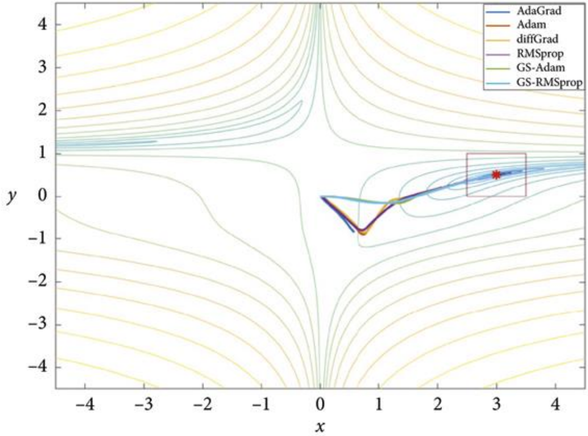


fonction de perte



https://playground.tensorflow.org

fonction de perte



https://playground.tensorflow.org

Bilan : MLP

- Choisir une architecture
- Choisir une fonction de perte
- Choisir une variante de la SGD
- Effectuer une SGD sur les données d’apprentissage (en utilisant les gradients calculés par pytorch)

Bilan : MLP

- Choisir une architecture MLP
- Choisir une fonction de perte
- Choisir une variante de la SGD
- Effectuer une SGD sur les données d'apprentissage (en utilisant les gradients calculés par pytorch)

PLAN

- COURS 1
  - MLP
  - Double descente ?
  - Apprentissage
- Cours 2
  - **Pytorch**
  - CNN
  - Transformer et perspective

Pytorch

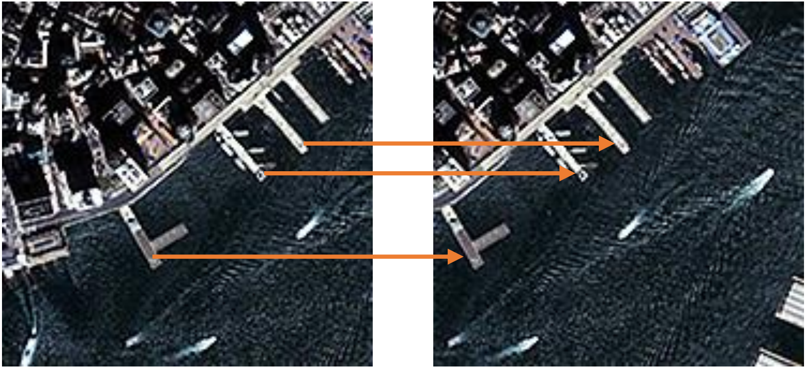
Voir notebooks

PLAN

- COURS 1
  - MLP
  - Double descente ?
  - Apprentissage
- Cours 2
  - Pytorch
  - **CNN**
  - Transformer et perspective

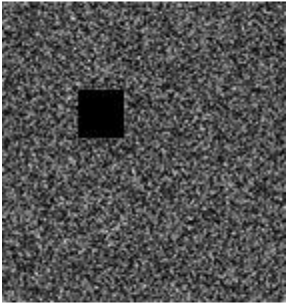
À la recherche de l'information visuelle

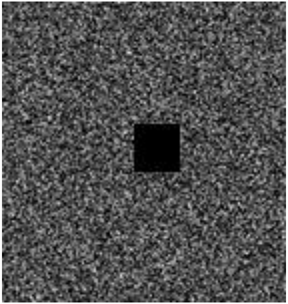
À la recherche de l'information visuelle



À la recherche de l’information visuelle

À la recherche de l'information visuelle





À la recherche de l’information visuelle

À la recherche de l'information visuelle : les contours ?

À la recherche de l’information visuelle : les contours ?

100	100	100	100	100
100	100	100	100	100
100	100	150	100	100
100	100	100	100	100
100	100	100	100	100

*

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

=

100	100	100	100	100
100	100	50	100	100
100	50	350	50	100
100	100	50	100	100
100	100	100	100	100

À la recherche de l'information visuelle : les contours ?

$$\mathbf{G_x} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \mathbf{A} \quad \text{et} \quad \mathbf{G_y} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * \mathbf{A}$$

