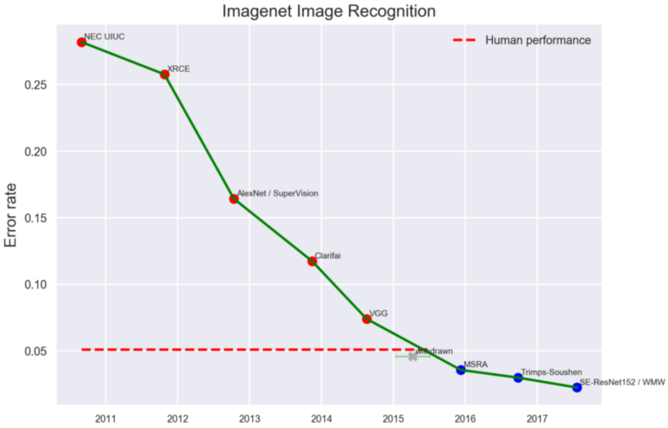
Concept de base du deep learning

Adrien CHAN-HON-TONG
ONERA

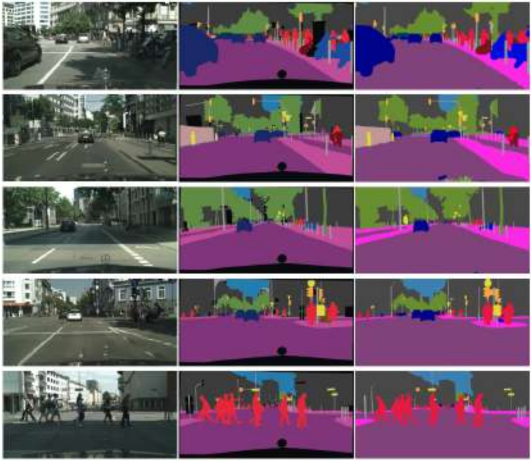
Introduction

La révolution deep learning



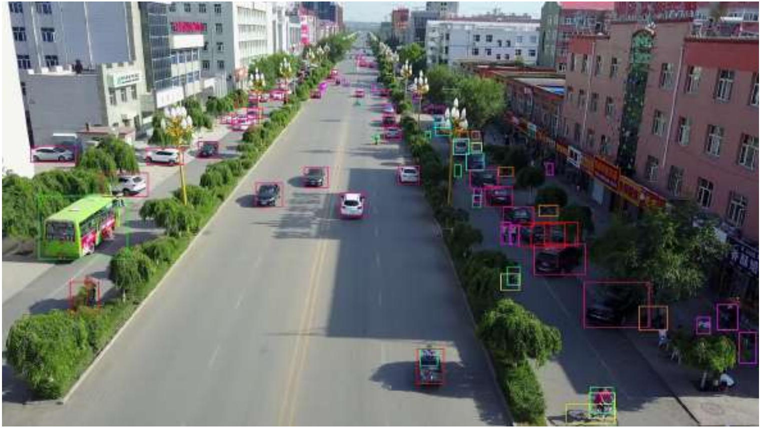
Introduction

Shock and Awe



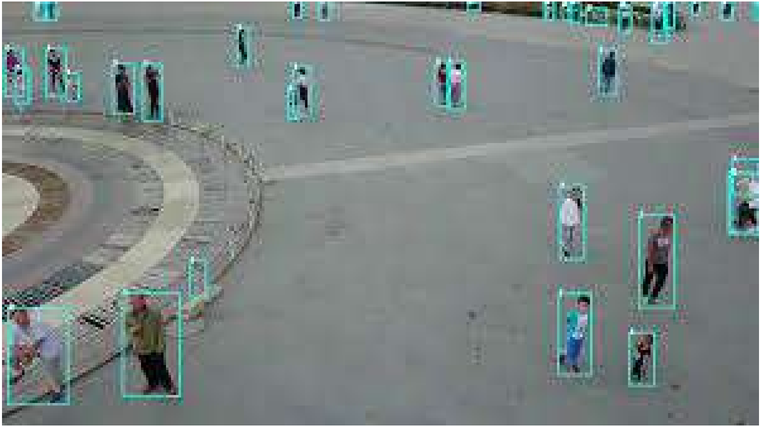
Introduction

Shock and Awe



Introduction

Shock and Awe



Introduction

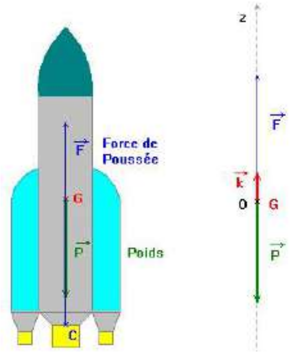
Shock and Awe

Introduction

Le premier satellite lancé dans l'espace (Spoutnik) - 1957

La théorie permettant de calculer la trajectoire de sortie
Newton et Descarte - 1650

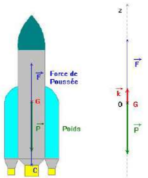
Introduction

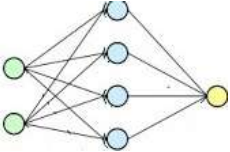


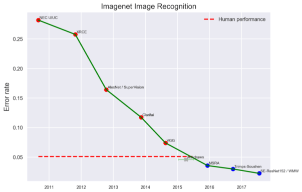
1650

1992

Introduction







Plan du cours

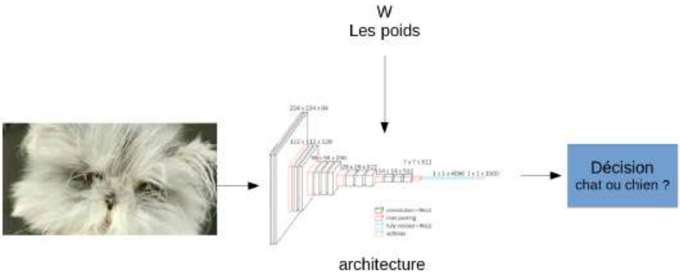
- Le neurone classique
- Le neurone convolutif
- L'apprentissage en pratique
  - loss function
  - optimiseur
  - backpropagation
- exemples de recherche

Plan du cours

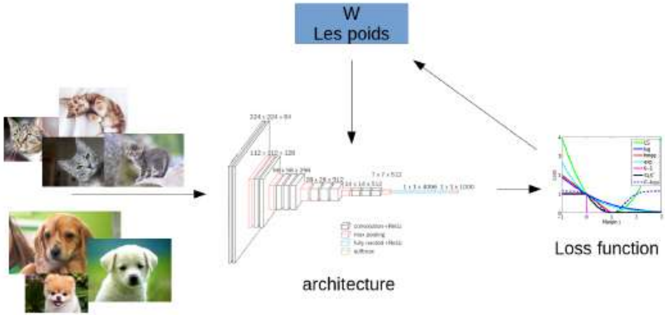
- Le neurone classique
- Le neurone convolutif
- L'apprentissage en pratique
  - loss function
  - optimiseur
  - backpropagation
- exemples de recherche

Mais le plus important c'est de comprendre le formalisme apprentissage vs utilisation

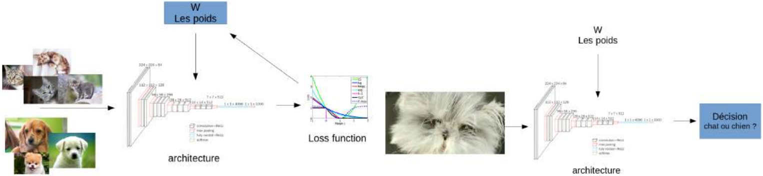
Test et/ou production et/ou inférence



Apprentissage



Apprentissage vs test



On le refait avec des maths au lieu d'un schéma...

La classification binaire

P une distribution sur X un espace, y la fonction cible dans {-1,1}

Apprentissage

- on tire une base d'apprentissage $x_1, \ldots, x_N$
- pour lesquels on va utiliser des humains pour connaitre $y(x_n)$
- on choisit une fonction f paramétrés par des poids
- l'apprentissage consiste (+/-) à trouver w tel que

$$e_a = \frac{1}{N} \sum_n \mathbf{1}_-(y(x_n) f(x_n, w)) \ll 1$$

La classification binaire

Test

- à la fin de l'apprentissage, on fixe w et on espère que

$$\forall \chi,\ y(\chi) f(\chi, w) > 0$$

- mais c'est peine perdu car plusieurs fonctions y conduisent à la même base d'apprentissage !
- il y a donc une erreur

$$e_r = \int_X \mathbf{1}_{-}(y(\chi) f(\chi, w)) P(\chi) \mathrm{d}\chi$$

La classification binaire

Erreur empirique

Mais comment calculer $e_r = \int_X \mathbf{1}_-(y(\chi)f(\chi, w))P(\chi)\mathrm{d}\chi$ alors qu'on ne connait ni P ni y ?

