Dr Aude Sportisse
Chercheuse dans l'équipe Maasai, Inria Sophia Antipolis
Enseignante à EFELIA

aude.sportisse@inria.fr

Introduction à l'Intelligence Artificielle

Appliquée à la biologie

Septembre-Décembre 2023

Programme du jour

Du cours de la semaine dernière:

1. Danger de sur- et sous- entrainement
2. TP sur les modèles linéaires

Nouveau (sur ces slides):

1. Réseaux de neurones
2. Réduction de dimension (rapidement)
3. TP: réseaux de neurones

1. Réseaux de neurones

Un neurone

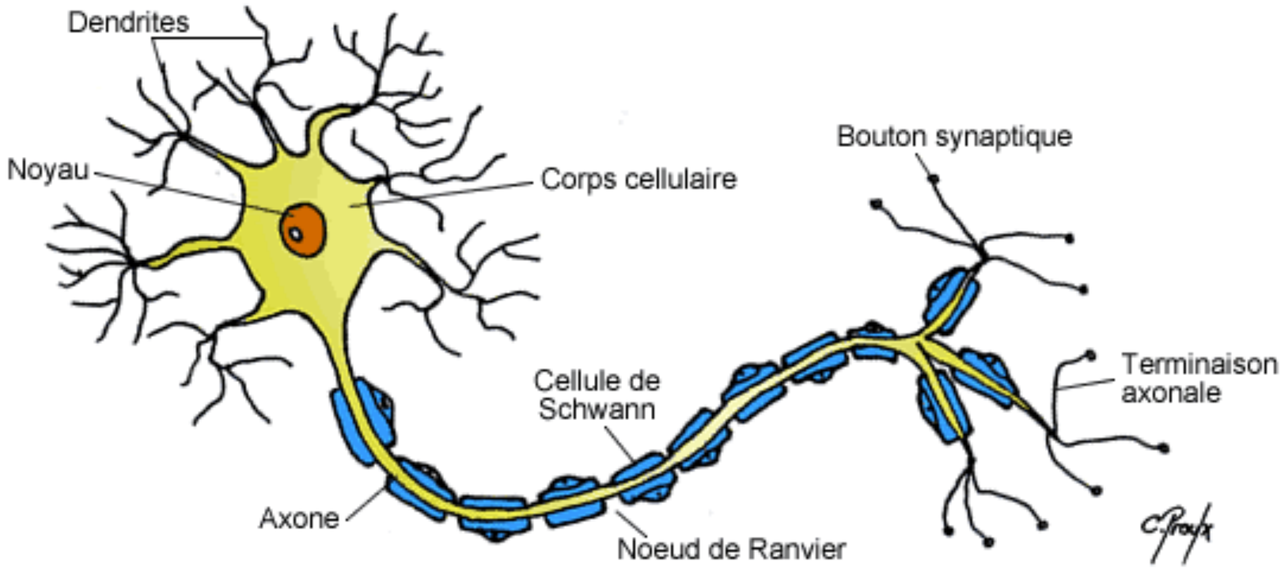


Schéma d'un neurone (cellule nerveuse)

Deux propriétés physiologiques d'un neurones

- Excitabilité d'un neurone: il répond aux stimulations et les transforme en impulsions nerveuses.
- Conductivité d'un neurone: il est capable de transmettre les impulsions.

- Approche de l'IA sub-symbolique
  - **Connexionnisme**: pour créer le modèle d'IA, on utilise des circuits connectés (et non pas des symboles), ici des réseaux de neurones.
- Début de la modélisation algorithmique des neurones: années 1950, **McCulloch & Pitts** (deux biologistes) étudiaient les neurones de grenouilles, ce sont les premiers qui propose un *modèle de neurone*

Le produit scalaire

Objet mathématique important

- Le produit scalaire n'utilise que deux opérations élémentaires: l'addition et la multiplication.
- À partir de deux vecteurs, on obtient un nombre.

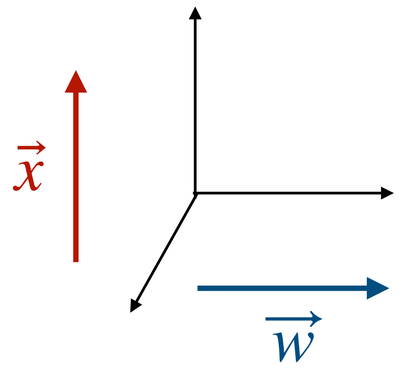
$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = x_1 * w_1 + x_2 * w_2 + x_3 * w_3 = \sum_{i=1}^{3} x_i * w_i$$

$\vec{x}$ $\overrightarrow{w}$

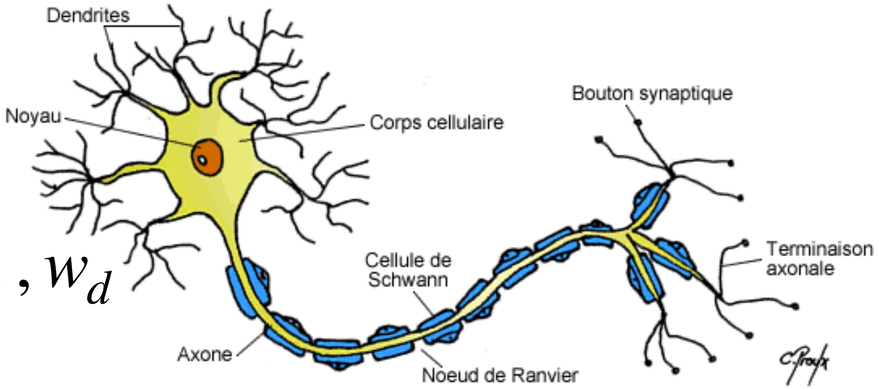
2 vecteurs
(à 3 coordonnées)

Nombre réel

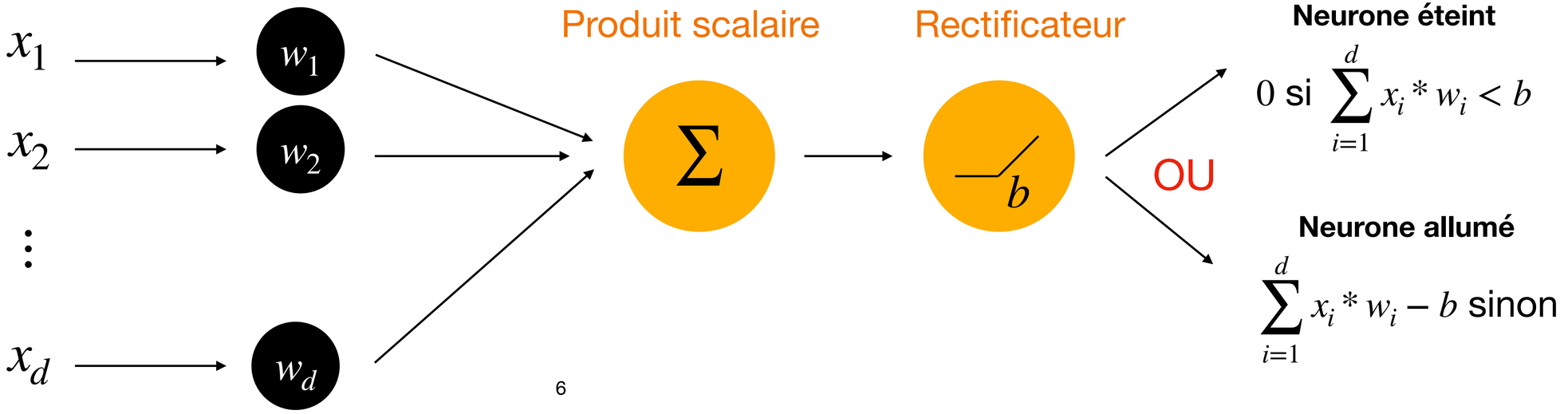
Interprétation géométrique du produit scalaire:
Si les vecteurs $\vec{x}$ et $\overrightarrow{w}$ sont sur un plan orthogonal, le produit scalaire est nul.



