

Petit historique

Même si les légendes d'êtres artificiels intelligents remontent à la nuit des temps (en particulier à l'Antiquité), l'intelligence artificielle en tant que domaine de recherche est assez récente : **1956**, à la suite d'une célèbre **conférence à Dartmouth**.

L'IA naît ainsi des conséquences des travaux d'*Alan Turing*, fondateur de l'informatique moderne, ainsi que des réalisations d'autres mathématiciens et programmeurs (dans les années 1950).

Elle connaît alors un premier **âge d'or** jusqu'en **1974** où l'on pense qu'une IA est accessible et où l'on investit des millions dans la recherche. De nombreuses théories sont formulées, mais la faible puissance des ordinateurs de l'époque va rapidement mettre un terme

A partir de là et jusqu'en **2011**, l'IA connaît un nouvel **âge d'or** durant lequel elle va prendre de l'ampleur, se diversifier, voir ses premiers succès médiatisés (dont la victoire de *Deep Blue* aux échecs contre Kasparov en 1997), et profiter de l'ère du Big Data pour s'apparenter à la Business Intelligence et profiter de nouvelles capacités de calculs et de jeux de données.

Depuis 2011, on est toujours dans l'**âge d'or** de l'IA mais on a dorénavant accès au Deep Learning (DL), un type d'algorithmes d'intelligence artificielle plus autonome et plus versatile que tout ce qui précède. Trop gourmand en données et en calculs, le DL peut enfin être appliqué à des cas concrets et remporte ses **premières compétitions** dans de la reconnaissance d'images (déterminer les nombres écrits à la main, octobre 2012, considérés par plusieurs comme le vrai départ du DL) ou encore dans la chimie (prédire la cible thérapeutique d'une molécule de médicament

IA = **Intelligence Artificielle** : désigne un algorithme qui **imite le comportement d'un humain**, non pas dans son calcul mais dans son résultat. Par exemple, une IA prédira la météo (un algorithme prendra en compte les vents, les nuages, etc... pour donner en résultat la probabilité qu'il fasse beau), dira si l'image est celle d'un chat

ML = **Machine Learning** (ou apprentissage automatique) : sous-branche de l'IA, le machine learning est un ensemble de **méthodes statistiques appliquées à l'IA**. Elles permettent en particulier à l'IA d'apprendre à partir de données d'exemple (apprentissage supervisé : on montre à l'IA plein d'appartements avec leur prix, puis elle peut deviner le prix d'un nouvel appartement). L'algorithme le plus simple et le plus connu du ML est la régression linéaire, qui consiste à trouver la droite approximant un nuage de points (grâce à ça, on peut facilement « classifier » des données en fonction de si elles sont en-dessous ou en-dessus).

DL = **Deep Learning** (ou apprentissage profond) : sous-branche du ML, le deep learning repose principalement sur l'utilisation de **réseaux de neurones** (on se rapproche donc de la neuroscience plutôt que des statistiques) pour résoudre les problèmes. Les applications les plus étonnantes de l'IA aujourd'hui sont possibles grâce au DL (dont la reconnaissance faciale et classification d'images, la génération de texte, la prédiction de pannes complexes)

La datascience dont tout le monde parle(ou science des données) (DS)

La Data Science est au cœur des Mathématiques, de l'Informatique et de l'Expertise

Il s'agit d'une matière transverse à l'IA car elle regroupe :

- la capacité à **gérer** et stocker les données (rôle du data engineer)
- la capacité à les **analyser**, grâce à des outils statistiques et plus généralement de Machine Learning
- la capacité à les **visualiser** ainsi que les résultats, via des graphes et d'autres modèles de représentation

De manière générale, la data science est très souvent vue comme du **machine learning appliqué** :

L'apprentissage supervisé

(en anglais : Supervised Learning) est le paradigme d'apprentissage le plus populaire en Machine Learning et en Deep Learning. Comme son nom l'indique, cela consiste à superviser l'apprentissage de la machine en lui montrant des exemples (des données) de la tâche qu'elle doit réaliser. Les applications sont nombreuses : Reconnaissance vocale, vision par ordinateur, régressions, classifications... La grande majorité des problèmes de Machine Learning et de Deep Learning utilisent l'apprentissage supervisé. Il est donc essentiel de bien comprendre le fonctionnement de cette mécanique.