La classification binaire

Erreurs empirique

- Mais comment calculer $e_r = \int_X \mathbf{1}_-(y(\chi)f(\chi, w))P(\chi)\mathrm{d}\chi$ alors qu'on ne connait ni P ni y ? $\Rightarrow$ **on ne peut PAS.**
- à la place on tire une base de test $\chi_1, ..., \chi_M$
- pour lesquels on va utiliser des humains pour connaitre $y(\chi_m)$ et on calcule une erreur de test (ou erreur empirique)

$$e_t = \frac{1}{M}\sum_m \mathbf{1}_-(y(\chi_m)f(\chi_m, w)) \underset{M\to\infty}{\to} e_r$$

La classification binaire

Erreurs

$$e_r = \int_X \mathbf{1}_{-}(y(\chi)f(\chi, w))P(\chi)\mathrm{d}\chi$$

$$e_t = \frac{1}{M}\sum_m \mathbf{1}_{-}(y(\chi_m)f(\chi_m, w)) \underset{M\to\infty}{\to} e_r$$

$$e_a = \frac{1}{N}\sum_n \mathbf{1}_{-}(y(x_n)f(x_n, w)) \ll 1$$

La classification binaire

Erreurs

$$e_r = \int_X \mathbf{1}_-(y(\chi)f(\chi, w))P(\chi)\mathrm{d}\chi$$

$$e_t = \frac{1}{M}\sum_m \mathbf{1}_-(y(\chi_m)f(\chi_m, w)) \underset{M\to\infty}{\to} e_r$$

$$e_a = \frac{1}{N}\sum_n \mathbf{1}_-(y(x_n)f(x_n, w)) \ll 1$$

0n veut e_r petit alors qu'on ne sait ni la calculer (on ne peut calculer que e_t) ni agir dessus (on ne peut que minimiser e_a).

En pratique l'objectif est de trouver une fonction f tel que **sur les problèmes qu'on veut résoudre** (non défini)

- $e_a \approx e_r$ (si $e_a \ll e_r$ on est en overfitting)
- on peut trouver w tel que $e_a \approx 0$ (si $e_a \gg 0$ on est en underfitting)

Aujourd'hui ce qui marche le mieux (mais néanmoins pas tout le temps) c'est le deep learning.

Synthèse

La classification binaire

P une distribution sur X un espace, y la fonction cible dans {-1,1}

- on approxime y par le signe de f (avec des poids w)
- erreur réelle $e_r = \int_X \mathbf{1}_-(y(\chi)f(\chi,w))P(\chi)\mathrm{d}\chi$
- $x_1, ..., x_N$ une base d'apprentissage tirée selon P
- l'erreur d'apprentissage $e_a = \frac{1}{N}\sum_n \mathbf{1}_-(y(x_n)f(x_n,w))$
- apprentissage : w optimisé tel que $e_a \approx 0$
- $\chi_1, ..., \chi_M$ une base de test tirée selon P
- w fixé, l'erreur de test $e_t = \frac{1}{M}\sum_m \mathbf{1}_-(y(\chi_m)f(\chi_m,w)) \approx e_r$

La classification binaire

Exemple

X c'est l'espace des images, P c'est leur probabilité d'être sur internet, y c'est la fonction qui dit si une image est un chat (+1) ou pas (-1).

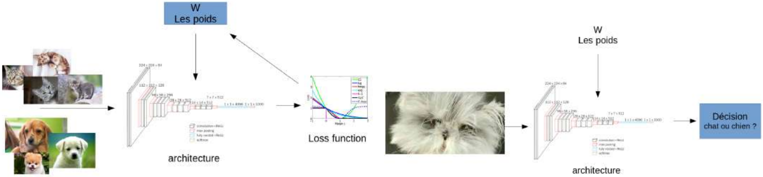
y n'a aucun sens mathématique c'est une fonction anthropique ! Même avec une puissance de calcul infini, vous ne pouvez ni définir ni calculer y, pourtant n'importe quelle humain peut l'évaluer !

Donc, on ne la connaît pas, mais avec un humain, on peut l'évaluer sur une image donnée.
Donc faire une base d'apprentissage et/ou de test !

Plan du cours

- *La classification binaire*
- Le neurone classique
- Le neurone convolutif
- L'apprentissage en pratique
  - loss function
  - optimiseur
  - backpropagation
- exemples de recherche

Du neurone au réseau



Si l'architecture est :

- un opérateur linéaire, c'est du SVM ou du boosting
- un arbre d'opérateurs simples, c'est du random tree
- un réseau de neurones, c'est du deep learning

Du neurone au réseau

Le neurone

Dans les architectures de réseaux de neurones (profond ou pas), le neurone est un filtre linéaire :

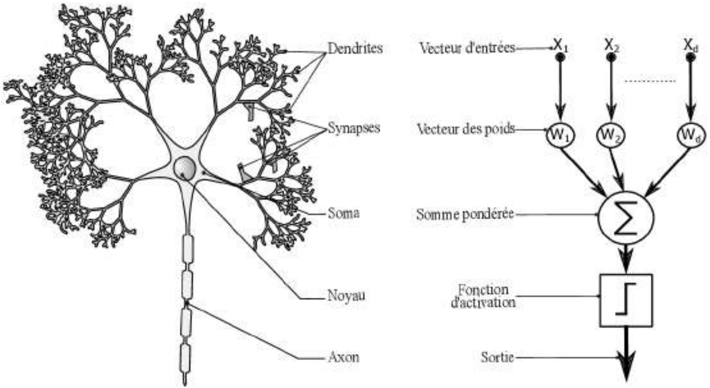
$$neurone_{\alpha,\beta} : \begin{array}{ccc} \mathbb{R}^{\phi} & \rightarrow & \mathbb{R} \\ u & \rightarrow & \alpha.u + \beta \end{array}$$

$\alpha \in \mathbb{R}^{\phi}$ et $\beta \in \mathbb{R}$ sont les **poids** du neurones.

Attention, le neurone n'est pas nécessairement connecté à x l'entrée - à ce stade, il a une entré de taille ϕ indépendante de D.

Du neurone au réseau

Le neurone



Du neurone au réseau

La couche de neurone

Une couche de ψ de neurones est une séquence de ψ neurones prenant la même entrée, et, dont les ψ sorties sont regroupées en 1 vecteur :

$$couche_{A,b} : \begin{array}{ccc} \mathbb{R}^{\phi} & \rightarrow & \mathbb{R}^{\psi} \\ u & \rightarrow & \begin{pmatrix} neurone_{A_1,b_1}(u) \\ ... \\ neurone_{A_\psi,b_\psi}(u) \end{pmatrix} \end{array}$$

$A \in \mathbb{R}^{\psi \times \phi}$ et $b \in \mathbb{R}^{\psi}$ sont les **poids** de chacun des ψ neurones.

La couche de neurone est aussi linéaire : $couche_{A,b}(u) = Au + b$ mais avec des tailles arbitraires en entrée et sorti.

Du neurone au réseau

Le réseau de neurone

Si on empile 2 couches de neurones c'est exactement comme s'il y en avait qu'une :

$$A'(Au + b) + b' = (A'A)u + (A'b + b')$$

Oui mais si on met une non linéarité entre les 2 c'est différents.

Laquelle ? Sur pytorch : il y en a un certain nombre relu, elu, leaky-relu, sigmoide, arctan, hard sigmoid, hard arctan, prelu, relu6, rrelu, celu, selu, gelu, hard shirk, soft shirk, log sigmoid, soft sign, tanh, tanhshirk

globalement avant on utilisait une sigmoide (lisse) et aujourd'hui c'est plutôt relu $relu(u) = [u]_+ = max(u, 0)$ car c'est rapide.

Du neurone au réseau

Le réseau de neurone

Un réseau de neurones entièrement connectées **MLP** (multi layer perceptron en anglais) de profondeur Q est un empilement de Q couche de neurones - séparé par des activations - la dernière est classiquement un seul neurone :

$$reseau_w : \begin{array}{ccc} \mathbb{R}^D & \rightarrow & \mathbb{R} \\ x & \rightarrow & C_{w_Q}(relu(C_{w_{Q-1}}(...relu(C_{w_1}(x))...))) \end{array}$$

c'est à dire

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

$w_1, ..., w_Q$, Q matrices dont la seule chose imposée étant que w_1 ait D colonnes, et w_Q 1 ligne (et que les tailles soit cohérentes entre elles)

Du neurone au réseau

Et c'est tout !

Exemple question examen

Considérons le réseau 2D déjà appris :

$$g(x) = relu\left(relu\left(\begin{pmatrix} 1 \\ 0 \end{pmatrix}^T x\right) - relu\left(\begin{pmatrix} 0 \\ 1 \end{pmatrix}^T x\right) + 5\right)$$

$$+relu\left(relu\left(\begin{pmatrix} 0 \\ -1 \end{pmatrix}^T x\right) - relu\left(\begin{pmatrix} -1 \\ 0 \end{pmatrix}^T x\right) - 5\right) - 10$$

Q1 : Que vaut $g(x)$ sur $\mathbb{Z}^2$?

