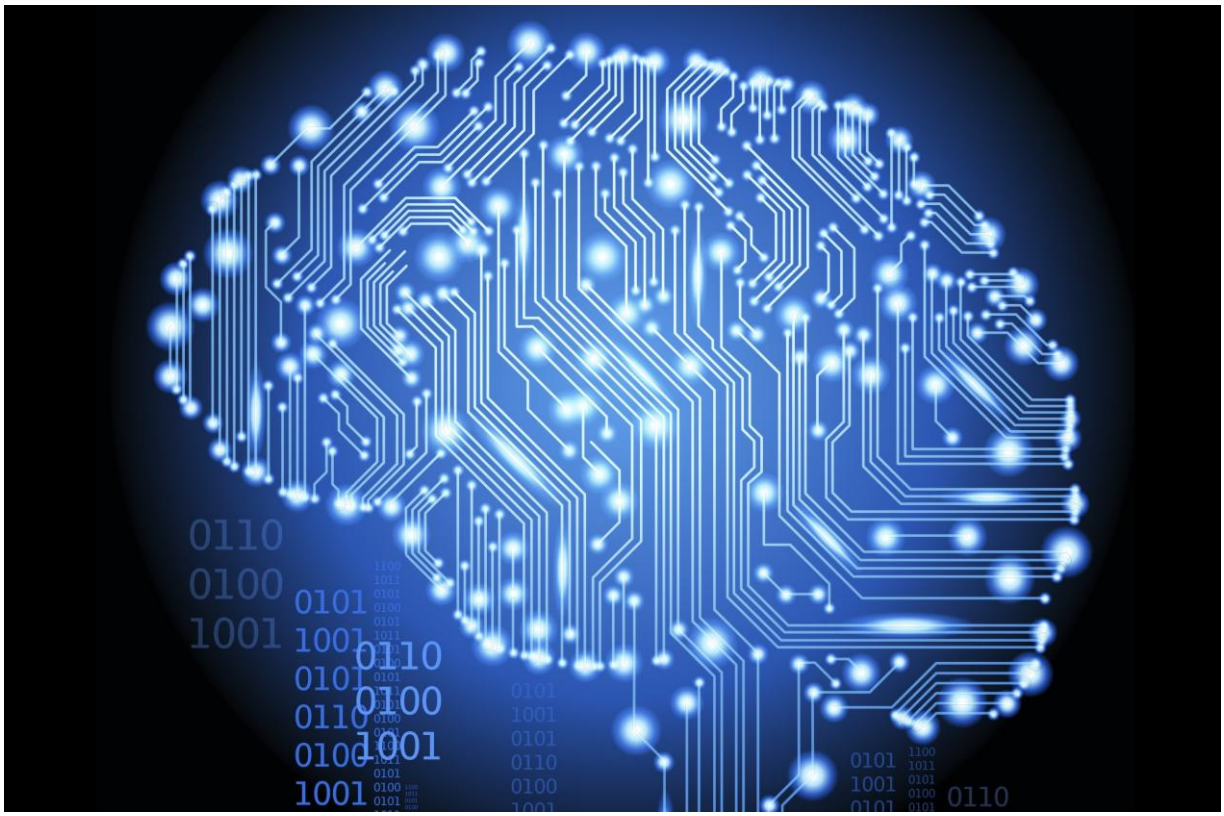
ALFRED COSTES

LYCÉE DES MÉTIERS DE LA COMMUNICATION ET DE L'INDUSTRIE GRAPHIQUE

Chef d'œuvre
Intelligence Artificielle (IA)



Théo OGER
TSN

2023 - 2024

Sommaire :

1) Introduction :

Dans le cadre du BAC Pro option RISC, nous avons choisi d’aborder l’intelligence artificielle comme sujet de projet. Ce projet était sur 2 ans et sera soutenu à la fin de classe de terminale devant un jury. Nous avons choisi ce sujet, car l’intelligence artificielle est aujourd’hui présente dans tous les domaines.

Nous allons donner une définition de l’intelligence artificielle, les bases théoriques, les algorithmes mathématiques qui s’y reportent, son utilisation dans les différents domaines, et pour finir l’utilisation que nous lui avons donné.

Le but de ce projet de chef-d'œuvre est de comprendre le fonctionnement de l'intelligence artificielle afin de pouvoir procéder à la création de notre propre IA. Cette IA permettra de déterminer, à partir d'une photo donnée, si celle-ci représente un chien ou un chat.