Modèle simple d'un neurone (McCulloch & Pitts, ~1950)

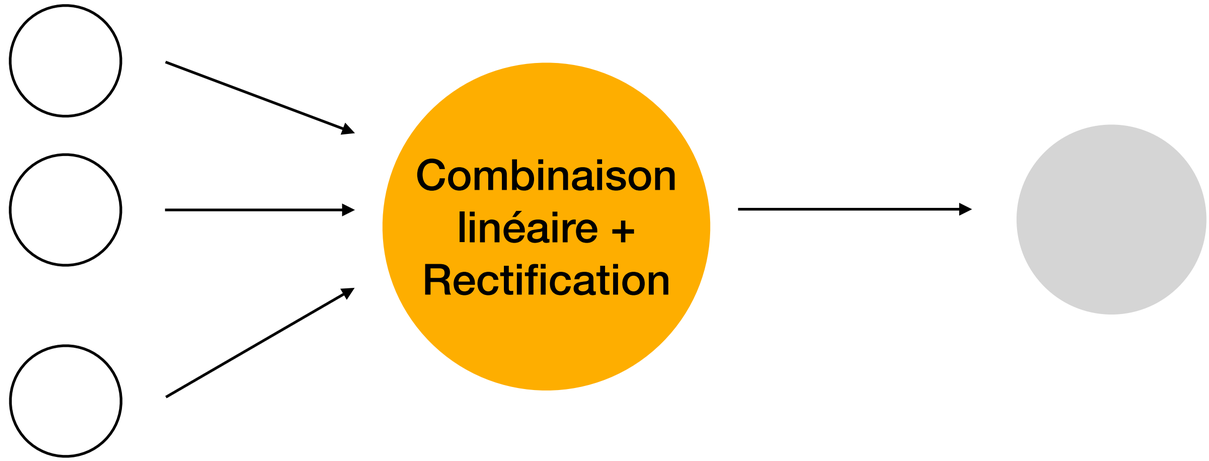


- Les dendrites du neurones sont *des entrées* $x_1, x_2, \ldots, x_d$
- Chacune de ses entrées est multipliée par *un poids* $w_1, w_2, \ldots, w_d$
- On fait un ***produit scalaire*** entre les entrées et les poids.
- On met une *non-linéarité*, par exemple un rectificateur: correspond à un **seuillage** - quand on est en dessous d'un seuil, le neurone ne renvoie aucune information, **principe de parcimonie.**



Modèle simple d'un neurone

- Modèle **très très simpliste** de ce que c'est un neurone.



Entrées (dendrites) **Axone** **Sortie**

- En faisant le produit scalaire puis la rectification, on regarde juste si on est à gauche ou à droite d'un hyperplan.
- Les paramètres sont seulement: les poids $w_1, w_2, \ldots, w_d$ et le seuil b.
- Le but est d'**estimer les paramètres** (cf algorithme perceptron dans les slides suivantes).

Fonction d'activation

Définition importante

***Définition* [fonction d'activation σ]: fonction appliquée à un signal de sortie d'un neurone.**

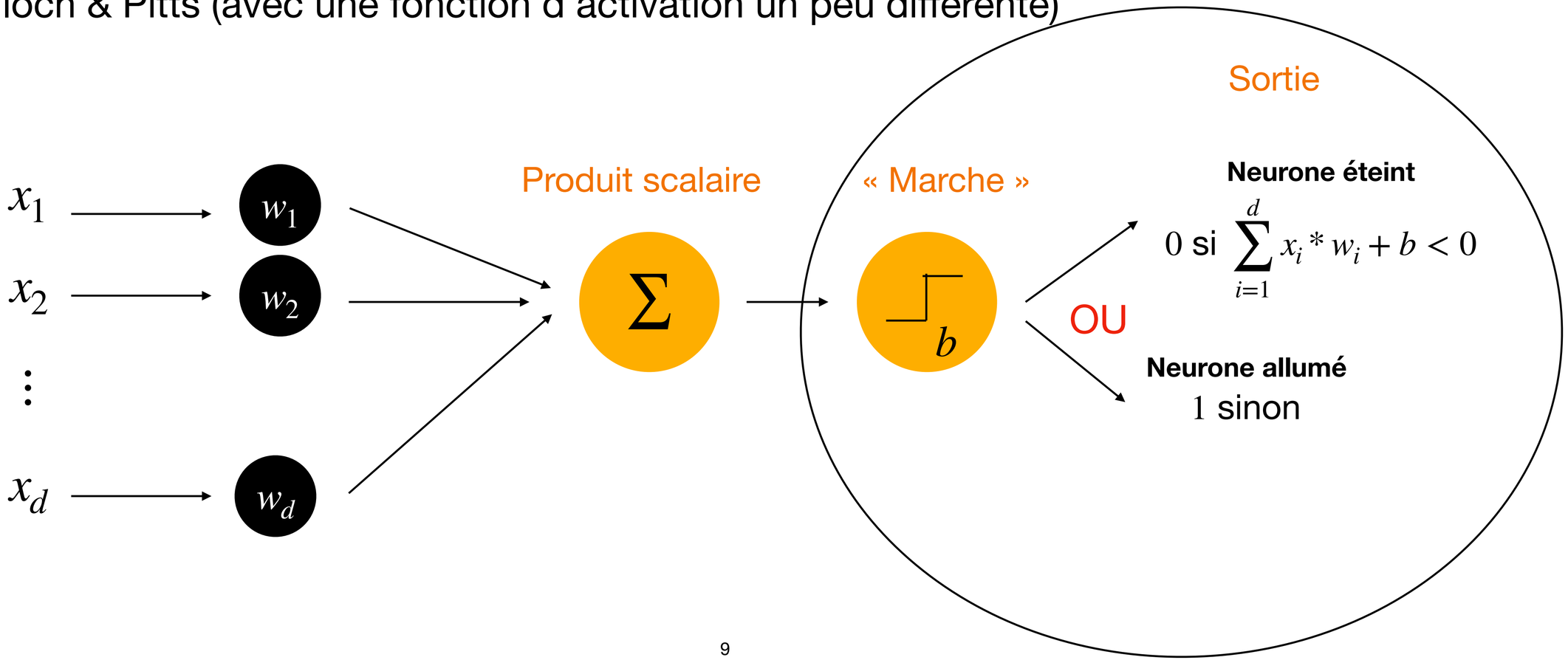
- Le rectificateur est un exemple de fonction d'activation:

$$\sigma(f(x)) = \begin{cases} 0 \text{ si } f(x) < 0 \\ f(x) \text{ si } f(x) \geq 0 \end{cases}$$

- Une fonction d'activation permet de **modéliser le phénomène biologique appelé potentiel d'activation**: c'est le seuil de stimulation qui, une fois atteint entraîne une réponse du neurone.

Le perceptron (Rosenblatt, 1958)

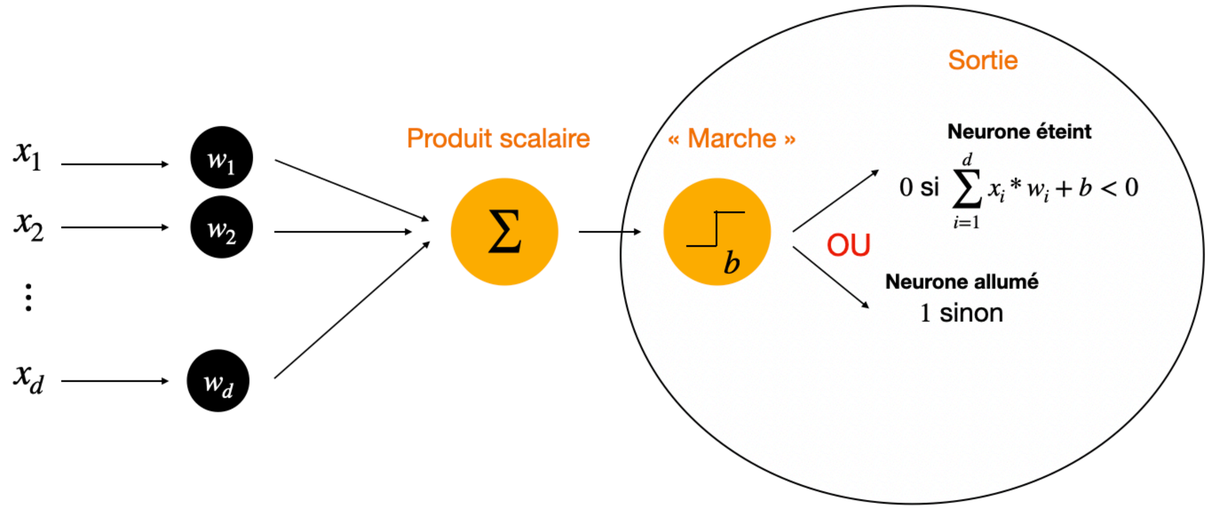
- En 1958, Rosenblatt (psychologue à Cornell) travaille sur les systèmes de décision présents dans l'œil d'une mouche, qui déterminent sa réaction de fuite.
- Il propose l'idée du perceptron, qui reprend la même modélisation que le neurone simple de McCulloch & Pitts (avec une fonction d'activation un peu différente)