Exemple question examen

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

$$g(x) = [[x_1]_+ - [x_2]_+ + 5]_+ + [-[-x_1]_+ + [-x_2]_+ - 5]_+ - 10$$

- $x_1 > 0, x_2 > 0 \Rightarrow g(x) = [x_1 - x_2 + 5]_+ - 10$
- $x_1 > 0, x_2 < 0 \Rightarrow g(x) = x_1 + [-x_2 - 5]_+ - 5$
- $x_1 < 0, x_2 > 0 \Rightarrow g(x) < 0 \ ([5 - x_2]_+ + [x_1 - 5]_+ - 10)$
- $x_1 < 0, x_2 < 0 \Rightarrow g(x) = [x_1 - x_2 - 5]_+ - 5$

Exemple question examen

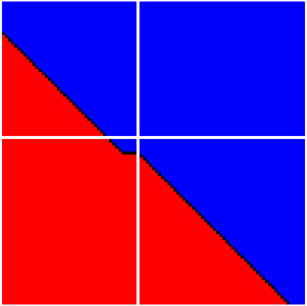
$$g(x) = [[x_1]_+ - [x_2]_+ + 5]_+ + [-[-x_1]_+ + [-x_2]_+ - 5]_+ - 10$$

- $x_1 > 0, x_2 > 0, x_1 - x_2 > 5 \Rightarrow g(x) > 0$
- $x_1 > 0, x_2 > 0, x_1 - x_2 < 5 \Rightarrow g(x) < 0$
- $x_1 > 0, -5 < x_2 < 0, x_1 > 5 \Rightarrow g(x) > 0$
- $x_1 > 0, -5 < x_2 < 0, x_1 < 5 \Rightarrow g(x) < 0$
- $x_1 > 0, x_2 < -5, x_1 - x_2 > 10 \Rightarrow g(x) > 0$
- $x_1 > 0, x_2 < -5, x_1 - x_2 < 10 \Rightarrow g(x) < 0$
- $x_1 < 0, x_2 > 0 \Rightarrow g(x) < 0$
- $x_1 < 0, x_2 < 0, x_1 - x_2 > 10 \Rightarrow g(x) > 0$
- $x_1 < 0, x_2 < 0, x_1 - x_2 < 10 \Rightarrow g(x) < 0$

Exemple question examen

$$g(x) = [[x_1]_+ - [x_2]_+ + 5]_+ + [-[-x_1]_+ + [-x_2]_+ - 5]_+ - 10$$

Q2 : représentation graphique des 8 zones



Exemple question examen

Considérons :

$$g(x) = relu\left(relu\left(\begin{pmatrix} 1 \\ 0 \end{pmatrix}^T x\right) - relu\left(\begin{pmatrix} 0 \\ 1 \end{pmatrix}^T x\right) + 5\right)$$

$$+relu\left(relu\left(\begin{pmatrix} -1 \\ 0 \end{pmatrix}^T x\right) + relu\left(\begin{pmatrix} 0 \\ -1 \end{pmatrix}^T x\right) - 5\right) - 10$$

Q3 : écrire g purement comme un MLP ?

Exemple question examen

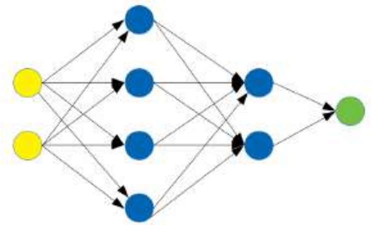
$$g(x) =$$

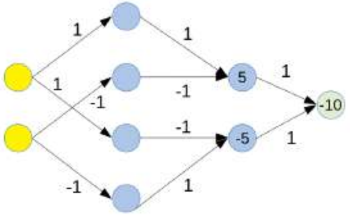
$$\mathbf{1}^T \left[\begin{pmatrix} 1 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \left[\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} x + \mathbf{0} \right]_+ + \begin{pmatrix} 5 \\ -5 \end{pmatrix} \right]_+ - 10$$

Exemple question examen

$$g(x) = [[x_1]_+ - [x_2]_+ + 5]_+ + [-[-x_1]_+ + [-x_2]_+ - 5]_+ - 10$$

Q4 : la même chose avec un dessin du réseau





Exemple question examen

La on s'est uniquement intéressé à la représentation et/ou au comportement en test. On peut aussi considérer des questions sur *l'apprentissage* c'est à dire sur le choix des poids (mais sans algorithme juste avec des solutions triviales).

Q5 : chercher w_1, w_2, w_3, b tel que le réseau 1D $h(x, w) = w_1[x]_+ + w_2[x-1]_+ + w_3[x-2]_+ + b$ vérifie

- $h(0, w) > 0$ (par exemple 1)
- $h(1, w) < 0$ (par exemple -1)
- $h(2, w) > 0$ (par exemple 1)
- $h(3, w) < 0$ (par exemple -1)

Exemple question examen

Q5 : chercher w_1, w_2, w_3, b tel que le réseau 1D $h(x, w) = w_1[x]_+ + w_2[x-1]_+ + w_3[x-2]_+ + b$ vérifie

- $h(0, w) > 0$ (par exemple 1)
- $h(1, w) < 0$ (par exemple -1)
- $h(2, w) > 0$ (par exemple 1)
- $h(3, w) < 0$ (par exemple -1)

$\Rightarrow$ réponse : $h(x) = 1 - 2[x]_+ + 4[x-1]_+ - 4[x-2]_+$

$$\Rightarrow h(x) = \begin{pmatrix} -1 & 1 & -1 \end{pmatrix} \left[\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} x + \begin{pmatrix} 0 \\ -1 \\ -2 \end{pmatrix} \right]_+ + 1$$

Plan du cours

- *La classification binaire*
- *Le neurone classique*
- Le neurone convolutif
- L'apprentissage en pratique
  - loss function
  - optimiseur
  - backpropagation
- exemples de recherche

La vision par ordinateur

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

La vision par ordinateur

Est ce qu'une image est $x \in \mathbb{R}^D$?
$\Rightarrow$ **Non !**
1 image RGB de 480x340 = 1 vecteur dans $[0, 255]^{163200}$
Il est impossible et stupide de traiter les images comme des points !

La vision par ordinateur

- 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.

CNN

- 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 $(\theta \star u)_{h,w}$:
$$\sum_{dh,dw,\rho \in [-\delta_H,\delta_H]\times[-\delta_H,\delta_W]\times[1,\phi]} x_{h+dh,w+dw,\rho} \times w_{dh,dw,\rho}$$

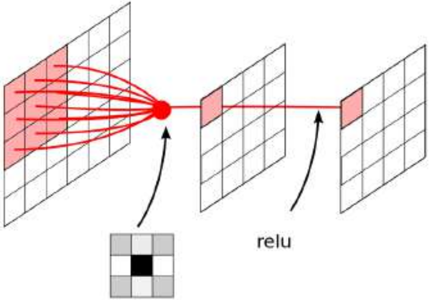
Le neurone convolutif c'est comme un neurone mais local !

CNN

- 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

CNN



MLP

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

ConvNet naïf

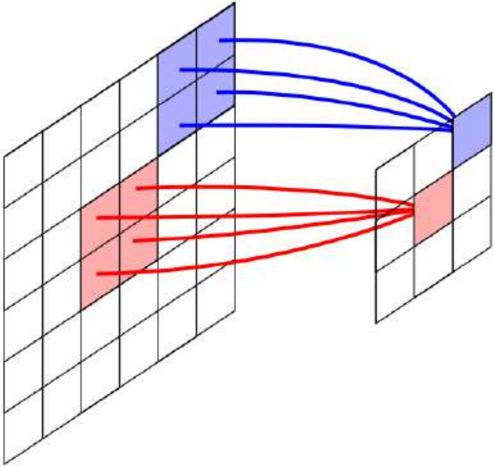
$$f(x, w) = w_Q \star relu(w_{Q-1} \star relu(...(relu(w_1 \star x))))$$

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 $pool(u)_{h,w,\rho} = \max\limits_{dh,dw \in [0,1]} x_{2h+dh,2w+dw,\rho}$

- 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 $c' \times \frac{h+\mathbf{2}*pad_h - kernel_h}{stride_h} + \mathbf{1} \times \frac{w+\mathbf{2}*pad_w - kernel_w}{stride_w} + \mathbf{1}$ 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).

https ://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1
https ://ensiwiki.ensimag.fr/index.php ?title=Fichier :Demo-Convolution.gif

MLP

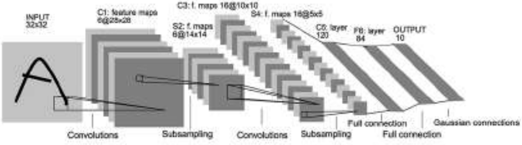
$$f(x, w) = w_Q \times relu(w_{Q-1} \times relu(...(relu(w_1 \times x))))$$

ConvNet sequentiel

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

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

Lenet

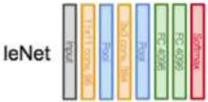


INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections

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



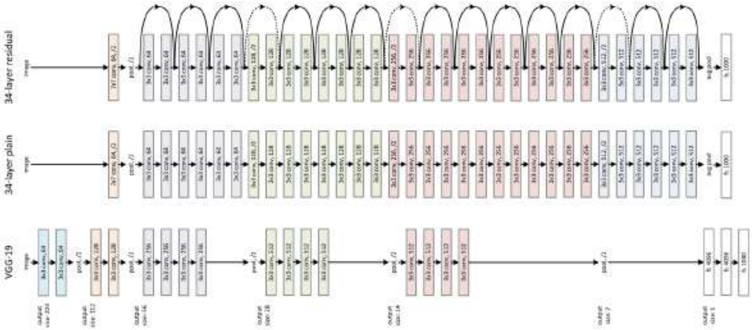
Alexnet

Quand le réseau a beaucoup de couche, on parle de réseau profond. Ce terme est apparu en 2012 avec un réseau à 10 couches dit Alexnet (*imagenet classification with deep convolutional neural networks*).

Les réseaux de neurones existent depuis 1950 mais la puissance de calcul n'était pas suffisante pour envisager des réseaux de neurones profonds.