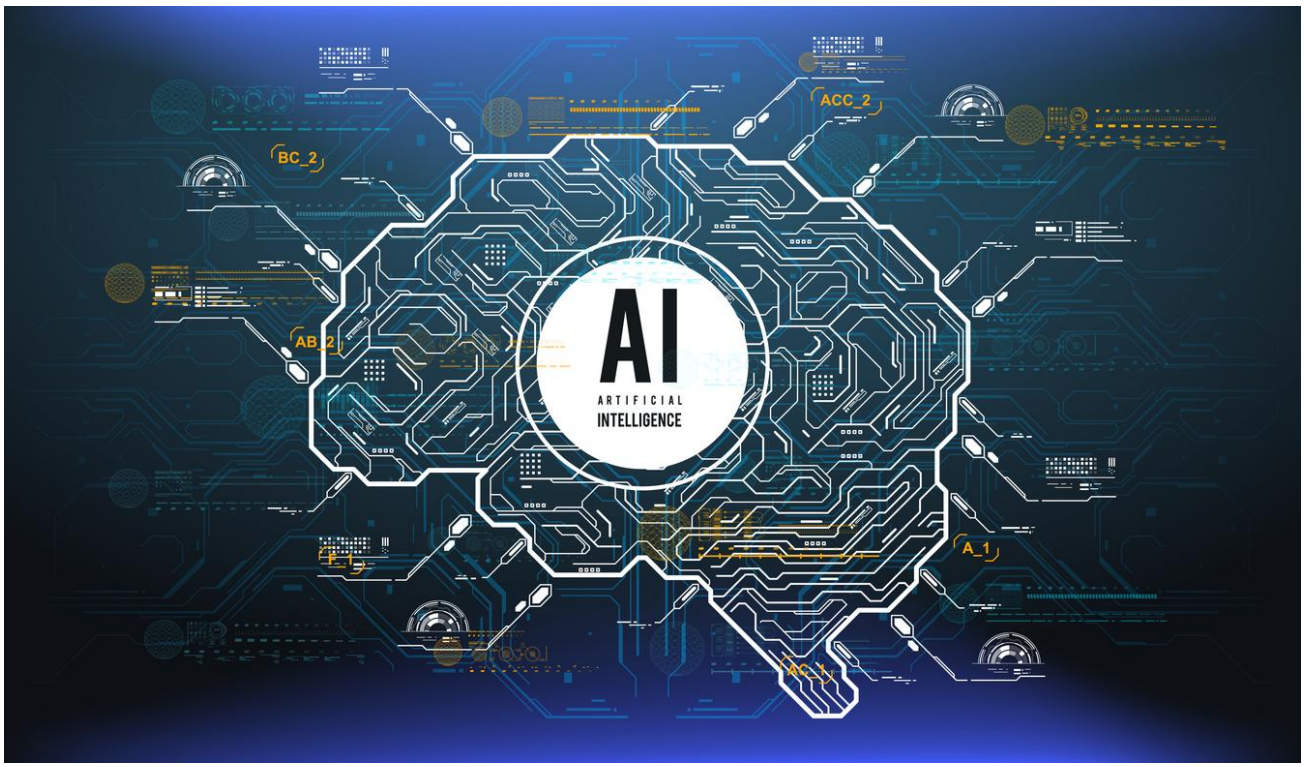
2) C’est quoi l’intelligence artificielle :

En 1950, le mathématicien Alan Turing se posait une question : les machines peuvent-elles penser ? En réalité, cette simple interrogation allait bouleverser le monde.

L’intelligence artificielle est une technologie vaste et révolutionnaire qu’il est difficile d’en donner une définition précise. Il y beaucoup de façons de la décrire, plusieurs définitions peuvent être donné.

On peut considérer qu’il s’agit d’une branche du domaine informatique, ayant pour but la création de machines capables d’effectuer des tâches nécessitant traditionnellement une intelligence humaine.

Pour le Parlement européen, l’intelligence artificielle représente tout outil utilisé par une machine afin de reproduire des comportements liés aux humains, tels que le raisonnement, la planification et la créativité.



3) L’intelligence artificielle (IA) : comment ça marche ?

L’IA avancée se nourrit de grands volumes de données, dont la quantité et la qualité forment le socle de l’efficacité de l’IA. Ses capacités lui permettent ensuite d’extraire de ces données certaines caractéristiques et de les traiter aux fins d’un résultat. Les systèmes d’apprentissage automatique requièrent une part d’intervention humaine afin d’indiquer à la machine les modalités d’extraction des caractéristiques recherchées. Les systèmes d’apprentissage profond, qui relèvent d’un niveau d’IA beaucoup plus perfectionné, sont capables de se former eux-mêmes à extraire et à classer des caractéristiques.

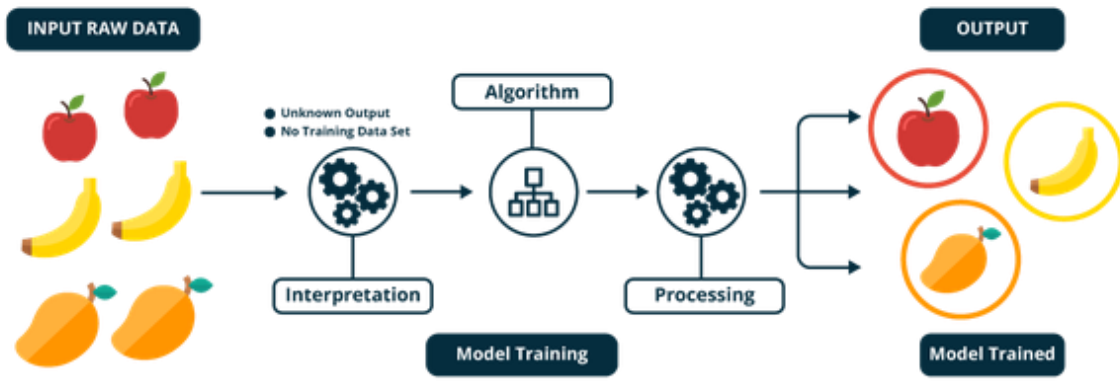
Prenons l’exemple de la conduite autonome. Un véhicule autonome reçoit des données visuelles par l’intermédiaire de caméras, mais s’appuie aussi sur des radars et différentes technologies de détection afin de reconnaître ce qui se passe dans son environnement. En même temps, il reçoit et surveille en permanence les données de suivi de son fonctionnement. L’IA utilise et traite ces données pour déterminer si le scénario auquel le véhicule est confronté nécessite une intervention et transmet le résultat de son analyse au véhicule afin d’assurer un pilotage sûr dans le scénario perçu.