Le perceptron (Rosenblatt, 1958)

- La « marche » est un autre exemple de fonction d'activation:

$$\sigma(f(x)) = \begin{cases} 0 \text{ si } f(x) < 0 \\ 1 \text{ si } f(x) \geq 0 \end{cases}$$



Algorithme d'optimisation: perceptron

- On considère la cas de la classification binaire: on essaie de **prédire y à partir de x**:
- (y, x) avec $y \in \{-1, 1\}, x = (x_1, \ldots, x_d)$
- On a plusieurs couples (y, x) (*si on ne considère qu'un seul couple, c'est comme si on avait un jeu de données avec une seule ligne !)*

Algorithme:

But: estimer les poids $w_1, w_2, \ldots, w_d$ et le seuil b

1. **Initialisation**: $w_1 = 0, \quad \ldots, \quad w_d = 0, \quad b = 0$
2. Répéter sur tous les couples: $\begin{cases} \text{si } y\left(\sum_{i=1}^{d} w_i x_i + b\right) \leq 0, \text{ modifier } w_1 \text{ en } w_1 + yx_1, \ldots, w_d \text{ en } w_d + yx_d \text{ et } b \text{ en } b + y \\ \text{sinon ne pas modifier les estimateurs.} \end{cases}$

Quand $y\left(\sum_{i=1}^{d} w_i x_i + b\right) \leq 0$, on a mal *classifié* la donnée car:

Pas le même signe !

$$y\left(\sum_{i=1}^{d} w_i x_i + b\right) \leq 0$$

$$\rightarrow \quad y \leq 0 \text{ et } \left(\sum_{i=1}^{d} w_i x_i + b\right) \geq 0$$

$$\rightarrow \quad y \geq 0 \text{ et } \left(\sum_{i=1}^{d} w_i x_i + b\right) \leq 0$$

Quand $y\left(\sum_{i=1}^{d} w_i x_i + b\right) > 0$, on a bien *classifié* la donnée car:

Même signe !

$$y\left(\sum_{i=1}^{d} w_i x_i + b\right) > 0$$

$$\rightarrow \quad y > 0 \text{ et } \left(\sum_{i=1}^{d} w_i x_i + b\right) > 0$$

$$\rightarrow \quad y < 0 \text{ et } \left(\sum_{i=1}^{d} w_i x_i + b\right) < 0$$

Fonction de coût

Définition importante

***Définition* [fonction de coût ℓ]: fonction qui donne une grande valeur si la sortie du réseau de neurone est loin de la vérité, une petite valeur sinon.**

Le but d'un algorithme d'apprentissage est donc de minimiser cette fonction de coût pour apprendre les paramètres d'intérêts.

- Un exemple de fonction de coût que l'on a déjà vu en régression linéaire est l'erreur quadratique moyenne.

- Ici, on veut minimiser la fonction de coût suivante:

$$\ell(w) = -\sum_{couples(y,x)} y\left(\sum_{i=1}^{d} w_i x_i + b\right)$$

Cas mal classifié: le terme est négatif * (-1) = résultat positif (*grande* valeur de $\ell(w)$)

La fonction pénalise bien les cas mal classifiés, on cherche à minimiser cette fonction.

Remarques sur l'algorithme du perceptron

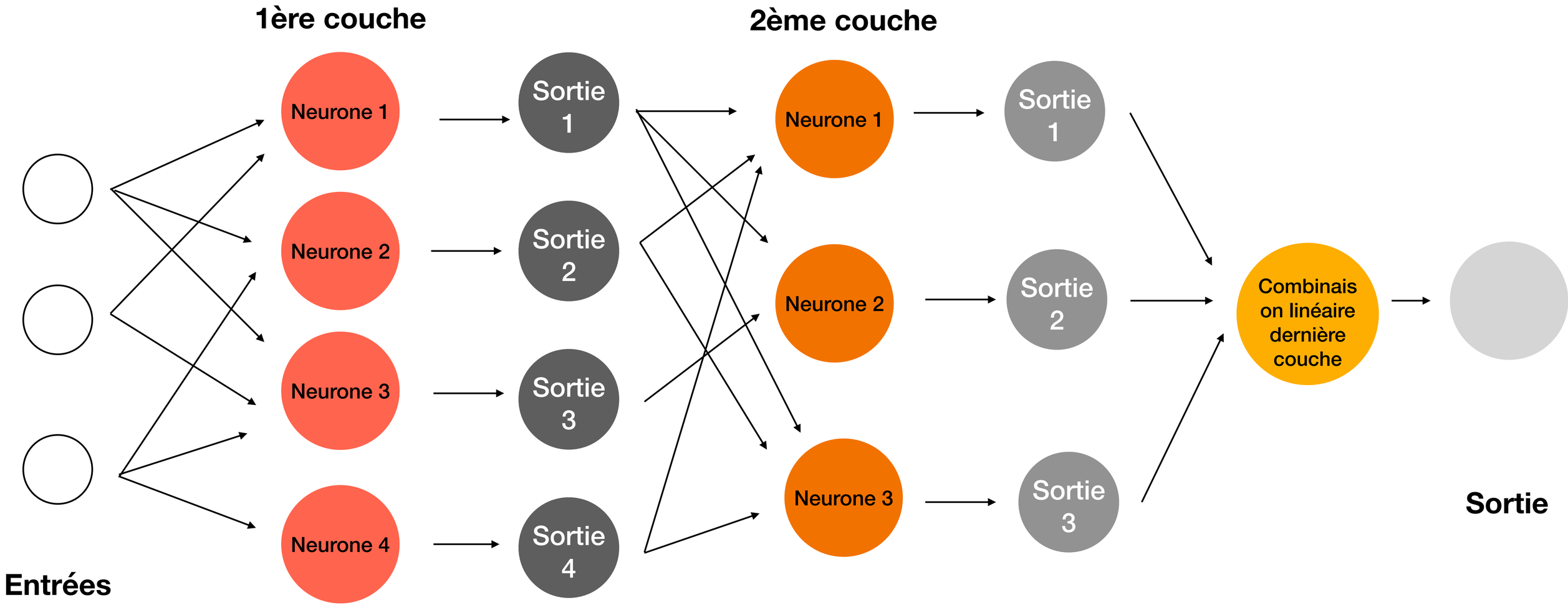
$$\ell(w) = -\sum_{couples(y,x)} y\left(\sum_{i=1}^{d} w_i x_i + b\right)$$

Cas mal classifié: le terme est négatif * (-1) = résultat positif (*grande* valeur de $\ell(w)$)

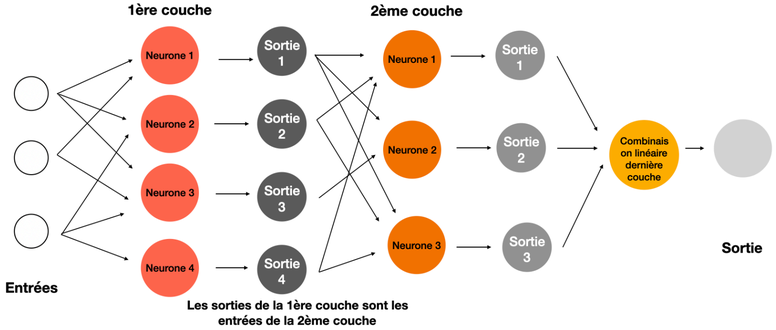
- On peut montrer que l'algorithme de perceptron est un **algorithme de descente de gradient stochastique** (la version stochastique, avec de l'aléatoire, de l'algorithme de gradient que nous avons vu la dernière fois).
- En pratique, on n'utilise plus ce modèle qui est trop simple: **quand les données ne sont pas linéairement séparables, l'algorithme ne marche pas**.