Lenet vs AlexNet vs VGG

VGG19 vs ResNet34



On peut en fait avoir n'importe quel graphe sans cycle.

Plan du cours

- *La classification binaire*
- *Le neurone classique*
- *Le neurone convolutif*
- L'apprentissage en pratique
  - loss function
  - optimiseur
  - backpropagation
- exemples de recherche

Rappel

- $f(x, w) = w_Q \times relu(w_{Q-1} \times relu(...(relu(w_1 \times x))))$
- En test, w est un paramètre
- χ est une entrée
- on calcule $f(\chi, w)$
- À l'apprentissage, l'entrée est la base $x_1, ..., x_N$
- on calcule w tel que $e_a = \sum_n \mathbf{1}_{-}(y(x_n)f(x_n, w)) \approx 0$

Comment ?

La descente de gradient

F est une fonction dérivable de $\mathbb{R}^D$ dans $\mathbb{R}$ alors
$\forall u, h \in \mathbb{R}^D$, $F(u+h) = F(u) + \nabla F_u | h + o(h)$
avec $ho(h) \underset{h \to 0}{\to} 0$ (notation petit o classique)
Donc si $\nabla F_u \neq 0$ alors il existe $\lambda > 0$ tel que $F\left(u - \lambda \nabla F_u\right) < F(u)$

La descente de gradient

pseudo code

input : F, u_0

1. $u = u_0$
2. calculer ∇F_u
3. si $\nabla F_u \approx 0$ ou early stopping alors sortir
4. $\lambda = 1$
5. tant que $F\left(u - \lambda \nabla F_u\right) \geq F(u)$ faire $\lambda = 0.5\lambda$
6. $u = u - \lambda \nabla F_u$
7. go to 2

La descente de gradient

pseudo code

input : F, u_0

1. $u = u_0$
2. calculer ∇F_u
3. si $\nabla F_u \approx 0$ ou early stopping alors sortir
4. $\lambda = 1$
5. tant que $F\left(u - \lambda \nabla F_u\right) \geq F(u)$ faire $\lambda = 0.5\lambda$
6. $u = u - \lambda \nabla F_u$
7. go to 2

cet algorithme converge vers un point u^* tel que $\nabla F_u = 0$

Apprentissage et descente de gradient

Appliquer à l'apprentissage :

- les variables (u en slide 57 - 58) sont les poids w du réseau
- la fonctionnelle (F) est (+/-) l'erreur d'apprentissage :
  $F(w) \approx \sum_n \mathbf{1}_-(y(x_n)f(x_n, w)) = e_a$

Apprentissage et descente de gradient

Test :

w fixé, on prend χ, et, on doit calculer $f(\chi, w)$

Apprentissage :

On prend $x_1, ..., x_N$, et, on doit **approximer**

$$\min_w \sum_n \mathbf{1}_-(y(x_n) f(x_n, w))$$

Fonction de perte

La descente de gradient ne marche qu'avec des fonctions globalement lisse.
Utiliser $F(w) = \min_w \sum_n \mathbf{1}_-(y(x_n) f(x_n, w))$ ne peut pas marcher

Fonction de perte

La descente de gradient ne marche qu'avec des fonctions globalement lisse.
Utiliser $F(w) = \min_w \sum_n \mathbf{1}_-(y(x_n)f(x_n, w))$ ne peut pas marcher
Il faut lisser l'erreur d'apprentissage via une loss function

$$F(w) = loss\,(w) = \sum_n l(y(x_n)f(x_n, w))$$

Fonction de perte

$$F(w) = loss\,(w) = \sum_n l(y(x_n)f(x_n, w))$$

- l doit être assez lisse
- l doit avoir une valeur proche de 0 si $y(x_n)f(x_n, w)$ est grand
- l doit avoir une valeur très supérieure à 0 si $y(x_n)f(x_n, w)$ est très petit

Fonction de perte

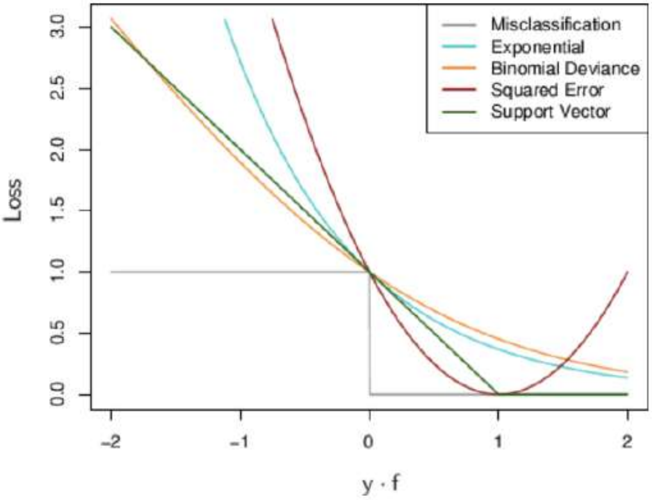
$$F(w) = loss\,(w) = \sum_n l(y(x_n)f(x_n, w))$$

- l doit être assez lisse
- l doit avoir une valeur proche de 0 si $y(x_n)f(x_n, w)$ est grand
- l doit avoir une valeur très supérieure à 0 si $y(x_n)f(x_n, w)$ est très petit

hinge loss :

$$loss\,(w) = \sum_n relu(1 - y_n f(x_n, w))$$

Fonction de perte



Limite de la descente de gradient

$$loss\,(w) = \sum_n relu(1 - y_n f(x_n, w))$$

Si $N = 1000000$ ça veut dire que pour calculer $loss(w)$ je dois appliquer f (plusieurs couches) à 1000000 points !

Descente de gradient stochastique

loss est une fonction dérivable de $\mathbb{R}^D$ dans $\mathbb{R}$
et que $loss\,(u) = \sum_{i=1} q_i\,(u)$
alors **dans le cas convexe**, il est possible de minimiser *loss* en faisant comme une descente de gradient mais en prenant une sous sommes des q_i tirée aléatoirement avec une politique $\lambda(t)$ fixée a priori (qui doit quand même vérifier certaines conditions).

Descente de gradient stochastique

pseudo code

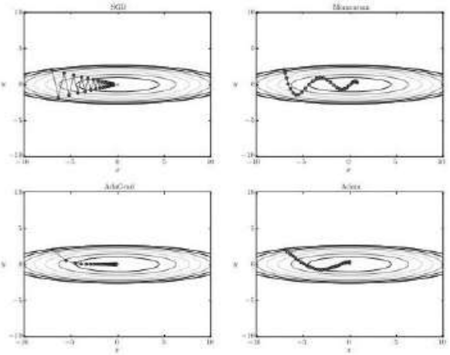
input : $x_1, y_1, ..., x_n, y_n,\ w_0$

1. $w = w_0$
2. $iter = 0$
3. tirer n au hasard dans 1,...,N
4. $partial\_loss = relu(1 - y_n f(x_n, w))$
5. calculer $\nabla_w partial\_loss$
6. $w = w - \lambda_{iter} \nabla_w partial\_loss$
7. $iter = iter + 1$
8. si condition d'arrêt alors sortir
9. go to 3

Descente de gradient stochastique

Optimizer

$w = w - \lambda_{iter} \nabla_w partial\_loss$ est une possibilité mais il y en a d'autres :



Synthèse

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

Synthèse

L'apprentissage

L'apprentissage consiste à appliquer la méthode de la descente de gradient stochastique (optimiseur à choisir) à une fonction de perte (à choisir) qui approxime l'erreur d'apprentissage
Par exemple

$$partial\_loss\,(w) = \sum_{n \in Batch} relu(1 - y_n f(x_n, w))$$

$$w = w - \lambda_{iter} \nabla_w partial\_loss$$

Forward - Backward

Mais ça suppose qu'on sache calculer le gradient ! ! ! !

Forward - Backward

objectif

$$partial\_loss(w) = \sum_{n \in Batch} relu(1 - y_n f(x_n, w))$$

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

$\Rightarrow$ on veut calculer

$$\partial \frac{\partial partial\_loss(w)}{\partial w_{t,i,j}}$$

Forward - Backward

Forward

```
for t
    for i
        for j
            A[t][i] += relu(A[t-1][j])*w[t-1][i][j]
```

Forward - Backward

Réduction w - α

$$\frac{\partial loss}{\partial w_{t,i,j}} = \frac{\partial loss}{\partial \alpha_{t,i,j}} \frac{\partial \alpha_{t,i}}{\partial w_{t,i,j}} = \frac{\partial loss}{\partial \alpha_{t,i}} x_{t,j}$$

Forward - Backward

Réduction α - α

$$\frac{\partial loss}{\partial \alpha_{t,j}} = \sum_i \frac{\partial loss}{\partial \alpha_{t+1,i}} \frac{\partial \alpha_{t+1,i}}{\partial \alpha_{t,j}} = \sum_i \frac{\partial loss}{\partial \alpha_{t+1,i}} w_{t,i,j} relu'\left(\alpha_{t,j}\right)$$

relu est une fonction linéaire par morçeau, sa *dérivé* est donc une constante par morçeau

Forward - Backward

Attention

La somme dans $\frac{\partial loss}{\partial \alpha_{t,j}} = \sum_i \frac{\partial loss}{\partial \alpha_{t+1,i}} \frac{\partial \alpha_{t+1,i}}{\partial \alpha_{t,j}}$ ne vient **pas** de la somme dans $\alpha_{t+1,i} = \sum_j x_{t,j} w_{t,i,j}$.

