Réseaux de neurones

IFT 780

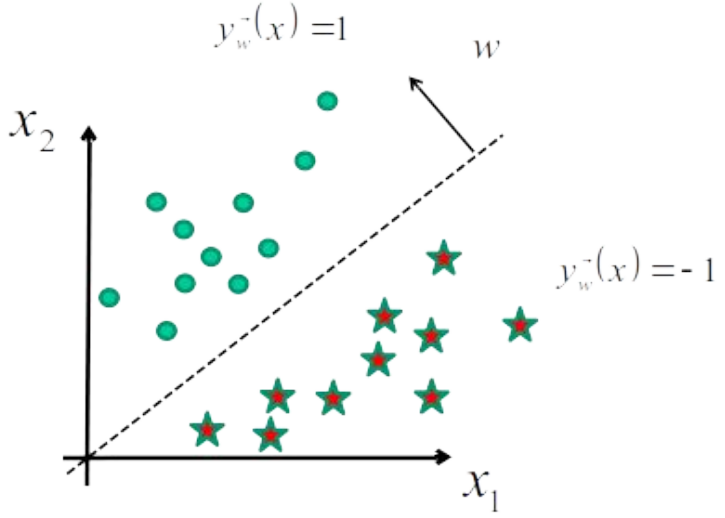
Conclusion

Par
Antoine Théberge

Chapitre 1

Séparation linéaire

(2D et 2 classes)



biais poids

$$y_w^{\rightarrow}(x) = w_0 + w_1 x_1 + w_2 x_2$$
$$= w_0 + w^T x$$
$$= w'^T x'$$

$$y_w^{\rightarrow}(x) = w^T x$$

Par simplicité

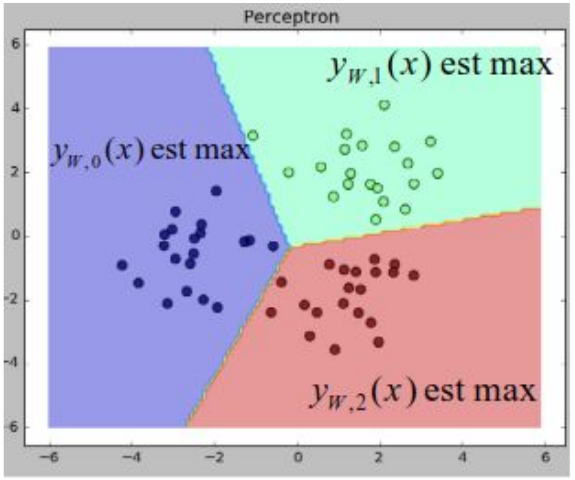
2 grands **advantages**. Une fois l'entraînement terminé,

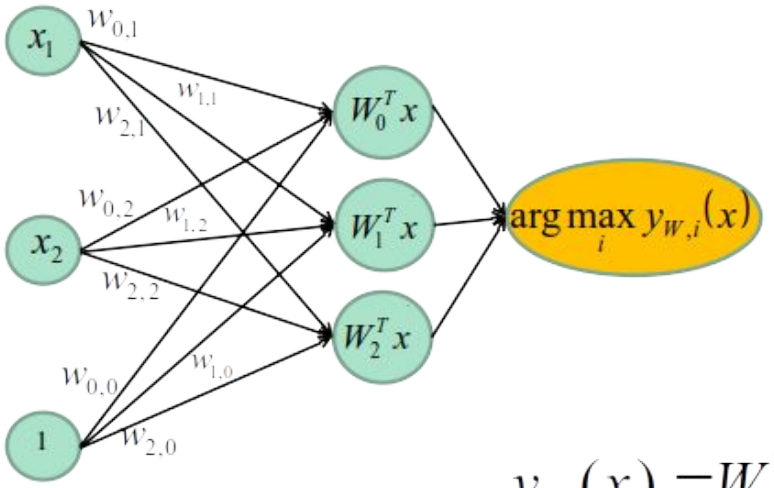
1. Plus besoin de données d'entraînement (vs. nearest neighbor p.e.)
2. Classification est très rapide (**produit scalaire** entre 2 vecteurs)

Chapitre 1

Perceptron Multiclasse

(2D et 3 classes)



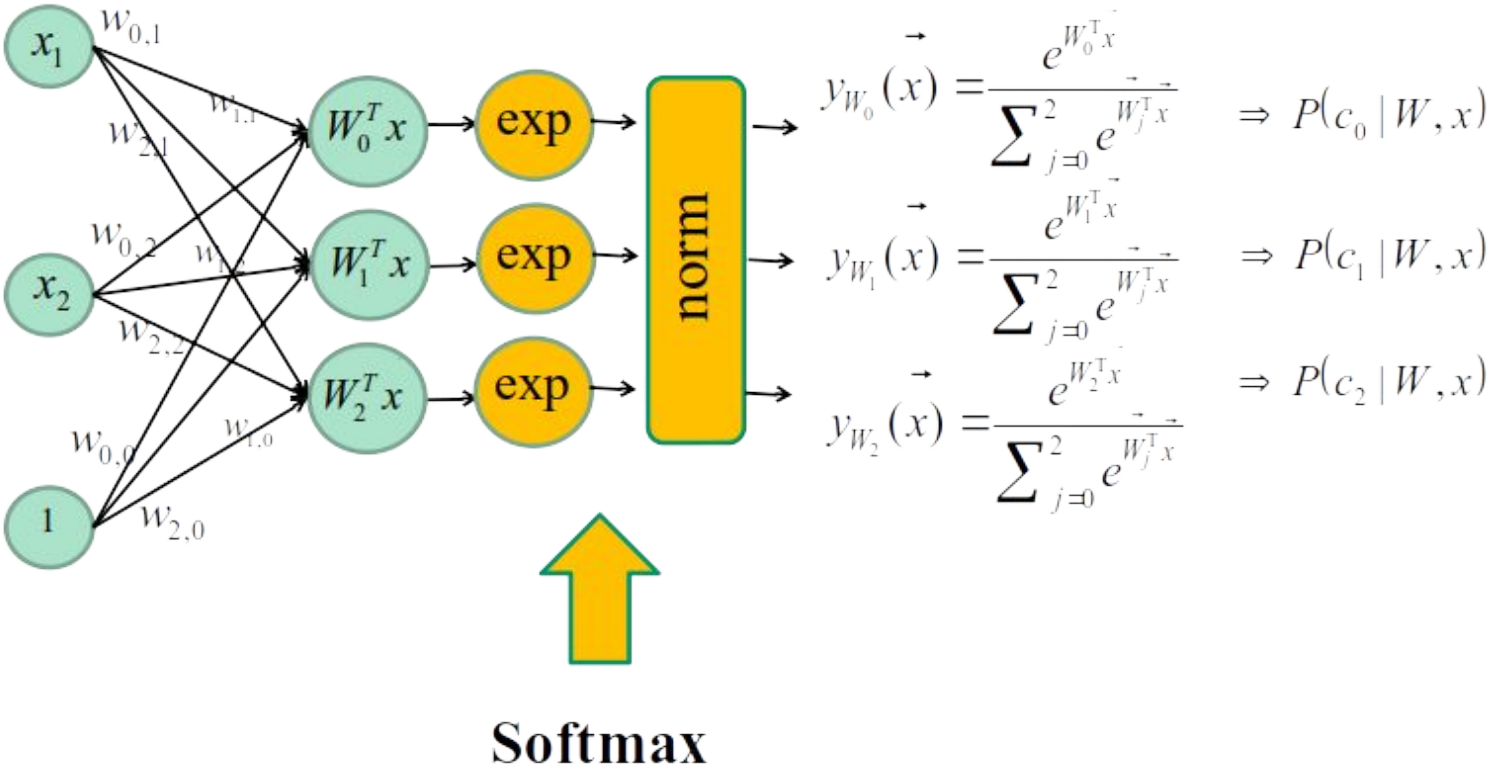


$$y_W(x) = W^T x$$

$$\vec{y_W}(x) = \begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} \\ w_{1,0} & w_{1,1} & w_{1,2} \\ w_{2,0} & w_{2,1} & w_{2,2} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

Chapitre 1

Régression logistique multiclasse



Chapitre 1

Régression logistique multiclasse

Fonction de coût est une **entropie croisée** (*cross entropy loss*)

$$E_D(\mathrm{W}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn} \ln y_{W_k}(\vec{x}_n)$$

$$\nabla E_D(\mathrm{W}) = \frac{1}{N}\sum_{n=1}^{N} \vec{x}_n \left(y_W(\vec{x}_n) - t_{kn} \right)$$

Chapitre 1

Descente de gradient

$$W^{[t+1]} = W^{[t]} - \eta \nabla E_{W^{[t]}}(D)$$

Chapitre 1

Optimisation

Descente de gradient

$$\mathbf{w}^{[k+1]} = \mathbf{w}^{[k]} - \eta^{[k]} \nabla E$$

∇E → Gradient de la function de coût

$\eta^{[k]}$ → Taux d'apprentissage ou "*learning rate*".

Optimisation par ***mini-batch***

Initialiser **w**
k=0
FAIRE k=k+1

FAIRE n=0 à N par sauts de ***MBS /*Mini-batch size*/***

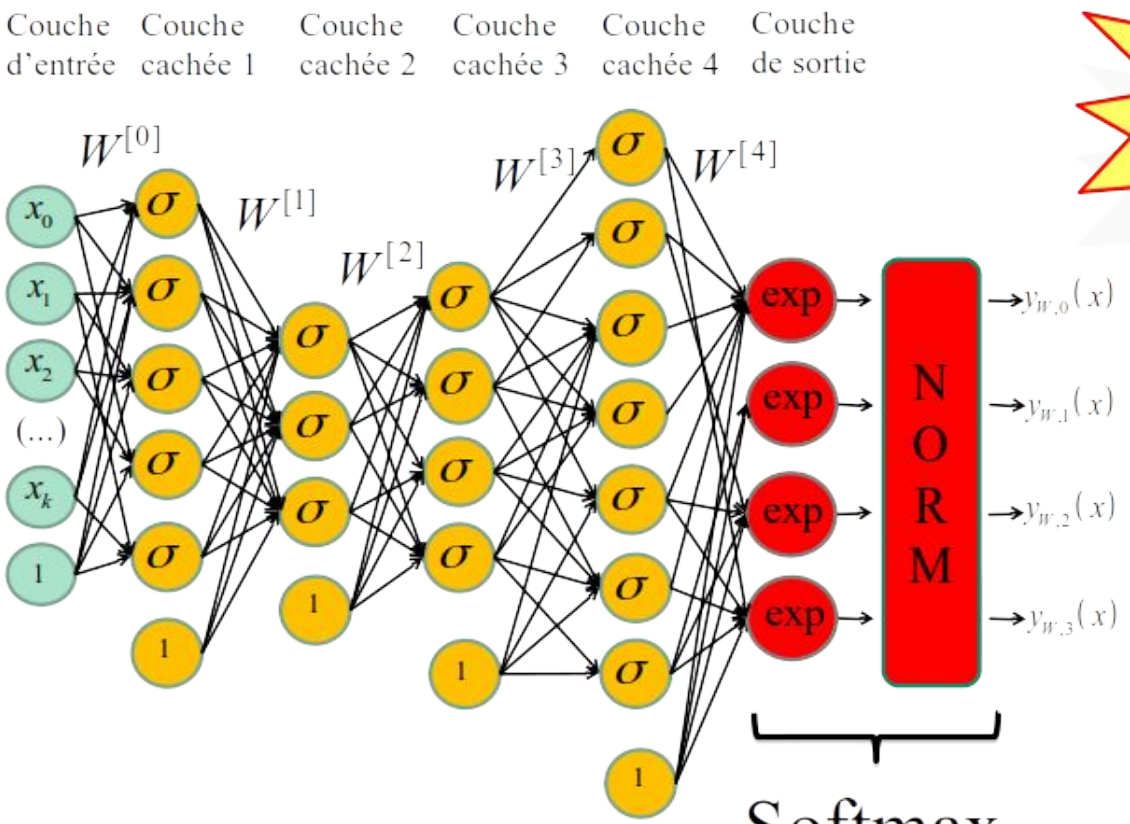
$$w = w - \eta^{[k]} \sum_{i=n}^{n+MBS} \nabla E(\vec{x_i})$$

Epoch

JUSQU'À ce que toutes les données sont bien classées ou
k==MAX_ITER

Chapitre 1

kD, **4 Classes**, Réseau à 4 couches cachées



Entropie Croisée

Softmax

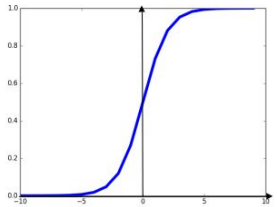
$$y_{W,i}(x) = \frac{f_{W,i}}{\sum_k f_{W,k}}$$

$$y_W(\vec{x}) = \text{softmax}\left(W^{[4]}\sigma\left(W^{[3]}\sigma\left(W^{[2]}\sigma\left(W^{[1]}\sigma\left(W^{[0]}\vec{x}\right)\right)\right)\right)\right)$$

Chapitre 1

Fonction d'activation

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



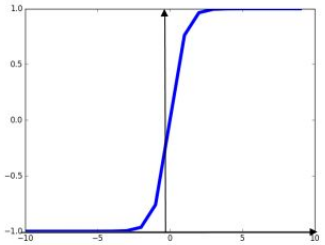
Sigmoïde

- Ramène les valeurs entre 0 et 1
- Historiquement populaire

3 Problèmes :

- Un neurone saturé a pour effet de **« tuer » les gradients**
- Sortie d'une sigmoïde n'est **pas centrée à zéro.**
- exp() est **coûteux** lorsque le nombre de neurones est élevé.

