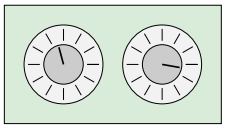
Apprentissage profond et intelligence artificielle

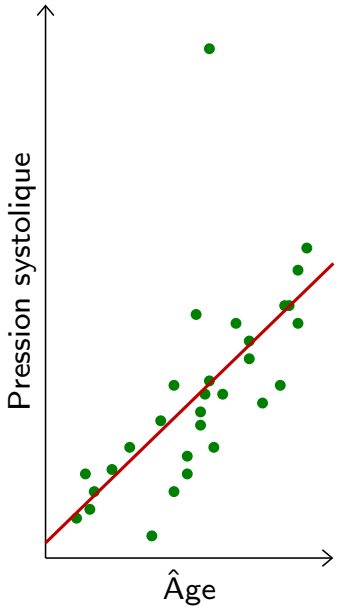
Formation CRM

François Fleuret
http://www.idiap.ch/~fleuret/
19 septembre 2019

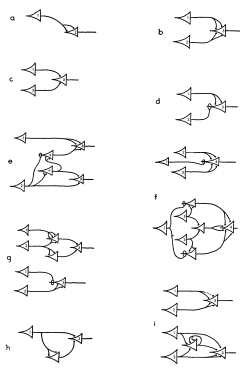
Apprentissage artificiel

L'intelligence artificielle "moderne" repose sur des méthodes d'apprentissage statistique capables d'adapter des modèles à des données.



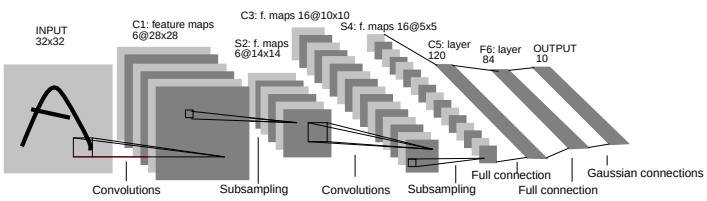


Les modèles les plus performants sont très proches des réseaux de neurones artificiels "traditionnels."

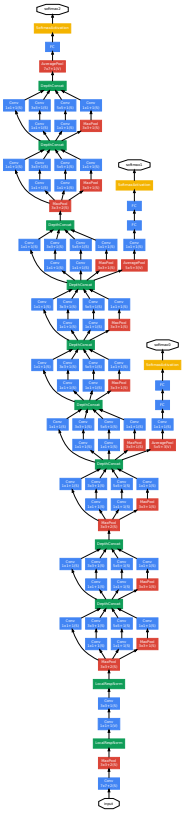


1943 – McCulloch et Pitts modèlent un réseau de neurones.
1949 – Hebb propose une règle d'apprentissage.
1951 – Minsky crée le premier réseau de neurones artificiel.
1958 – Rosenblatt crée un neurone qui classe des images.
1959 – Hubel et Wiesel analysent le cortex visuel du chat.
1982 – Werbos propose la rétro-propagation de l'erreur.

LeNet-5



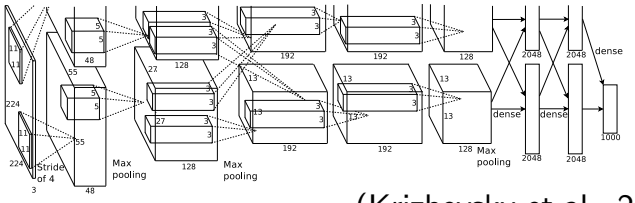
(leCun et al., 1998)



(Szegedy et al., 2015)

. . . 1990–2010 “éclipse des réseaux de neurones” . . .

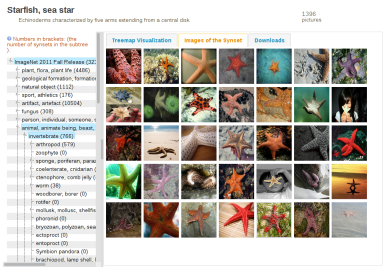
AlexNet

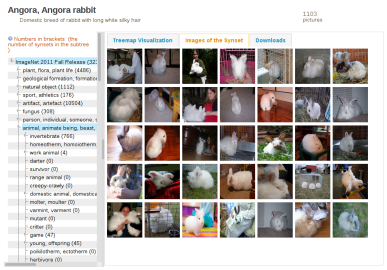


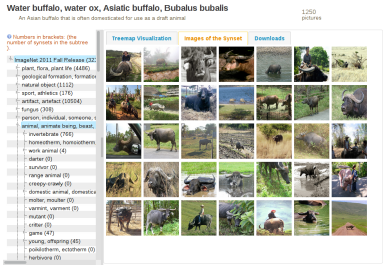
(Krizhevsky et al., 2012)

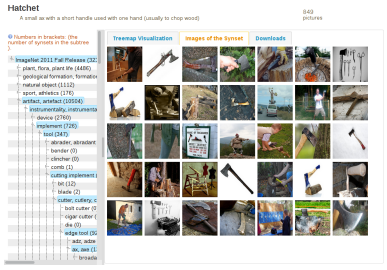
ImageNet Large Scale Visual Recognition Challenge.

1000 categories, $>$ 1M images

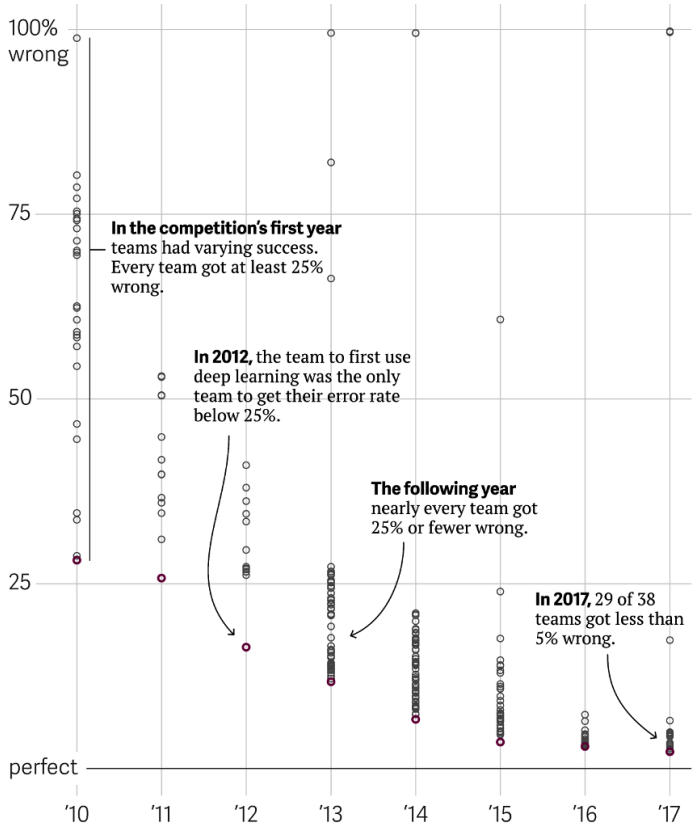








(http://image-net.org/challenges/LSVRC/2014/browse-synsets)

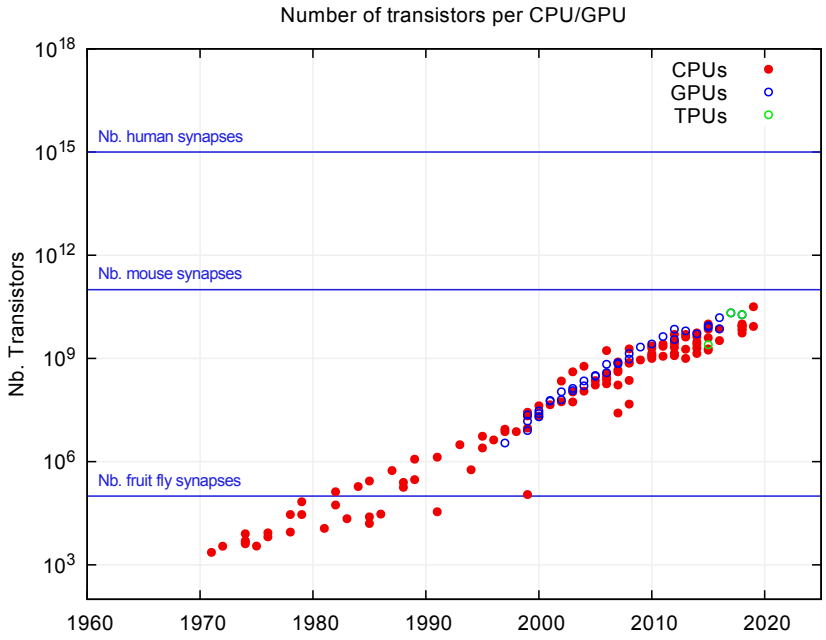


(Gershgorn, 2017)

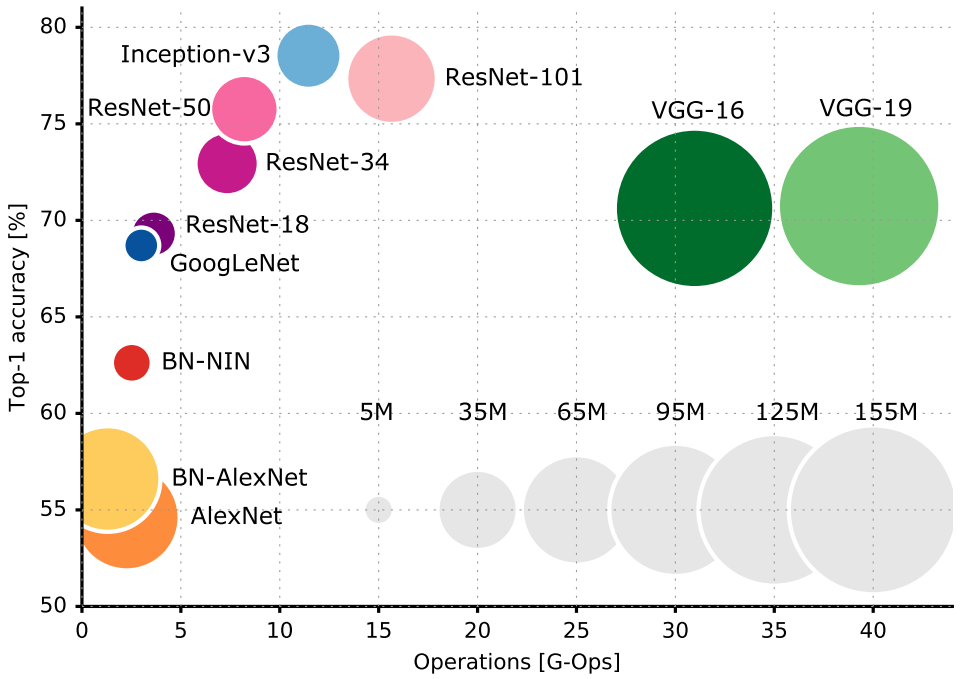
La cause des progrès récents et spectaculaires de l'IA est multi-factorielle:

- Cinq décennies de recherche en intelligence artificielle,
- des ordinateurs développés pour d'autres raisons,
- d'énormes quantités de données grâce à "internet",
- une culture de développement collaboratif de logiciels libres,
- des ressources et des efforts de grands groupes industriels.

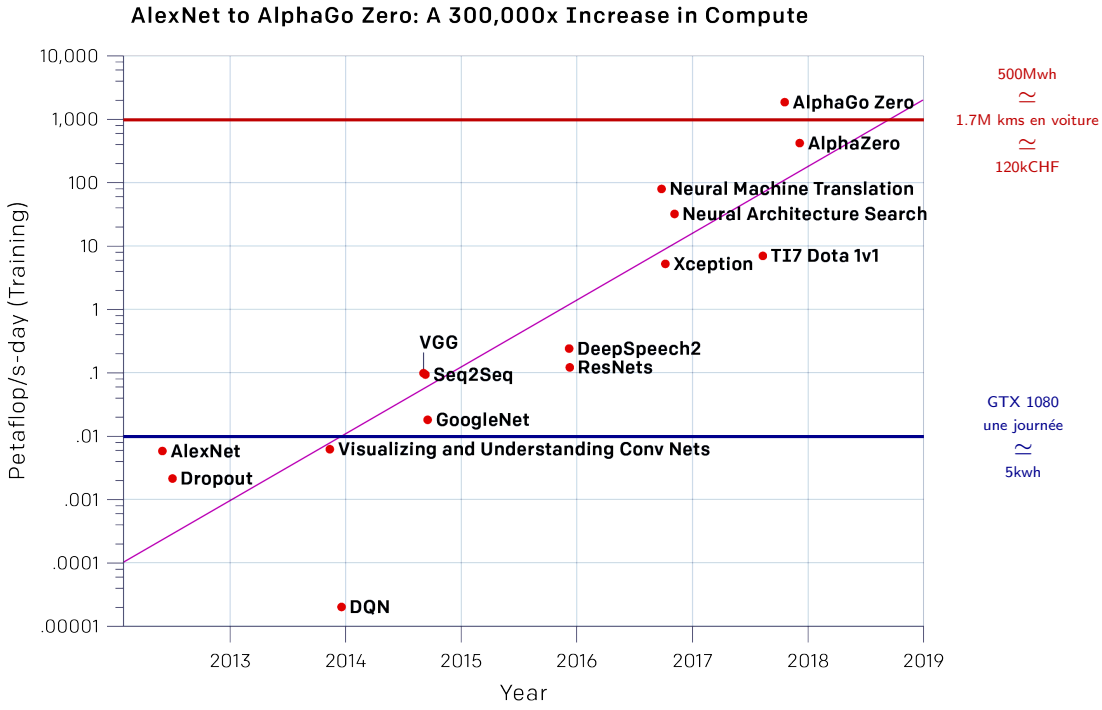
En particulier la puissance de calcul et les capacités de stockage des ordinateurs n'ont cessé de croître exponentiellement depuis près de 50 ans.