Elle vient de $f(u) = a(b(u), c(u))$ implique $\frac{\partial f}{\partial u} = \frac{\partial a}{\partial b}\frac{\partial b}{\partial u} + \frac{\partial a}{\partial c}\frac{\partial c}{\partial u}$.
Lui même vient de $f(u+h) = f(u) + f'(u)h$

Forward - Backward

```
for t
    for i
        for j
            A[t][i] += relu(A[t-1][j])*w[t-1][i][j]
DA[z][1] = partial_loss
for t from z to 1
    for j
        for i
            DA[t][j] += DA[t+1][i]*w[t][i][j]*relu'(A[t][j])
```

Une conséquence inattendue

Structurellement dans un réseau de neurones, le gradient *se propage* partout.
On peut donc calculer le gradient vis à vis de l'entrée !

$$\nabla_x loss(y_n, f(x_n, w))$$

(tout aussi facilement qu'on peut calculer $\nabla_x loss(y_n, f(x_n, w))$)

Une conséquence inattendue

L'apprentissage consiste à calculer

$$\nabla_w loss(y_n, f(x_n, w))$$

et à **actualiser** w de sorte que

$$loss(y_n, f(x_n, w)) \approx 0$$

Une conséquence inattendue

Mais avec le même outils, on peut calculer

$$\nabla_x loss(y_n, f(x_n, w))$$

et **actualiser** x de sorte que

$$loss(y_n, f(x_n, w)) \gg 0$$

Une conséquence inattendue

Mais avec le même outils, on peut calculer

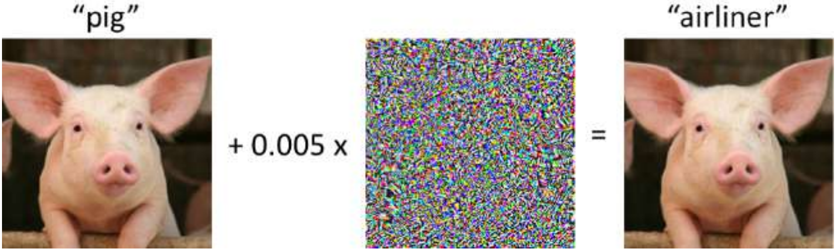
$$\nabla_x loss(y_n, f(x_n, w))$$

et **actualiser** x_n de sorte que

$$loss(y_n, f(x_n, w)) \gg 0$$

$\Rightarrow$ ce qui permet de construire une image x_n spécifiquement perturbée pour échapper au réseau : *adversarial exemple*.

Une conséquence inattendue



Synthèse

La classification binaire

P une distribution sur X un espace, y la fonction cible dans {-1,1}

- on approxime y par le signe de f (avec des poids w)
- erreur réelle $e_r = \int_X \mathbf{1}_-(y(\chi)f(\chi,w))P(\chi)\mathrm{d}\chi$
- $x_1, ..., x_N$ une base d'apprentissage tirée selon P
- l'erreur d'apprentissage $e_a = \frac{1}{N}\sum_n \mathbf{1}_-(y(x_n)f(x_n,w))$
- apprentissage : w optimisé tel que $e_a \approx 0$
- $\chi_1, ..., \chi_M$ une base de test tirée selon P
- w fixé, l'erreur de test $e_t = \frac{1}{M}\sum_m \mathbf{1}_-(y(\chi_m)f(\chi_m,w)) \approx e_r$

Synthèse

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

Synthèse

L'apprentissage

L'apprentissage consiste à appliquer la méthode de la descente de gradient stochastique (optimiseur à choisir) à une fonction de perte (à choisir) qui approxime l'erreur d'apprentissage
Par exemple

$$loss\,(w) = \sum_n relu(1 - y_n f(x_n, w))$$

$$w = w - \lambda_{iter} \nabla_w partial\_loss$$

Où ∇_w se calcule facilement en utilisant le fait que la dérivé par rapport à une couche s'exprime en fonction de la dérivé par rapport à la couche suivante.

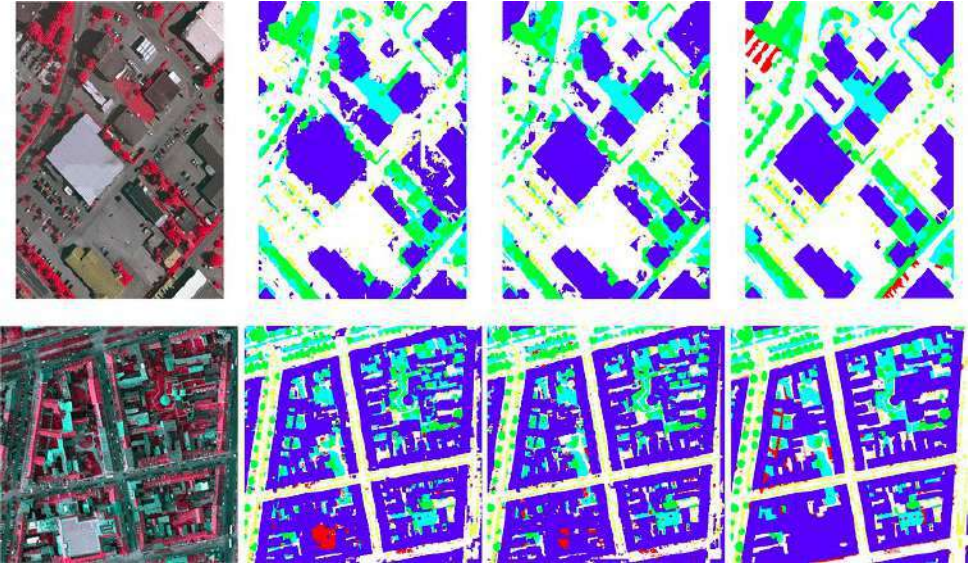
Plan du cours

- *La classification binaire*
- *Le neurone classique*
- *Le neurone convolutif*
- *L'apprentissage en pratique*
- exemples de recherche

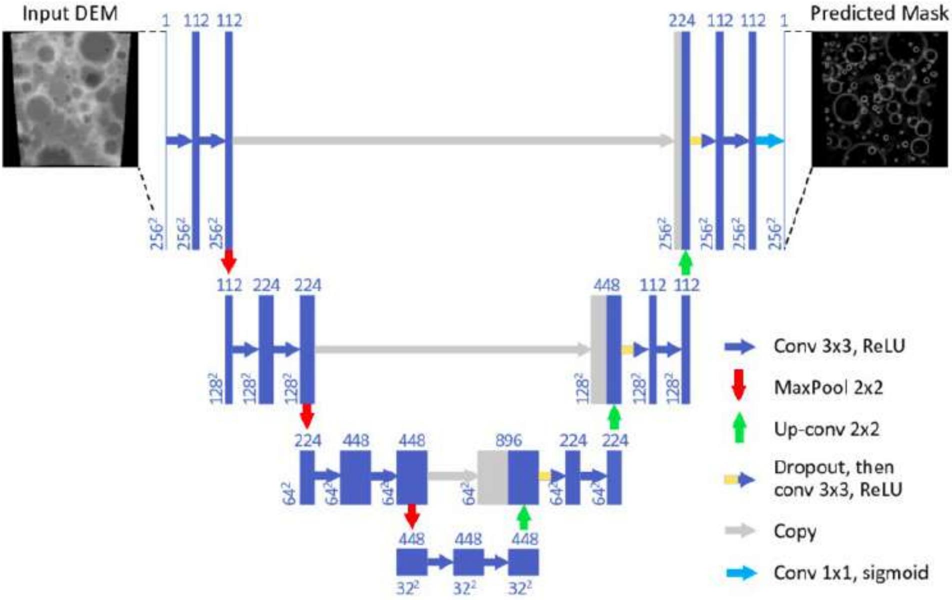
Segmentation sémantique

Segmentation sémantique

Segmentation sémantique

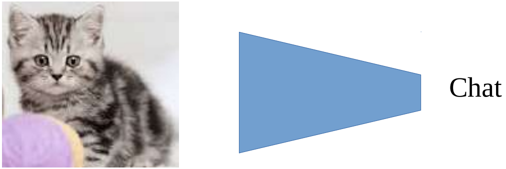


Segmentation sémantique

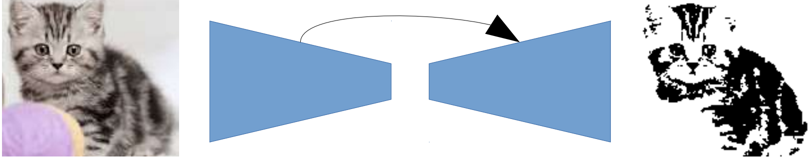


Segmentation sémantique

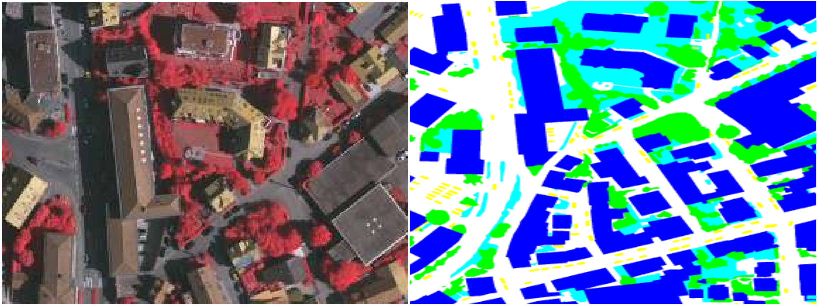
Lenet, Alexnet, VGG, Resnet...