Fonction d'activation

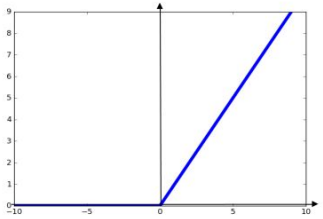


Tanh(x)

- Ramène les valeurs entre -1 et 1
- **Sortie centrée à zéro**
- **Disparition du gradient** lorsque la fonction sature

Fonction d'activation

$$\mathrm{ReLU}(x) = \max(0, x)$$



ReLU(x)
(Rectified Linear Unit)

- Aucune **saturation**
- Super **rapide**
- **Converge plus rapide** que sigmoïde/tanh (5 à 10x)

- Sortie **non centrée à zéro**
- **Un inconvénient :** qu'arrive-t-il au gradient lorsque x<0?

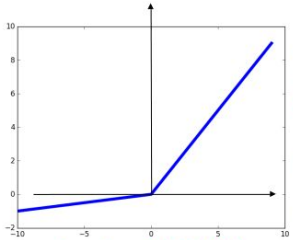
$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$$

[Krizhevsky et al., 2012]

Fonction d'activation

[LeCun et al., 1991]

$$\mathrm{LReLU}(x) = \max(0.01x, x)$$



Leaky ReLU(x)

- Aucune **saturation**
- Super **rapide**
- **Converge plus rapide** que sigmoïde/tanh (5 à 10x)
- **Gradients ne meurent pas**

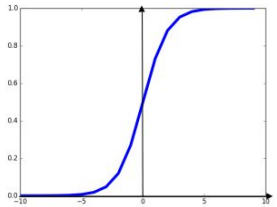
- 0.01 est un **hyperparamètre**

[Mass et al., 2013]
[He et al., 2015]

Chapitre 1

Fonction d'activation

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



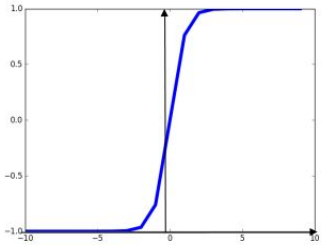
Sigmoïde

- Ramène les valeurs entre 0 et 1
- Historiquement populaire

3 Problèmes :

- Un neurone saturé a pour effet de **« tuer » les gradients**
- Sortie d'une sigmoïde n'est **pas centrée à zéro.**
- exp() est **coûteux** lorsque le nombre de neurones est élevé.

Fonction d'activation

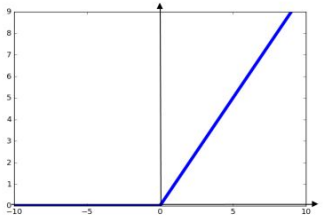


Tanh(x)

- Ramène les valeurs entre -1 et 1
- **Sortie centrée à zéro**
- **Disparition du gradient** lorsque la fonction sature

Fonction d'activation

$$\mathrm{ReLU}(x) = \max(0, x)$$



ReLU(x)
(Rectified Linear Unit)

- Aucune **saturation**
- Super **rapide**
- **Converge plus rapide** que sigmoïde/tanh (5 à 10x)

- Sortie **non centrée à zéro**
- **Un inconvénient :** qu'arrive-t-il au gradient lorsque x<0?

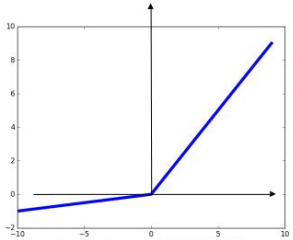
$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$$

[Krizhevsky et al., 2012]

Fonction d'activation

[LeCun et al., 1991]

$$\mathrm{LReLU}(x) = \max(0.01x, x)$$



Leaky ReLU(x)

- Aucune **saturation**
- Super **rapide**
- **Converge plus rapide** que sigmoïde/tanh (5 à 10x)
- **Gradients ne meurent pas**

- 0.01 est un **hyperparamètre**

[Mass et al., 2013]
[He et al., 2015]

Chapitre 1

Descente de gradient + **Momentum**

Descente de gradient stochastique

$$w_{t+1} = w_t - \eta \nabla \vec{E}_{x_n}(w_t)$$

Descente de gradient stochastique + **Momentum**

$$v_{t+1} = \rho v_t + \nabla \vec{E}_{x_n}(w_t)$$

$$w_{t+1} = w_t - \eta v_{t+1}$$

Provient de l'équation de la vitesse

ρ exprime la « friction », en général $\in [0.5, 1[$

Chapitre 1

RMSProp (AdaGrad amélioré)

AdaGrad

$$dE_t = \nabla E_{x_n}^{\rightarrow}(w_t)$$

$$m_{t+1} = m_t + |dE_t|$$

$$w_{t+1} = w_t - \frac{\eta}{m_{t+1} + \varepsilon} dE_t$$

RMSProp

$$dE_t = \nabla E_{x_n}^{\rightarrow}(w_t)$$

$$m_{t+1} = \gamma m_t + (1 - \gamma)|dE_t|$$

$$w_{t+1} = w_t - \frac{\eta}{m_{t+1} + \varepsilon} dE_t$$

- η décroit lorsque le gradient est élevé
- η augmente lorsque le gradient est faible

Chapitre 1

Adam (Version complète)

$$v_{t=0} = 0$$

$$m_{t=0} = 0$$

```
for t=1 à num_iterations
    for n=0 à N
```

$$dE_t = \nabla E_{x_n}(w_t)$$

$$v_{t+1} = \alpha v_t + (1 - \alpha)\, dE_t$$

$$m_{t+1} = \gamma m_t + (1 - \gamma)\, |dE_t|$$

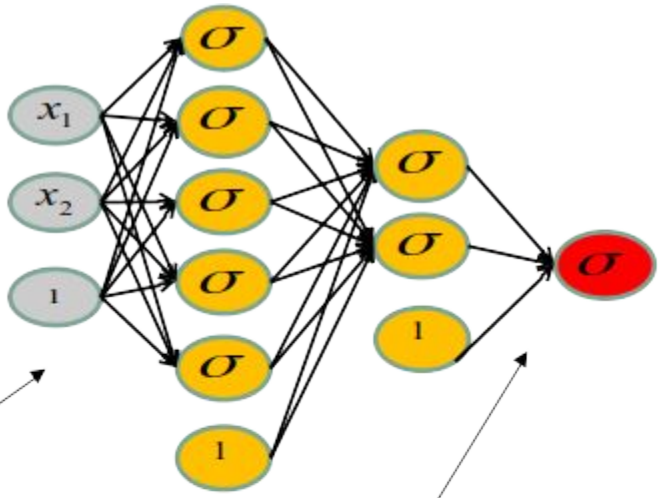
$$v_{t+1} = \frac{v_{t+1}}{1 - \beta_1^{\,t}}, \quad m_{t+1} = \frac{m_{t+1}}{1 - \beta_2^{\,t}}$$

$\beta_1 = 0.9, \beta_2 = 0.99$

$$w_{t+1} = w_t - \frac{\eta}{m_{t+1} + \varepsilon} v_{t+1}$$

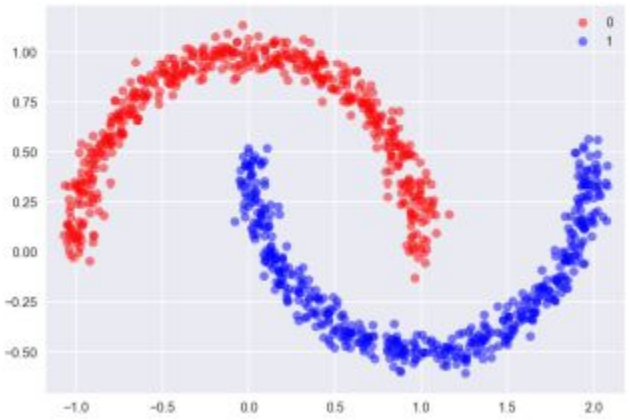
Réseaux convolutifs

Example

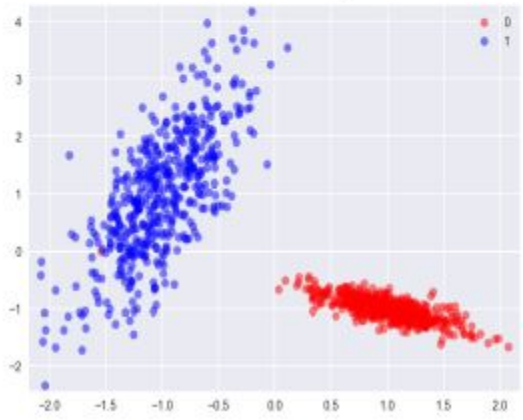


x

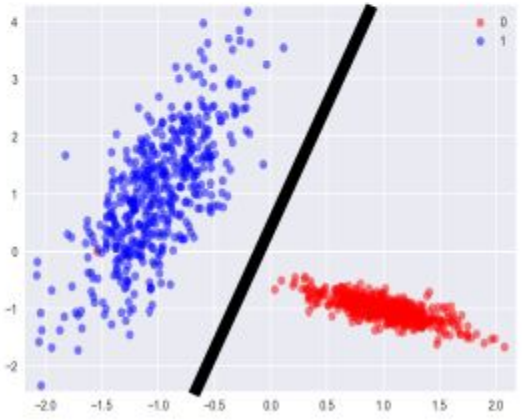
$y_W(x)$



Données en entrée



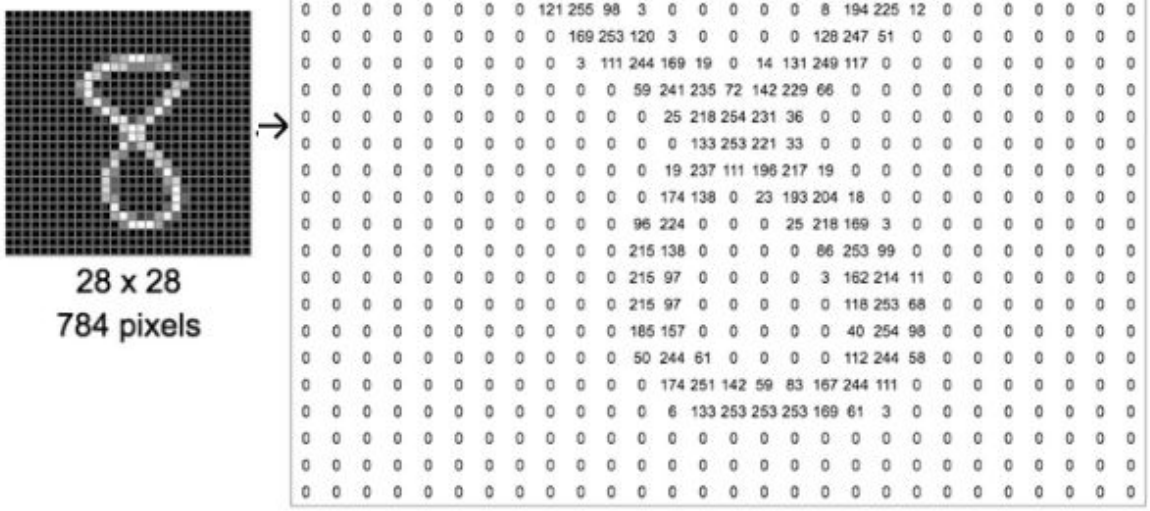
Sortie de la dernière couche



Sortie du réseau

Réseaux convolutifs

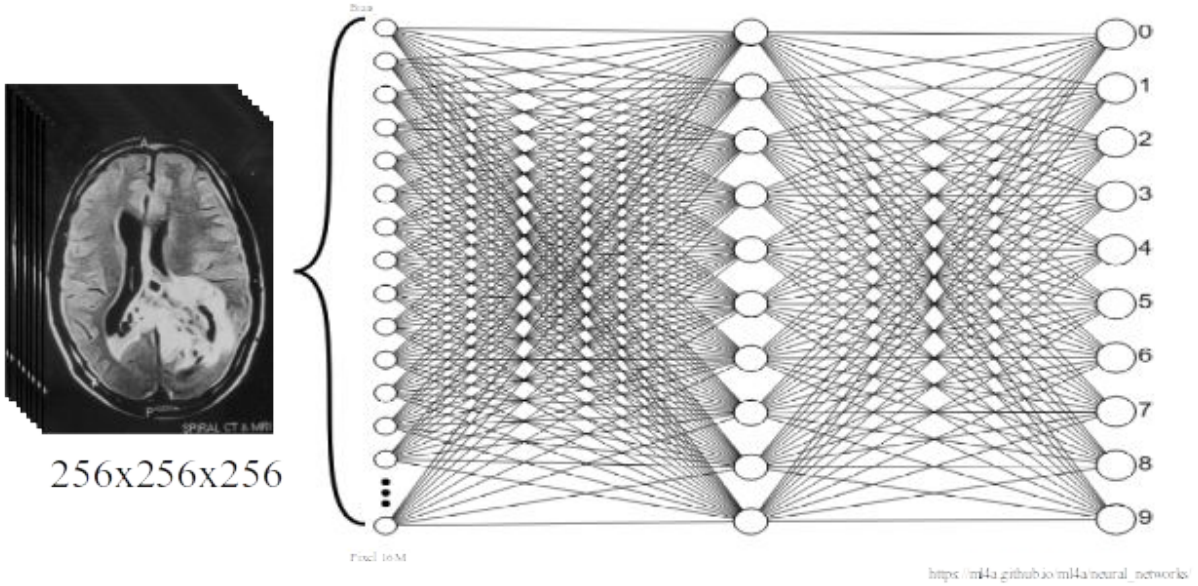
Comment classifier des images?



https://ml4a.github.io/ml4a/neural_networks/

Réseaux convolutifs

Beaucoup **TROP** de paramètres (160M de params dans la couche 1)



256x256x256

Image médicale 3D (IRM de cerveau)