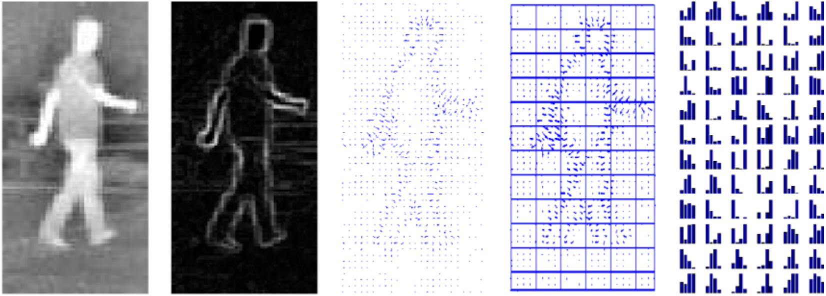
À la recherche de l'information visuelle : les contours ?

Image

Bords

Reconnaissance des formes

À la recherche de l'information visuelle : les contours ?



a b c d e

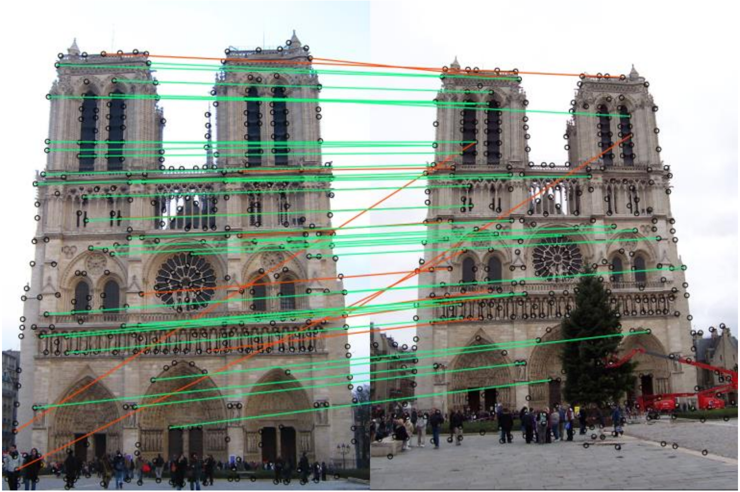
À la recherche de l'information visuelle : les contours ?

Image → Bords → Histogrammes

→ Représentation globale → Classification

À la recherche de l'information visuelle : coin de Harris

À la recherche de l'information visuelle : coin de Harris



À la recherche de l’information visuelle : coin de Harris

Robert Collins
CSE486, Penn State

Harris Corner Detection Algorithm

1. Compute x and y derivatives of image

$$I_x = G_\sigma^x * I \quad I_y = G_\sigma^y * I$$

2. Compute products of derivatives at every pixel

$$I_{x2} = I_x . I_x \quad I_{y2} = I_y . I_y \quad I_{xy} = I_x . I_y$$

3. Compute the sums of the products of derivatives at each pixel

$$S_{x2} = G_{\sigma'} * I_{x2} \quad S_{y2} = G_{\sigma'} * I_{y2} \quad S_{xy} = G_{\sigma'} * I_{xy}$$

4. Define at each pixel (x, y) the matrix

$$H(x,y) = \begin{bmatrix} S_{x2}(x,y) & S_{xy}(x,y) \\ S_{xy}(x,y) & S_{y2}(x,y) \end{bmatrix}$$

5. Compute the response of the detector at each pixel

$$R = Det(H) - k(Trace(H))^2$$

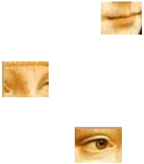
6. Threshold on value of R. Compute nonmax suppression.

À la recherche de l'information visuelle : rappel

$$\mathbf{G_x} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \mathbf{A} \quad \text{et} \quad \mathbf{G_y} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * \mathbf{A}$$

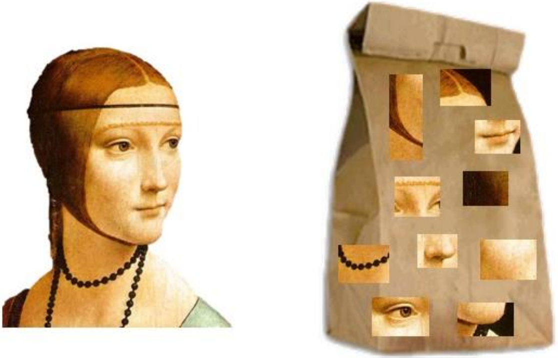
À la recherche de l’information visuelle