FPN, Segnet, Unet, PSP...



Segmentation sémantique

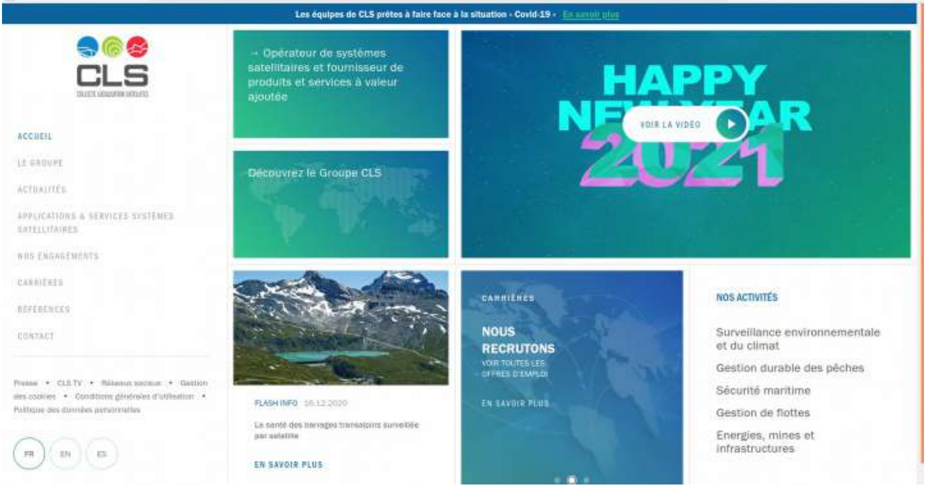


Renseignement militaire, politique d’urbanisme, impacte climatique, analyse économique, détection de zone d’atterrissage d’urgence, analyse de données médicale...

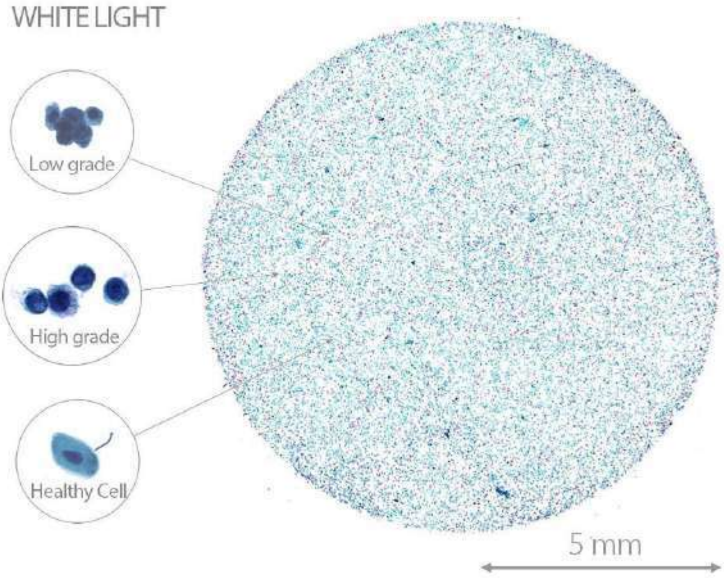
Segmentation sémantique

Segmentation sémantique

Segmentation sémantique



Segmentation sémantique



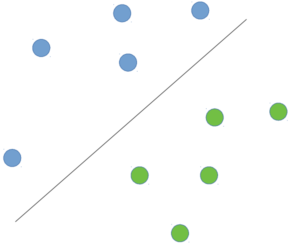
Segmentation sémantique

Les aspects militaires/surveillances restent les moteurs principaux

Adversarial examples

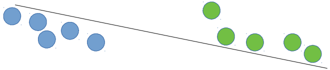
Adversarial examples

La frontière de décision
sur la variété



« vu du dessus »

La frontière de décision
hors de la variété



« vu de coté »

Adversarial examples

AIRS

Apprentissage \ Performance	avant perturbation	après perturbation
avant perturbation	94	86
après perturbation	92	91

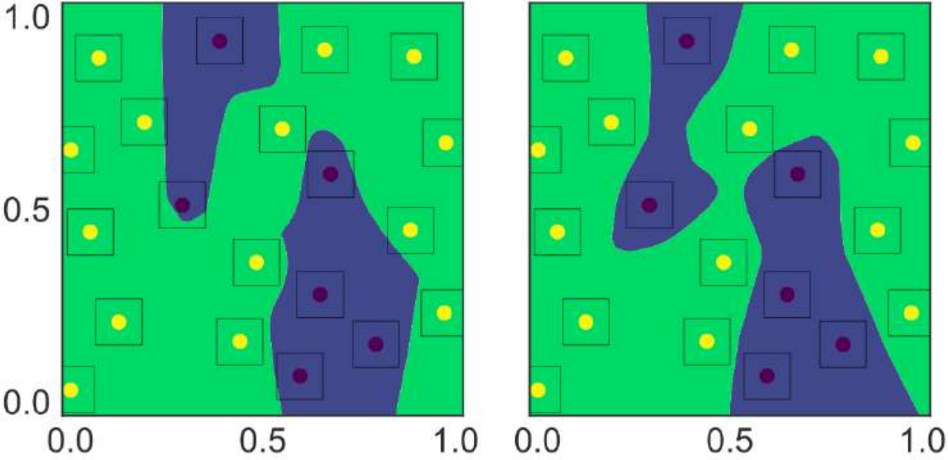
Adversarial examples

Performance / Apprentissage	avant perturbation	après perturbation
avant perturbation	94	86
après perturbation	92	91

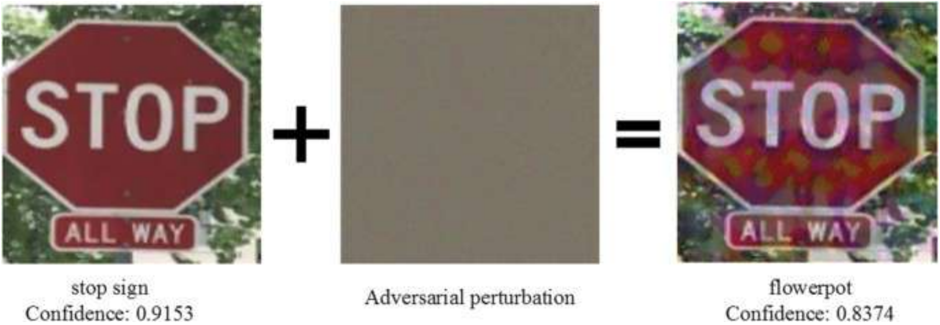
→ il ne faut pas avoir peur non plus, il est possible de faire des réseaux robuste !

Adversarial examples

TRADE



Adversarial examples



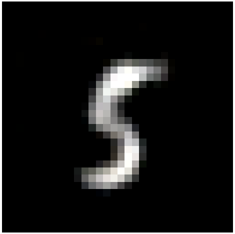
→ L’image de gauche est-elle possible ?

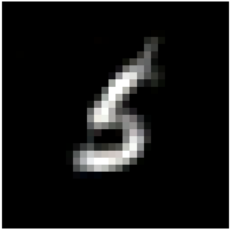
Adversarial examples

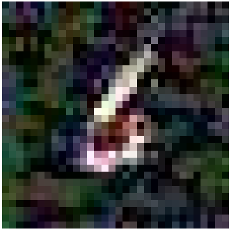
Real hard examples

→ Le hacking / camouflage doit être pris en compte

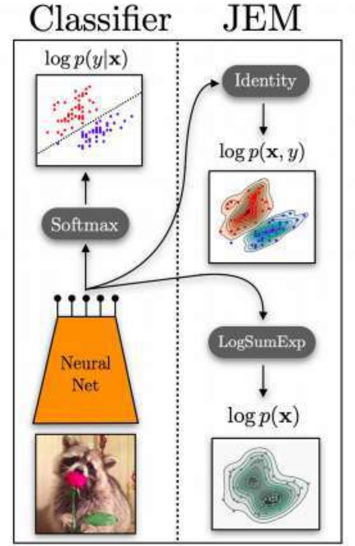
Mais rien ne démontre aujourd’hui qu’il soit réellement possible





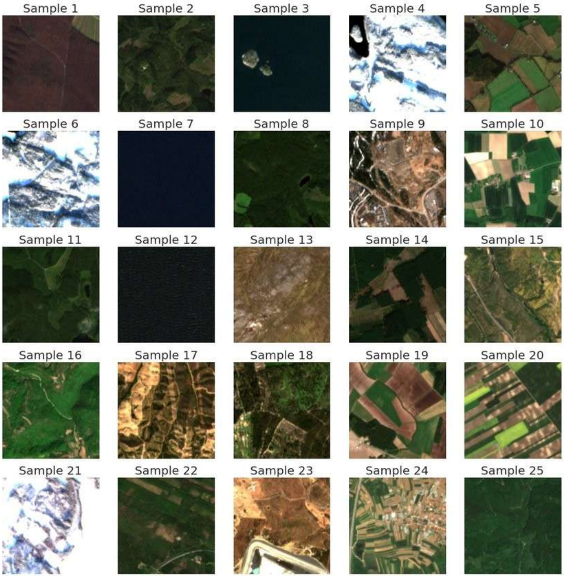


Real hard examples

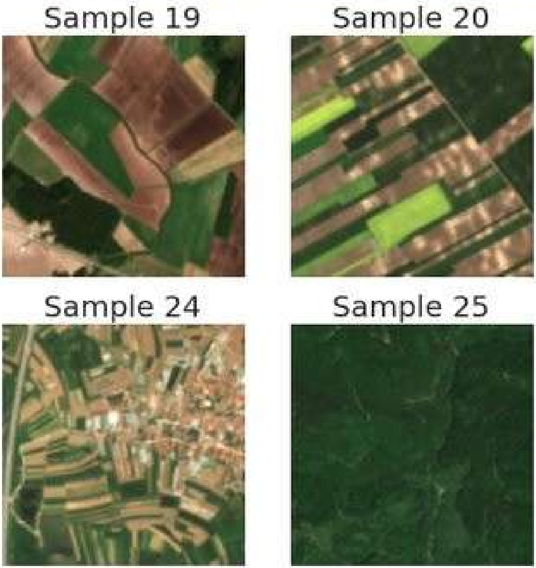


Apprendre
à la fois la classe
et la vraisemblance ?