4) Les différents types d’apprentissage de l’IA

Effectivement les IA ne fonctionnent pas tous pareille elles ont différente manière d’apprentissage

L’apprentissage supervisé

L’IA a ainsi fréquemment recours à l’apprentissage supervisé. Par exemple, on « nourrit » un programme avec des milliers de photos de voitures, étiquetées. Après cet « entraînement », le programme peut reconnaître, seul, des voitures de tous types sur les nouvelles images qui lui seront présentées.



Le Machine Learning

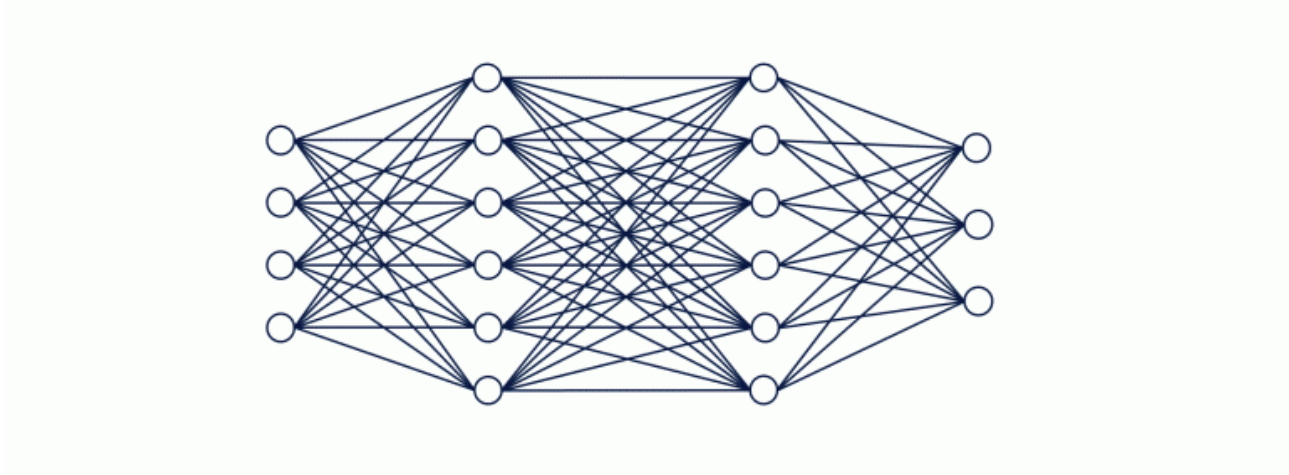
Autre composant de l’intelligence artificielle, le « Machine Learning ». Cette fois, on donne aux ordinateurs l’accès à des données, puis on les laisse apprendre par eux-mêmes, sans intervention humaine ou reprogrammation logicielle. Ce qui leur permet de s’améliorer progressivement, de manière autonome. Et de dépasser ainsi les fonctions et les capacités initialement programmées.
Certains algorithmes ne se contentent plus de reconnaître des images, mais se montrent capables de les produire et de donner des yeux aux machines.

Il y en a une dernière « la plus importante » qu’on va laisser pour un chapitre a part

5) Les réseau de neurones

En intelligence artificielle, les réseaux de neurones sont des assemblages de plusieurs couches de neurones. Il existe des centaines de types de réseaux de neurones différents. Le principe est toujours le même, c'est l'architecture qui est différente à chaque fois.

De manière générale, les réseaux de neurones sont un ensemble de neurones, organisés en plusieurs couches et reliés entre eux. La première couche est appelée *input layer*, le dernier output *layer* et les couches qui se situent entre les deux sont appelées *hidden layers*. Il peut y en avoir un très grand nombre, selon les domaines d'applications.

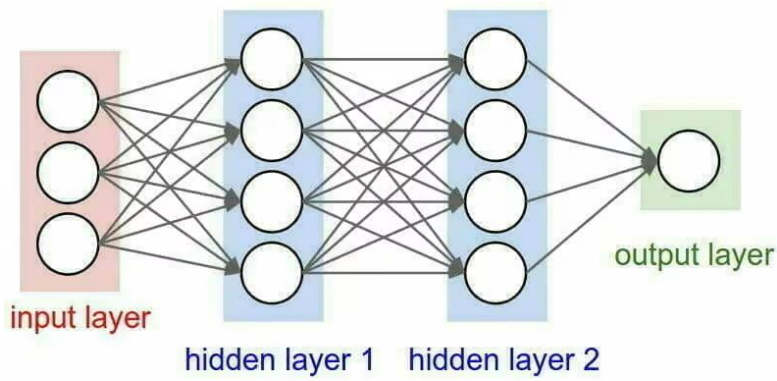


En pratique, les neurones de la première couche vont transmettre des informations simples aux neurones de la couche suivante. Ce qui exercera une influence sur leur degré d'activation et ainsi de suite. La dernière couche nous fournira alors des informations sur le résultat.