Un PC standard peut effectuer 10'000 milliards d'opérations scalaires par *seconde*, et stocker 4'000 milliards de caractères.



(Canziani et al., 2016)



(OpenAI blog, 2018)

1 petaflop/s-day $\simeq$ 100 GPUs pendant un jour, $\simeq$ 500kwh, $\simeq$ 100CHF d'électricité

Fonctions de perte et apprentissage

Étant donné un ensemble d'apprentissage

$$(x_n, y_n) \in \mathbb{R}^D \times \mathscr{C},$$

une fonction paramétrique

$$\forall w \in \mathbb{R}^Q,\ x \mapsto f(x; w),$$

et une fonction de perte

$$\mathscr{L}(w) = \frac{1}{N} \sum_{n=1}^{N} \ell(f(x_n; w), y_n) = \hat{\mathbb{E}}(\ell(f(X; w), Y)),$$

l'apprentissage consiste à estimer

$$w^* = \operatorname*{argmin}_w \mathscr{L}(w).$$

Ou plus simplement "Pour quelles valeurs des paramètres le modèle fait-il le moins d'erreurs ?"

Les deux fonctions de perte standards sont:

- La "mean square error" pour la regression:

$$\mathscr{L}(w) = \frac{1}{ND} \sum_{n=1}^{N} \sum_{d=1}^{D} \left(f_d(x_n; w) - y_{n,d}\right)^2.$$

- La "cross entropy" pour la classification:

$$\mathscr{L}(w) = \frac{1}{N} \sum_{n=1}^{N} \log \left(\frac{e^{f_{y_n}(x_n; w)}}{\sum_y e^{f_y(x_n; w)}} \right).$$

Modèles linéaires et MLP

Un perceptron combine une transformation linéaire paramétrique et une application non-linéaire

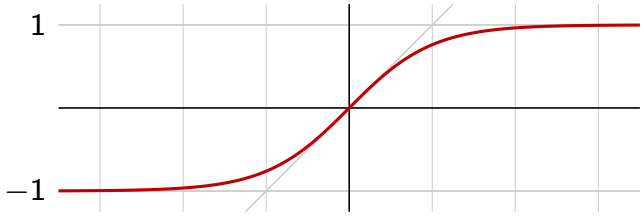
$$\begin{aligned} f_w : \mathbb{R}^D &\to \mathbb{R} \\ x &\mapsto \sigma(w^T x + b), \end{aligned}$$

avec $w \in \mathbb{R}^D$, $b \in \mathbb{R}$, and $\sigma : \mathbb{R} \to \mathbb{R}$.

On appelle σ traditionnellement la "fonction d'activation".

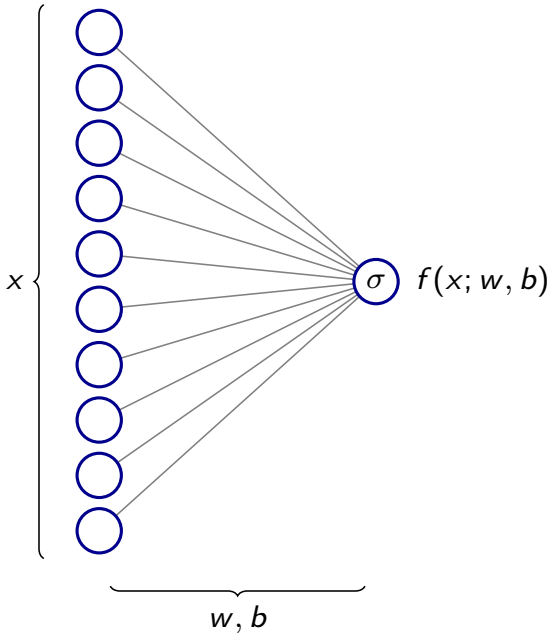
Deux fonctions d'activation classiques sont la tangente hyperbolique

$$x \mapsto \frac{2}{1+e^{-2x}} - 1$$



et la "rectified linear unit" (ReLU)

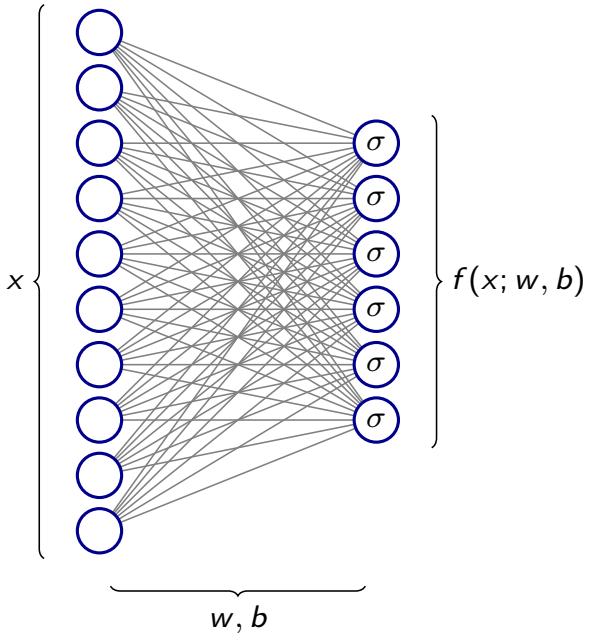
$$x \mapsto \max(0, x)$$



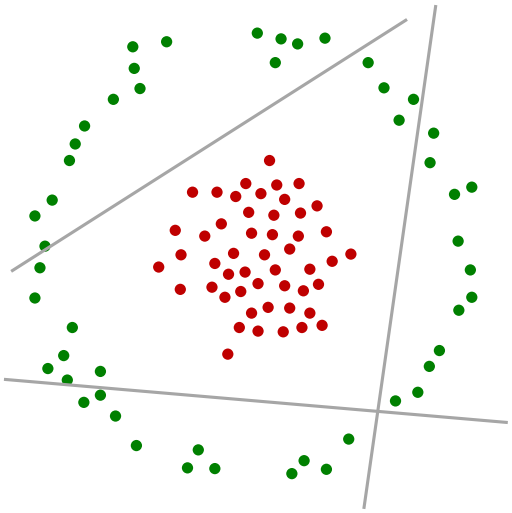
Ceci peut être naturellement étendu à plusieurs dimensions de sortie en répliquant autant que nécessaire une unité de ce type

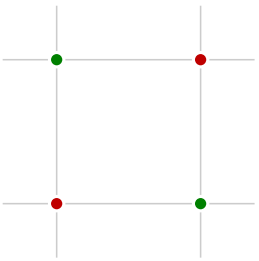
$$\begin{aligned} \mathbb{R}^D &\to \mathbb{R}^C \\ x &\mapsto \sigma(w\,x + b), \end{aligned}$$

avec $w \in \mathbb{R}^{C \times D}$, $b \in \mathbb{R}^C$, et σ est appliquée sur chaque composante.



Un prédicteur de ce type ne peut évidemment représenter qu'une classe limitée de fonctions, et par exemple ne peut classer correctement que des populations **linéairement séparables**.

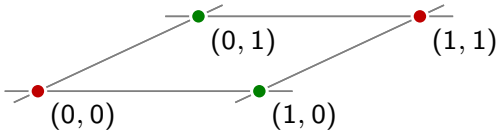


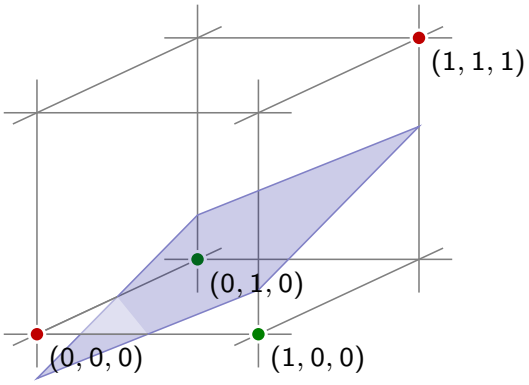


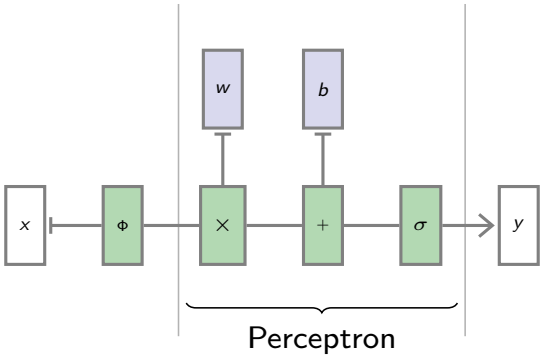
"ou exclusif"

Le "ou exclusif" peut être calculé en pré-processant l'entrée pour rendre les population linéairement séparables.

$$\Phi : (x_u, x_v) \mapsto (x_u, x_v, x_u x_v).$$







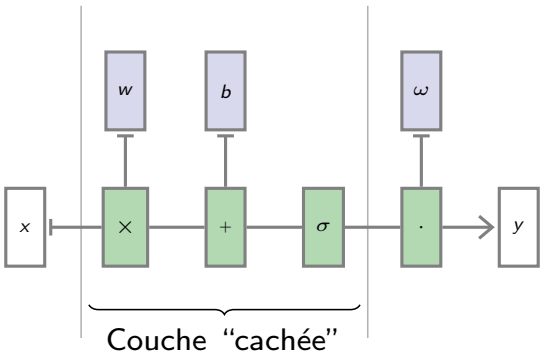
Il est possible d'approcher toute fonction continue sur un compact

$$\psi : [0,1]^D \to \mathbb{R}$$

avec un réseau à une couche cachée

$$x \mapsto \omega \cdot \sigma(w\,x + b),$$

où $b \in \mathbb{R}^K$, $w \in \mathbb{R}^{K \times D}$, et $\omega \in \mathbb{R}^K$.



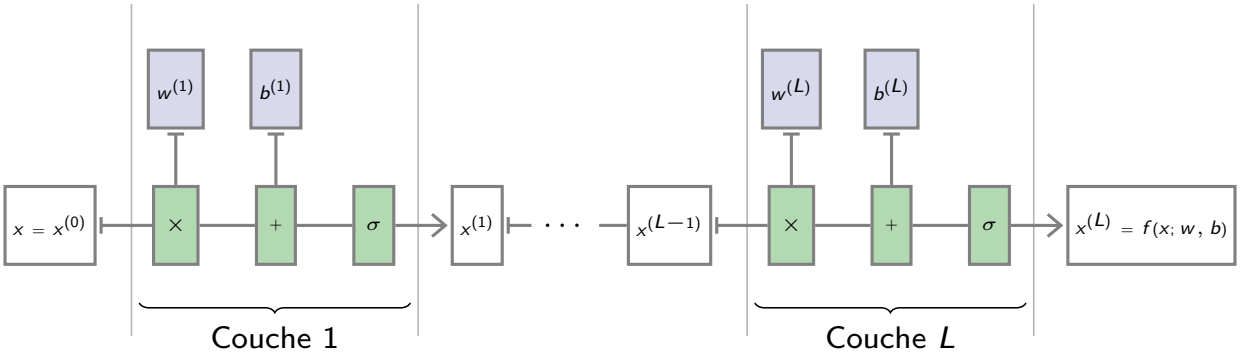
Ceci constitue le **théorème d'approximation universelle** des réseaux de neurones.

Ce résultat motive l'utilisation de fonctions composées de plusieurs couches.

Avec $x^{(0)} = x$:

$$\forall l = 1, \ldots, L, \; x^{(l)} = \sigma\left(w^{(l)} x^{(l-1)} + b^{(l)}\right)$$

et $f(x; w, b) = x^{(L)}$.



Un modèle de ce type est appelé un **Multi-Layer Perceptron (MLP).**

L'implémentation de ces modèles se fait à l'aide de librairies (PyTorch, TensorFlow, Torch7, CNTK, MXNet), la plupart dans le langage Python.