Réseaux convolutifs

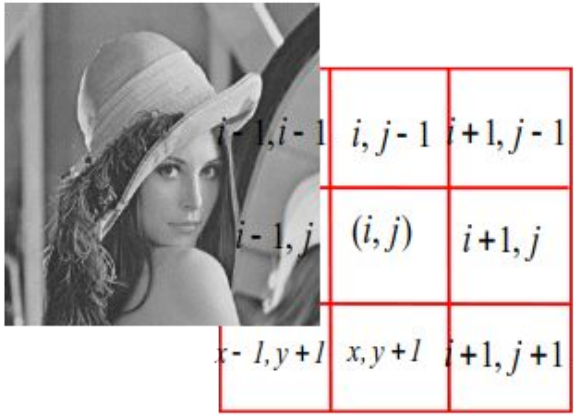
Filtrage 2D

(sans flip de filtre)

1D: $(\vec{f} * \vec{w})(v) = \sum_u \vec{f}(u)\vec{w}(v-u) = \sum_u \vec{f}(v+u)\overleftarrow{w}(u)$

$$(x * W)(i,j) = \sum_u \sum_v f(i+u, j+v) W(u,v)$$

$x(i,j)$



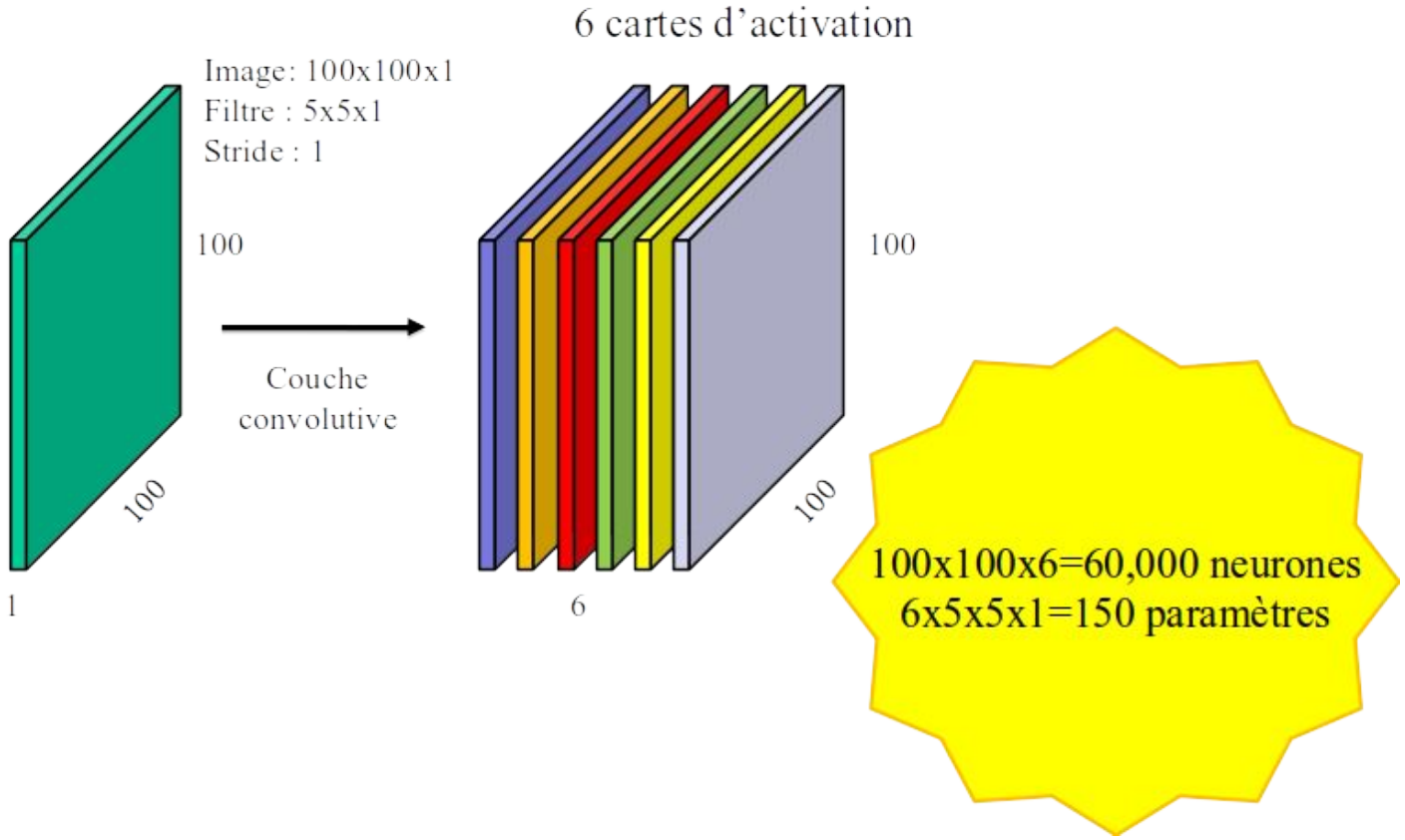
$W(u,v)$

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

$$\begin{aligned}(x * W)(i,j) = {} & w_1 x(i-1, j-1) + w_2 x(i, j-1) + w_3 x(i+1, j-1) \\ & + w_4 x(i-1, j) \quad + w_5 x(i, j) \quad + w_6 x(i+1, j) \\ & + w_7 x(i-1, j+1) + w_8 x(i, j+1) + w_9 x(i+1, j+1)\end{aligned}$$

Réseaux convolutifs

Représentation schématique
(6 filtres et 6 carte d'activation, convolution « *same* »)



Réseaux convolutifs

Max pooling

1	2	4	4	9	3	1	2
6	7	8	4	-3	-3	6	3
9	-9	8	-4	5	5	3	0
8	-8	9	-9	5	5	0	1
0	0	1	2	7	9	7	8
-1	-3	3	6	8	8	7	6
9	9	8	2	1	5	-1	-1
1	1	-2	8	3	7	4	-2

Max pool par filtre 2x2 avec stride =2

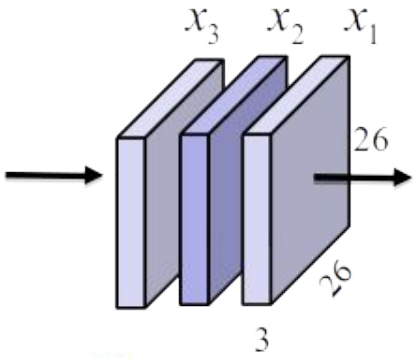
7	8	9	6
9	9	5	3
0	6	9	8
9	8	7	4

Réseaux convolutifs

Normalisation par lot (*Batch norm*)

[Ioffe and Szegedy, 2015]

Ex. : normalisation de la couche 1



x_i : les cartes d'activations du ième lot

→ **Contient 26x26x3= 2028 neurones**

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$ → **Taille 26 x 26 x 3**

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$ → **Taille 26 x 26 x 3**

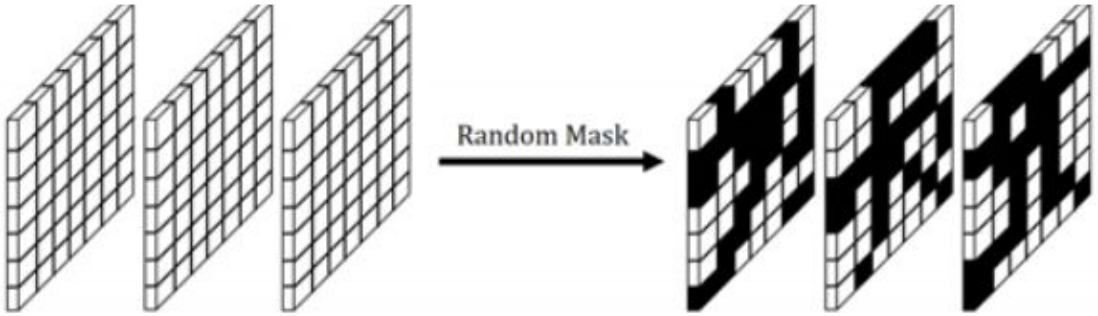
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$$ → **Taille 26 x 26 x 3**

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$$ → **γ et β et taille 26 x 26 x 3**

Réseaux convolutifs

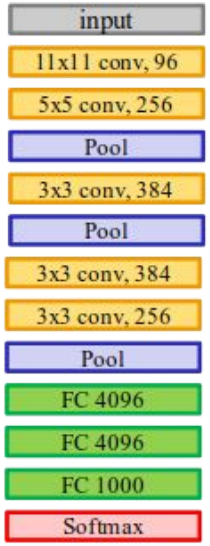
Autre bonne pratique : *Dropout*

Dans le cas d'un réseau à convolution, dropout revient à appliquer un **masque binaire aléatoire** à chaque carte d'activation.

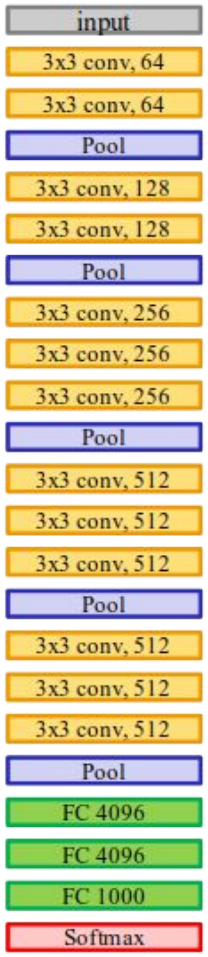


Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, Christoph Bregler
"Efficient Object Localization Using Convolutional Networks", in proc of CVPR 2015

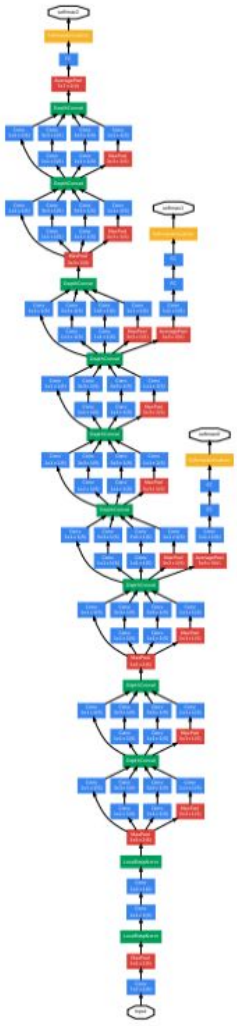
Réseaux convolutifs



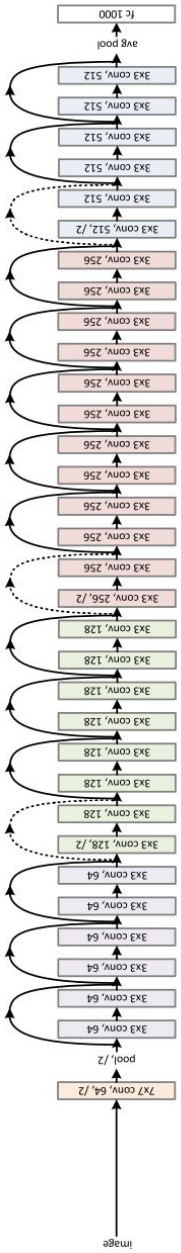
AlexNet



VGG16



GoogLeNet

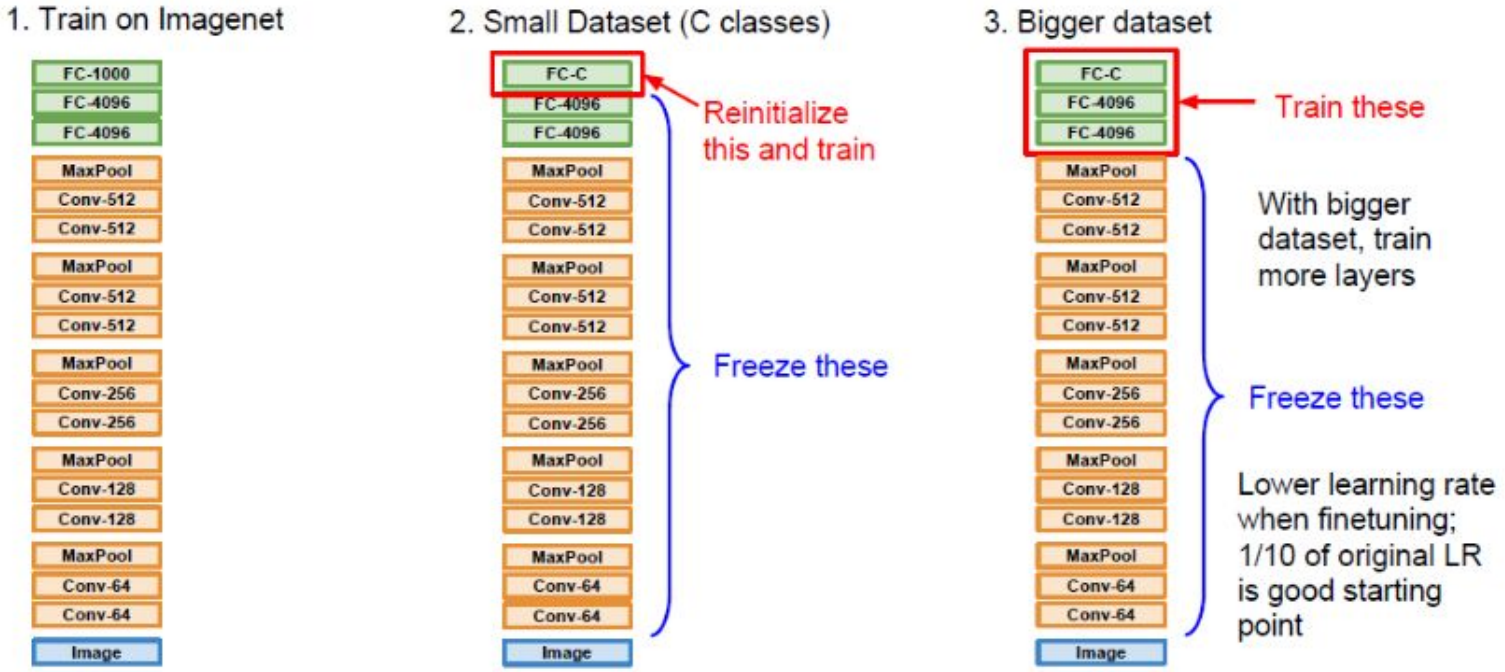


ResNet

Réseaux convolutifs

Transfert d'apprentissage

(Transfer learning)