6) Le Deep Learning

Deep Learning (apprentissage profond), il repose sur un réseau de neurones artificiels

Voici un exemple ; les neurones sont représentés par des ronds ici.

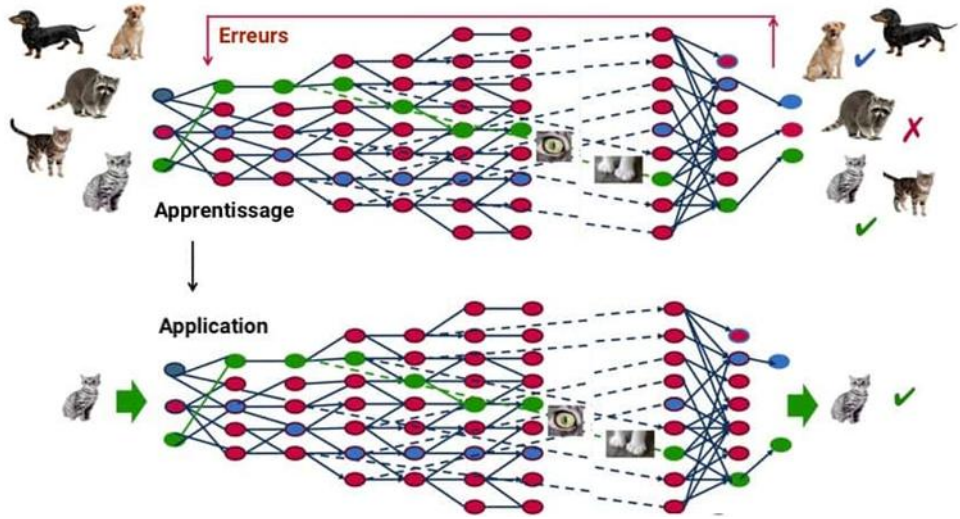


La **couche d'entrée** (input layer) reçoit les données d'entrée. Dans notre cas, nous avons trois neurones dans la couche d'entrée : L'aéroport d'origine, l'aéroport de destination, la date de départ et la compagnie aérienne. La couche d'entrée transmet les données d'entrée à la première couche cachée.

Les **couches cachées** (hidden layers) effectuent des calculs mathématiques sur nos entrées. L'un des défis de la création de réseaux de neurones consiste à décider du nombre de couches cachées, ainsi que du nombre de neurones pour chaque couche.

Ce qui imitent le fonctionnement de notre cerveau. Et ce système crée une machine virtuelle composée de milliers d'unités.

Le ***deep Learning*** s'appuie sur un réseau de neurones artificiels s'inspirant du cerveau humain. Ce réseau est composé de dizaines voire de centaines de « couches » de neurones, chacune recevant et interprétant les informations de la couche précédente. Le système apprendra par exemple à reconnaître les lettres avant de s'attaquer aux mots dans un texte, ou détermine s'il y a un visage sur une photo avant de découvrir de quelle personne il s'agit.


Erreurs
Apprentissage
Application

7) Le Dataset

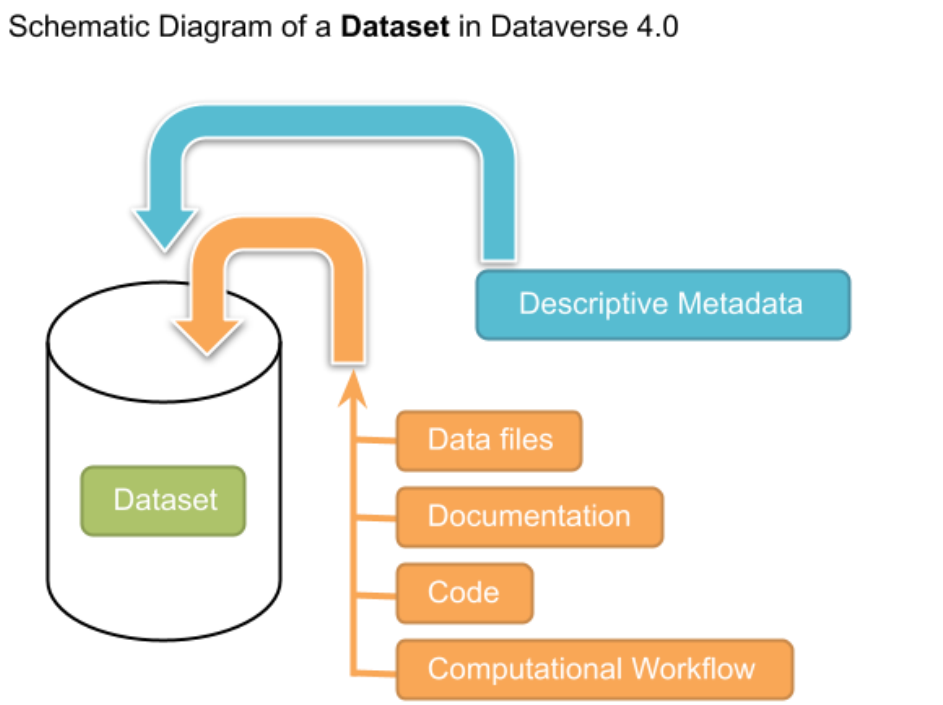
En Machine Learning, tout démarre d'un Dataset qui contient nos données. Dans l'apprentissage supervisé, le Dataset contient les questions (x) et les réponses (y) au problème que la machine doit résoudre. Les datasets (ou jeux de données) sont couramment utilisés en machine learning. Ils regroupent un ensemble de données cohérents qui peuvent se présenter sous différents formats (textes, chiffres, images, vidéos etc…).