E.g

$$f : \mathbb{R}^3 \to \mathbb{R}$$
$$x \mapsto w^T x + b$$

```
>>> f = nn.Linear(3, 1)
>>> x = torch.empty(5, 3).normal_()
>>> f(x)
tensor([[-0.4264],
        [-0.0265],
        [-0.2168],
        [ 0.2507],
        [-0.3525]], grad_fn=<AddmmBackward>)
>>> for t in f.parameters(): print(t.size())
...
torch.Size([1, 3])
torch.Size([1])
```

```
>>> f = nn.Sequential(nn.Linear(3, 9), nn.ReLU(),
...                   nn.Linear(9, 2), nn.ReLU(),
...                   nn.Linear(2, 2))
>>> x = torch.empty(5, 3).normal_()
>>> f(x)
tensor([[-0.2195, -0.4222],
        [-0.2234, -0.4257],
        [-0.2217, -0.4241],
        [-0.2195, -0.4222],
        [-0.2465, -0.4459]], grad_fn=<AddmmBackward>)
>>> for t in f.parameters(): print(t.size())
...
torch.Size([9, 3])
torch.Size([9])
torch.Size([2, 9])
torch.Size([2])
torch.Size([2, 2])
torch.Size([2])
```


Couches convolutionnelles

S'ils étaient traités comme des vecteurs sans structure particulière, les signaux de grandes dimensions comme les images ou le sons demanderaient des modèles de tailles excessives.

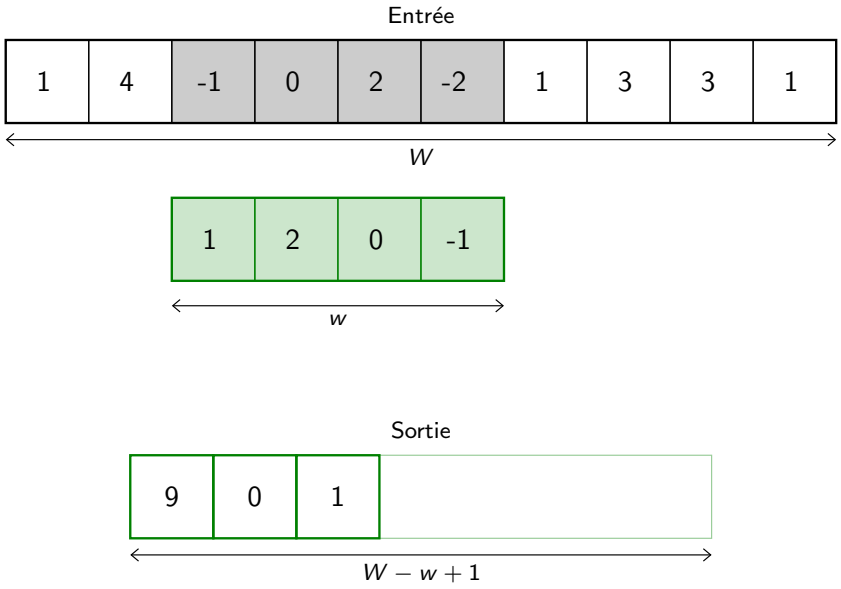
Par exemple un modèle linéaire prenant une image 256×256 couleur en entrée et générant un signal similaire demanderait

$$(256 \times 256 \times 3)^2 \simeq 3.87e{+}10$$

paramètres ($\simeq$150Gb !)

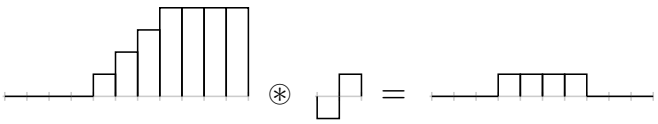
De plus un tel modèle n'exploiterait pas l'intuition que nous avons que ces signaux sont stationnaires et qu'une représentation du signal qui est adéquate à une position dans une image ou un son l'est partout ailleurs.

Une couche "convolutionnaire" implémente précisément cette idée.

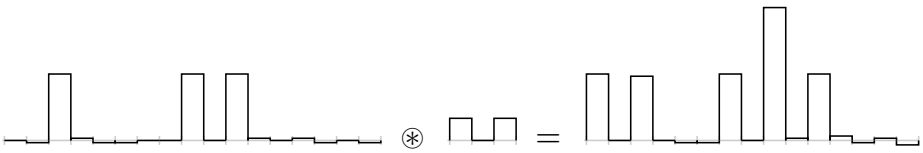


Une convolution peut implémenter un opérateur différentiel, *e.g.*

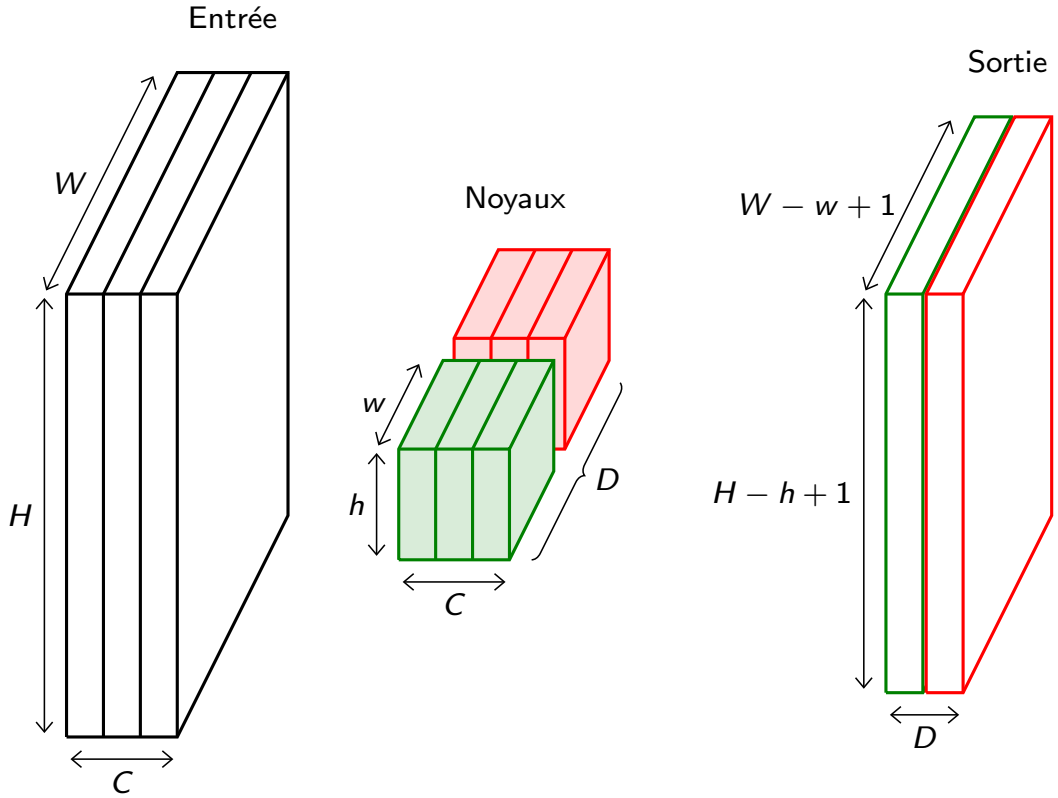
$$(0,0,0,0,1,2,3,4,4,4,4) \circledast (-1,1) = (0,0,0,1,1,1,1,0,0,0).$$

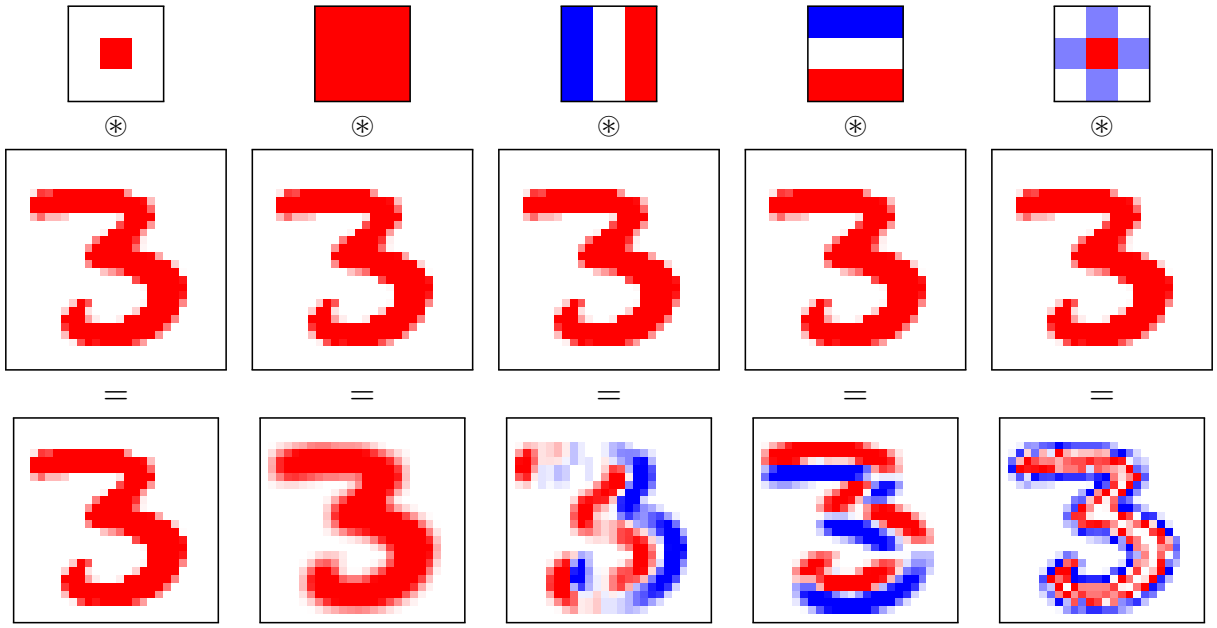


ou un simple détecteur de motif, *e.g.*



Le même concept peut être généralisé à un signal 2d multi-canaux:





Un réseau de neurones classique de traitement d'images, par exemple pour faire de la classification, combine des couches convolutionnelles, des couches qui réduisent la taille du signal ("pooling"), et des couches linéaires classiques.

```
model = nn.Sequential(
    nn.Conv2d( 1, 32, 5), nn.ReLU(), nn.MaxPool2d(3),
    nn.Conv2d(32, 64, 5), nn.ReLU(), nn.MaxPool2d(2),
    Shape1D(),
    nn.Linear(256, 200), nn.ReLU(),
    nn.Linear(200, 10)
)
```

Entraînement

Finalement, la fonction f_w correspondant à un "réseaux de neurones" s'exprime comme un graphe d'opérateurs tensoriels qui sont essentiellement de deux types:

- Paramétriques et linéaires,
- Non-paramétriques et non-linéaires.

Le vecteur w est l'ensemble de tous les paramètres de tous les opérateurs du graphe et sa dimension est de l'ordre de $10^5 - 10^9$.

Sauf dans de rares cas (*e.g.* régression linéaire), la quantité

$$\operatorname*{argmin}_{w} \mathscr{L}(w)$$

n'a pas d'expression analytique, et on doit l'estimer à l'aide de méthodes numériques, telles que la "descente de gradient".

Étant donnée une fonction

$$\begin{aligned} f : \mathbb{R}^D &\to \mathbb{R} \\ x &\mapsto f(x_1, \ldots, x_D), \end{aligned}$$

son gradient est l'application

$$\begin{aligned} \nabla f : \mathbb{R}^D &\to \mathbb{R}^D \\ x &\mapsto \left(\frac{\partial f}{\partial x_1}(x), \ldots, \frac{\partial f}{\partial x_D}(x) \right). \end{aligned}$$

Pour minimiser une fonction

$$\mathscr{L} : \mathbb{R}^D \to \mathbb{R}$$

la descente de gradient utilise itérativement une approximation linéaire locale pour améliorer la meilleure solution obtenue jusque là.

Considérons l'approximation suivante de $\mathscr{L}$ dans le voisinage de $w_0 \in \mathbb{R}^D$

$$\tilde{\mathscr{L}}_{w_0}(w) = \mathscr{L}(w_0) + \nabla\mathscr{L}(w_0)^T(w - w_0) + \frac{1}{2\eta}\|w - w_0\|^2.$$

Notons en particulier que le terme quadratique ne dépend pas de $\mathscr{L}$.