Real hard examples

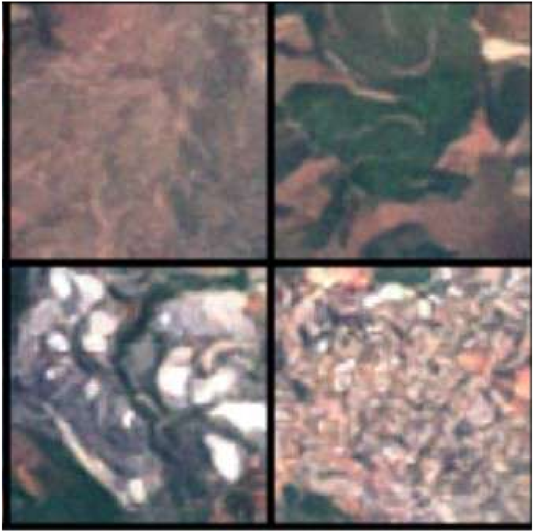


Real hard examples



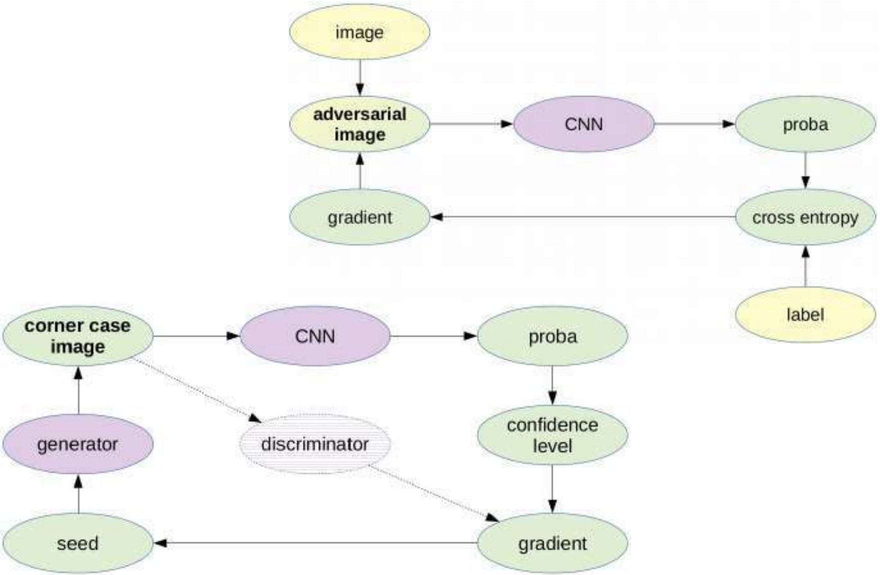
Real hard examples

Real hard examples

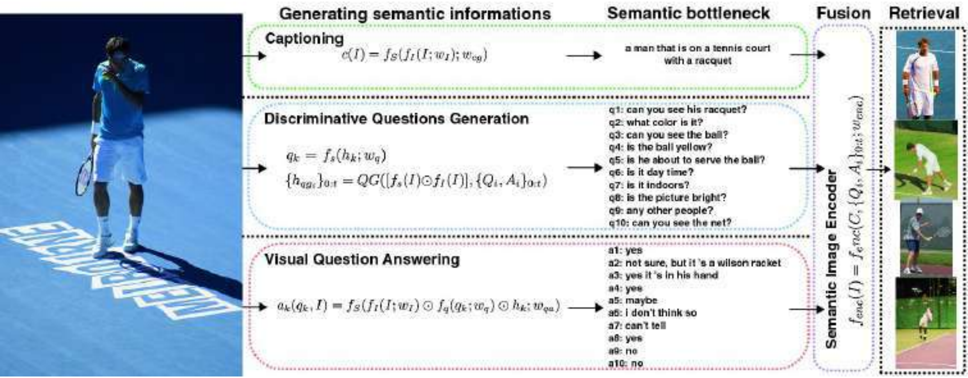


Real hard examples

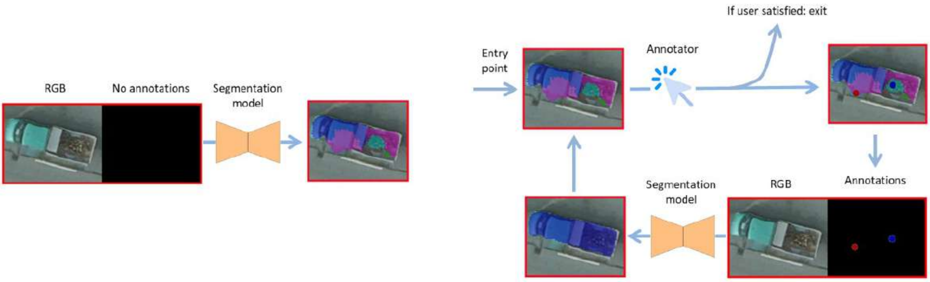
Real hard examples



Explicabilité



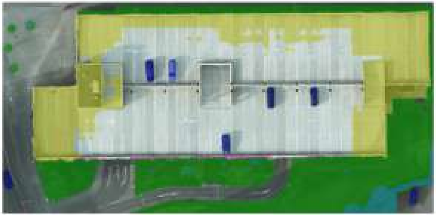
Interaction et deep learning



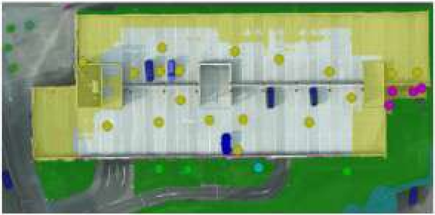
Initialization phase

Interactive loop

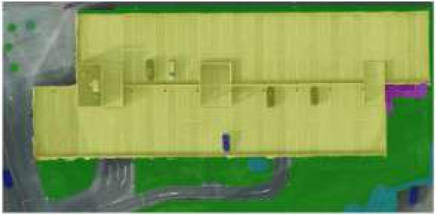
Interaction et deep learning



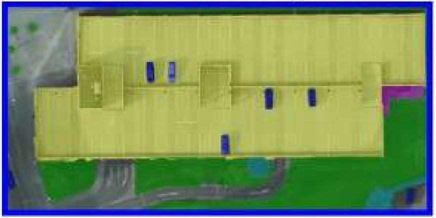
1 - Initial segmentation



2 - Annotation phase

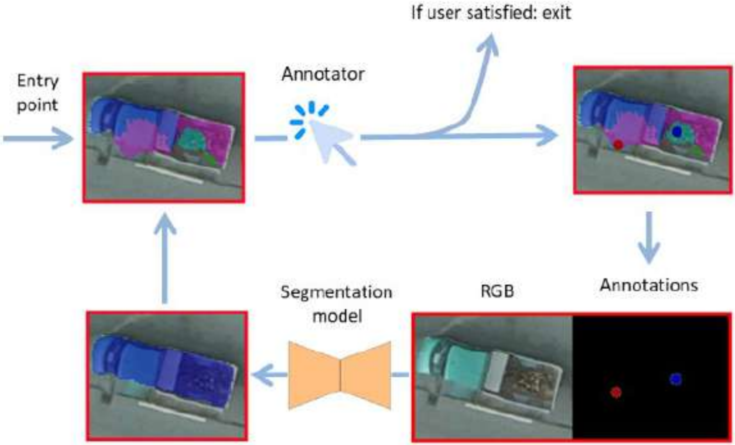


3 - Refined segmentation



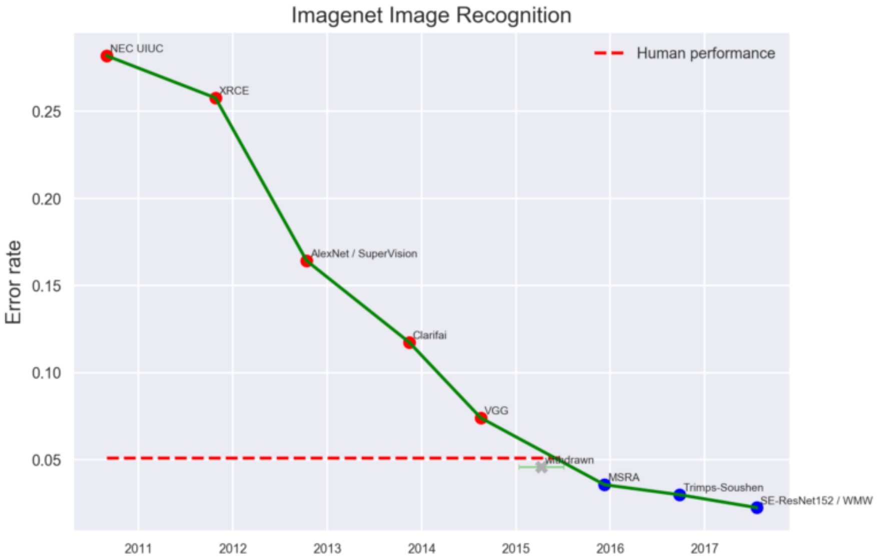
Ground-truth

Interaction et deep learning



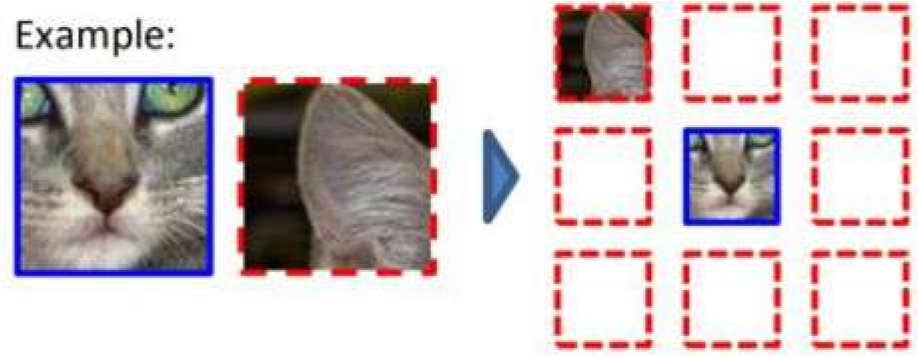
Interactive loop