Réseaux de neurones

Pour modéliser de manière plus complexe:
on met les neurones en réseaux

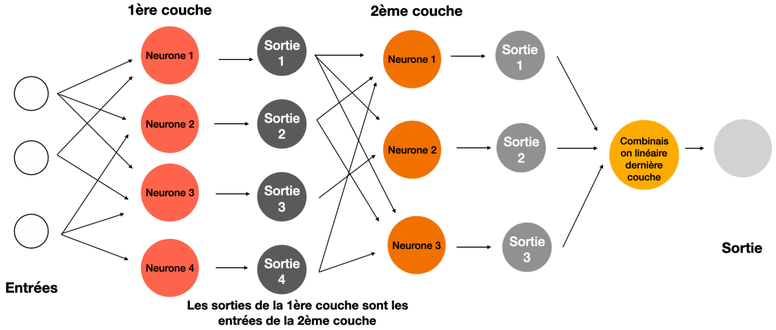


Optimisation plus complexe



- Les paramètres sont les poids du réseau.
- Pour les estimer, **on ne peut plus utiliser l'algorithme du perceptron !**
- On utilise des algorithmes de type descente de gradient.

Architecture du réseau



- Est-ce que cela marche ? **En général, NON !**

- Si on relie toutes les entrées à tous les neurones (réseaux de neurones complètement connectés), c'est trop complexe. Cela ne va généralement pas marcher, l'optimisation des paramètres (les poids du réseau) va être mauvaise.

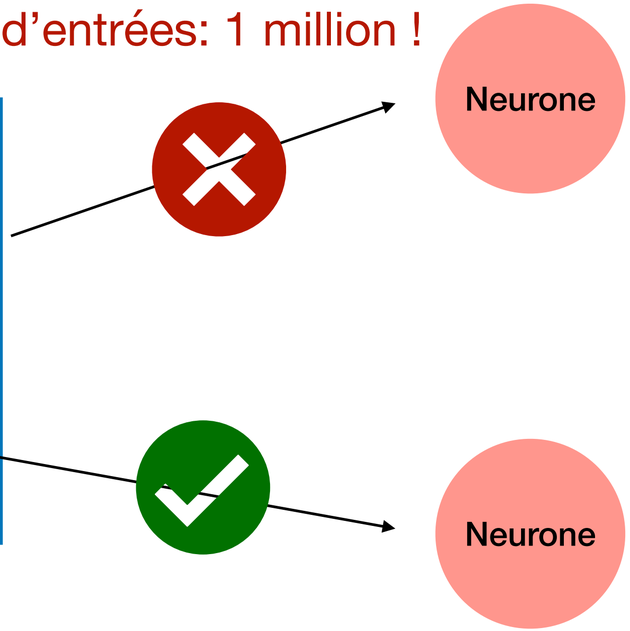
- On doit structurer les réseaux de neurones, **trouver une architecture** (souvent empiriquement). Trouver l'architecture revient à trouver les propriétés du problème qui vont permettre de réduire la complexité.

Réseaux Convolutionnels Profonds (LeCun, 2015)

- Avancée fondamentale: utiliser des **informations a priori** pour structurer les réseaux.

Une image en entrée (~1 million de pixel)

Trop d'entrées: 1 million !

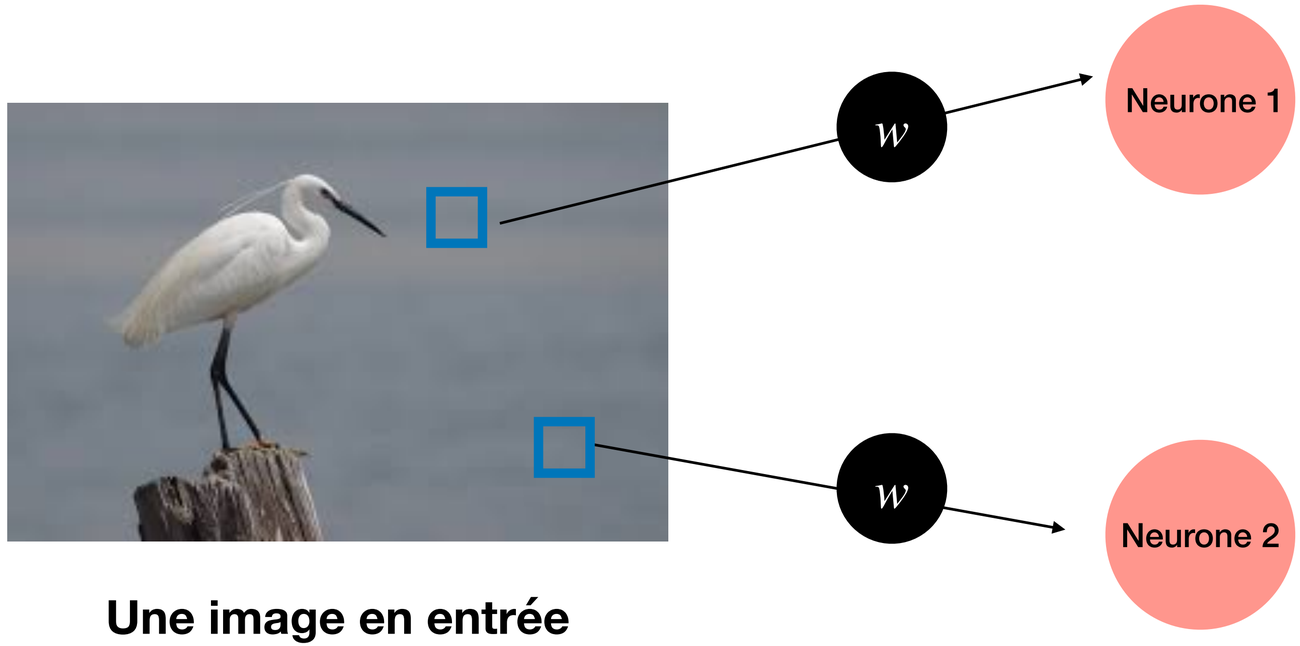


On donne à chaque neurone des sous-images de taille 3*3 (au lieu d'un million d'entrées, on a 9 entrées)

Idée 1: Au lieu de donner à chaque neurone toutes les entrées de l'image (ce qui reviendrait à donner les un million de pixels à chaque neurone), **on donne à chaque neurone une toute petite partie de l'image initiale**.

Réseaux Convolutionnels Profonds (LeCun, 2015)

- Avancée fondamentale: utiliser des **informations a priori** pour structurer les réseaux.