Nous avons

$$\nabla\tilde{\mathscr{L}}_{w_0}(w) = \nabla\mathscr{L}(w_0) + \frac{1}{\eta}(w - w_0),$$

qui entraîne

$$\operatorname*{argmin}_w \tilde{\mathscr{L}}_{w_0}(w) = w_0 - \eta\nabla\mathscr{L}(w_0).$$

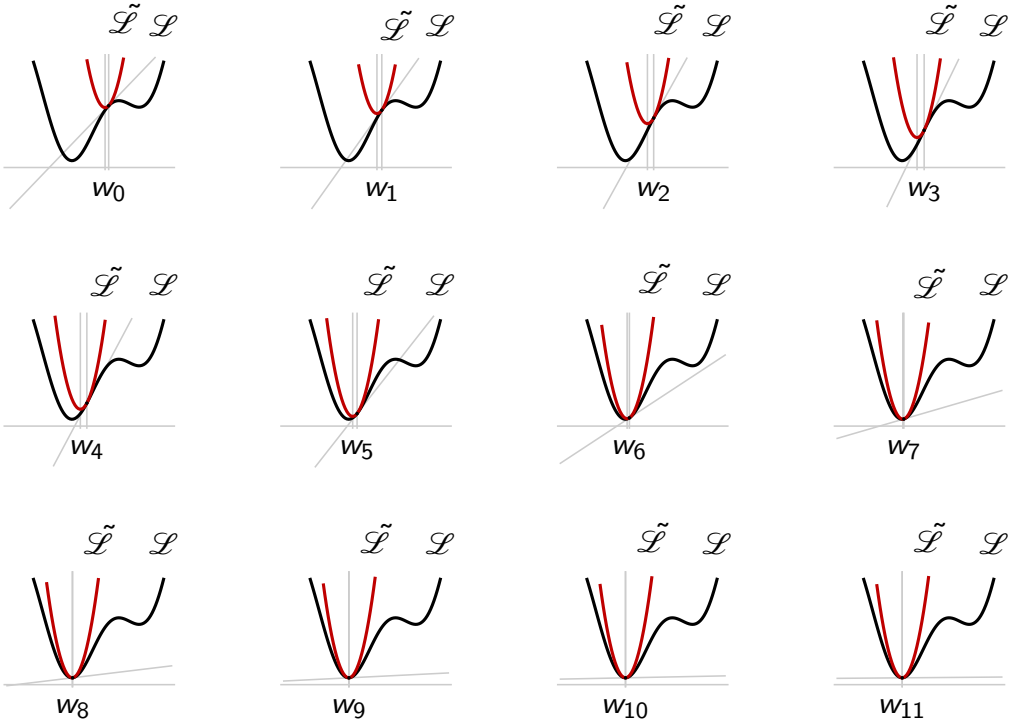
La règle itérative qui consiste à aller au minimum de cette approximation prend donc la forme:

$$w_{t+1} = w_t - \eta\nabla\mathscr{L}(w_t),$$

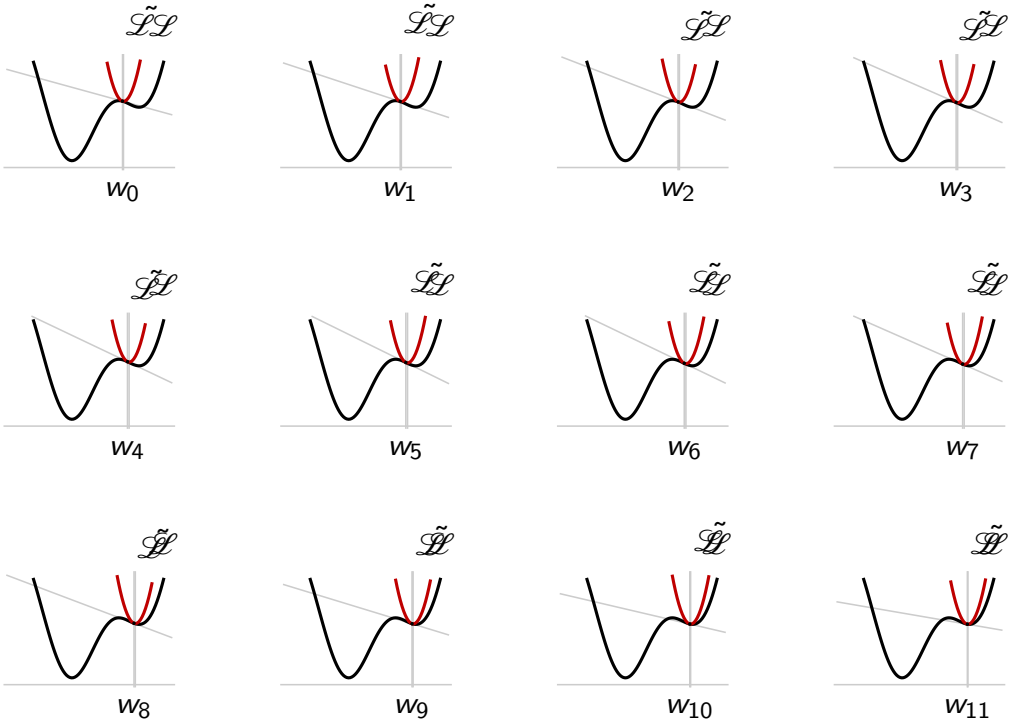
ce qui correspond intuitivement à "suivre la pente maximale".

Cette stratégie mène [la plupart du temps] à un minimum local, qui dépend fortement de w_0 et η.

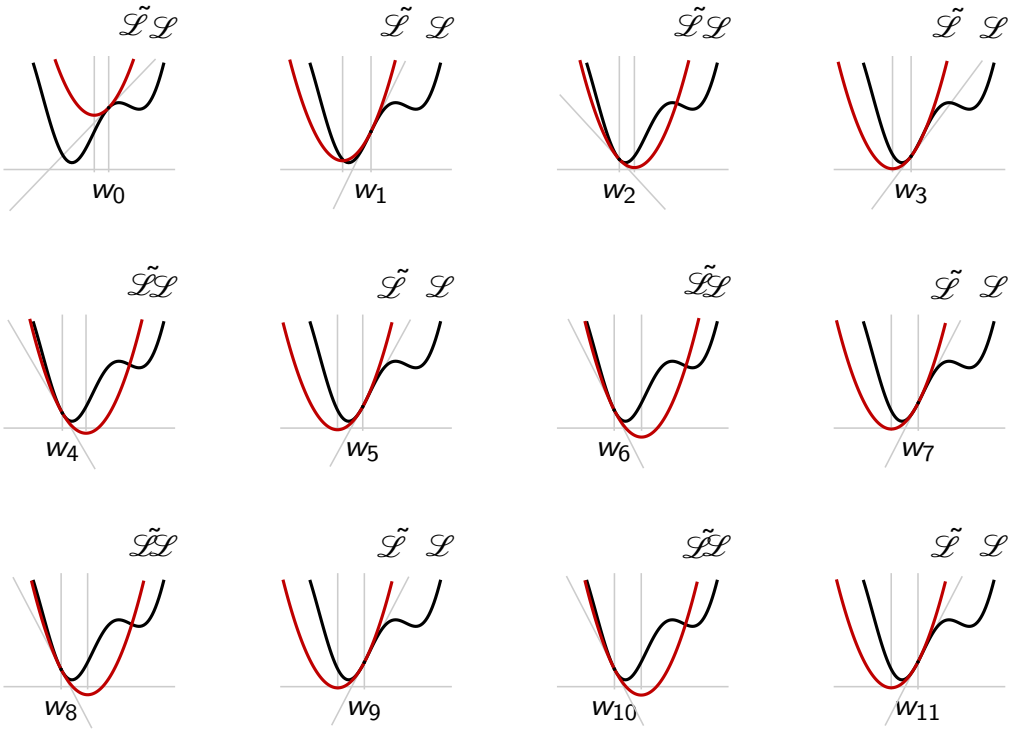
$\eta = 0.125$

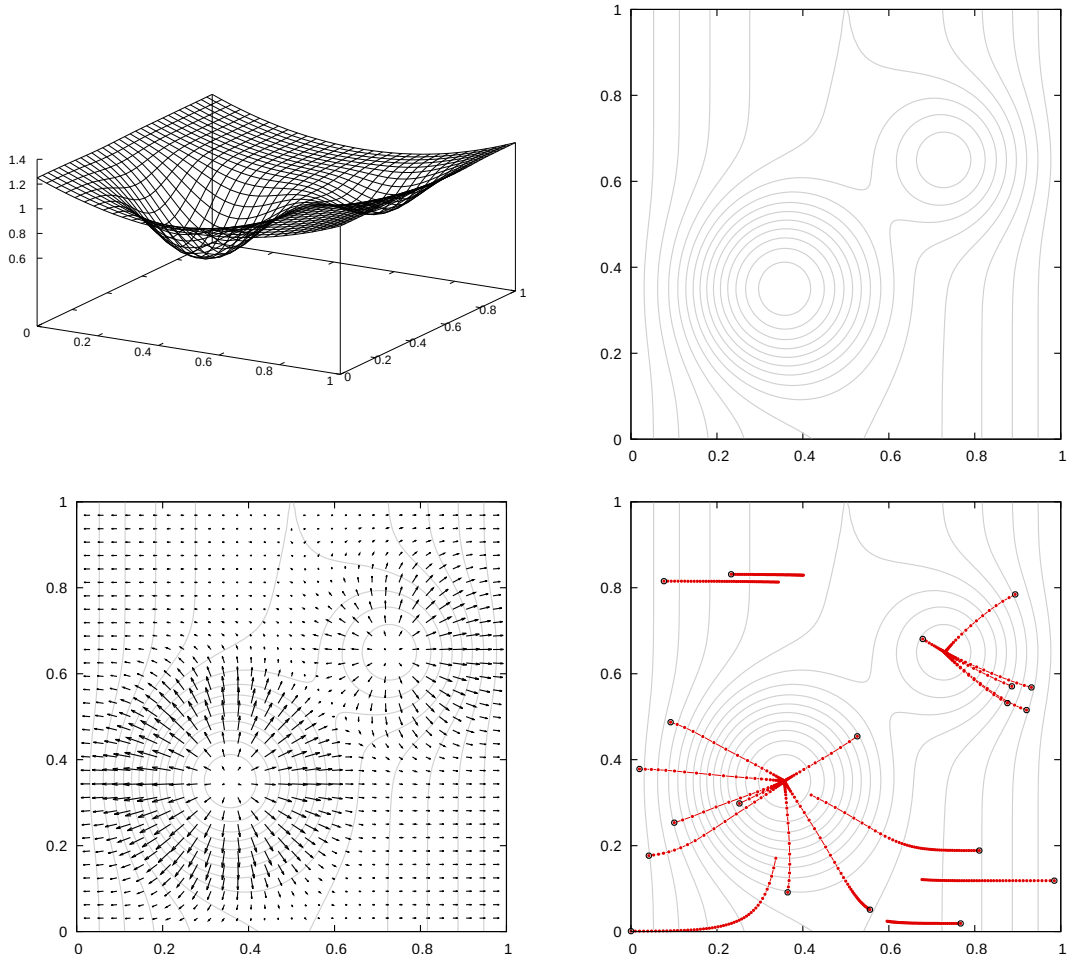


$\eta = 0.125$



$\eta = 0.5$


$\mathscr{L}\tilde{\mathscr{L}}$
w_0
$\mathscr{L}\tilde{\mathscr{L}}$
w_1
$\mathscr{L}\tilde{\mathscr{L}}$
w_2
$\mathscr{L}\tilde{\mathscr{L}}$
w_3
$\mathscr{L}\tilde{\mathscr{L}}$
w_4
$\mathscr{L}\tilde{\mathscr{L}}$
w_5
$\mathscr{L}\tilde{\mathscr{L}}$
w_6
$\mathscr{L}\tilde{\mathscr{L}}$
w_7
$\mathscr{L}\tilde{\mathscr{L}}$
w_8
$\mathscr{L}\tilde{\mathscr{L}}$
w_9
$\mathscr{L}\tilde{\mathscr{L}}$
w_{10}
$\mathscr{L}\tilde{\mathscr{L}}$
w_{11}



Différentiation automatique et optimisation

La descente de gradient peut donc être utilisée en générale, quelle que soit la fonction f et la fonction de perte $\mathscr{L}$, **à condition de pouvoir calculer les dérivées de cette dernière par rapport aux paramètres w.**

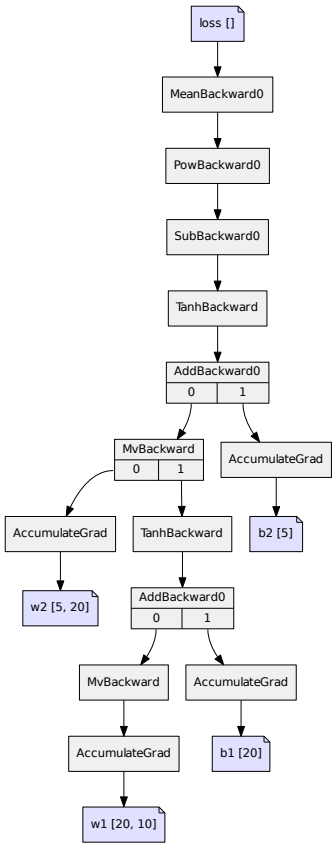
Les librairies de deep learning fournissent des mécanismes de différentiation automatique et des implémentations de la descente de gradient.

PyTorch enregistre dynamiquement les séquences de calculs tensoriels et permet de calculer n'importe quel gradient.