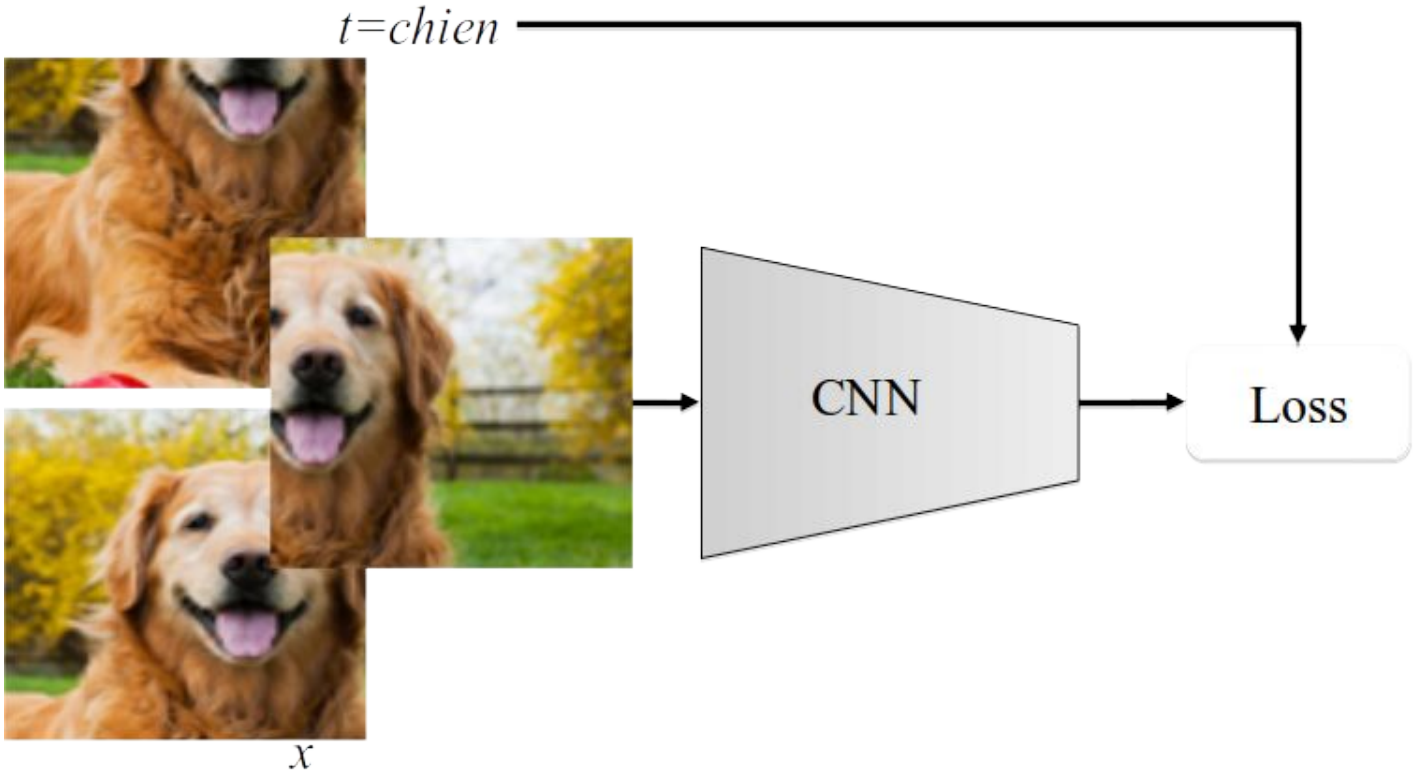
Crédit : http://cs231n.stanford.edu

Réseaux convolutifs

Augmentation de données

(Data augmentation)

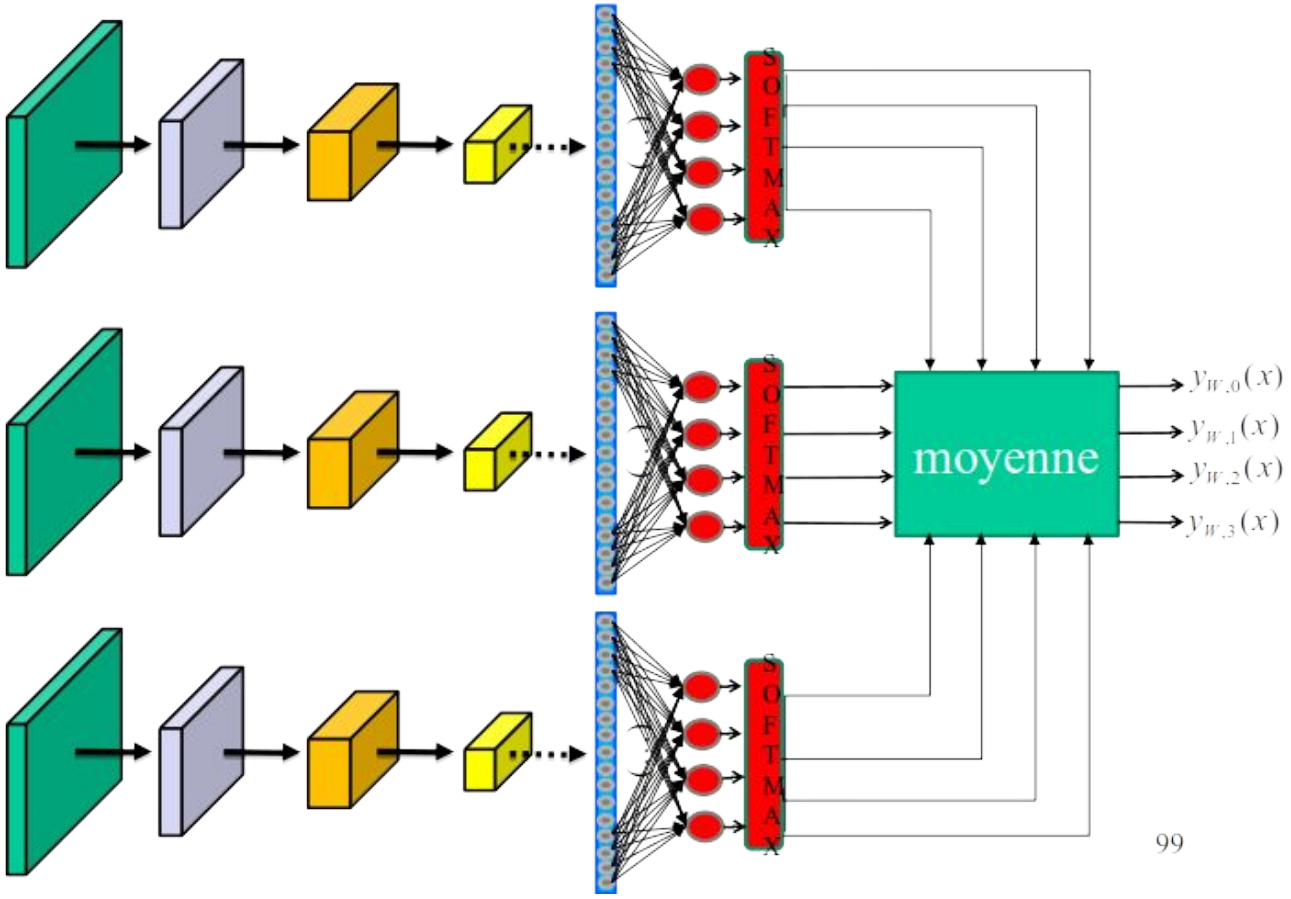
Exemple de transformation : **crop aléatoire + redimension**



Réseaux convolutifs

Ensemble de modèles

Une fois les modèles entraînés, on combine leur sortie



Segmentation et localisation

Segmentation sémantique

Image

Segmentation

But: Assigner la bonne étiquette de classe à **chaque pixel de l'image d'entrée**

Peut être vu comme un problème de classification **dense** et **structurée**

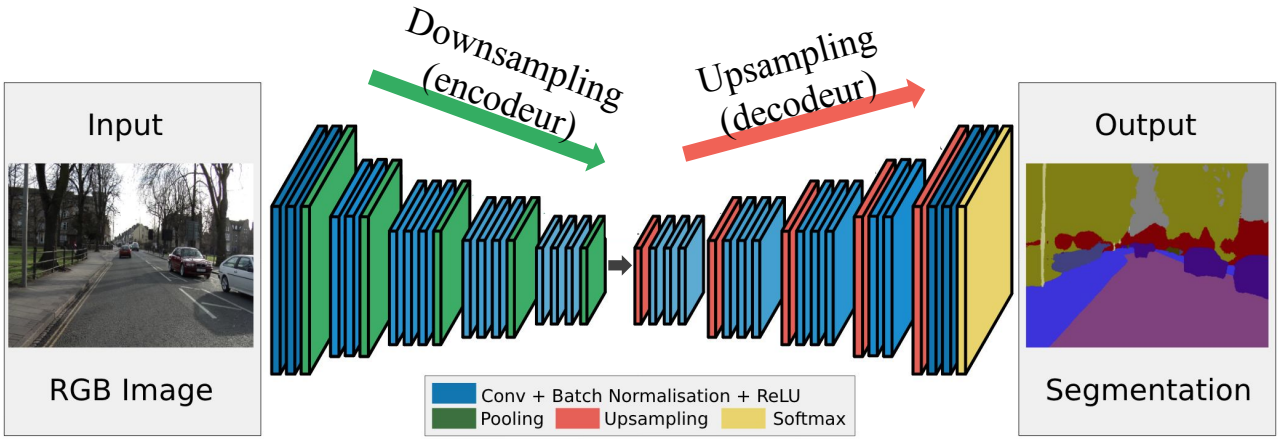
- **dense** : Possiblement des millions de pixels
- **structurée** : Prédictions structure spatiale

Segmentation et localisation

Encodeur-décodeur

Encodeur: projette l'image d'entrée vers un espace de plus faible dimension
Décodeur: projette l'image de faible dimension vers l'espace souhaité

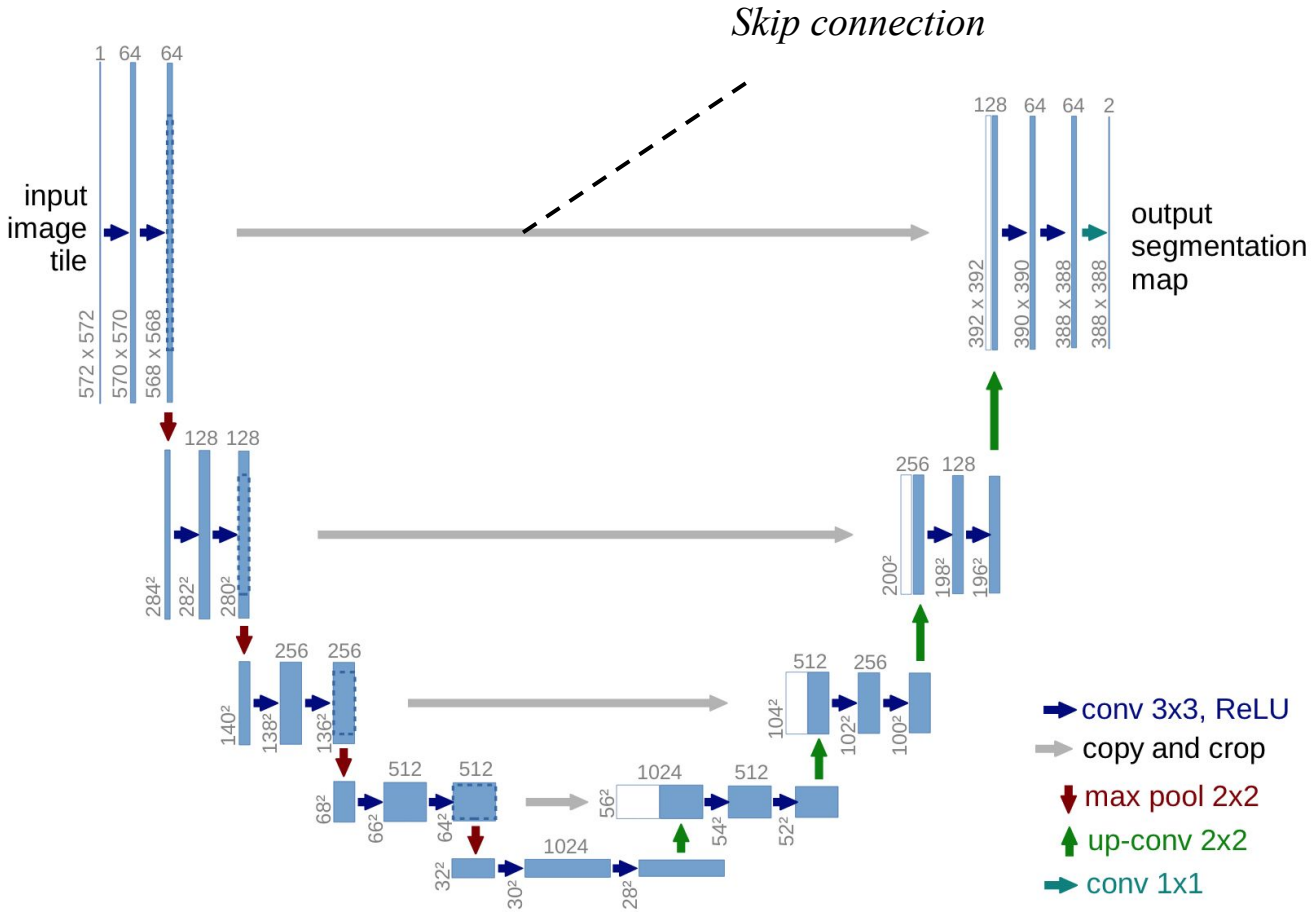
Architecture *généralement* symétrique



Adapté de:
Badrinarayanan et al. "Segnet: A deep convolutional encoder-decoder architecture for image segmentation." PAMI, 2017.