Les datasets peuvent être représentés sous différents types, que ce soient des tableaux, des graphes, des arbres ou autres. On travaille **souvent avec des structures de tableaux** dans les algorithmes de machine Learning.

Chaque valeur présente dans un dataset est associée à un **attribut** et à une **observation**.

Prenons par exemple des données sur différentes personnes atteintes ou non du Covid-19.

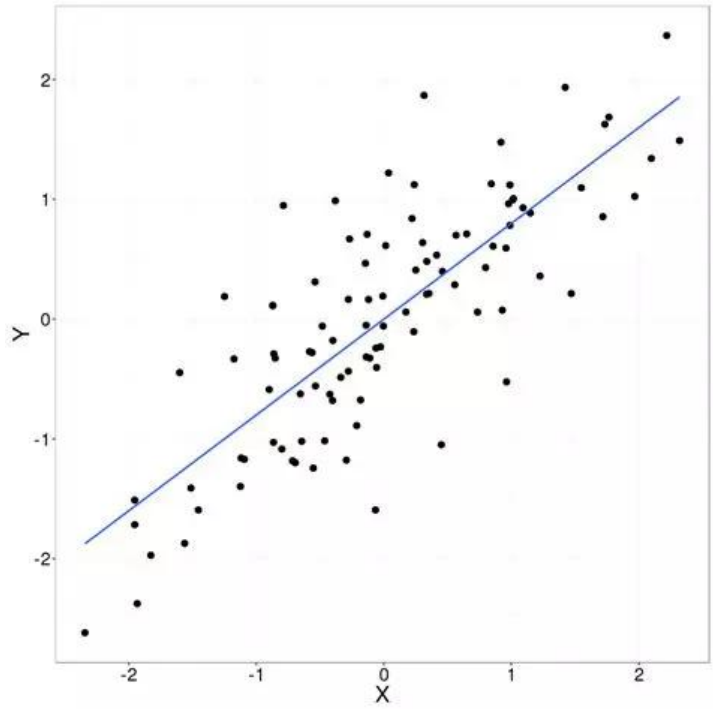
Les attributs correspondront à différentes caractéristiques telles que l'âge, le poids, la taille, la ville de résidence, les symptômes… Alors que chaque observation sera associée à une personne différente.



8) La régression linéaire

La régression linéaire un modèle linéaire $\boldsymbol{f(x) = ax + b}$ est un algorithme d'apprentissage automatique utilisé pour prédire une variable cible numérique à partir de variables indépendantes. Il utilise une droite (ou un hyperplan) pour modéliser la relation entre les variables. La régression linéaire est souvent utilisée pour les tâches de prévision et de modélisation, comme la prédiction des ventes d'un produit en fonction du nombre de clients intéressé ou la prédiction de la consommation d'énergie en fonction de la température. C'est un des algorithmes de base dans l'IA.

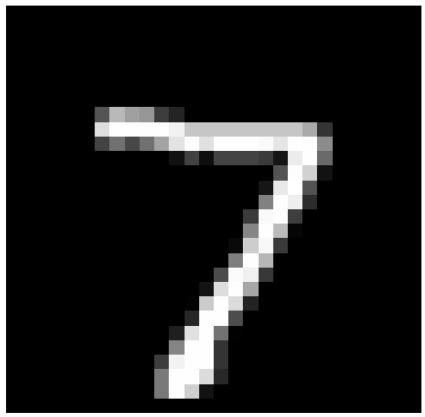
La régression linéaire est utilisée dans l'IA pour modéliser et prédire les relations entre des variables numériques. Il peut être utilisé pour résoudre des tâches de prévision en utilisant des données historiques pour prédire les résultats futurs, ainsi que des tâches de modélisation pour comprendre les relations causales entre différentes variables.



9) Etude de cas :

Pour plonger davantage dans le sujet, le professeur nous a fourni deux codes à recopier, répertoriés en Annexe 1 et 2. La réplication de ces codes nous a permis d'appliquer les diverses connaissances acquises au cours des deux dernières années. L'objectif de ces codes était le suivant : l'Annexe 1 visait à permettre à notre programme de recevoir une photo de chien ou de chat et de la reconnaître grâce à l'apprentissage supervisé vue ultérieurement.