Une image en entrée (~1 million de pixel)

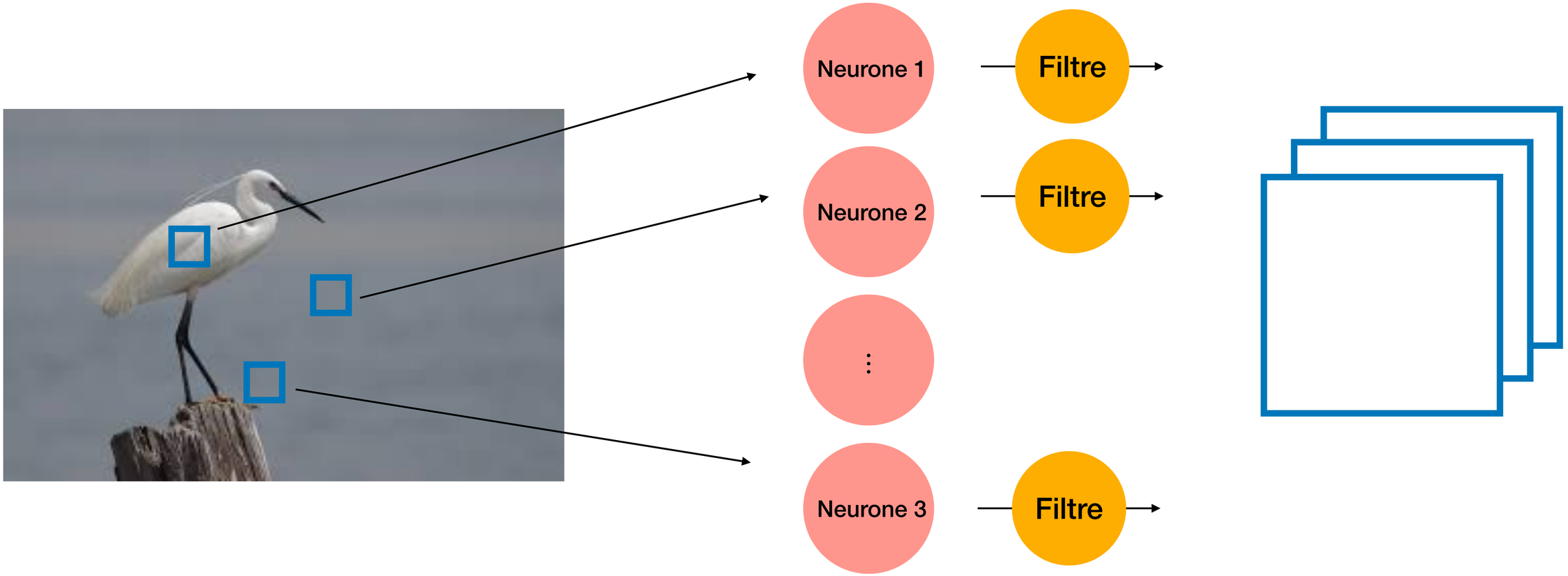
On a donc moins de paramètres !
(Certains poids sont égaux)

Idée 2: On considère qu'il y a une certaine **symétrie du problème par translation** (le même problème se retrouve dans plusieurs carrés). On peut donc considérer que les poids associé aux neurones qui répondent à ces carrés sont les mêmes.

Réseaux Convolutionnels Profonds (CNN)

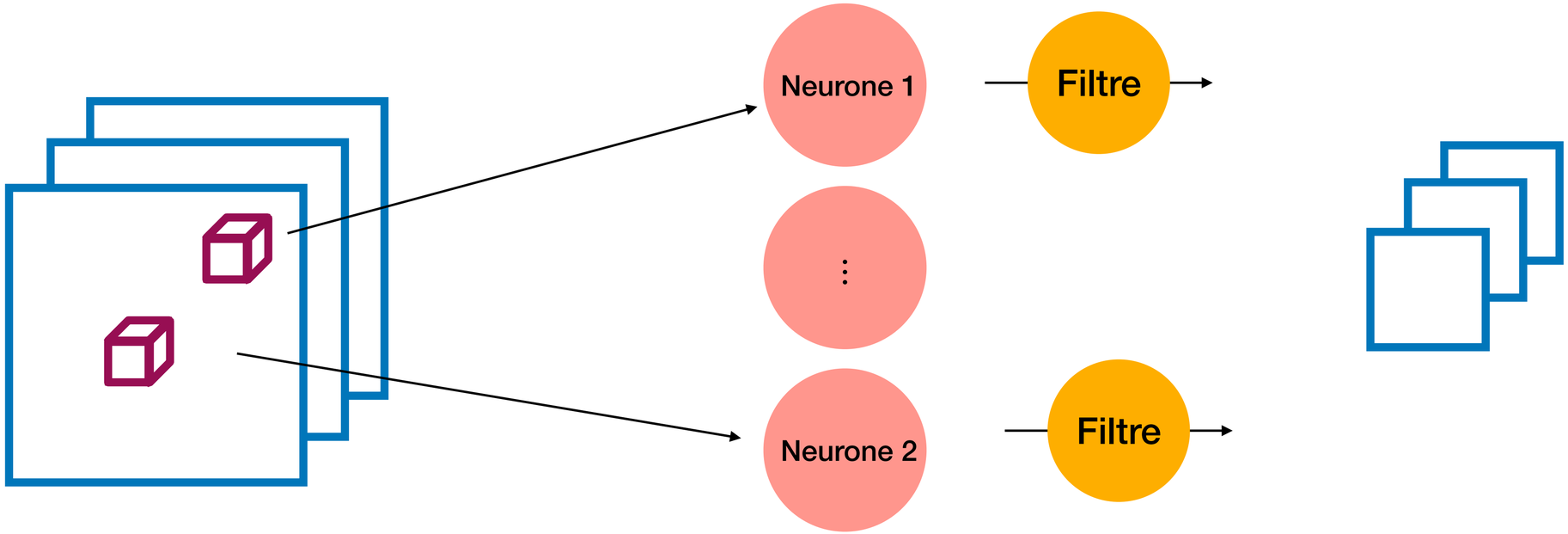
- Avec ces deux idées, la **réduction de la complexité du problème** est énorme:
  - On a beaucoup moins d'entrées pour chaque neurone.
  - On utilise des poids égaux.
- En faisant une combinaison linéaire invariante par translation, on a fait ce que l'on appelle une **convolution**.
- Une convolution + une fonction d'activation (seuillage) est appelé un **filtre.** On applique donc différents filtres aux sous-images d'entrées.

Première couche d'un CNN



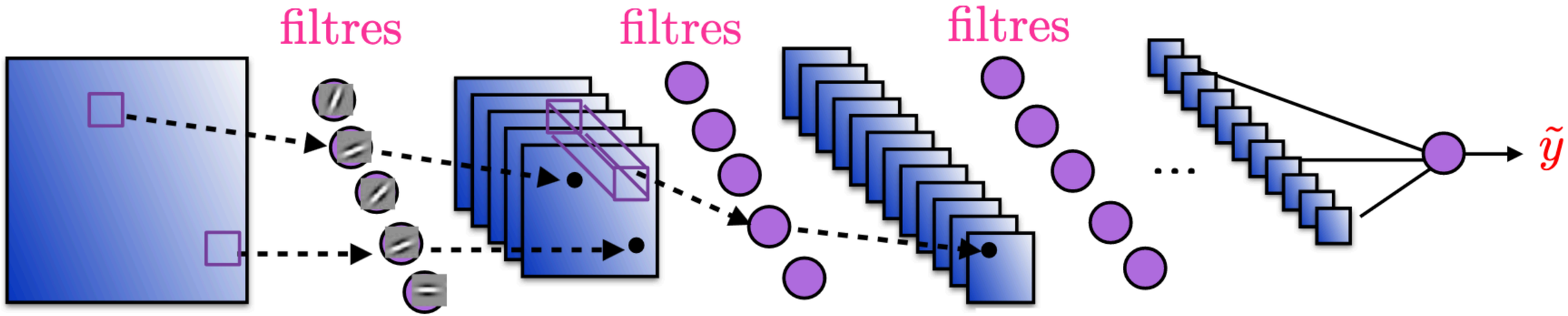
En sortie de la couche, on a plein de petites images (transformées avec les filtres)

Deuxième couche d'un CNN



Dans la deuxième couche, chaque neurone prend **toutes les images dans une petite région** en entrée.

Réseaux Convolutionnels Profonds (CNN)



Source: cours de Stéphane Mallat au Collège de France

Pour la dernière couche, toutes les sorties sont agrégées linéairement et on applique une fonction d'activation.