Comment fonctionne l'apprentissage supervisé ?

Avec l'apprentissage supervisé, la machine peut apprendre à faire une certaine tâche en étudiant des exemples de cette tâche. Par exemple, elle peut apprendre à reconnaître une photo de chien après qu'on lui ait montré des millions de photos de chiens. Ou bien, elle peut apprendre à traduire le français en chinois après avoir vu des millions d'exemples de traduction français-chinois.

D'une manière générale, la machine peut apprendre une relation $f : x \rightarrow y$ qui relie x à y en ayant analysé des millions d'exemples d'associations $x \rightarrow y$.

L'apprentissage non supervisé

À la différence de l'apprentissage supervisé, l'apprentissage non supervisé est celui où l'algorithme doit opérer à partir d'**exemples non annotés**. En effet, dans ce cas de figure, l'apprentissage par la machine se fait de manière entièrement indépendante. Des données sont alors renseignées à la machine sans qu'on lui fournisse des exemples de résultats.

On attend donc de la machine qu'elle crée elle-même les réponses grâce à différentes analyses et au classement des données.

Les modèles d'apprentissage non supervisé sont notamment utilisés pour :

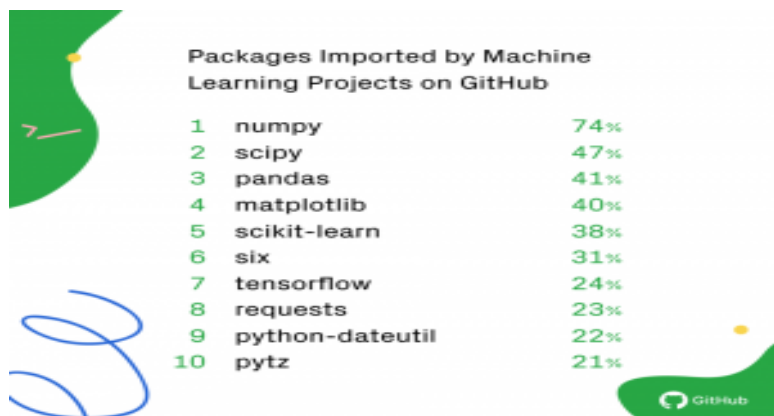
- Le classement des données
- Le calcul approximatif de la densité de distribution
- La réduction des dimensions

Quel langage utiliser ? Python 3 ? Java ? Javascript ?



- **Python** : le langage de référence absolue en IA. Il permet de gérer beaucoup de choses (en plus de l'IA, dont par exemple des micro-services, des bases de données), il est facile à installer, n'est pas compilé, est rapide/léger... Tous les outils d'IA sont disponibles en Python, généralement bien avant les autres langages !

Quelles sont les bibliothèques principales de l'IA ?



Voici les principaux modules à connaître ainsi que leur utilisation générale :

- **Numpy** : package supportant les opérations mathématiques sur des données multidimensionnelles. On en utilise quasiment tout le temps les « array » (pour faire des matrices).
- **Scipy** : ressemble un peu à numpy, à ceci près qu'il est dédié aux calculs scientifiques
- **Pandas** : qui gère les jeux de données (par exemple, importer un CSV et effectuer certaines opérations sur les données ou en extraire des informations basiques)
- **Matplotlib** : pour la visualisation des données par le biais de graphiques (plots)
- **Scikit-learn** : contient de nombreux algorithmes de machine learning. Il s'oppose aux frameworks de deep learning
- **NLTK** : pour manipuler du texte
- **OpenCV** : pour manipuler des images (les transformer)