Pour sa part, l'Annexe 2 permet de générer des chiffres de manière peu compréhensible pour ensuite permettre à la machine de faire des prédictions.



```
Prédiction du modèle: 7
Etiquette réelle: 7
```

```
Prédiction du modèle: 2
Etiquette réelle: 2
```

Nous avons débuté notre code en important plusieurs bibliothèques essentielles, telles que TensorFlow, Sequential et mnist. Ces bibliothèques nous ont permis de créer un réseau de neurones et de charger nos données d'entraînement.

Ensuite, nous avons construit notre réseau de neurones en associant les images d'entraînement (x_train) avec leurs étiquettes correspondantes (y_train). Cette étape nous a permis de créer un modèle capable d'apprendre à partir des données fournies.

Nous avons ensuite procédé à l'entraînement du modèle en lui fournissant l'ensemble des images disponibles dans la bibliothèque MNIST. Pendant cette phase, le modèle a appris à associer correctement les images avec les étiquettes correspondantes, afin d'améliorer sa capacité à faire des prédictions précises.

Après l'entraînement, nous avons évalué les performances du modèle en calculant son propre taux de réussite. Cela nous a permis de déterminer à quel point le modèle était capable de prédire correctement les étiquettes des images.

Enfin, nous sommes passés au test du modèle en lui présentant une image au hasard. Le modèle a alors produit une prédiction en se basant sur son entraînement précédent. Nous avons comparé cette prédiction avec l'étiquette réelle de l'image pour évaluer l'exactitude de la prédiction.

Dorénavant cela fonctionnant, j’ai voulu me donner un nouvel exercice plutôt complexe qui était de pouvoirs donné moi-même une image à notre IA et qu’elle la reconnaisse. (Annexe 3)

Pour commencer j’ai ajouté la bibliothèque Cv2 qui me permet de donner ma propre image à notre IA. J’ai donc automatisé la mise de l’image avec le chemin direct de mon ordinateur, convertir l’image (car MNIST ne lis uniquement les images 28 x 28) ainsi qu’un convertisseur d’image en Numpy 2d afin que MNIST puisse reconnaitre l’image.

```
resized_image = cv2.resize(image, (28, 28))


normalized_image = resized_image / 255.0

input_image = np.expand_dims(normalized_image, axis=0)
```

Cependant travaillant sur google collaborate le chemin ne marchais pas. J’ai donc ajouté une nouvel bibliothèque « files » qui allais me permettre de transférer directement des images dans mon programme.

Etant fait il me suffit de rajouter une commande qui me permet d'uploader mes fichiers. Je peux maintenant uploader des fichiers dans mon IA

```
Epoch 1/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2605 - accuracy: 0.9250
Epoch 2/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1109 - accuracy: 0.9671
Epoch 3/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0769 - accuracy: 0.9771
Epoch 4/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0573 - accuracy: 0.9826
Epoch 5/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0451 - accuracy: 0.9863
313/313 [==============================] - 1s 2ms/step - loss: 0.0810 - accuracy: 0.9752
Précision sur l'ensemble de test: 0.9751999974250793
Modèle sauvegardé avec succès.
Modèle chargé avec succès
```

Parcourir... Aucun fichier sélectionné. Cancel upload

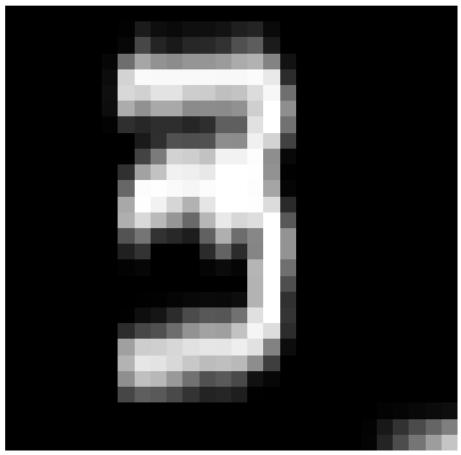
Il me suffit maintenant de dessiner un chiffre sur fond noir et le chiffre en blanc (car l'IA apprend sur un fond noir et une écriture blanche). Transformer mon image en 28 pixels par 28.

```
Modèle sauvegardé avec succès.
Modèle chargé avec succès
```

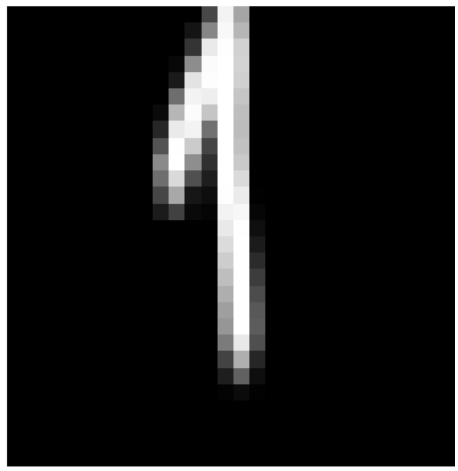
Parcourir... Sans titre.png

Sans titre.png(image/png) - 547 bytes, last modified: n/a - 100% done

```
Saving Sans titre.png to Sans titre (1).png
1/1 [==============================] - 0s 154ms/step
```



```
Prédiction du modèle: 3
```



```
Prédiction du modèle: 1
```

10) Conclusion :

En conclusion, ce projet de chef-d'œuvre nous a plongé au cœur de l'univers passionnant de l'intelligence artificielle. À travers notre exploration des concepts fondamentaux tels que les réseaux de neurones, le deep learning, et les différents types d'apprentissage, nous avons acquis une compréhension approfondie du fonctionnement de cette technologie révolutionnaire.