Segmentation et localisation

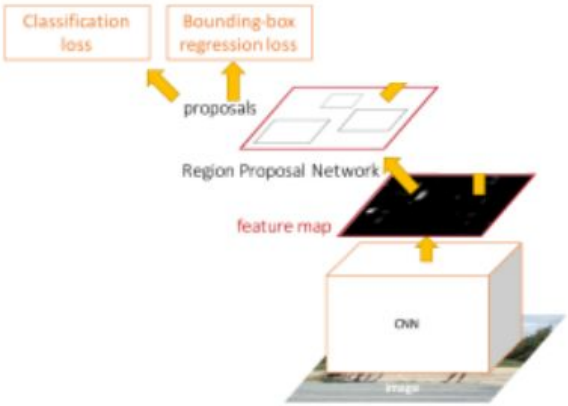
U-Net [Ronneberger et al., 2015]



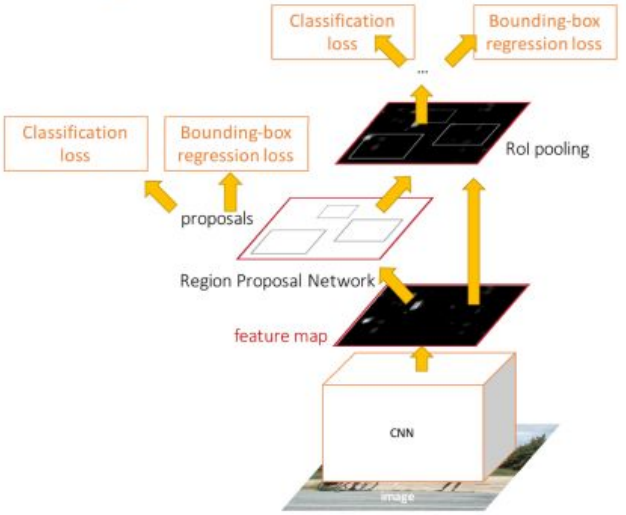
Segmentation et localisation

Object Detection: Single Stage vs Two Stage

Single-Stage:
YOLO, SSD, RetinaNet
Make all predictions with a CNN



Two-Stage:
Faster R-CNN
Use RPN to predict proposals, classify them with second stage

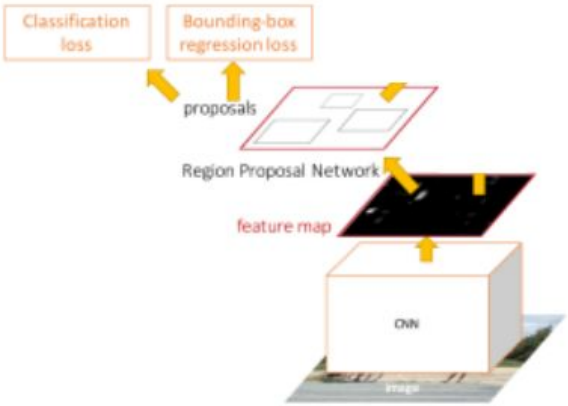


source: Justin Johnson, cs231n

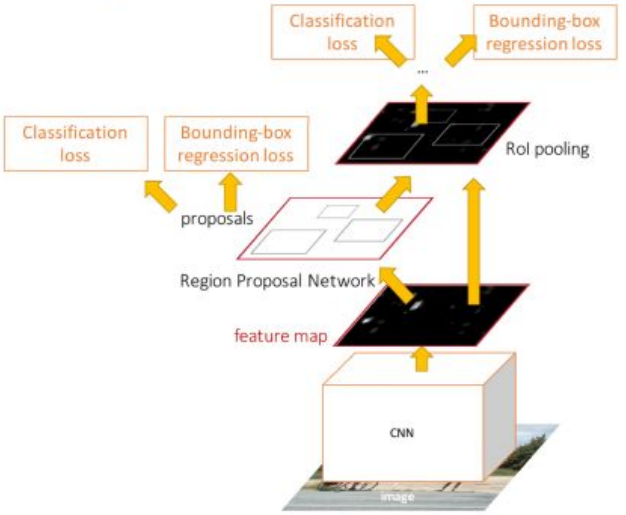
Segmentation et localisation

Object Detection: Single Stage vs Two Stage

Single-Stage:
YOLO, SSD, RetinaNet
Make all predictions with a CNN

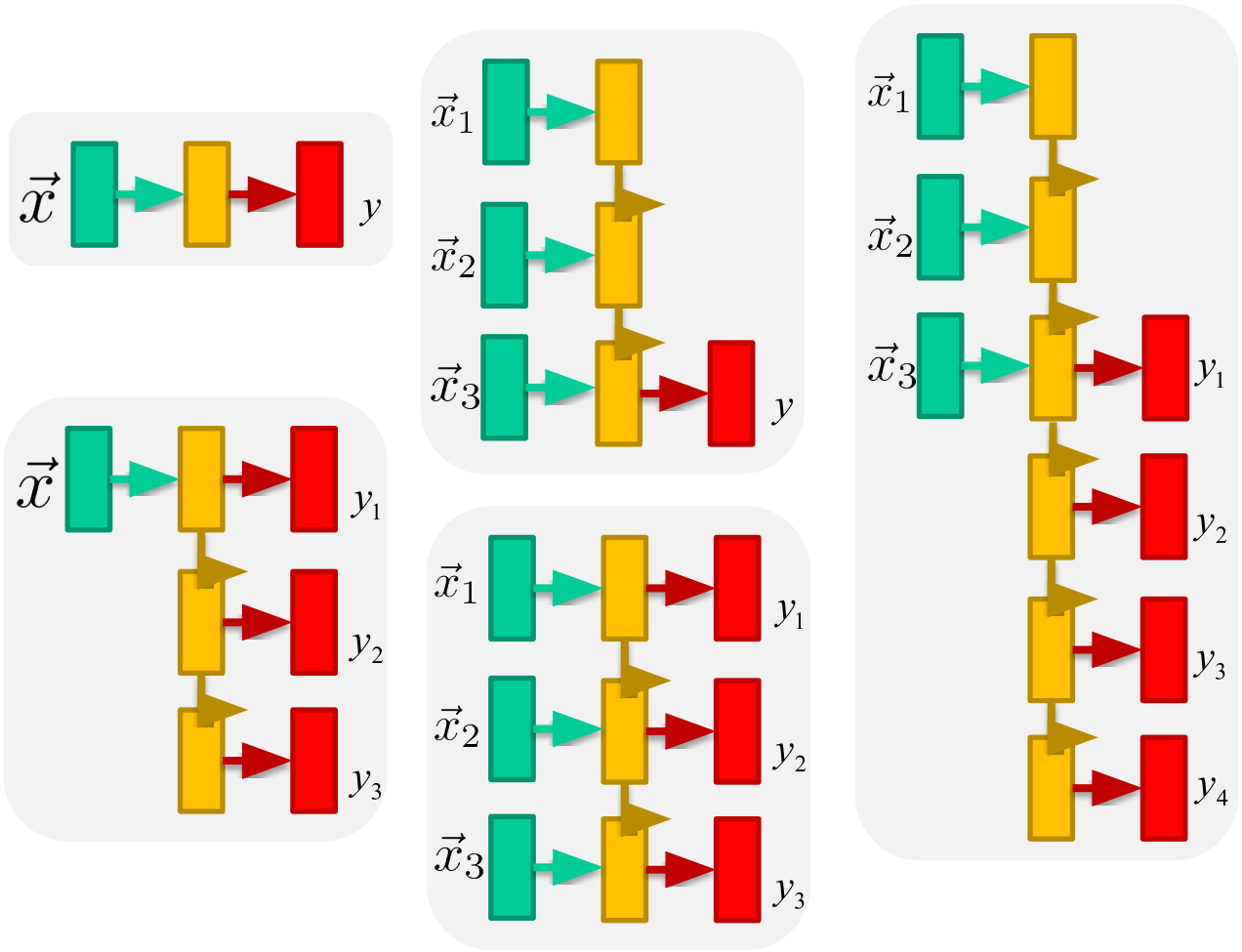


Two-Stage:
Faster R-CNN
Use RPN to predict proposals, classify them with second stage

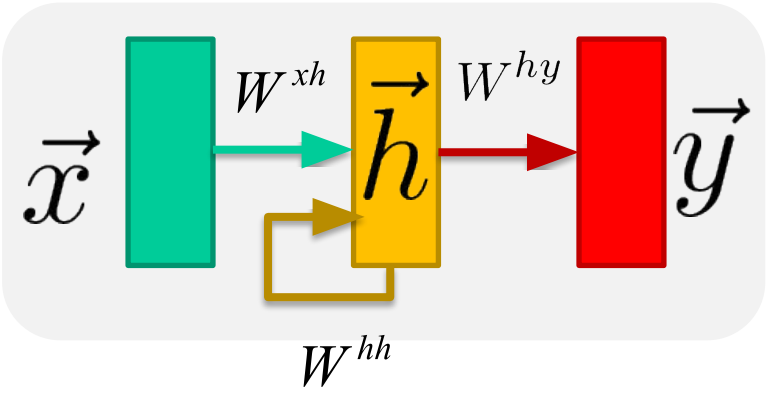


source: Justin Johnson, cs231n

Réseaux récurrents



Réseaux récurrents



$$y(\vec{x}) = W^{hy} f_a(W^{xh}\vec{x} + W^{hh}\vec{h})$$

$$\vec{h} = f_a(W^{xh}\vec{x} + W^{hh}\vec{h})$$

$$\hat{y} = W^{hy}\vec{h}$$

$$\vec{y}(\vec{x}) = \text{softmax}(\hat{y})$$

Réseaux récurrents

PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

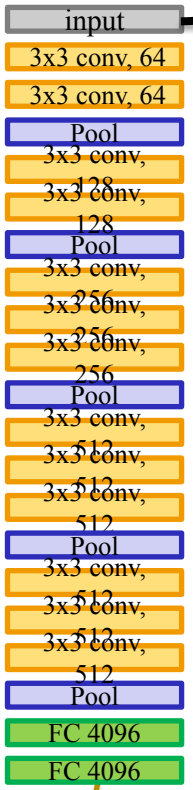
VIOLA:
I'll drink it.

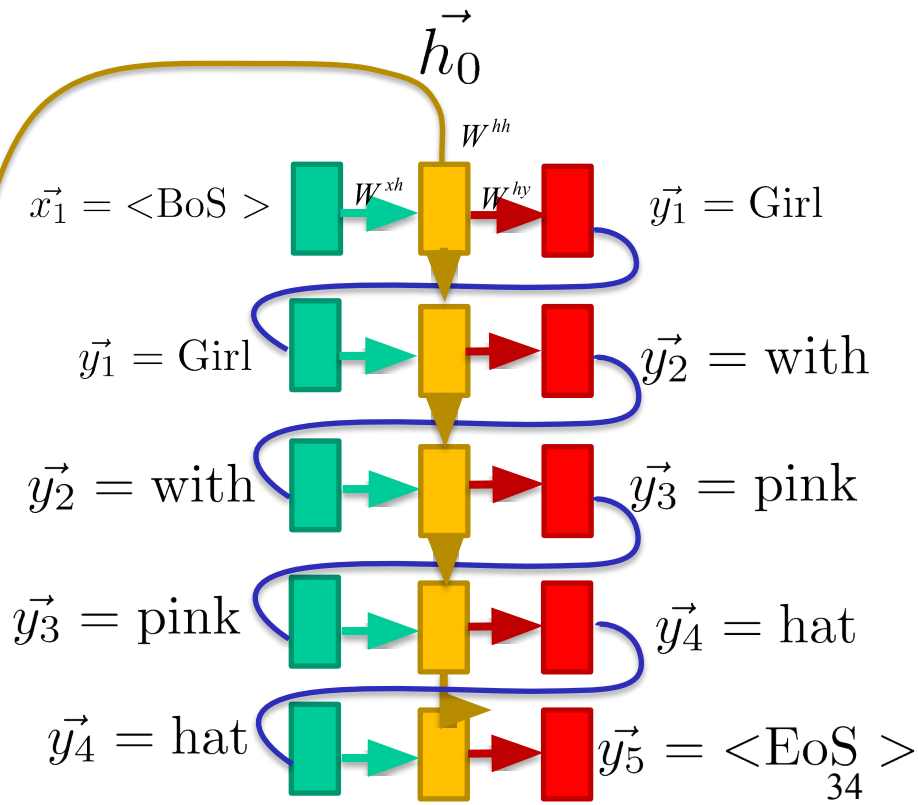
VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.