À la recherche de l’information visuelle : sac de mots

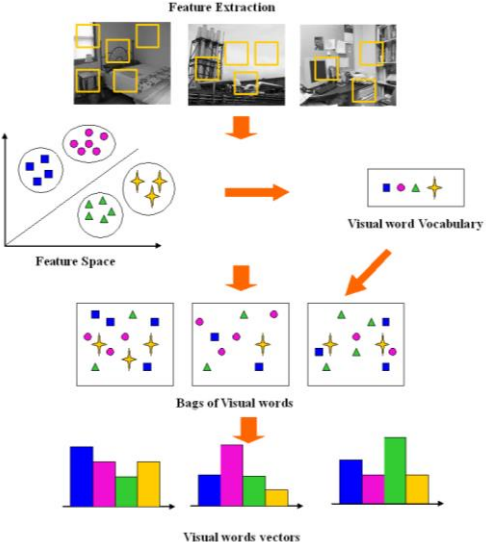


À la recherche de l'information visuelle : sac de mots

À la recherche de l'information visuelle : sac de mots



À la recherche de l'information visuelle : sac de mots



À la recherche de l’information visuelle : sac de mots

Phase1

Images → Mot**s** → Dictionnaire

À la recherche de l'information visuelle : sac de mots

Phase1

Images → Mots → Dictionnaire

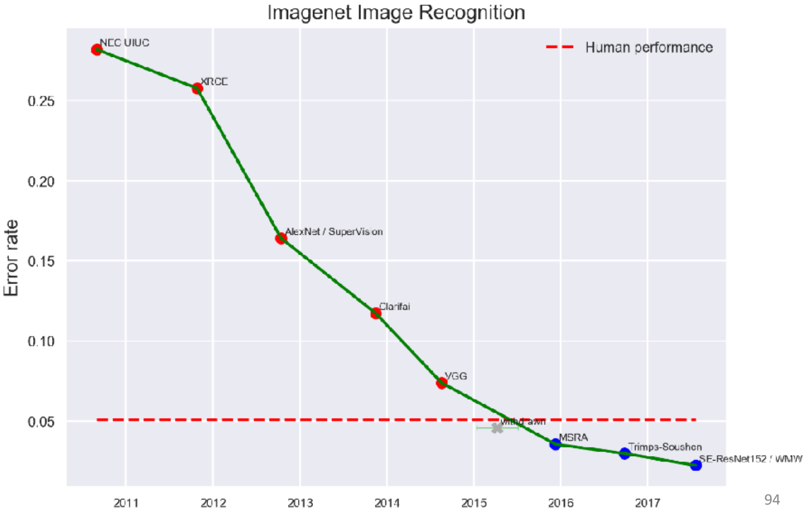
Phase2

Image → Mots → Histogramme

↓

Classification

À la recherche de l'information visuelle : apprentissage profond



À la recherche de l'information visuelle : apprentissage profond

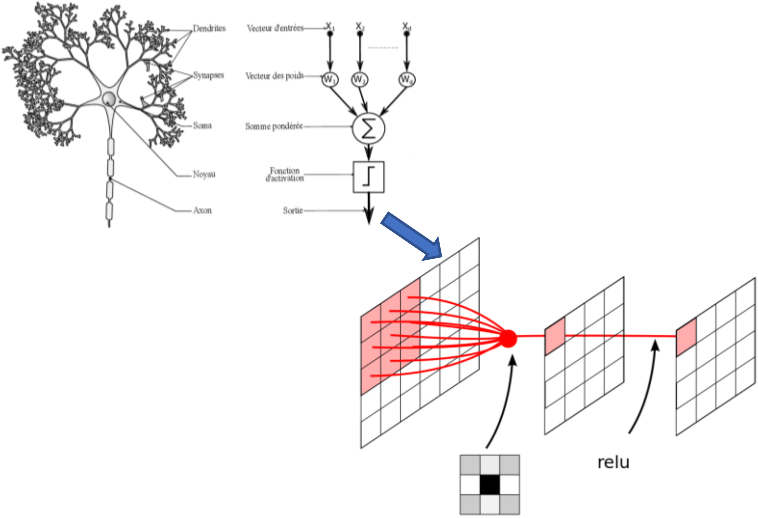