Réseaux Convolutionnels Profonds

Réduction de la complexité du problème en utilisant beaucoup de paramètres

- On a **réduit la complexité du problème**: à la dernière couche, le nombre de pixel de chaque petite image est très inférieur à l'image de départ.
- À chaque couche, on apprend les structures sous-jacentes de l'image qui permettent de construire des frontières entre les classes. Ces frontières sont de plus en plus aplaties (**dernière couche: la frontière entre les classes est un hyperplan**, comme dans le cas du modèle de neurone simple précédent).
- Mais attention, on a **beaucoup de paramètres** ! Le nombre de paramètres d'un CNN est généralement supérieur à 1 million (supérieur aux nombre de pixels de l'image d'origine, ce qui peut être contre-intuitif). Le problème d'optimisation est complexe derrière.

Application possible: Classification d'image

- **Oeil humain est très développé.** Par exemple, en 150ms on est capable de dire si dans une image il y a une animal ou non. Source: Speed of processing in the human visual system, 1996, Nature

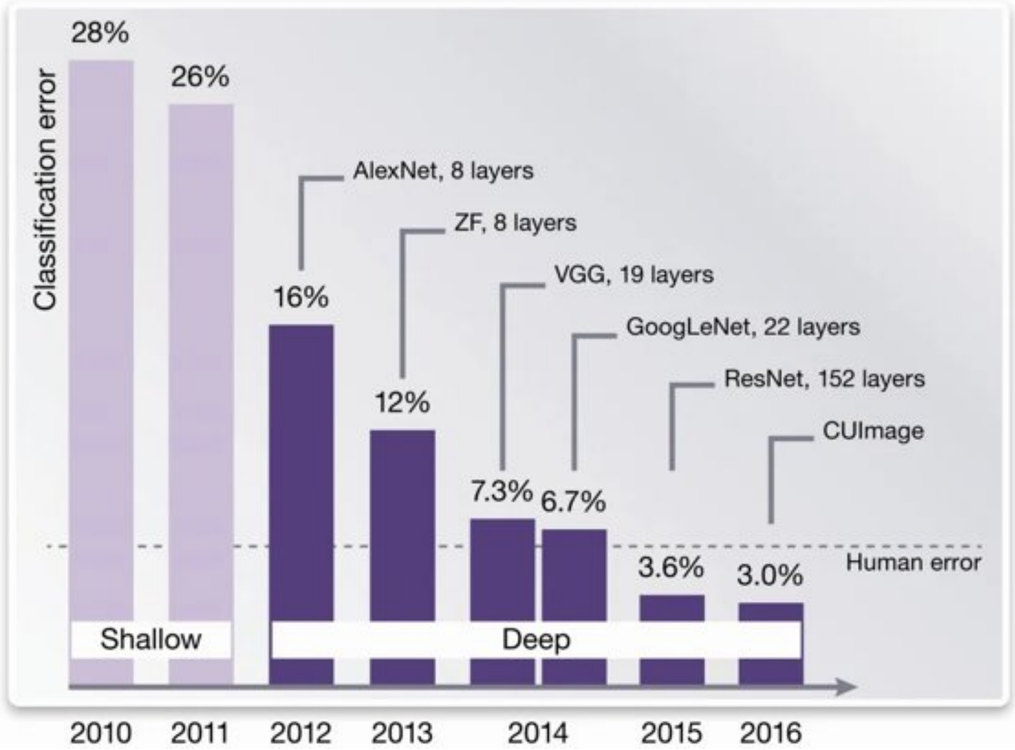
- Dans les années 80, une première approche (IA symbolique) pour reconnaître les objets a été développée. L'idée était de **détecter les contours** puis de définir des métriques pour comparer les contours d'un image à une autre. Ce problème a été abordé en linguistique pour **définir une grammaire des images** (caractérisation d'une forme).

- Echec de cette approche: ça marche bien sur des problèmes simples, mais pas sur les problèmes complexes.

Application possible: Classification d'image

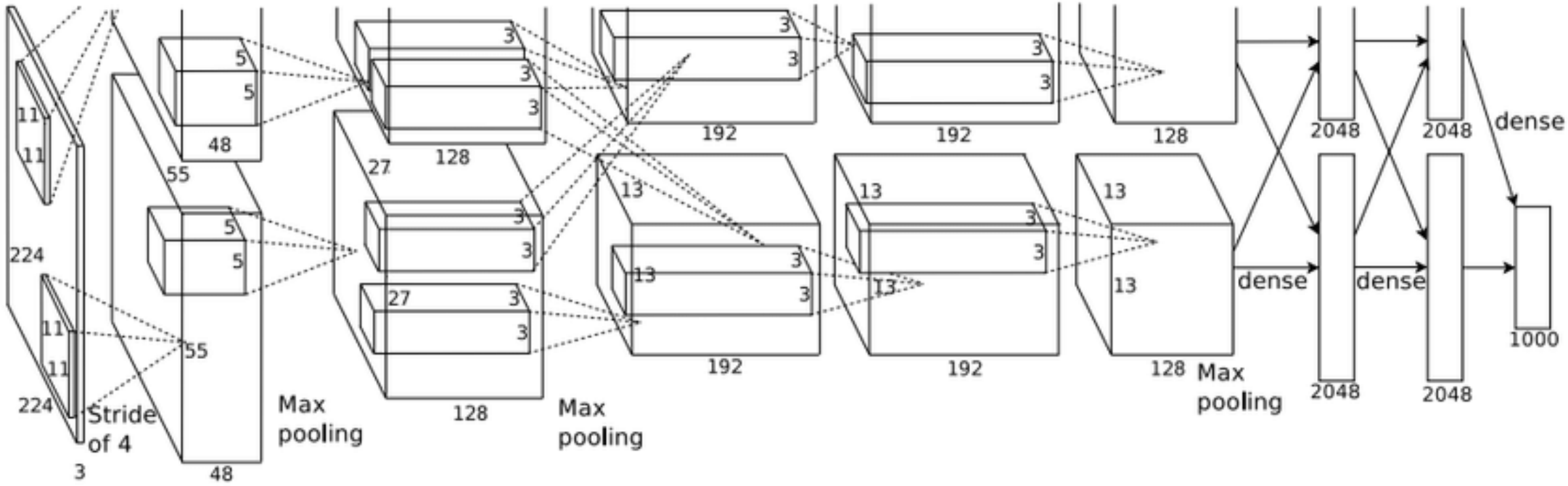
ImageNet

- Environ 1,5 millions d'images
- Compétition ILVSRC qui évalue les algorithmes du traitement d'images.
- Avant 2012 (utilisation de l'apprentissage profond), l'erreur était supérieur à 25%, elle atteint 16% en 2012.



Application possible: Classification d'image

Architecture du réseau AlexNet



Source: A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 25, pages 1097–1105. Curran Associates, Inc., 2012

Application possible: Classification d'image

ImageNet

		Erreur
AlexNet	2012	15 %
GoogLeNet	2016	3 %
Humain	-	~5 %

Cela ne veut pas dire que les réseaux de neurones marche mieux que le système visuel humain ! Les erreurs ne sont pas de même nature.

Sur une tâche très spécialisée (reconnaître les aigrettes), le réseau de neurones sera peut-être meilleur, mais il va pouvoir faire des erreurs beaucoup plus « bêtes » que l'humain.