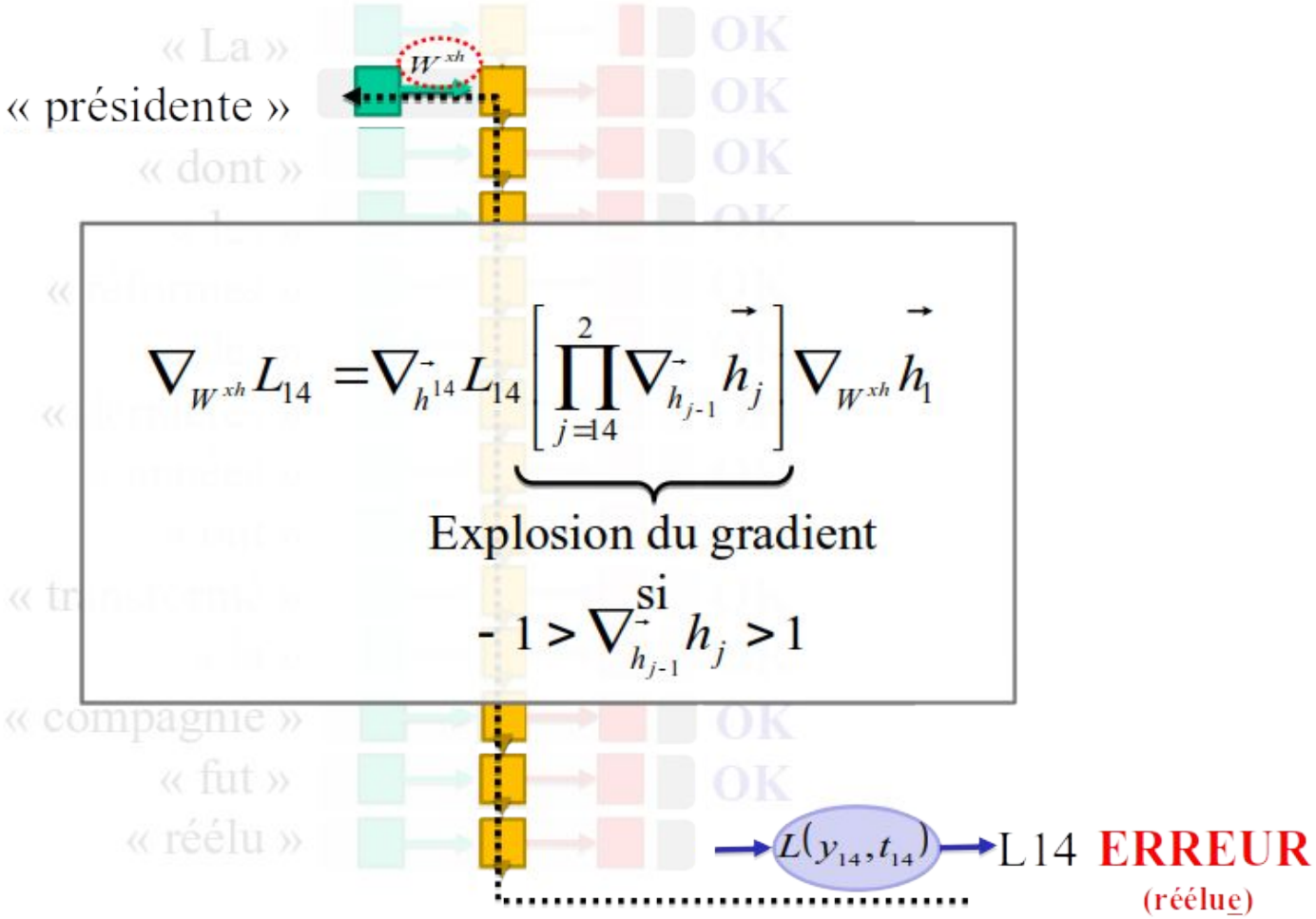
Réseaux récurrents

Captioning

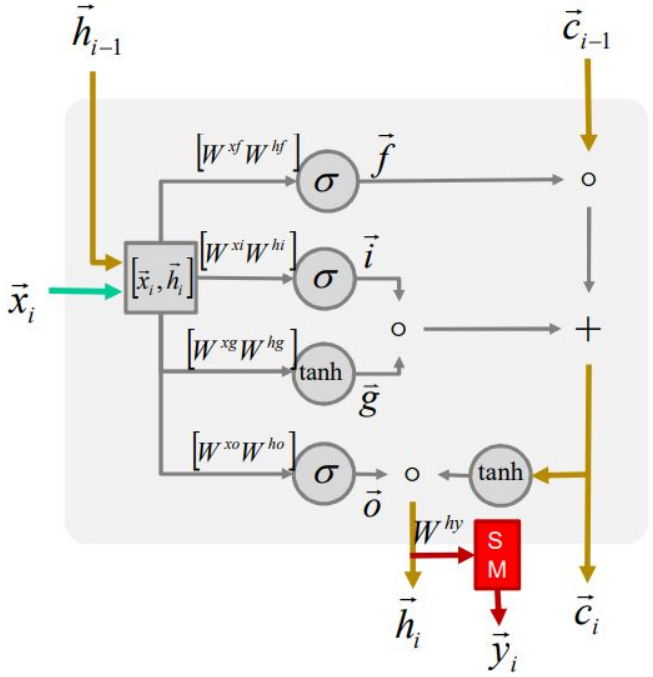




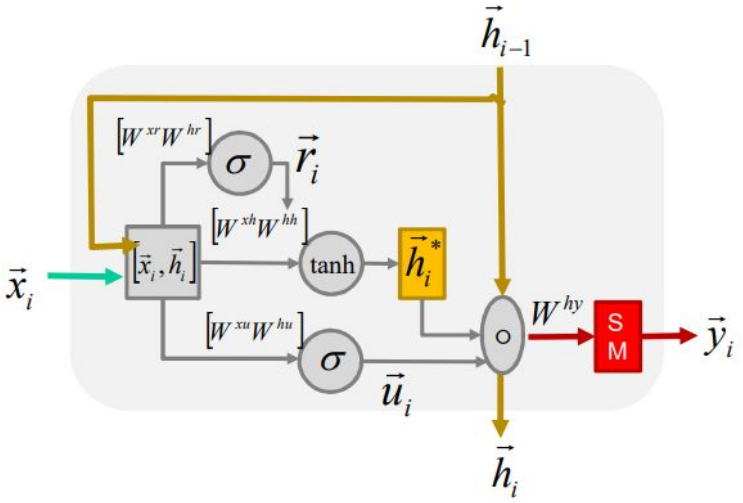
Réseaux récurrents



Réseaux récurrents



LSTM



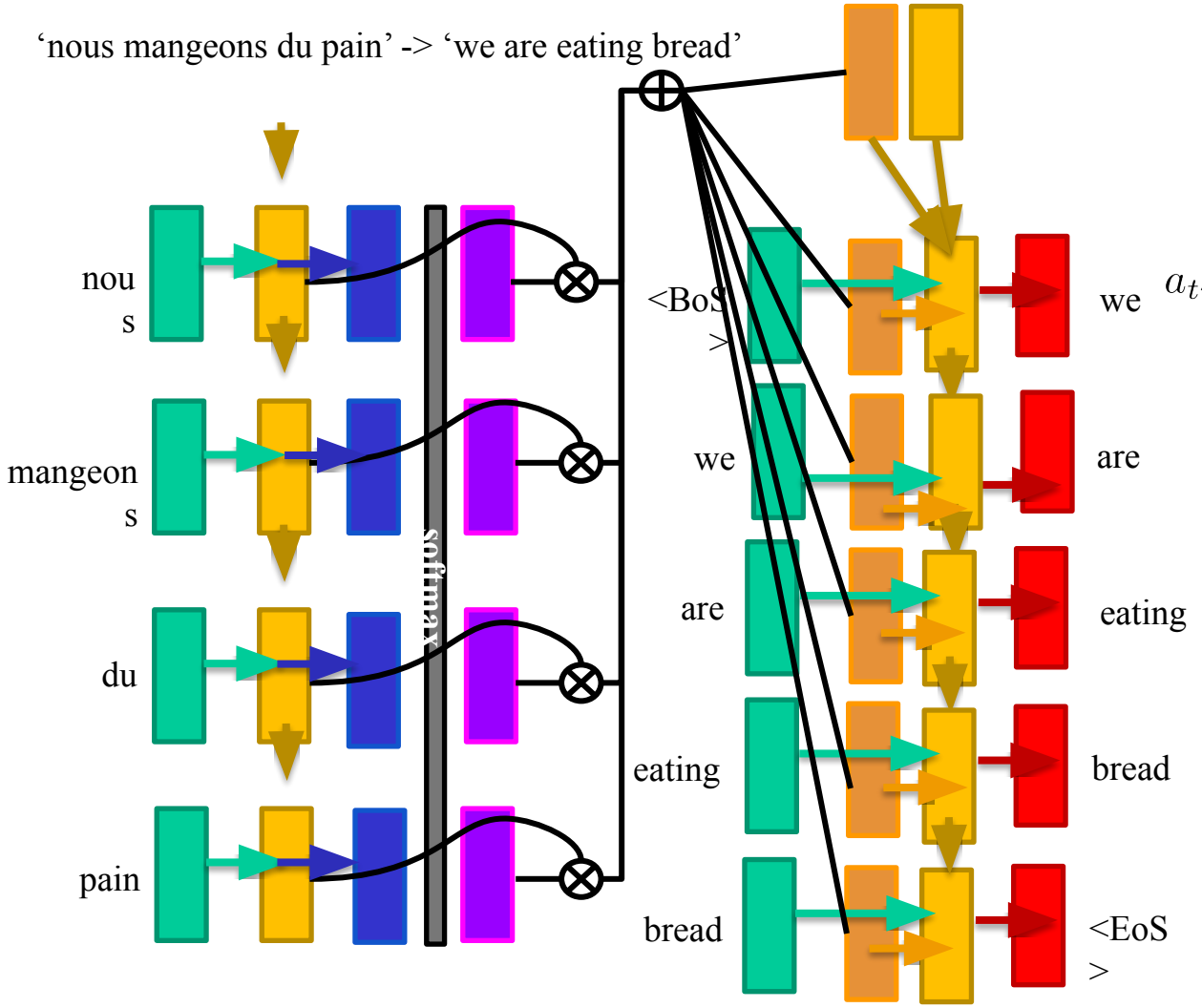
GRU

Réseaux récurrents

Seq2Seq avec attention

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

‘nous mangeons du pain’ -> ‘we are eating bread’