Image $\longrightarrow$ Classification

À la recherche de l'information visuelle : apprentissage profond

Image → Classification

C'est le réseau qui trouve l'information visuelle et l'exploite en même temps

ConvNet : convolution



ConvNet

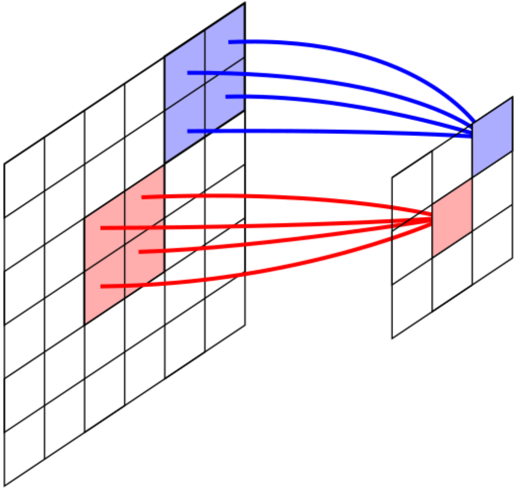
1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

	1	
1	-4	1
	1	

4	3	2	1	-6
-5	0	0	0	-11
-10	0	0	0	-16
-15	0	0	0	-21
-46	-27	-28	-29	-56

4	3	2	1	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

ConvNet : pooling

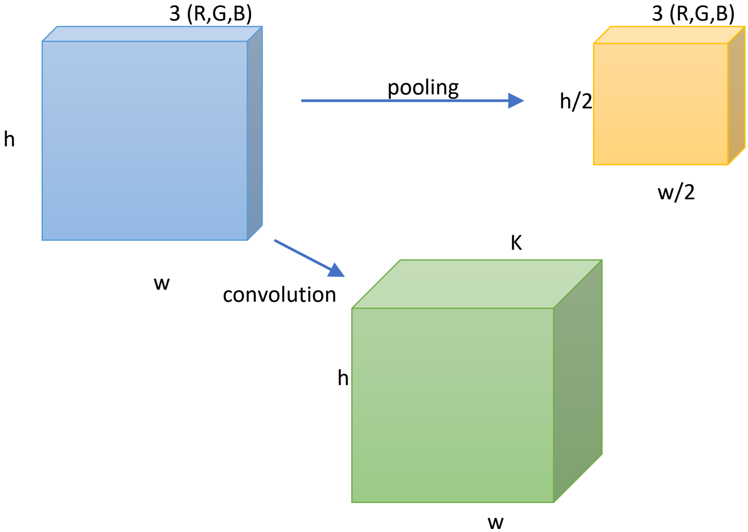


ConvNet

1	2	3	4
6	7	8	9
11	12	13	14
16	17	18	19

7	9
17	19

Attention : on oublie souvent la dimension spectrale



Attention, on manipule des paquets (donc 4D)