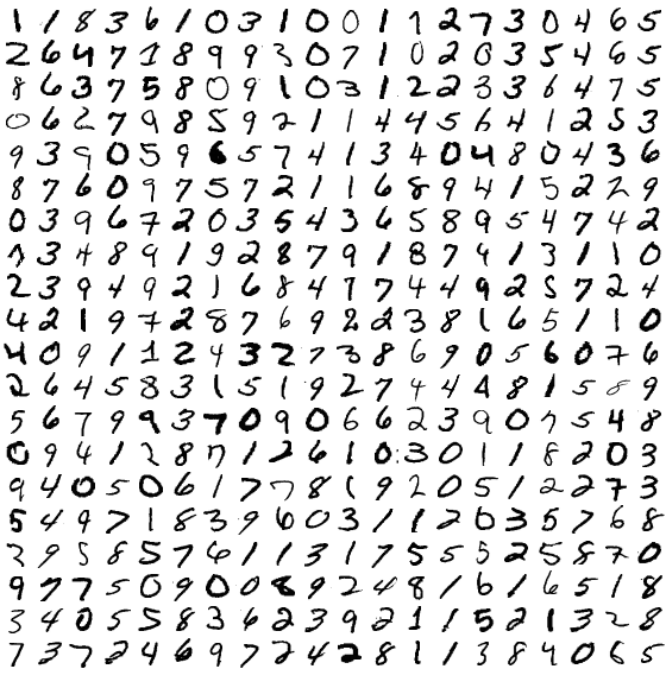
```
>>> t = torch.tensor([1., 2., 4.]).requires_grad_()
>>> a = t.pow(2).sum()
>>> torch.autograd.grad(a, (t,))
(tensor([2., 4., 8.]),)
```



```
w1 = torch.rand(20, 10).requires_grad_()
b1 = torch.rand(20).requires_grad_()
w2 = torch.rand(5, 20).requires_grad_()
b2 = torch.rand(5).requires_grad_()

x = torch.rand(10)
h = torch.tanh(w1 @ x + b1)
y = torch.tanh(w2 @ h + b2)

target = torch.rand(5)

loss = (y - target).pow(2).mean()
```



MNIST



(leCun et al., 1998)


```
model = nn.Sequential(
    nn.Conv2d(1, 32, 5),
    nn.ReLU(),
    nn.MaxPool2d(3),
    nn.Conv2d(32, 64, 5),
    nn.ReLU(),
    nn.MaxPool2d(2),
    Shape1D(),
    nn.Linear(256, 256),
    nn.ReLU(),
    nn.Linear(256, 10)
)

criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.SGD(model.parameters(), lr = 1e-2)

for e in range(10):
    for input, target in data_loader_iterator(train_loader):
        output = model(input)
        loss = criterion(output, target)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Temps d'apprentissage $<$10s, erreur $\simeq$1%

Les librairies de “Deep Learning” tells que PyTorch ou TensorFlow simplifient énormément l'utilisation de ces techniques.

```
import PIL, torch, torchvision

img = torchvision.transforms.ToTensor()(PIL.Image.open('blacklab.jpg'))
img = img.view(1, img.size(0), img.size(1), img.size(2))
img = 0.5 + 0.5 * (img - img.mean()) / img.std()

alexnet = torchvision.models.alexnet(pretrained = True)
alexnet.eval()
output = alexnet(Variable(img))
scores, indexes = output.data.view(-1).sort(descending = True)

class_names = eval(open('imagenet1000_clsid_to_human.txt', 'r').read())
for k in range(15):
    print('#{:d} ({:.02f}) {:s}'.format(k, scores[k], class_names[indexes[k]]))
```