Semi/self supervised / apprentissage incremental

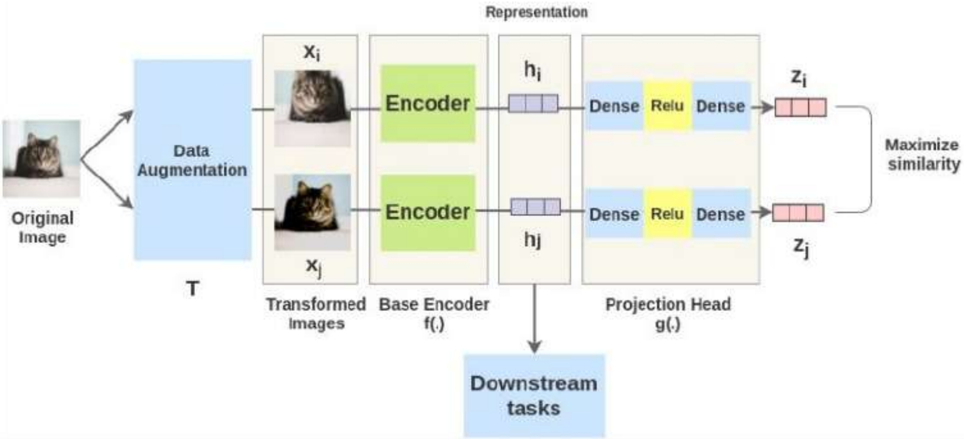


Semi/self supervised / apprentissage incremental

Semi/self supervised / apprentissage incremental

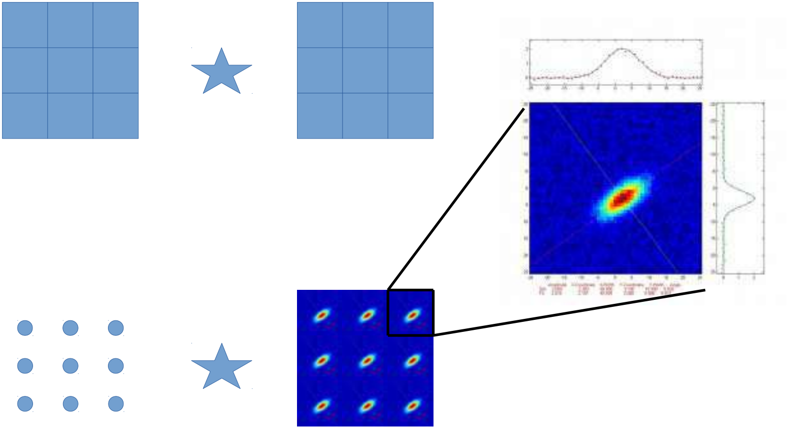


Semi/self supervised / apprentissage incremental

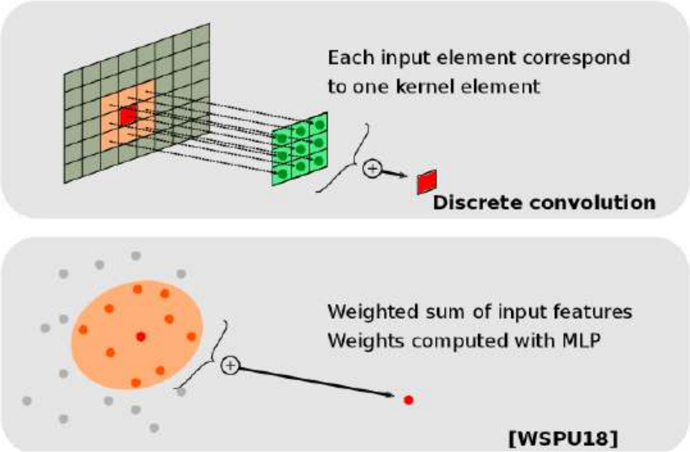


3D

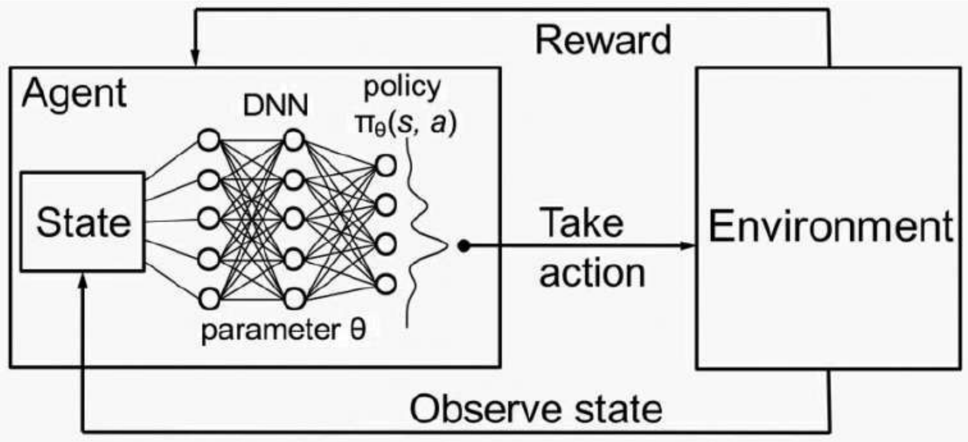
3D



3D


Each input element correspond
to one kernel element
Discrete convolution
Weighted sum of input features
Weights computed with MLP
[WSPU18]

Deep reinforcement learning

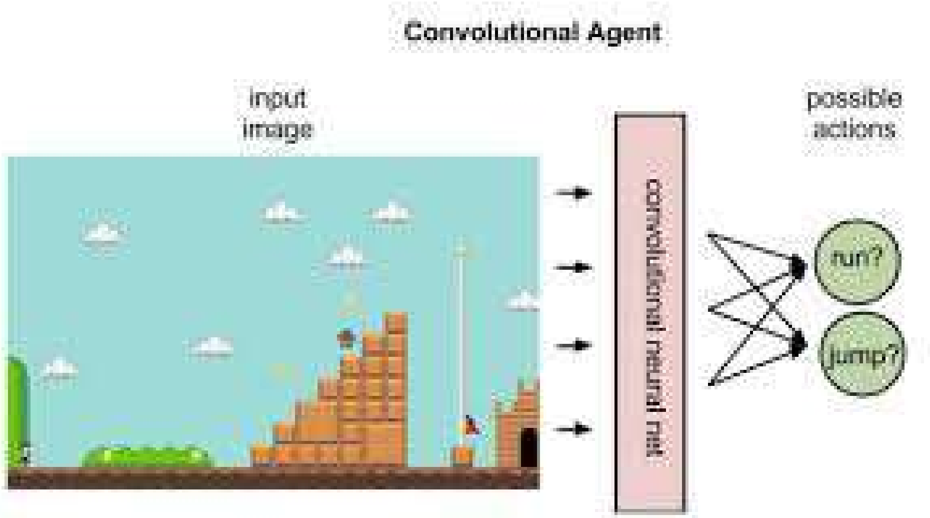


Deep reinforcement learning

$$Q(s,a) = r(s,a) + \gamma \max_{a} Q(s',a)$$

$$\nabla_{\theta_i} L(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)\nabla_{\theta_i} Q(s,a;\theta_i)\right].$$

Deep reinforcement learning



Deep reinforcement learning