Nous avons également pris conscience des multiples domaines dans laquelle l'IA peut être utile ainsi que tout le savoir et les étapes permettant à sa création.

En fin de compte, ce projet nous a non seulement permis d'acquérir des connaissances techniques précieuses, mais il nous a également incités à réfléchir aux implications plus larges de l'IA dans notre société.

Annexe 1 :

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
!wget --no-check-certificate -O /tmp/cats_and_dogs_filtered.zip
https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip
import os
import zipfile
local_zip = '/tmp/cats_and_dogs_filtered.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/tmp')
zip_ref.close()
base_dir = '/tmp/cats_and_dogs_filtered'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
train_cats_dir = os.path.join(train_dir, 'cats')
train_dogs_dir = os.path.join(train_dir, 'dogs')
validation_cats_dir = os.path.join(validation_dir, 'cats')
validation_dogs_dir = os.path.join(validation_dir, 'dogs')
train_cat_fnames = os.listdir(train_cats_dir)
train_dog_fnames = os.listdir(train_dogs_dir)
print(train_cat_fnames[:11])
print(train_dog_fnames[:12])
samples_dogs = np.random.choice(os.listdir(train_dogs_dir), size = 12)
plt.figure(figsize = (12, 8))
for i,dog in enumerate(samples_dogs):
        ax = plt.subplot(3,4,i+1)
        img = mpimg.imread(os.path.join(train_dogs_dir,dog))
        plt.axis("off")
        plt.imshow(img)
        ax.set_title(dog)

samples_cats = np.random.choice(os.listdir(train_cats_dir), size = 12)
plt.figure(figsize = (12, 8))
for i,cat in enumerate(samples_cats):
        ax = plt.subplot(3,4,i+1)
        img = mpimg.imread(os.path.join(train_cats_dir,cat))
        plt.axis("off")
        plt.imshow(img)
        ax.set_title(cat)
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
train_datagen = ImageDataGenerator(rescale = 1/255.0)
validation_datagen = ImageDataGenerator(rescale = 1/255.0)
train_generator = train_datagen.flow_from_directory(train_dir, target_size = (150,150), batch_size = 20,
class_mode = 'binary')
validation_generator = validation_datagen.flow_from_directory(train_dir, target_size = (150,150), batch_size =
20, class_mode = 'binary')
import tensorflow as tf
model = tf.keras.models.Sequential([
        tf.keras.layers.Conv2D(32,(3,3), input_shape = (150, 150, 3), activation = 'relu'),
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.Conv2D(64,(3,3), activation = 'relu'),
        tf.keras.layers.MaxPooling2D((2,2)),
        tf.keras.layers.Conv2D(128,(3,3), activation = 'relu'),
        tf.keras.layers.MaxPooling2D((2,2)),
        tf.keras.layers.Conv2D(128,(3,3), activation = 'relu'),
        tf.keras.layers.MaxPooling2D((2,2)),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(512, activation = 'relu'),
        tf.keras.layers.Dense(1, activation = 'sigmoid')
])
model.summary()
model.compile(optimizer = tf.keras.optimizers.RMSprop(learning_rate = 0.001), loss = 'binary_crossentropy',
metrics = ['acc'])
#%tensorflow_version 2.x
import tensorflow as tf
#device_name = tf.test.T4GPU_device_name()
#if device_name != '/device:T4GPU:0':
  # raise SystemError("T4GPU not found")
#print("T4GPU at: {}".format(device_name))
history = model.fit(
        train_generator,
        steps_per_epoch=100,
        epochs=10,
        validation_data=validation_generator,
        validation_steps=50
)
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = len(acc)
import seaborn as sns
sns.set()
```

```
plt.figure(figsize = (12,8))
plt.plot(np.arange(epochs),acc,label = "train acc")
plt.plot(np.arange(epochs),val_acc,label = "validation acc")
plt.title("train acc vs validation acc")
plt.legend()
plt.show()
plt.figure(figsize= (12,8))
plt.plot(np.arange(epochs),loss,label="train loss")
plt.plot(np.arange(epochs),val_loss,label="validation loss")
plt.legend()
plt.show()
datagen = ImageDataGenerator(
        rotation_range = 40,
        width_shift_range=0.2,
        height_shift_range = 0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode="nearest"
)
from tensorflow.keras.preprocessing import image
sample_cat = np.random.choice(os.listdir(train_cats_dir))
img = image.load_img(os.path.join(train_cats_dir, sample_cat), target_size = (150,150))
x = image.img_to_array(img)
x = x[np.newaxis]
datagen_img = datagen.flow(x, batch_size=1)
x.shape
plt.imshow(img)
plt.axis("off")
plt.figure(figsize = (12,6))
for i,batch in enumerate(datagen.flow(x, batch_size = 1)):
    ax = plt.subplot (3,3,i+1)
    plt.imshow(np.squeeze(batch).astype("int64"))
    plt.axis("off")
    if (i == 8):
        break
plt.show()
# @title Titre par défaut
from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(
        rescale = 1/255.0,
        rotation_range = 40,
        width_shift_range=0.2,
```