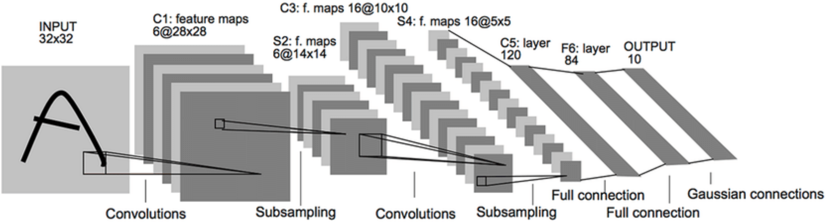
paquet

dimensions spatiales

$$x \in \mathbb{R}^{B \times C \times H \times W}$$

dimension spectrale

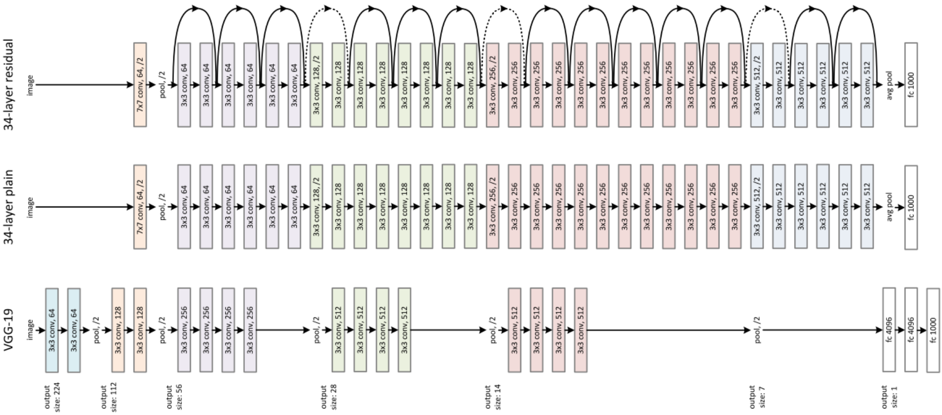
Lenet



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

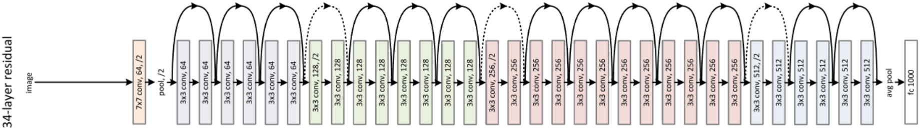
Alexnet et VGG

Resnet



34-layer residual
34-layer plain
VGG-19
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
avg pool
fc 1000
3x3 conv, 64
3x3 conv, 128
3x3 conv, 256
3x3 conv, 512
fc 4096
output size: 224
output size: 112
output size: 56
output size: 28
output size: 14
output size: 7
output size: 1

Resnet

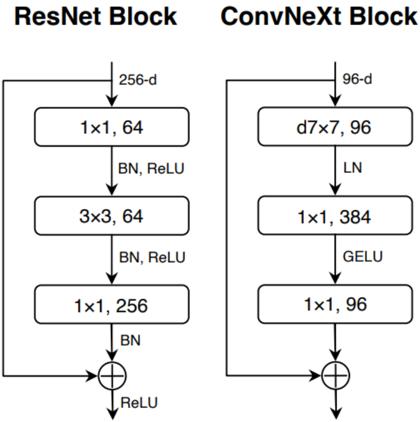


$$BN(x) = \frac{x - \left(\sum_b x_b\right)}{\sqrt{\sum_b \left(x_b - \left(\sum_{b'} x_{b'}\right)\right)^2} + 0.0000001}$$

EfficientNet, ConvNext

https://pytorch.org/vision/stable/models.html

EfficientNet, ConvNext



EfficientNet, ConvNext

https://pytorch.org/vision/stable/models.html

PLAN

- COURS 1
  - MLP
  - Double descente ?
  - Apprentissage
- Cours 2
  - Pytorch
  - CNN
  - **Transformer et perspective**

Transformer

Si $H = h_1, \dots, h_R$ sont *R* vecteurs de dimension *D*, la self attention s'exprime comme

$$\mathrm{S}(H) = softmax\left(\frac{(QH)(KH)^T}{\sqrt{L}}\right)(VH)$$

avec $\quad \mathrm{Q}, \mathrm{K} \in \mathbb{R}^{L \times D}$

Transformer

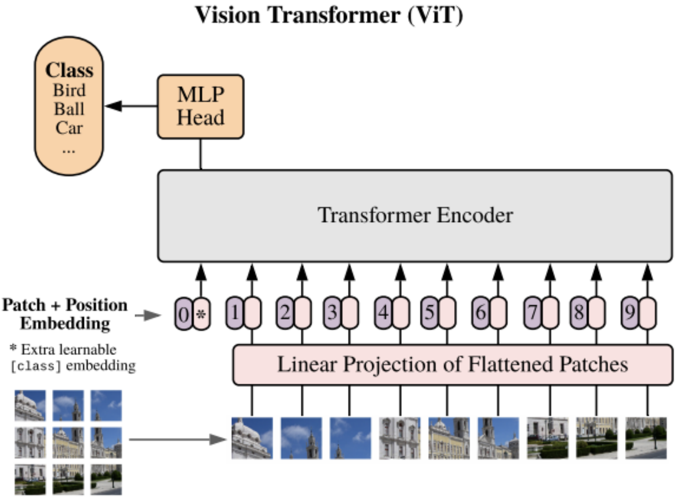
Si $H = h_1, \ldots, h_R$ sont *R* vecteurs de dimension *D*, la self attention s'exprime comme

$$\mathrm{S}(H) = softmax\left(\frac{(QH)(KH)^T}{\sqrt{L}}\right)(VH)$$

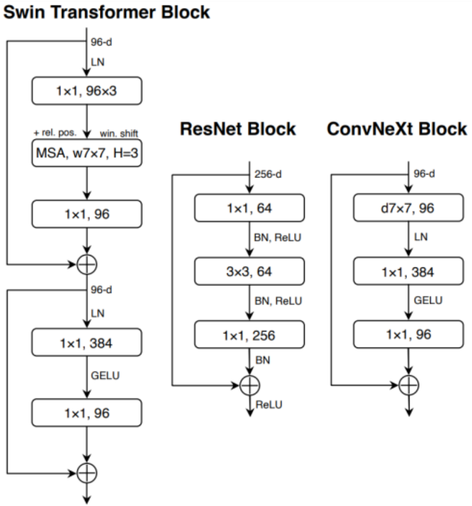
avec $\quad \mathrm{Q}, \mathrm{K} \in \mathbb{R}^{L \times D}$

⇨ Le nombre de poids ne dépend pas de *R* !