```
#1 (12.26) Weimaraner
#2 (10.95) Chesapeake Bay retriever
#3 (10.87) Labrador retriever
#4 (10.10) Staffordshire bullterrier, Staffordshire bull terrier
#5 (9.55) flat-coated retriever
#6 (9.40) Italian greyhound
#7 (9.31) American Staffordshire terrier, Staffordshire terrier
#8 (9.12) Great Dane
#9 (8.94) German short-haired pointer
#10 (8.53) Doberman, Doberman pinscher
#11 (8.35) Rottweiler
#12 (8.25) kelpie
```

Weimaraner

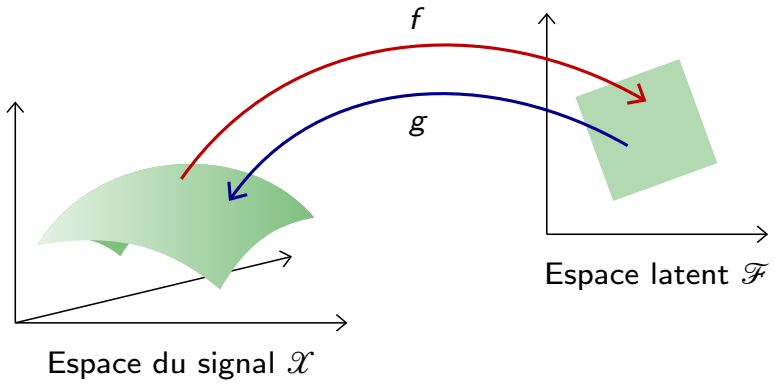
Chesapeake Bay retriever

Auto-encodeurs

Certaines applications comme la synthèse de signaux ou le débruitage, nécessitent de disposer d'une représentation en petite dimension d'un signal.

La construction d'une telle représentation repose en général sur un auto-encodeur, composé de deux modèles optimisés ensemble.

Un auto-encodeur (Bourlard and Kamp, 1988; Hinton and Zemel, 1994) combine un **encodeur** f depuis l'espace du signal $\mathcal{X}$ vers un espace **latent** $\mathcal{F}$, et un **décodeur** g qui projette réciproquement vers $\mathcal{X}$, de manière à ce que $g \circ f$ soit [proche de] l'identité sur les données.



Un bon auto-encodeur capture la "structure" du signal, et en particulier les dépendances statistiques entre ses différentes composantes.

Soit q la distribution des données sur $\mathcal{X}$. La qualité d'un auto-encodeur peut être estimée à l'aide de l'erreur quadratique

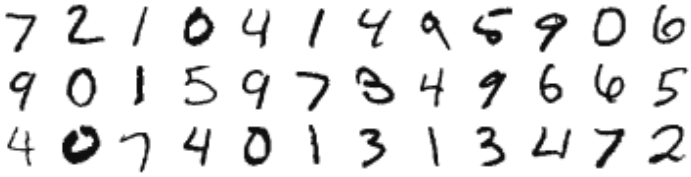
$$\mathbb{E}_{X \sim q}\left[\|X - g \circ f(X)\|^2\right] \simeq 0.$$

Étant données deux fonctions paramétriques $f(\cdot\,; w)$ et $g(\cdot\,; w)$, l'entraînement consiste à minimiser une estimation empirique de cette quantité

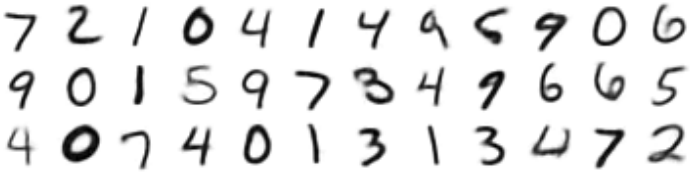
$$\hat{w}_f, \hat{w}_g = \operatorname*{argmin}_{w_f, w_g} \frac{1}{N} \sum_{n=1}^{N} \|x_n - g(f(x_n; w_f); w_g)\|^2 .$$

Un exemple simple d'auto-encodeur serait composé de deux modèles linéaires, auquel cas la solution est celle du PCA. De bien meilleurs résultats peuvent être obtenus avec des fonctions plus complexes comme des réseaux de neurones.

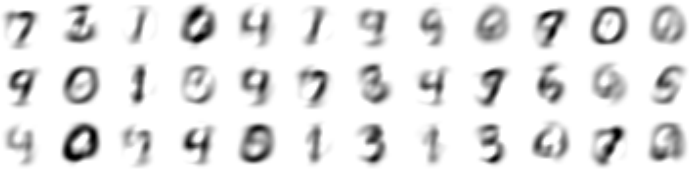
X (examples originaux)



$g \circ f(X)$ (CNN, $d = 8$)

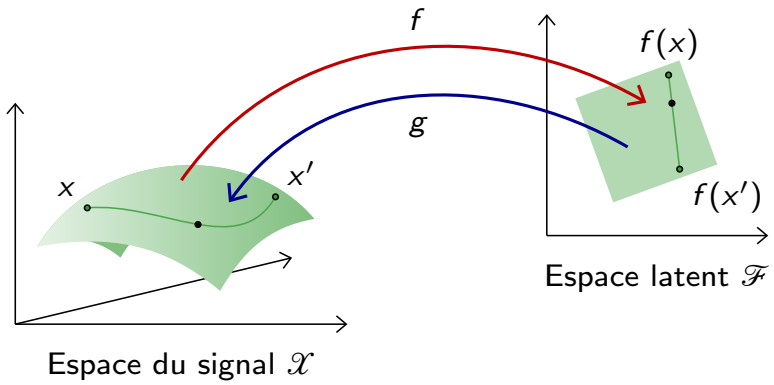


$g \circ f(X)$ (PCA, $d = 8$)



Pour avoir une intuition de la représentation latente, nous pouvons choisir deux exemples x et x' au hasard et interpoler linéairement dans l'espace latent.

$$\forall x, x' \in \mathcal{X}^2, \ \alpha \in [0,1], \quad \xi(x, x', \alpha) = g((1-\alpha)f(x) + \alpha f(x')).$$



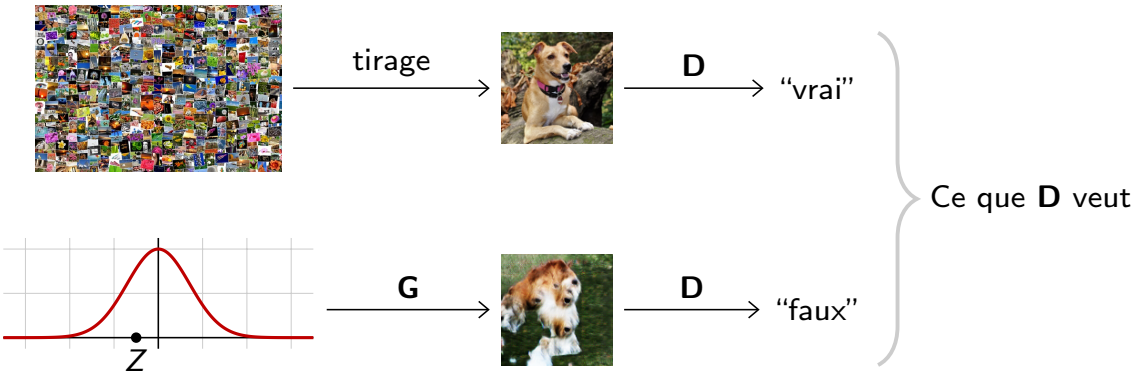
Interpolation avec PCA ($d = 32$)

Interpolation avec un auto-encodeur ($d = 8$)

Modèles génératifs adverseriaux

Une approche puissante pour apprendre un modèle génératif en grande dimension consiste à entraîner deux modèles avec des objectifs adverseriaux (Goodfellow et al., 2014).

- Un **discriminateur D** pour prédire si un exemple est "vrai" ou "faux",
- un **générateur G** pour transformer des échantillons d'une distribution fixe en exemples qui trompent **D**.



Cette stratégie est **adverseriale** car les modèles ont des objectifs antagonistes.

Soit $\mathcal{X}$ l'espace du signal et D la dimension de l'espace latent.

- Le **générateur**

$$\mathbf{G} : \mathbb{R}^D \to \mathcal{X}$$

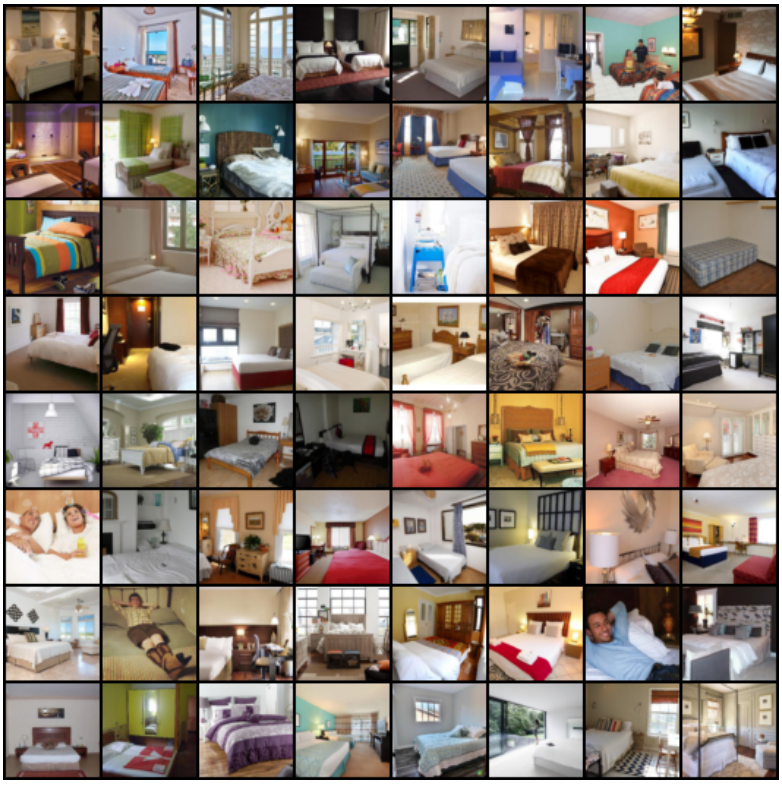
est optimisé de manière à ce que [idéalement] s'il a une entrée Z qui suit une loi normale, il génère un exemple qui suit la distribution des données en sortie.

- Le **discriminateur**

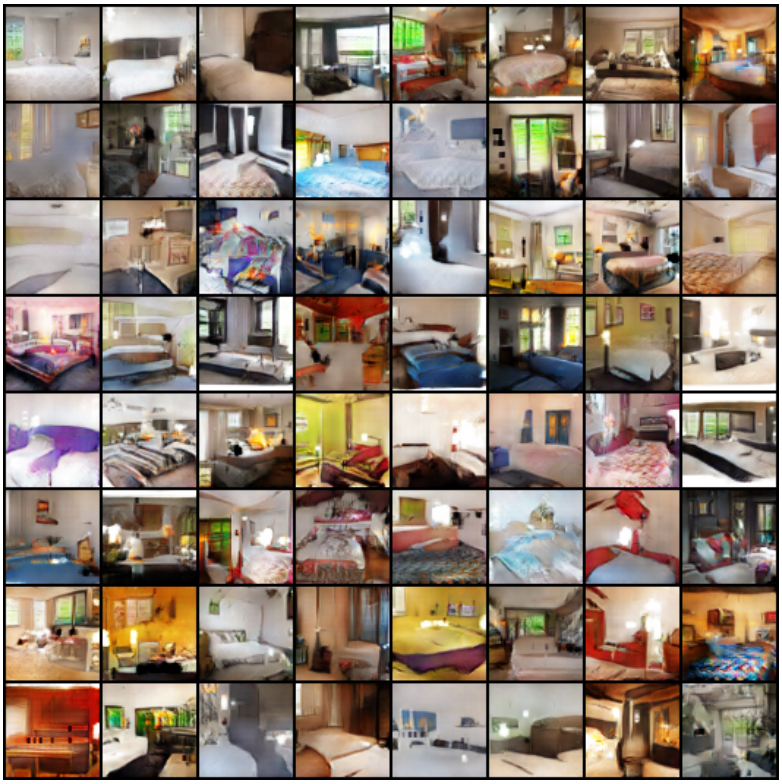
$$\mathbf{D} : \mathcal{X} \to [0, 1]$$