Puis dans les utilitaires transverses nous avons :

- **Requests** : sert à faire des appels http
- **BeautifulSoup4** : qui parse efficacement des pages html

- **Pillow** : qui manipule des images sans problème
- **Six** : qui permet d'écrire du code compatible Python 2 et 3 en même temps !
- **Redis** : un petit serveur avec lequel on peut communiquer, notamment pour stocker des données
- **Pickle** : sert à sauvegarder et charger des objets Python dans des fichiers

Un exemple simple de classification d'images en python

De manière générale, un modèle de classification d'images fonctionne comme présenté sur la figure 1 :

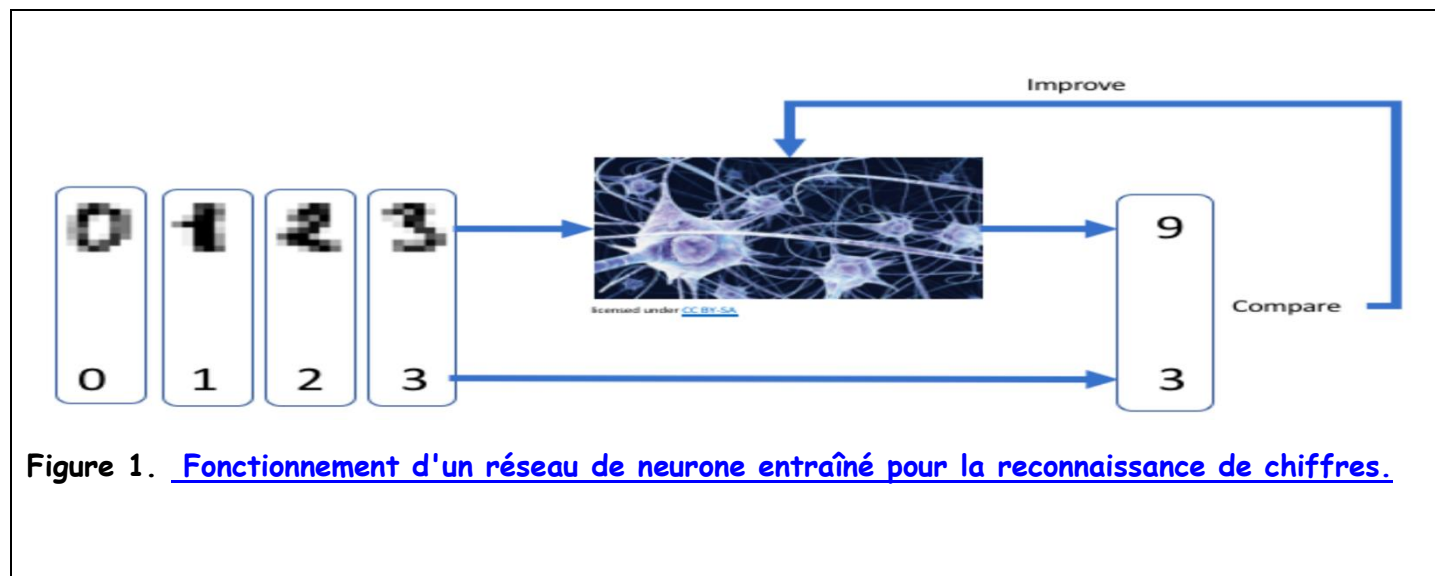


Figure 1. [Fonctionnement d'un réseau de neurone entraîné pour la reconnaissance de chiffres.](#)

L'utilisateur fournit au modèle un échantillon d'images étiquetées par un humain, ici des chiffres manuscrits.