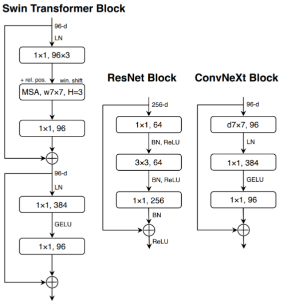
Visual transformer



ViT vs ConvNet



ViT vs ConvNet



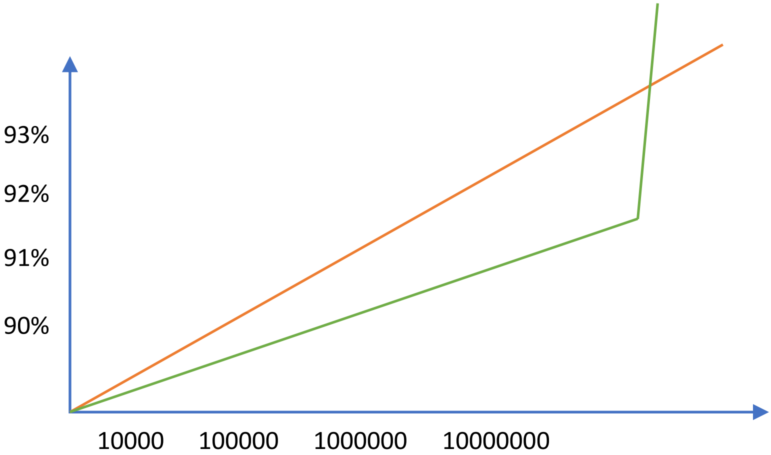
$$\mathrm{S}(H) = softmax\left(\frac{(QH)(KH)^T}{\sqrt{L}}\right)(VH)$$

Augmenter brutalement la taille n’apporte pas tant que cela sur des architectures convolutives, alors que l’augmentation de performance semble constante avec des architectures Transformer !

L’état de l’art de la vision par ordinateur

https://pytorch.org/vision/stable/models.html

Performance vs tailles-données-modèles



??

Fondation models

EfficientNet

100 Mo

SAM (Segment Everything by Meta)

2Go de poids

21Go pour faire un batch de 2 sur le petit modèle

LLAMA (un concurrent de chatGPT)

128 Go de poids !!!!!!!!!!!!!!!!!!!!!!!

8 x 32Go pour le faire tourner

200000€

Fondation models

Sam Scarpino
@svscarpino

"Most good ideas still come from academia." 💯💯💯

Prof @ylecun on the role of academia in AI and the importance of good ideas (even if you don't have access to 50k gpus for compute). Fireside chat @Northeastern with @Experiential_AI's @usamaf.

Traduire le Tweet

8:32 PM · 24 mai 2023 depuis Boston, MA · **25,5 k** vues

Fondation models

- Lenet	AT&T
- Alexnet	toronto / google
- VGG	Oxford
- Resnet	Microsoft
- Gpipe	Google
- Efficientnet	Google
- ViT	Google
- SWIM	Microsoft
- ConvNext	Meta

Sam Scarpino
@svscarpino

"Most good ideas still come from academia." 💯💯💯

Prof @ylecun on the role of academia in AI and the importance of good ideas (even if you don't have access to 50k gpus for compute). Fireside chat @Northeastern with @Experiential_AI's @usamaf.

Traduire le Tweet

8:32 PM · 24 mai 2023 depuis Boston, MA · **25,5 k** vues

Perspectives

Il existe de très bon modèles convolutifs (EfficientNet, ConvNext) mais leurs performances saturent

L’utilisation de couche Transformer
est autant prometteuse que couteuse !

Le passage à l’échelle ne se fera pas dans les labos,
car les ordres de grandeur mis en jeux ont explosé !

Merci pour votre attention