est optimisé pour classifier correctement si un exemple provient de la base d'apprentissage ou a été généré par **G**.

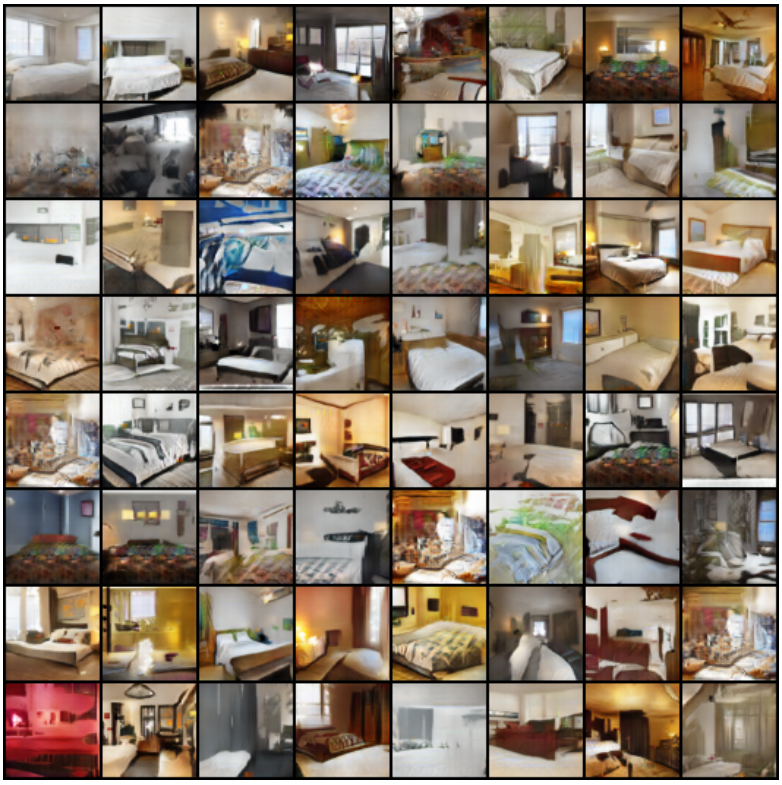
On peut montrer que sous certaines conditions l'optimisation jointe minimise $\mathbb{D}_{\text{JS}}(\mu, \mu_{\mathbf{G}})$ où μ est la "vraie" distribution et $\mu_{\mathbf{G}}$ celle des exemples synthétisés.



Images de la classe "chambres à coucher" de la base LSUN.



Images synthétiées après 1 epoch (3M images)



Images synthétiées après 20 epochs

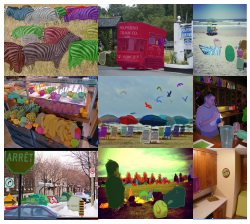
(Brock et al., 2018)

(Brock et al., 2018)

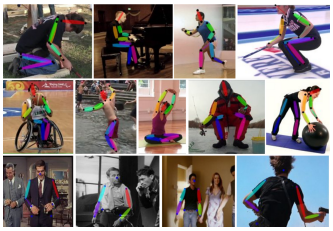
(Karras et al., 2018)

Applications

Les applications vont du traitement d'images à celui de la langue.

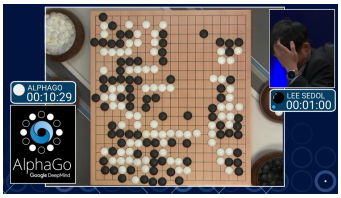


Détection d'objets



Estimation de poses

Planification de l'action



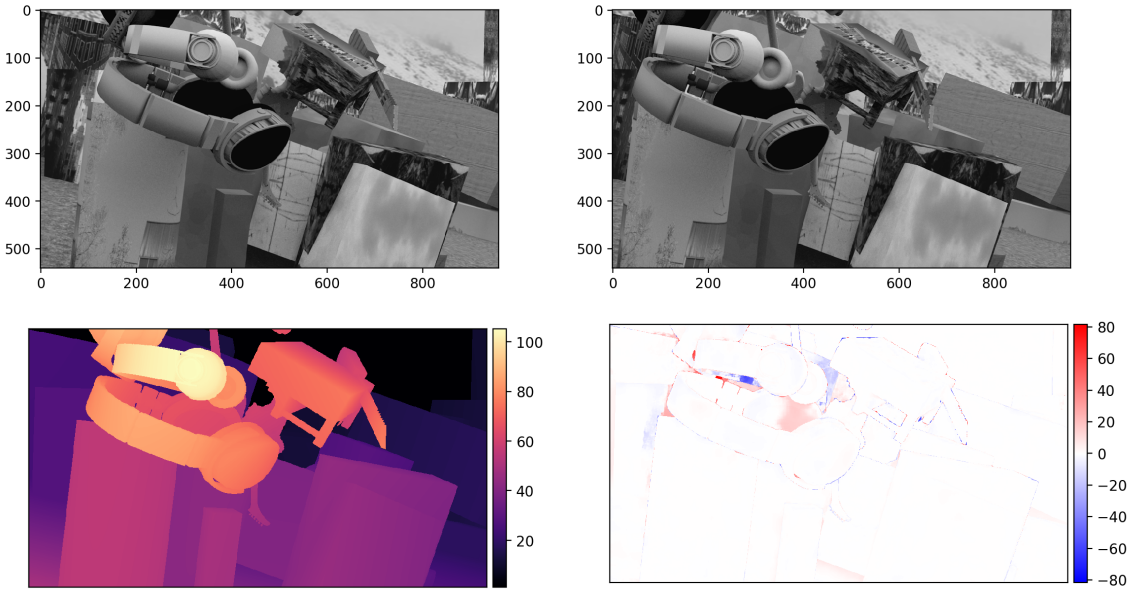
Stratégie

Description d'images

I: Jane went to the hallway.
I: Mary walked to the bathroom.
I: Sandra went to the garden.
I: Daniel went back to the garden.
I: Sandra took the milk there.
Q: Where is the milk?
A: garden

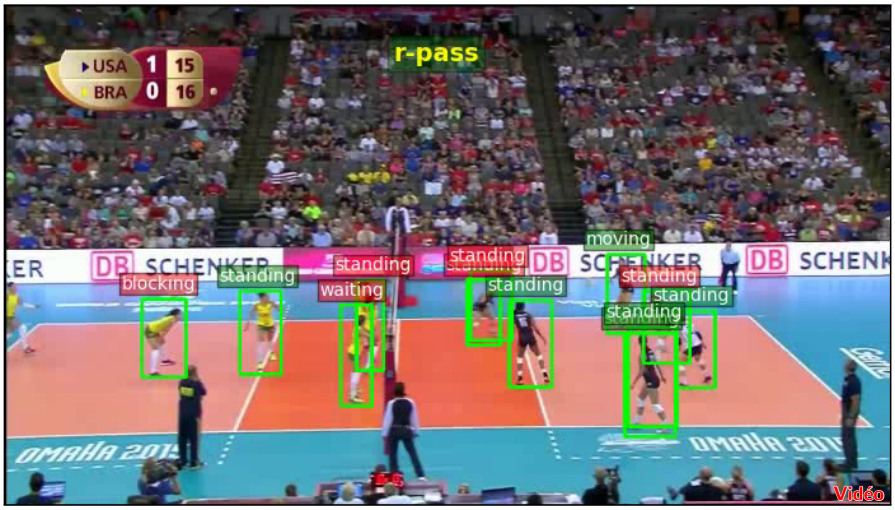
Compréhension de textes

Estimation de profondeur à partir d'une paire d'images.



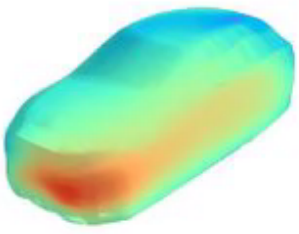
(S. Tulyakov)

Classification d'actions.

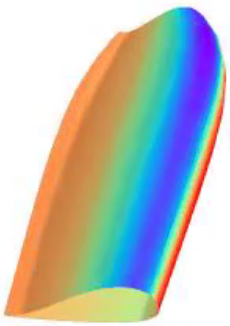


(T. Bagautdinov)

Optimisation de formes aérodynamiques.



Vidéo



Vidéo

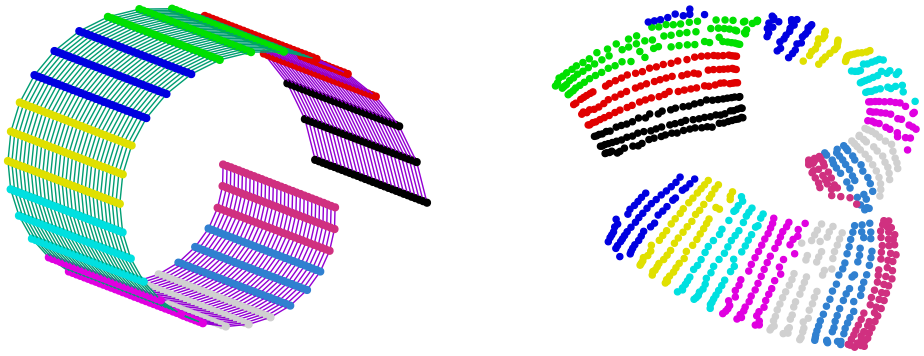
(Neural Concept, P. Baqué)

Compréhension des modèles

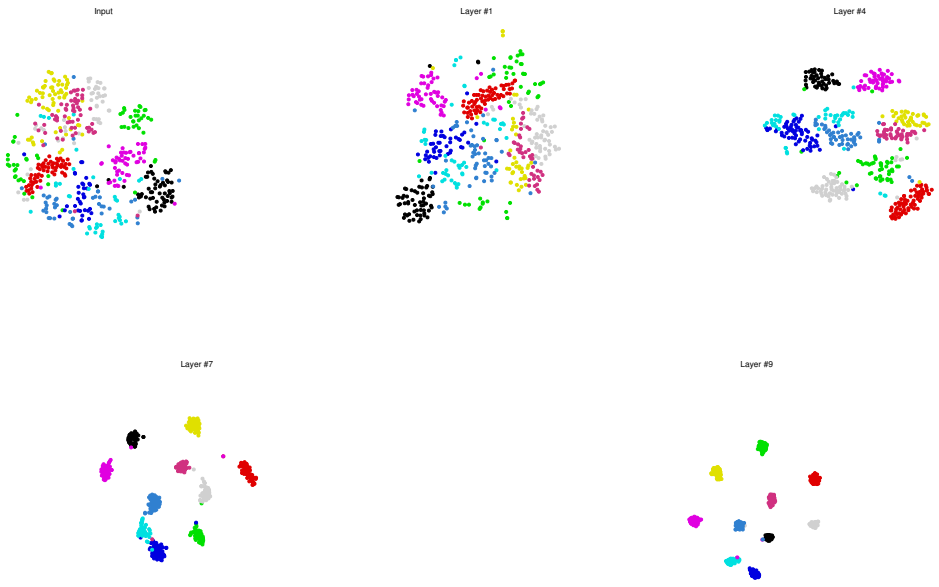
Les classes de fonctions considérées ont des propriétés d'approximateurs universels, et sont en pratiquement extrêmement complexes. Le "fonctionnement" du f appris n'est que partiellement compris.

Nombres de techniques ont étés développées pour analyser les grandeurs intermédiaires qui sont calculées dans un modèle.

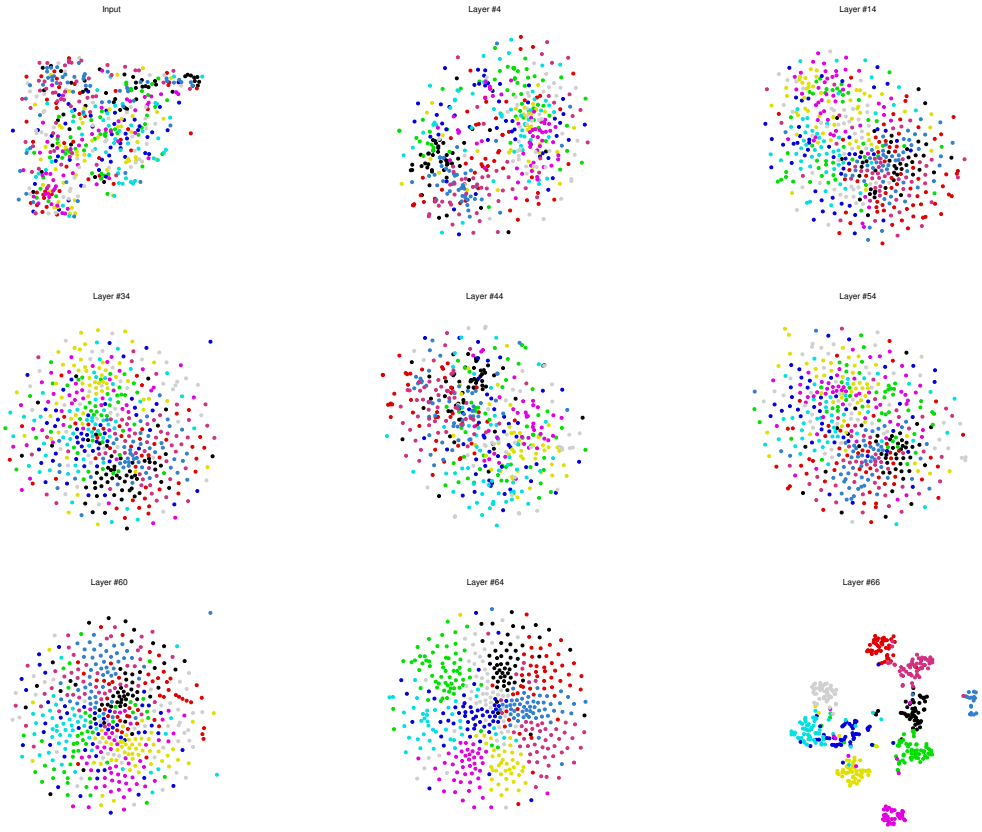
L'algorithme t-SNE permet de visualiser en 2d des ensembles de points appartenant à des espaces de très grandes dimensions.



Nous pouvons l'utiliser pour visualiser les encodages réalisés par les couches successives d'un réseau de neurones.



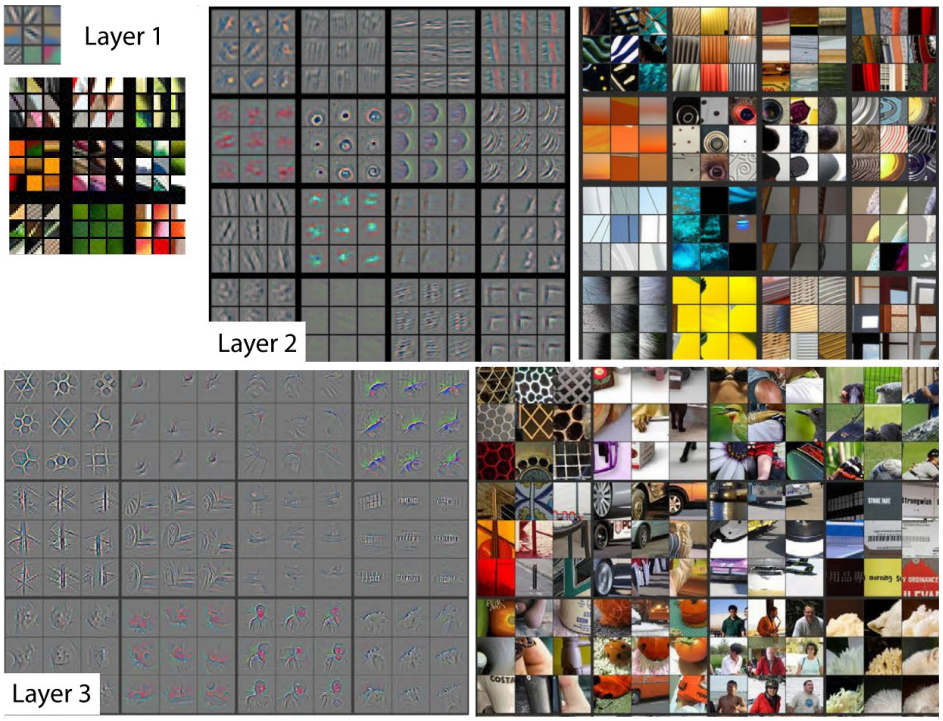
Encodages de MNIST dans les couches d'un LeNet



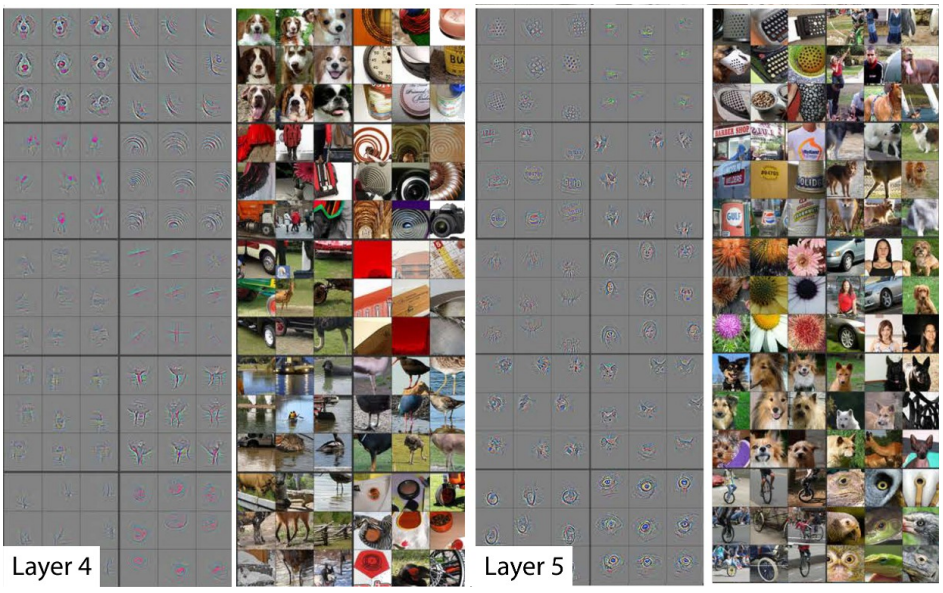
Encodages de CIFAR10 dans les couches d'un ResNet

Images originales

VGG16, rétro-propagation guidée

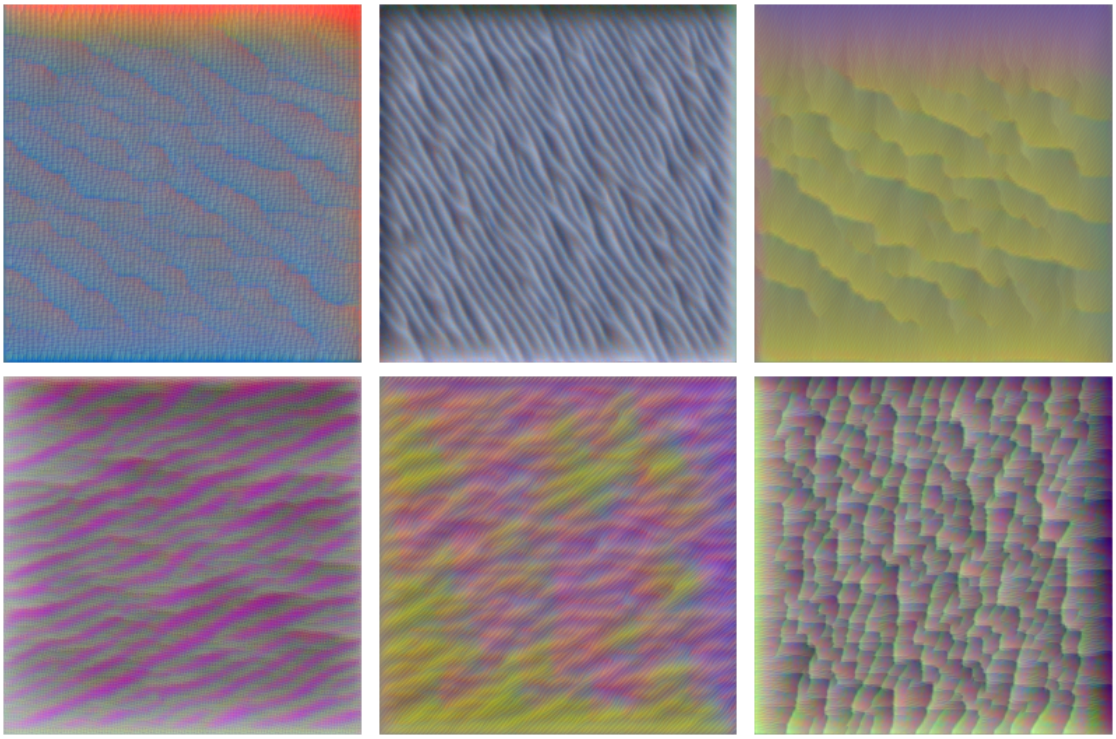


(Zeiler and Fergus, 2014)

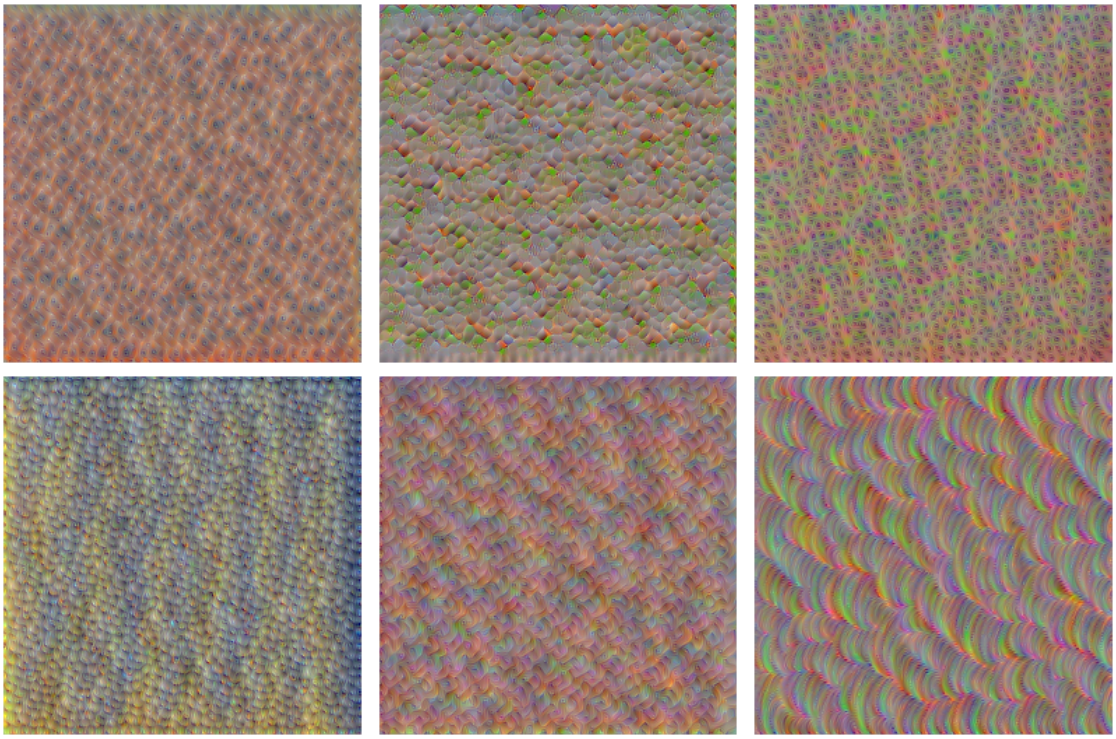


(Zeiler and Fergus, 2014)

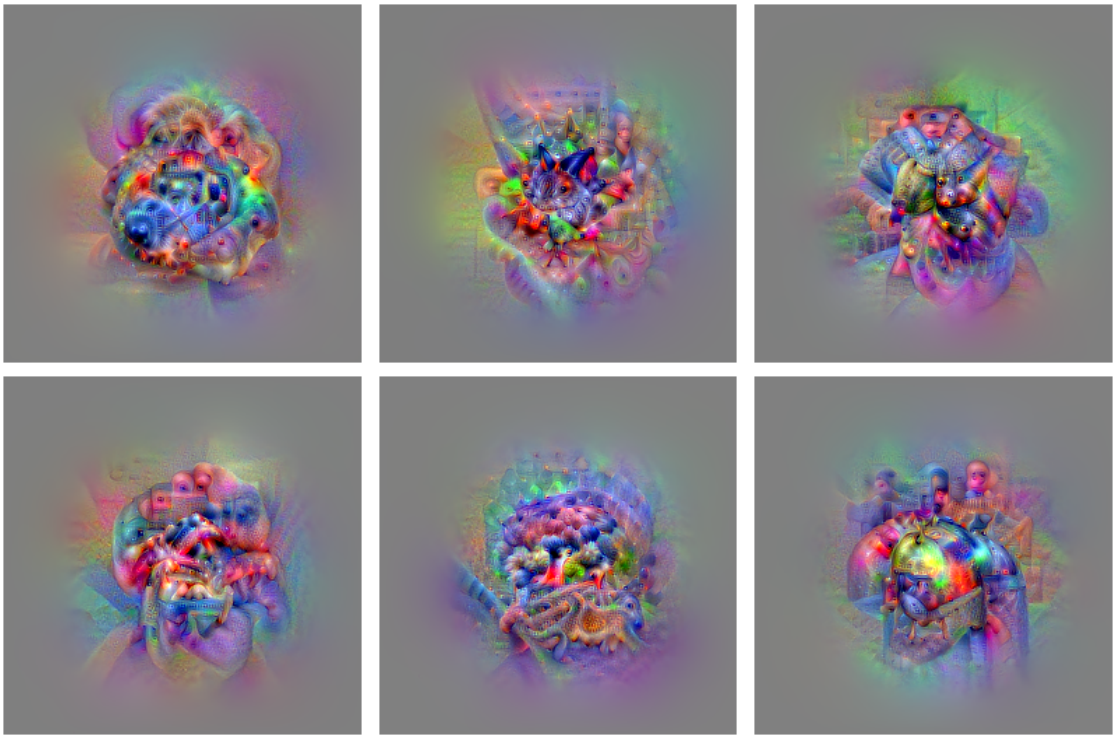
VGG16, maximisation d'un canal de la 4ème couche convolutionnelle



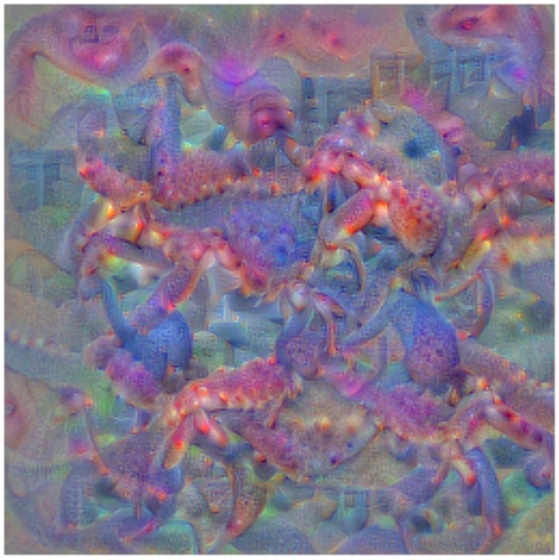
VGG16, maximisation d'un canal de la 7ème couche convolutionnelle



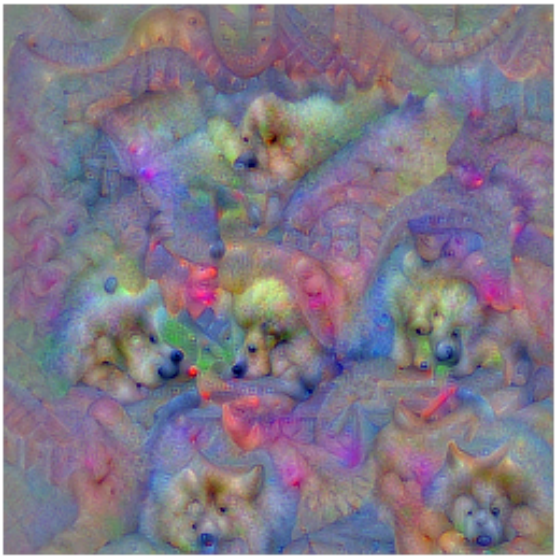
VGG16, maximisation d'une unité de la 13ème couche



VGG16, maximisation d'une unité de la couche de sortie

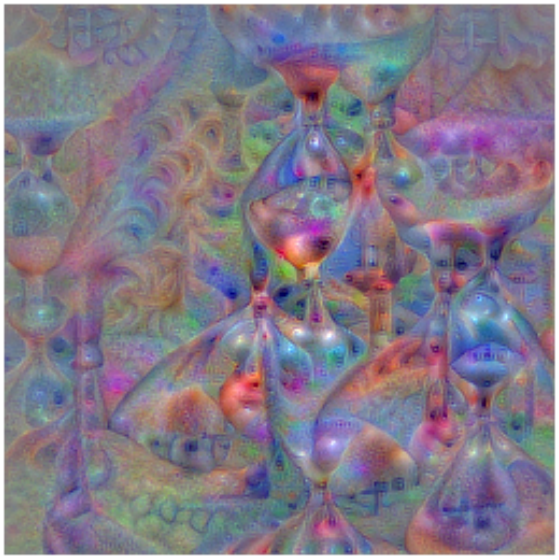


"King crab"

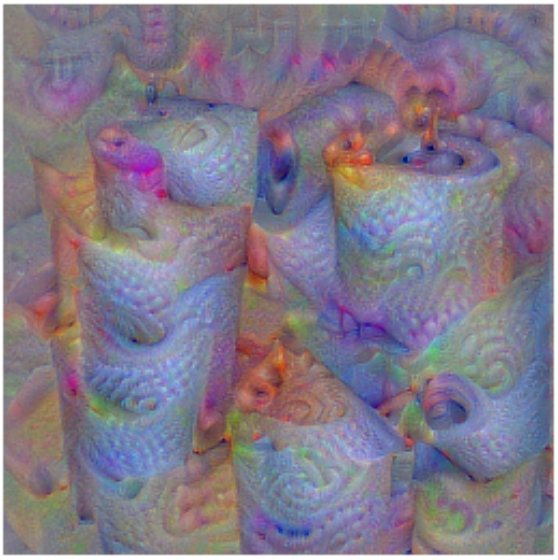


"Samoyed"

VGG16, maximisation d'une unité de la couche de sortie

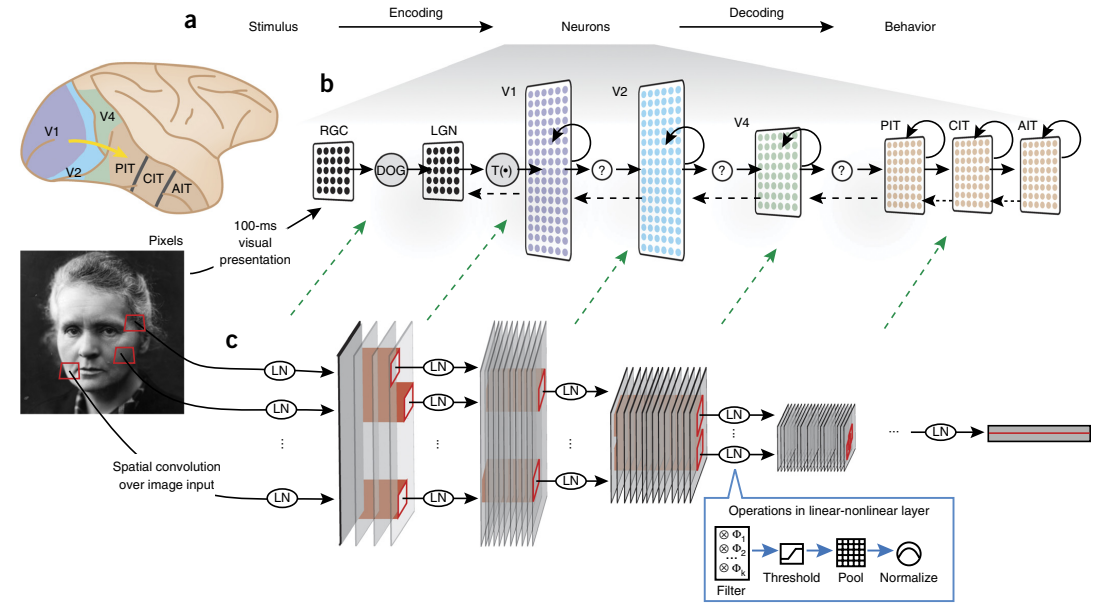


"Hourglass"



"Paper towel"

Relation avec la neurobiologie.



(Yamins and DiCarlo, 2016)

Promesses et risques

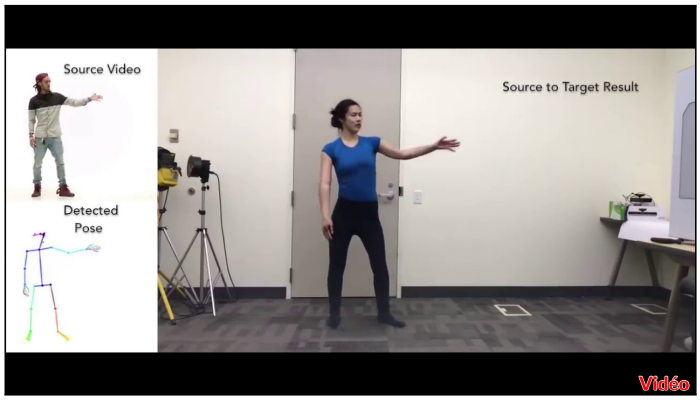
Performances surhumaines:

- Reconnaissance d'images.
- Lecture de panneaux de signalisation.
- Reconnaissance de visages.
- Go et échecs ("from first principles"), poker.
- Jeux vidéos des années 80.

Performances au niveau des humains:

- Détection de cancers de la peau.
- Compréhension de la parole.