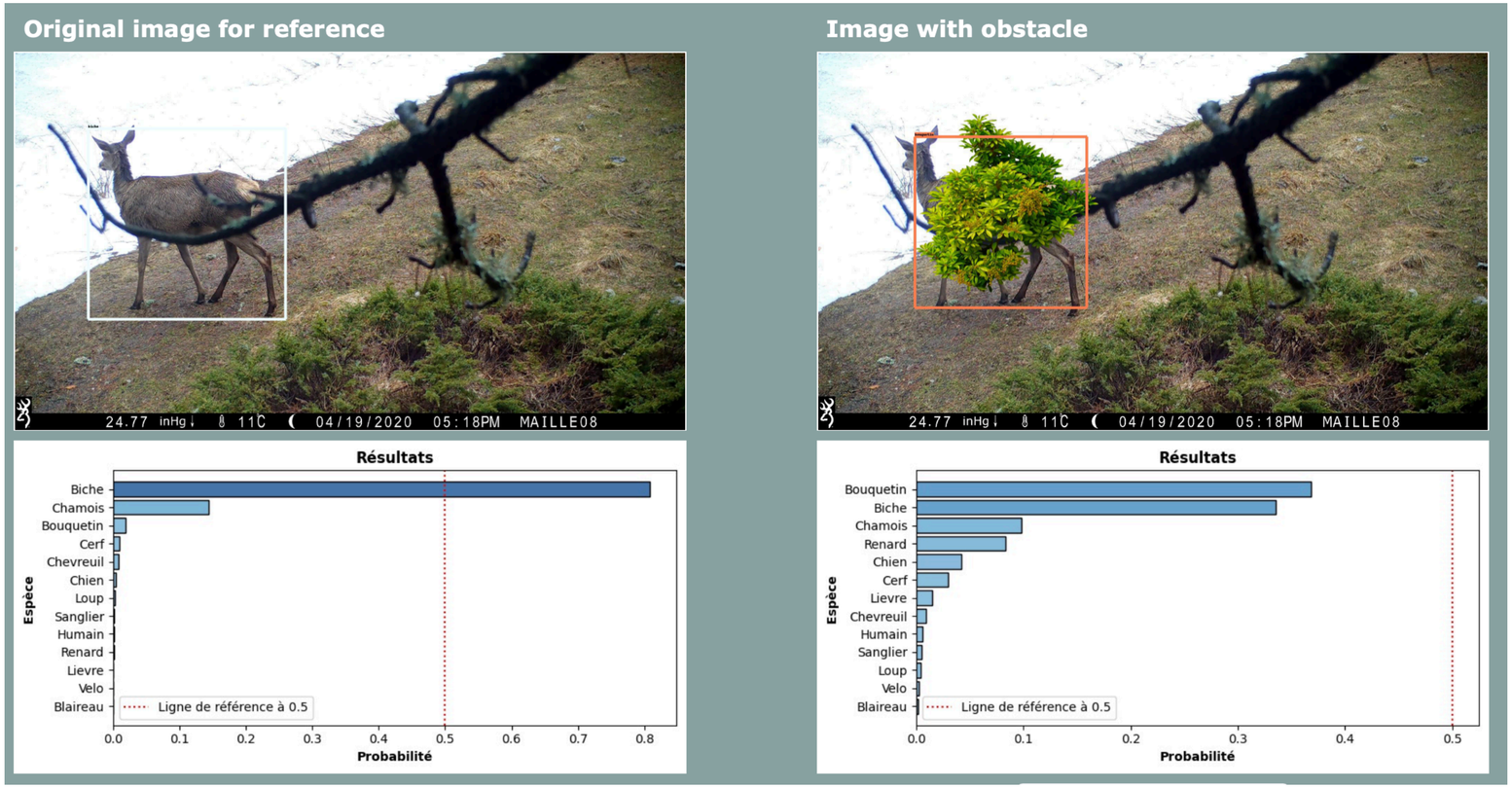
Application possible: Classification d'image

Vérifier que l'algorithme est robuste

Robustesse d'un algorithme: on doit vérifier que si l'entrée est un peu bruitée, on retrouve une sortie proche

2 applications pour comprendre:

- http://playground.tensorflow.org/
- https://3ia-demos.inria.fr/mercantour/



2. Réduction de dimension

Analyse à Composantes Principales (ACP)

Réduction de dimension classique

- Le but est de synthétiser l'information contenue dans les données. L'ACP va **trouver un espace où les données sont le mieux représentées possible**, notamment:
  - en débruitant: on oublie les axes qui ne nous aident pas et ajoutent du bruit.
  - en décolérant: les nouveaux axes donnent des informations distinctes sur les données, ils ne sont pas corrélés.
- L'ACP va donner un pourcentage de « la variance du jeu de donnée » expliquée par chacun des axes trouvés: cela donne une idée de combien d'axes il faut pour bien représenter les données.

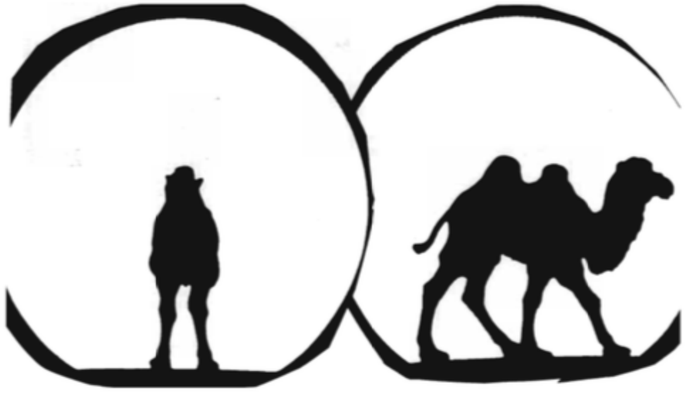
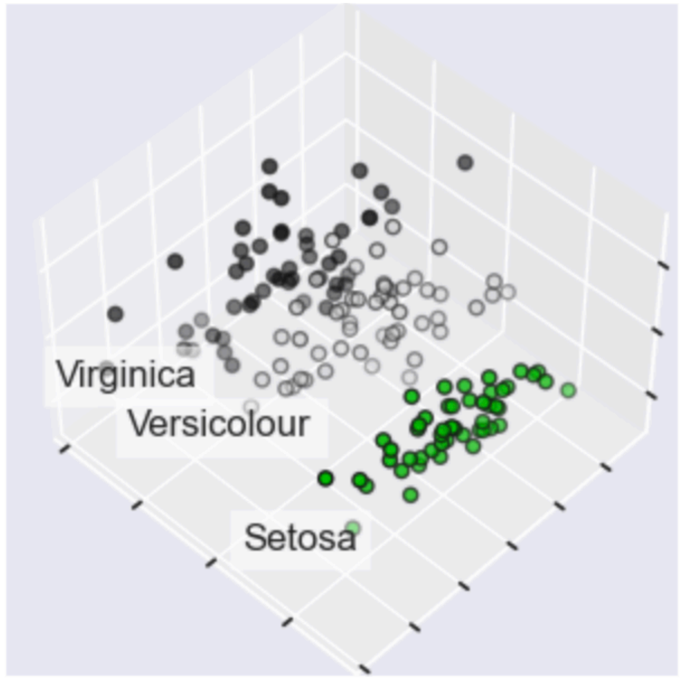


Figure 1: Camel or dromedary? source J.P. Fénelon

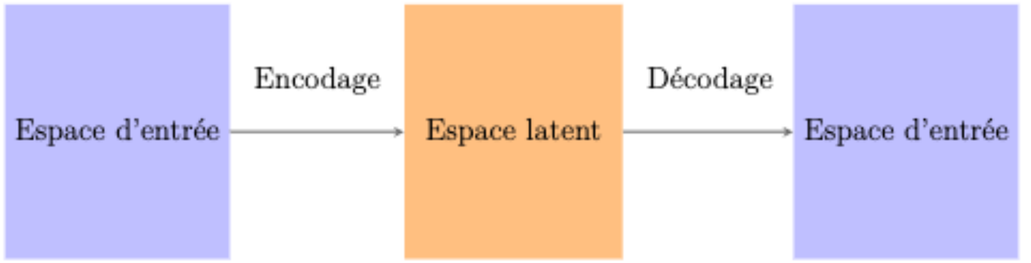
Pour iris, les 3 premiers axes expliquent 99,5% de la variance du jeu de données.