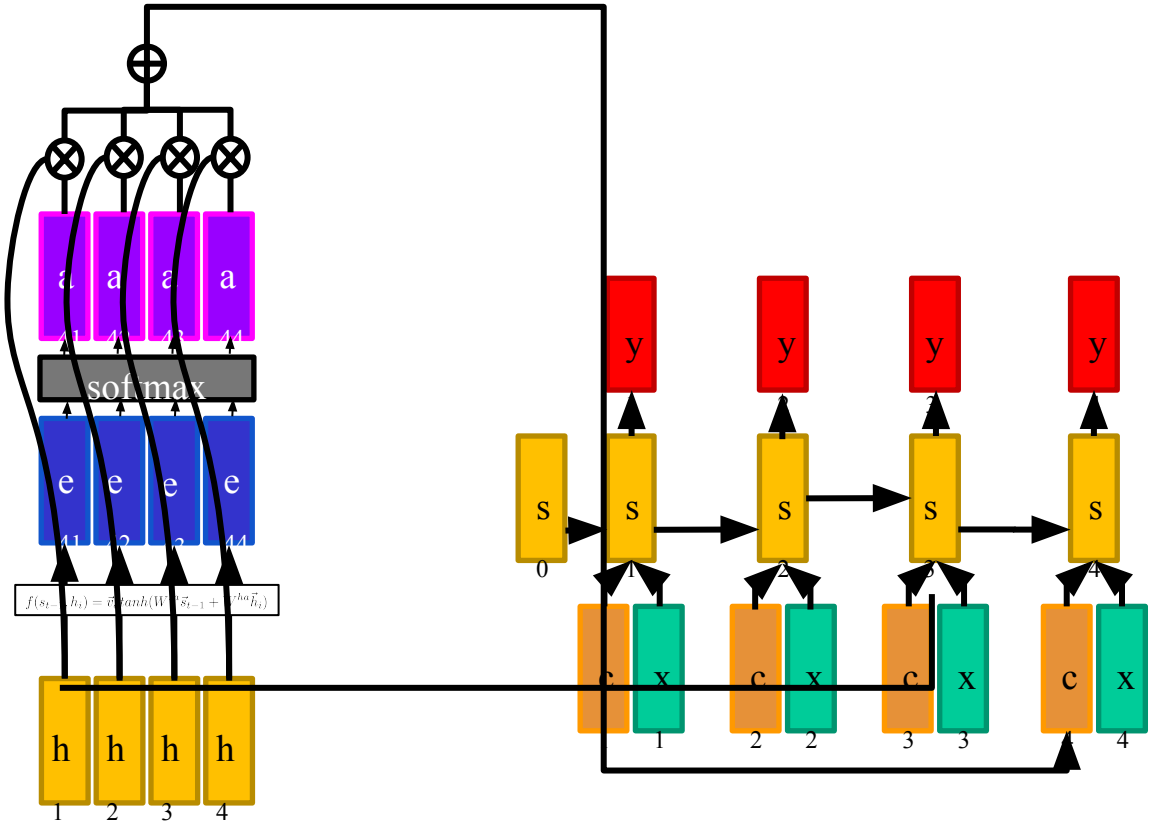
$$e_{ti} = f(s_{t-1}, h_i)$$

$$a_{ti} = \frac{exp(e_{ti})}{\sum_{j=1}^{T} exp(e_{tj})}$$

$$c_t = \sum_{i=1}^{T} a_{ti} h_i$$

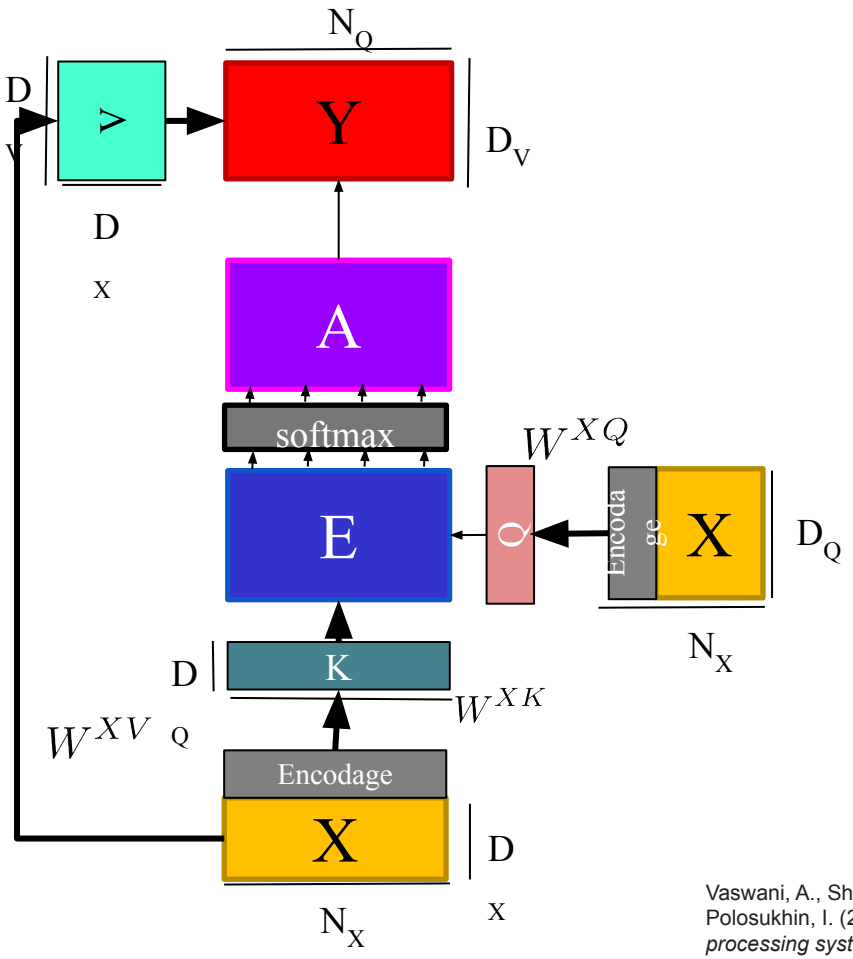
Attention

Attention: version simplifiée



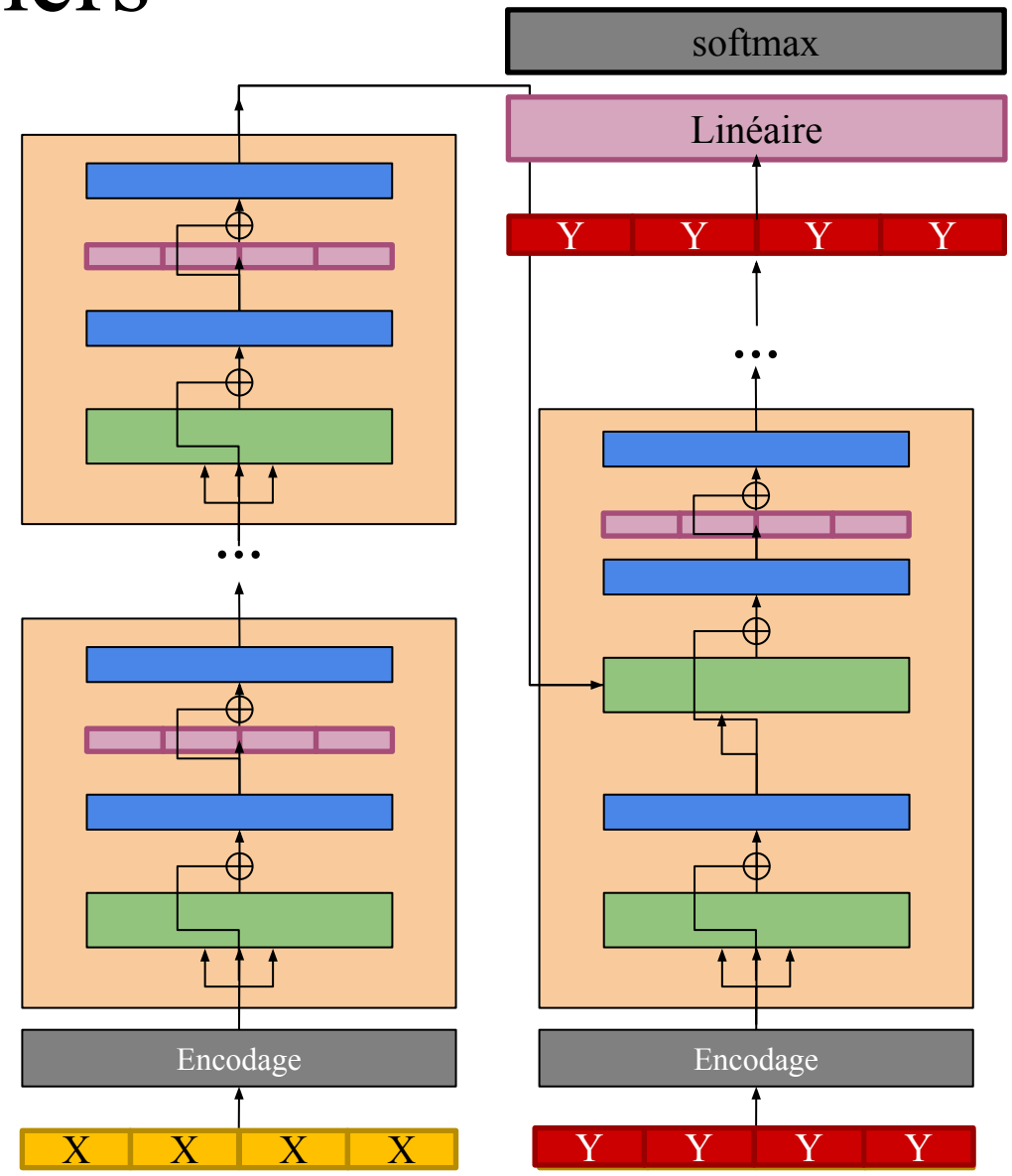
Attention

Self-attention (intra-attention)



Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Transformers



Modèles génératifs

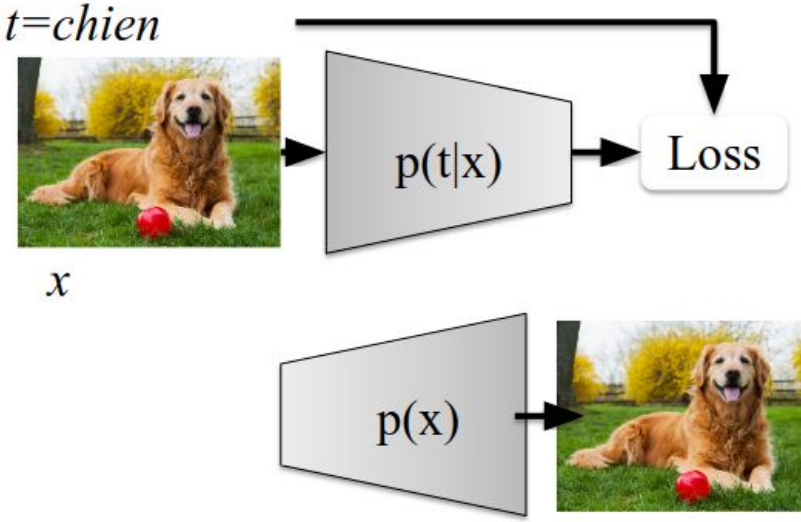
Modèles discriminatifs vs. génératifs

p(t|x) -> p(chien| )

probabilité de la classe t en sachant x

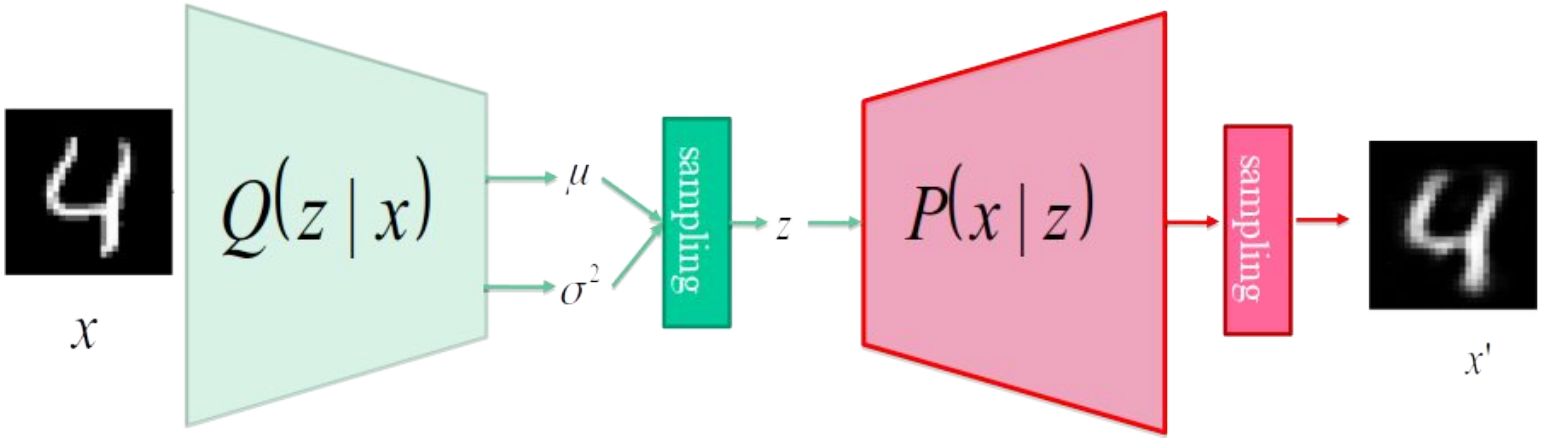
p(x) -> p()

probabilité de x



Modèles génératifs

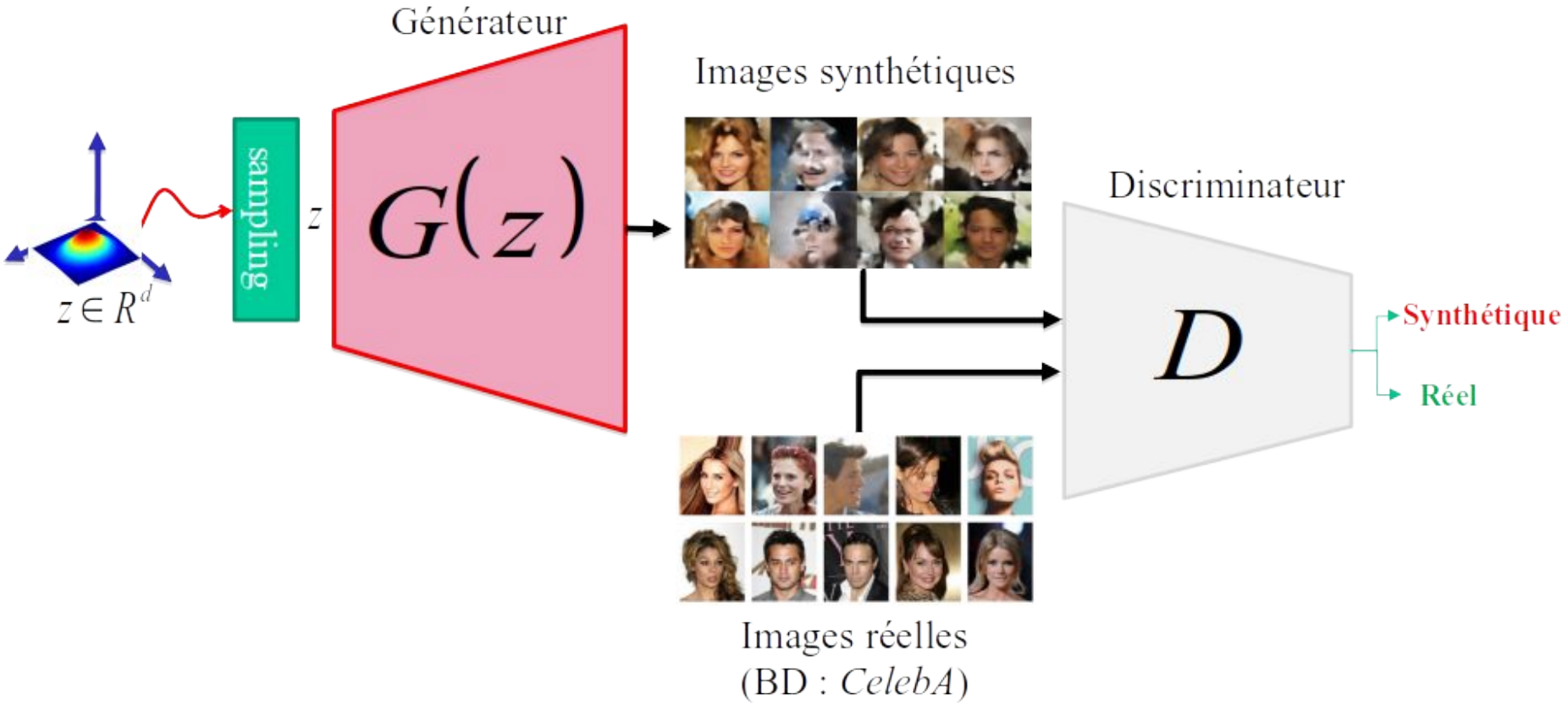
Autoencodeur
variationnel



ELBO loss : Evidence Lower Bound

$$Loss = \underbrace{KL(N(z;0,1), N(z;\mu,\Sigma))}_{\text{Perte encodeur}} - \underbrace{\log(P(x \mid z))}_{\text{Perte décodeur}}$$