Il est extrêmement difficile de comprendre les modèles actuels, et de prévoir les développements et les progrès possibles de l'IA à 10-20 ans.



`https://talktotransformer.com/`

The meeting about deep fakes is an important event since it will give consumers and journalists the facts before these companies rush to exploit them. With the meeting, we hope that we will reach a good balance between protecting the legitimate business and being fair to companies that do not follow traditional guidelines. This is a very important issue and is now going to be discussed at many more international conferences. And I can only hope that the government will take the initiative to address it urgently so that consumers get a right to know the true nature of their product."

"Obama: If properly harnessed, it can generate enormous prosperity and opportunity. **But it also has some downsides that we're gonna have to figure out in terms of not eliminating jobs. It could increase inequality.** It could suppress wages."

(WIRED, Aug. 24, 2016)

"Deep-learning systems for breast and heart imaging have already been developed commercially. 'It's just completely obvious that in five years deep learning is going to do better than radiologists,' he went on. 'It might be ten years. I said this at a hospital. It did not go down too well.'

Hinton's actual words, in that hospital talk, were blunt: **'They should stop training radiologists now.'** "

(The New-Yorker, Apr. 3, 2017)

"Putin, speaking Friday at a meeting with students, said the development of AI raises 'colossal opportunities and threats that are difficult to predict now.'

He warned that **'the one who becomes the leader in this sphere will be the ruler of the world.'** /.../

The Russian leader predicted that **future wars will be fought by drones,** and 'when one party's drones are destroyed by drones of another, it will have no other choice but to surrender.' "

(Associated Press, Sep. 1, 2017)

"There was no faltering psychologically and [AlphaGo] focused relentlessly. Really, I don't know, even if I were to play again, I wonder if I will be able to win. Rather than the technique, in **its psychological aspects no human can match it.** I cannot accept a technical superiority, but in that area, I believe it will be difficult to beat for humans." '

(Lee Sedol answer to NHK question, Mar. 15, 2016)

References

H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. Biological Cybernetics, 59(4):291–294, 1988.

A. Brock, J. Donahue, and K. Simonyan. Large scale GAN training for high fidelity natural image synthesis. CoRR, abs/1809.11096, 2018.

A. Canziani, A. Paszke, and E. Culurciello. An analysis of deep neural network models for practical applications. CoRR, abs/1605.07678, 2016.

D. Gershgorn. The data that transformed ai research—and possibly the world, July 2017.

I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. CoRR, abs/1406.2661, 2014.

G. E. Hinton and R. S. Zemel. Autoencoders, minimum description length and helmholtz free energy. In Neural Information Processing Systems (NIPS), pages 3–10, 1994.

T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks. CoRR, abs/1812.04948, 2018.

A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In Neural Information Processing Systems (NIPS), 2012.

Y. leCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In Conference on Computer Vision and Pattern Recognition (CVPR), 2015.

D. L. K. Yamins and J. J. DiCarlo. Using goal-driven deep learning models to understand sensory cortex. Nature neuroscience, 19:356–65, Feb 2016.

M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In European Conference on Computer Vision (ECCV), 2014.