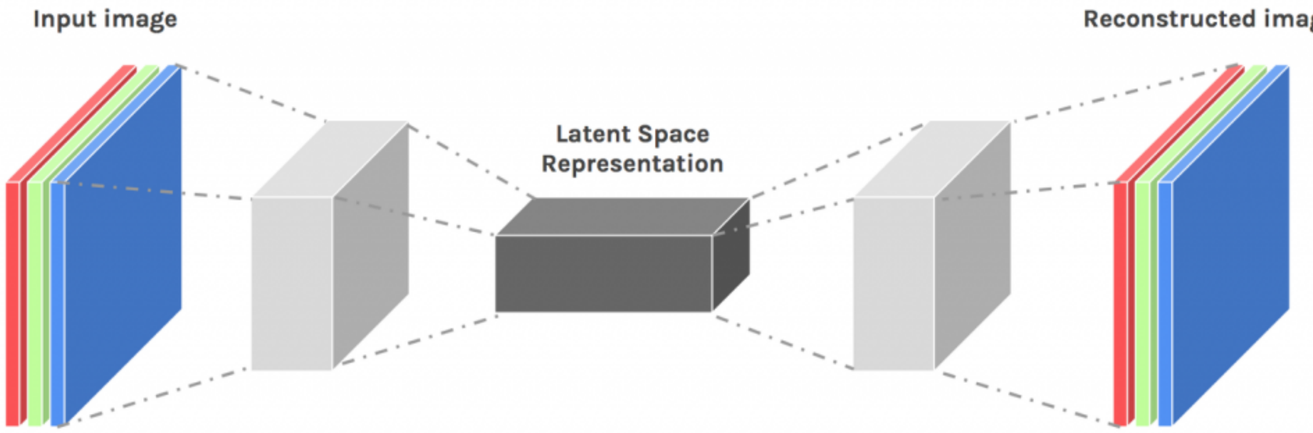
Source: scikit-learn

Auto-encodeurs

Réduction de dimension utilisant les réseaux de neurones



- Les auto-encodeurs fonctionnent ainsi:
  - On **encode les données** qui se retrouve dans un **espace latent**. Cet espace latent permet de condenser les informations contenues dans les données et de filtrer le bruit.
  - On **décode les données** pour que la sortie soit dans le même espace que l'entrée.
  - Les paramètres de l'encodeur et du décodeur sont appris en utilisant deux **réseaux de neurones**.

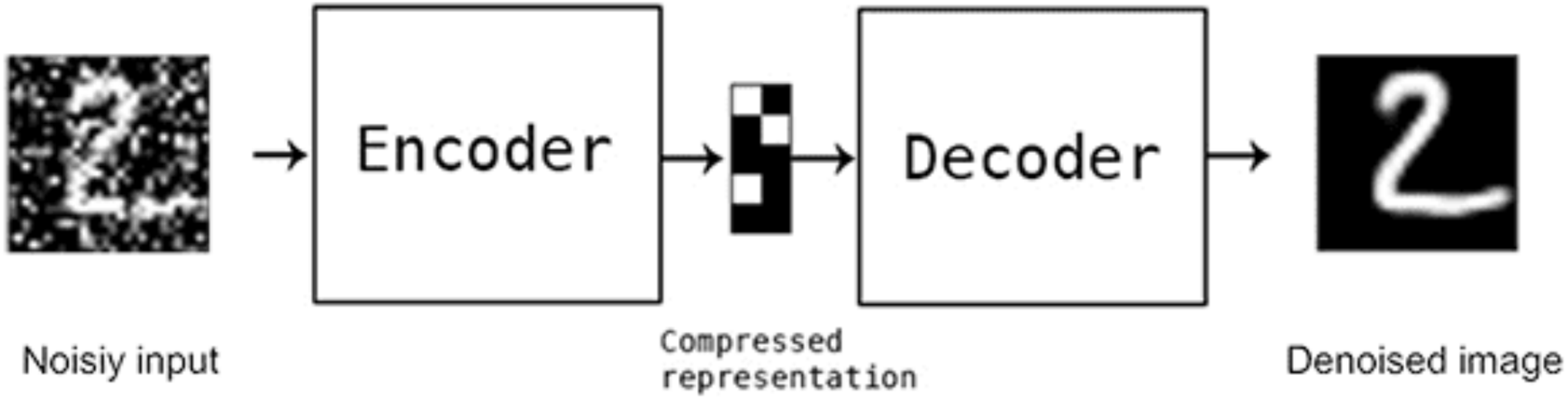


Source: Introduction aux auto-encoders sur La revue IA

Auto-encodeurs

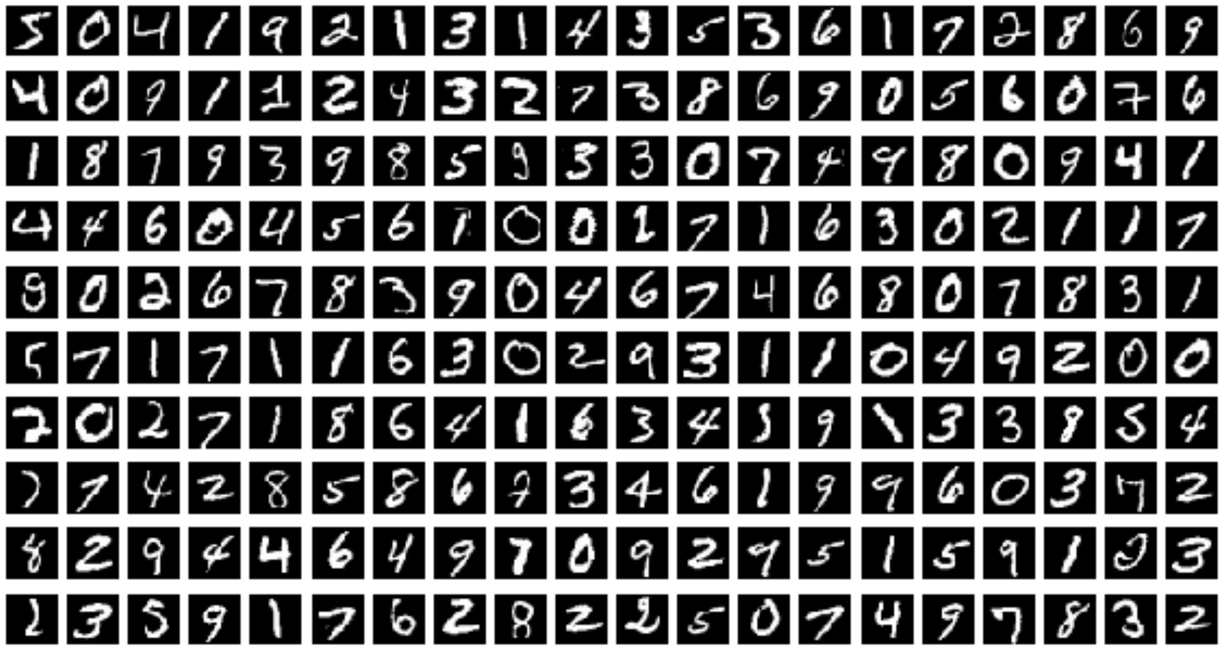
Réduction de dimension utilisant les réseaux de neurones

- Quelques applications pour les auto-encodeurs: **débruiter des images**, compresser des images, générer des nouvelles images, …



Source: Introduction aux auto-encoders sur La revue IA

Jeu de données MNIST



2018: taux d'erreur 0,18%

- C'est un jeu de données **très utilisé** pour tester un algorithme d'apprentissage statistique.
- Le but est de **reconnaître l'écriture manuscrite:** on a une image en noir et blanc d'un chiffre entre 0 et 9 écrit à la main et on veut trouver son étiquette (0, 1, 2, 3, 4, 5, 6, 7, 8 ou 9).
- Les données d'entrainement sont étiquetées (**problème d'apprentissage supervisé**): on a accès à leurs étiquettes.
- Il y a **60 000** images d'entrainement.

3. TP: réseaux de neurones

https://mybinder.org/v2/gh/AudeSportisse/Efelia-cours/HEAD?labpath=TP2_neural_networks.ipynb

Dernière séance (9/11)

1. **La prochaine séance (19/10) sera une séance de TP avec Djampa Kozlowski**
2. Programme prévisionnel:
   - Forêts aléatoires
   - Limites et biais de l'IA
3. **Préparer l'exposé (session de 2h sur la lecture des articles)**
4. **Faire le QCM (maison - je le mettrai sur modèle)**

Bibliographie (liens cliquables)

- Cours Introduction to Neural Network, **Erwan Scornet**, professeur à la Sorbonne, LPSM.
- Introduction aux réseaux de neurones profonds, **Stéphane Mallat**, Professeur du Collège de France
- Introduction aux auto-encoders sur La revue IA, Ilyes Talbi, 2021