Modèles génératifs



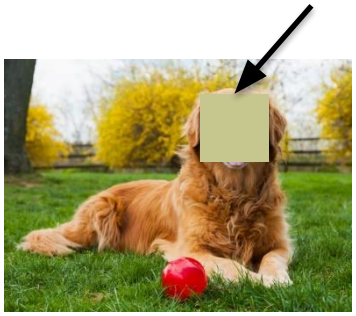
Visualisation

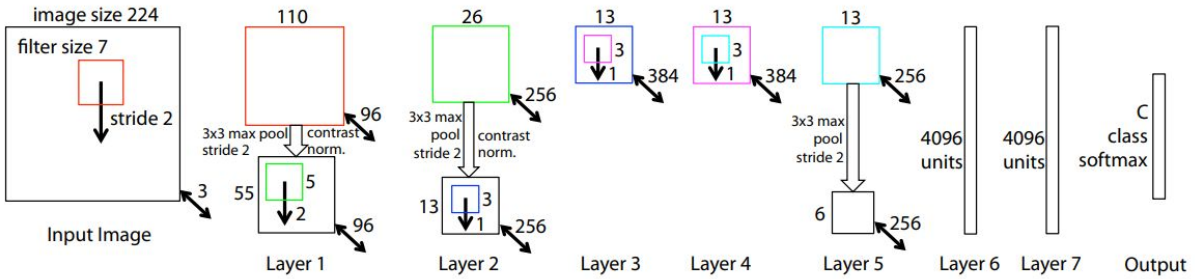
input

Boîte fermée

p(chien) = 0.9

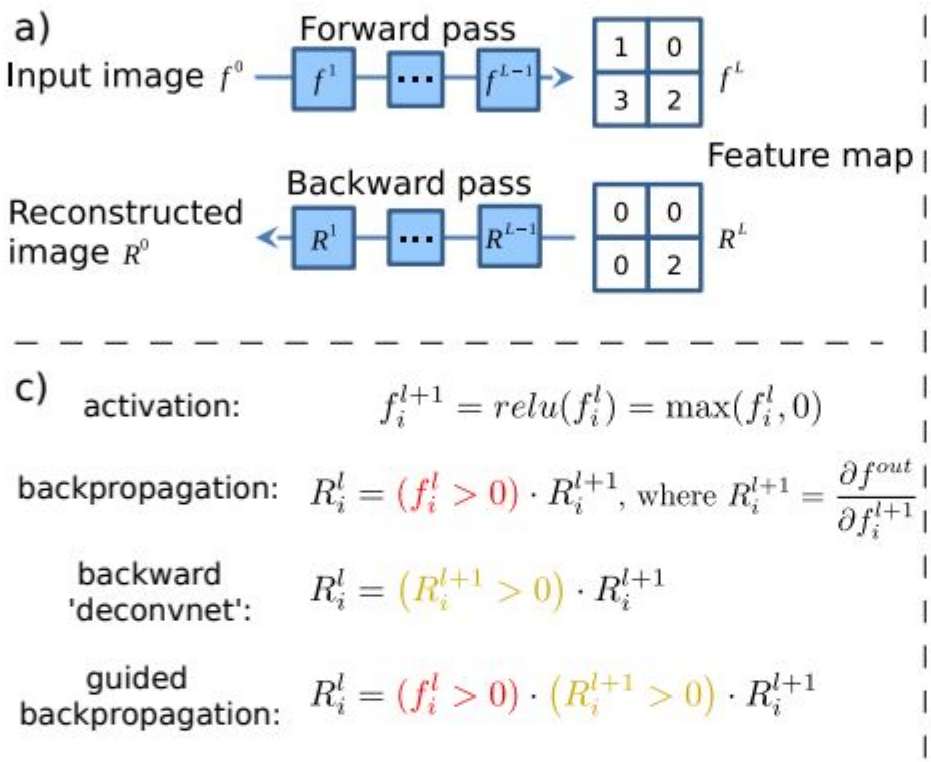
Visualisation

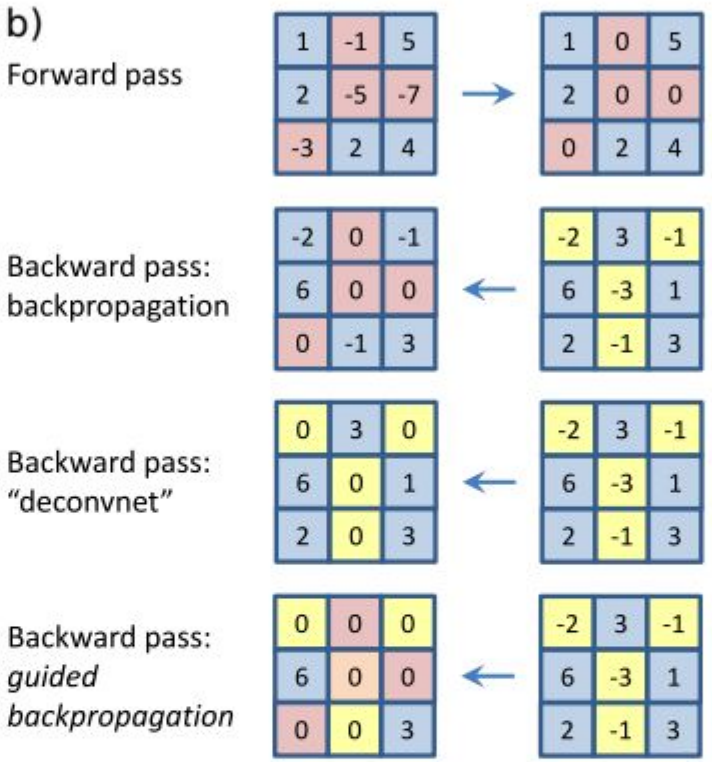




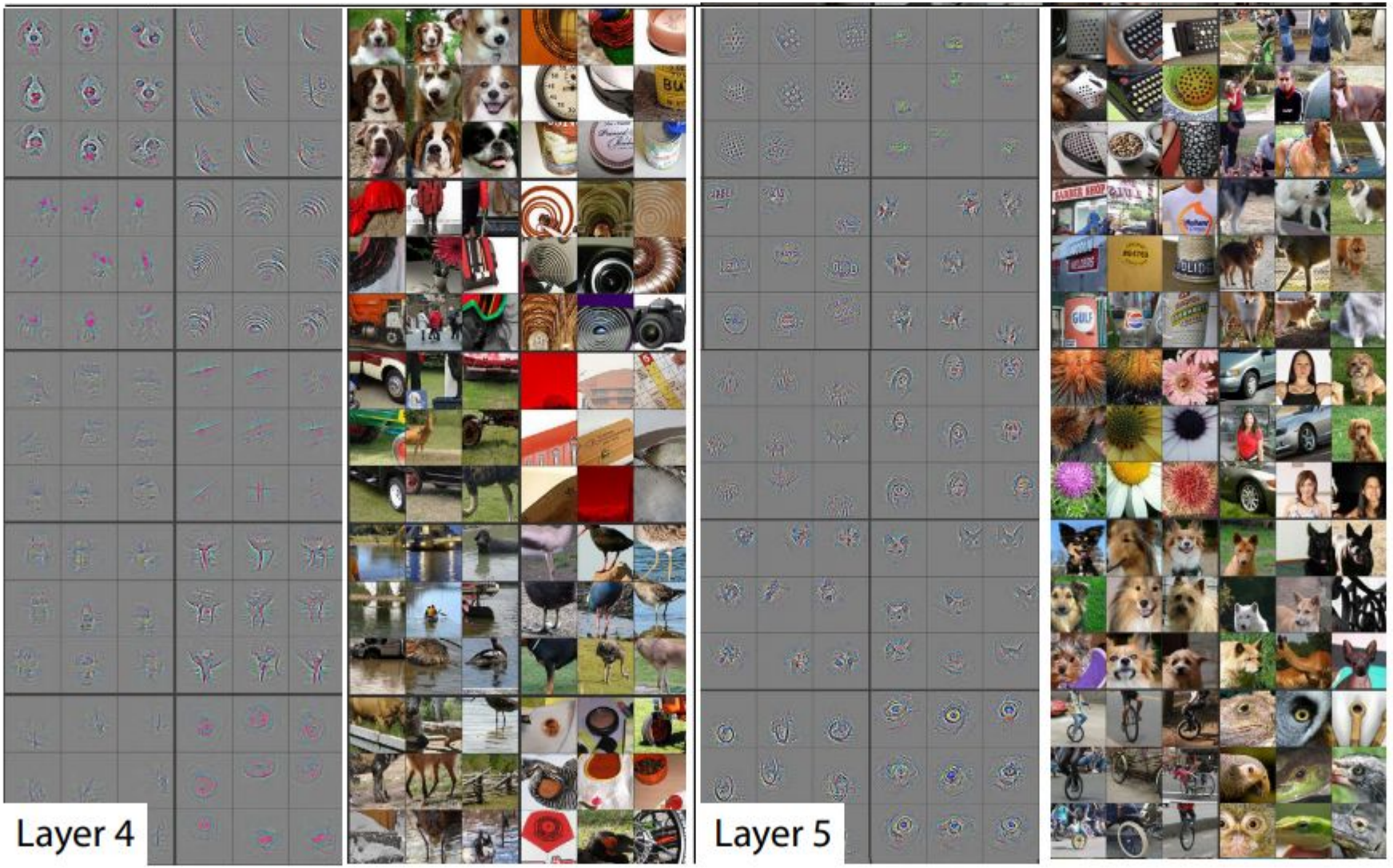
‘lion .34’

Visualisation

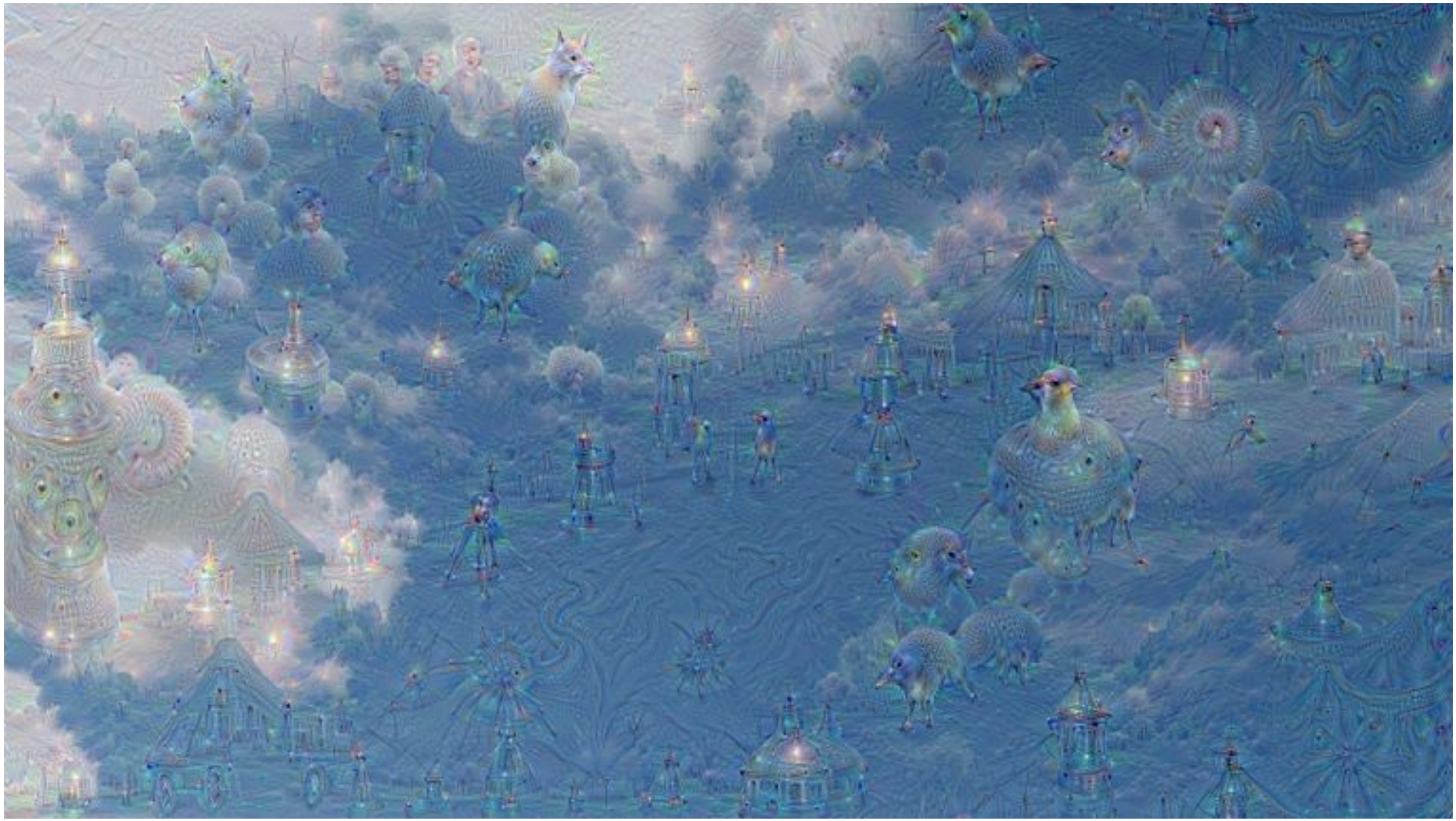




Visualisation

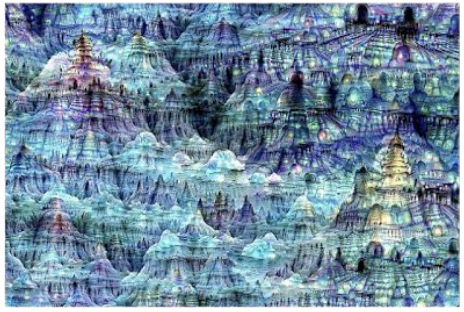
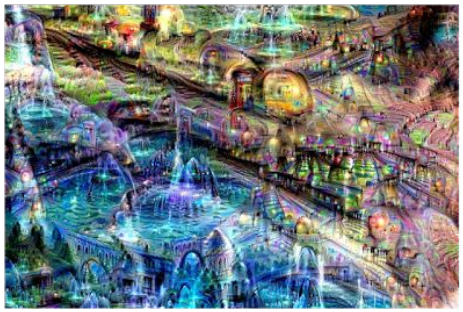


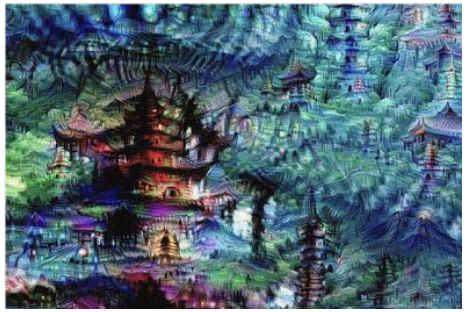
Visualisation



Visualisation





Apprentissage par renforcement