Chaque image est constituée de pixels, avec dans chaque pixel un niveau de gris, ou trois niveaux de couleurs. Ci-dessus, nos images sont en noir et blanc, et font 8x8 pixels. Chaque image est donc représentée par 64 valeurs. Ces images sont des points dans un espace à 64 dimensions.

Le modèle est une fonction de ces 64 valeurs, qui fournit une unique valeur en sortie, sa prédiction pour le chiffre représenté par l'image.

Ici, on fournit d'abord une image du chiffre 3 au modèle. Le modèle prédit que cette image correspond au chiffre 9, et donc se trompe. Le programme compare cette prédiction à l'étiquette correspondante (3), et quantifie l'erreur commise par le modèle. À partir de cette erreur, le programme adapte l'ensemble des paramètres du modèle pour se rapprocher de la prédiction désirée. Puis il passe aux images suivantes. À la longue, le modèle devient capable de reconnaître de nouveaux chiffres avec précision.

Tutoriel en python

Nous proposons un petit tutoriel dans lequel vous pourrez entraîner vous-même un réseau de neurones à reconnaître des chiffres manuscrits. Tout d'abord on importe le set d'images de chiffres que l'on stocke dans *digits*.

```
[ ] from sklearn import datasets
    digits = datasets.load_digits()
```


- On affiche la première image. Attention ici *digits.images[0]* indique que l'on prend le premier élément de la matrice *digits.images*, il se trouve qu'ici le premier élément est un '0'.

À l'aide de la fonction `print`, on affiche une matrice donnant les valeurs de niveaux de l'image du chiffre en 8x8 pixels (à gauche). À l'aide de `matplotlib`, on affiche sa représentation graphique (à droite).

```
[ ] print(digits.images[0])

[[ 0.  0.  5. 13.  9.  1.  0.  0.]
 [ 0.  0. 13. 15. 10. 15.  5.  0.]
 [ 0.  3. 15.  2.  0. 11.  8.  0.]
 [ 0.  4. 12.  0.  0.  8.  8.  0.]
 [ 0.  5.  8.  0.  0.  9.  8.  0.]
 [ 0.  4. 11.  0.  1. 12.  7.  0.]
 [ 0.  2. 14.  5. 10. 12.  0.  0.]
 [ 0.  0.  6. 13. 10.  0.  0.  0.]]
```

```
[ ] import matplotlib.pyplot as plt
plt.imshow(digits.images[0], cmap='binary')
plt.title(digits.target[0])
plt.axis('off')
plt.show()
```



- Nous souhaitons entraîner un réseau de neurones simple à reconnaître les chiffres dans ces images. Ce réseau va prendre en entrée des tableaux 1D de 8x8=64 valeurs. Nous devons donc convertir nos images 2D en tableaux 1D.

La matrice `x` comprend maintenant les échantillons des chiffres sous forme de vecteurs de 64 valeurs. Ici, on affiche le vecteur correspondant au premier chiffre du set d'échantillon, le '0'.

```
[ ] x = digits.images.reshape((len(digits.images), -1))
# x contient toutes les images en version 1D.
# nous imprimons ici la première, que nous avons déjà vue :
print(x[0])

[ 0.  0.  5. 13.  9.  1.  0.  0.  0.  0. 13. 15. 10. 15.  5.  0.  0.  3.
 15.  2.  0. 11.  8.  0.  0.  4. 12.  0.  0.  8.  8.  0.  0.  5.  8.  0.
  0.  9.  8.  0.  0.  4. 11.  0.  1. 12.  7.  0.  0.  2. 14.  5. 10. 12.
  0.  0.  0.  0.  6. 13. 10.  0.  0.  0.]
```

Le réseau va agir comme une fonction permettant de passer d'un tableau de 64 valeurs en entrée à une valeur en sortie qui est son estimation du chiffre. Les valeurs de sortie sont stockées dans la variable `y`, cela correspond à "la cible".

```
[ ] y = digits.target
```

- Nous décidons de créer un réseau de neurones relativement simple utilisant 15 neurones. Avec le langage python et ses bibliothèques de machine learning, il est aujourd'hui simple et rapide d'entraîner ses propres réseaux de neurones. Par exemple, `scikit-learn`^[1] fournit des outils de machine learning de haut niveau avec simplement deux lignes de code :

```
[ ] from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes=(15,))
```

- Nous allons entraîner ce réseau sur les 1000 premières images de notre set d'échantillons, et réserver les images suivantes pour tester les performances du réseau.

On définit `x_train` comme les 1000 premiers vecteurs de `x` (donc correspondant aux 1000 premières images), et `x_test` comme les vecteurs de `x` mais à partir du millième élément, pour réaliser les tests. De la même manière `y_train` et `y_test` comme les vecteurs de `y` mais à partir du millième élément, pour réaliser les tests.

L'entraînement se fait en une ligne de code : `mlp.fit(x_train, y_train)`

```
[ ] x_train = x[:1000]
    y_train = y[:1000]
    x_test = x[1000:]
    y_test = y[1000:]

[ ] mlp.fit(x_train, y_train)
```

Il est possible de connaître le nombre total d'échantillon de la banque de données à l'aide de la fonction `len` (pour `length`). ici 1780 images sont disponibles.

```
print(len(digits.images))

1797
```

- Nous pouvons maintenant regarder ce que donne le réseau pour les images suivantes, qui n'ont pas été vues par le réseau lors de l'entraînement. Nous réalisons le test pour les 10 premières images de test (`x_test[:10]`) et nous comparons les résultats avec la cible (`y_test[:10]`).

```
[ ] mlp.predict(x_test[:10])

array([1, 4, 0, 5, 3, 6, 9, 6, 1, 7])

[ ] y_test[:10]

array([1, 4, 0, 5, 3, 6, 9, 6, 1, 7])
```

Pour les 10 premières images de test, les estimations sont excellentes !

- Nous pouvons maintenant évaluer le réseau pour toutes les images de test.

Le vecteur `y_pred` contient l'ensemble des prédictions sur les images de test. On calcule le nombre d'images avec erreur en comparant les valeurs estimées (`y_pred`) avec les cibles (`y_test`). Le taux d'erreur s'écrit comme la somme du nombre d'images pour lesquelles il y a une erreur de prédiction, divisée par le nombre total d'images testées.

```
y_pred = mlp.predict(x_test)
error = (y_pred != y_test)

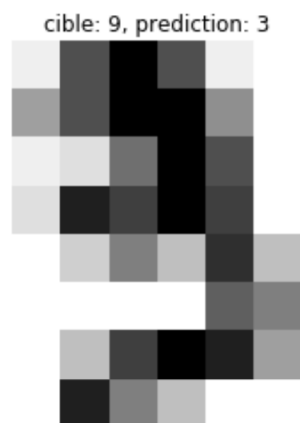
import numpy as np
np.sum(error) / len(y_test)

0.11794228356336262
```

Dans l'exemple présenté ici, on a un taux d'erreur d'environ 11,8%, ce qui signifie que 88,2% des prédictions sont correctes.

Nous pouvons enfin sélectionner les mauvaises prédictions pour les afficher. Ici nous choisissons le 2^{ème} élément dont la prédiction est éronnée (i=1, attention on commence à compter à partir de 0).

```
x_error = x_test[error].reshape((-1, 8,8))
y_error = y_test[error]
y_pred_error = y_pred[error]
i = 1
plt.imshow(x_error[i], cmap='binary')
plt.title(f'cible: {y_error[i]}, prediction: {y_pred_error[i]}')
plt.axis('off')
plt.show()
```



Il est aussi possible d'utiliser notre réseau pour reconnaître de nouveaux chiffres manuscrits.

Dans cet exercice, nous avons utilisé un réseau de neurones extrêmement simple et classifié des images de basse résolution.