```
        height_shift_range = 0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode="nearest"
)
test_datagen = ImageDataGenerator(rescale = 1/255.0)
train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size = (150, 150),
        batch_size = 20,
        class_mode = "binary"
)
validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size = (150, 150),
        batch_size = 20,
        class_mode = "binary"
)
model = tf.keras.models.Sequential([
        tf.keras.layers.Conv2D(32,(3,3), input_shape = (150, 150, 3), activation = 'relu'),
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.Conv2D(64,(3,3), activation = 'relu'),
        tf.keras.layers.MaxPooling2D((2,2)),
        tf.keras.layers.Conv2D(128,(3,3), activation = 'relu'),
        tf.keras.layers.MaxPooling2D((2,2)),
        tf.keras.layers.Conv2D(128,(3,3), activation = 'relu'),
        tf.keras.layers.MaxPooling2D((2,2)),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(512, activation = 'relu'),
        tf.keras.layers.Dense(1, activation = 'sigmoid')
])
model.compile(optimizer = tf.keras.optimizers.RMSprop(learning_rate=0.0001),loss="binary_crossentropy",
metrics=["acc"])

history = model.fit_generator(
        train_generator,
        steps_per_epoch=100,
        epochs=50,
        validation_data=validation_generator,
        validation_steps=50
)
acc = history.history['acc']
```

```
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = len(acc)
plt.figure(figsize = (12,8))
plt.plot(np.arange(epochs),acc,label = "train acc")
plt.plot(np.arange(epochs),val_acc,label = "validation acc")
plt.title("train acc vs validation acc")
plt.legend()
plt.show()
plt.figure(figsize= (12,8))
plt.plot(np.arange(epochs),loss,label="train loss")
plt.plot(np.arange(epochs),val_loss,label="validation loss")
plt.legend()
plt.show()
from tensorflow.keras.preprocessing import image
imgname = f'/tmp/cats_and_dogs_filtered/validation/cats/{np.random.choice(os.listdir(validation_cats_dir))}'
print(imgname)
img = image.load_img(imgname, target_size=(150,150))
x = image.img_to_array(img)
x = x[np.newaxis]
x.shape
model.predict(x)
train_generator.class_indices
```

Annexe 2 :

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
import matplotlib.pyplot as plt

(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train, x_test = x_train / 255.0, x_test / 255.0

model = Sequential([
        Flatten(input_shape=(28, 28)),
        Dense(128, activation='relu'),
        Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
                        loss='sparse_categorical_crossentropy',
                        metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)

test_loss, test_acc = model.evaluate(x_test, y_test)
print("Précision sur l'ensemble de test:", test_acc)

model.save('model_mnist.h5')
print("Modèle sauvegardé avec succès.")

trained_model = tf.keras.models.load_model('model_mnist.h5')
print("Modèle sauvegardé avec succès")
num_images = 4
predictions = trained_model.predict(x_test[:num_images])
for i in range(num_images):
    plt.imshow(x_test[i], cmap='gray')
    plt.axis('off')
    plt.show()
    predicted_label = tf.argmax(predictions[i]).numpy()
    true_label = y_test[i]
    print ("Prédiction du modèle:", predicted_label)
    print ("Etiquette réelle:", true_label)
```

Annexe 3 :

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
import matplotlib.pyplot as plt
import cv2
import numpy as np


(x_train, y_train), (x_test, y_test) = mnist.load_data()


x_train, x_test = x_train / 255.0, x_test / 255.0

model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])


model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])


model.fit(x_train, y_train, epochs=5)


test_loss, test_acc = model.evaluate(x_test, y_test)
print("Précision sur l'ensemble de test:", test_acc)


model.save('model_mnist.h5')
print("Modèle sauvegardé avec succès.")


trained_model = tf.keras.models.load_model('model_mnist.h5')
print("Modèle chargé avec succès")
```

```
# Chargez la librairie
from google.colab import files
uploaded = files.upload()

# Récupérer le nom de fichier de l'image téléchargée
image_path = list(uploaded.keys())[0]

# Lire l'image téléchargée
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

#pixel 2_ par 28
resized_image = cv2.resize(image, (28, 28))

#adapter l'image a NMIST
normalized_image = resized_image / 255.0

#la mettre sous forme de tableau python
input_image = np.expand_dims(normalized_image, axis=0)

# Faire une prédiction avec le modèle
predictions = trained_model.predict(input_image)<

# Obtenir l'étiquette prédite
predicted_label = np.argmax(predictions)

# Afficher l'image et la prédiction
plt.imshow(resized_image, cmap='gray')
plt.axis('off')
plt.show()
print("Prédiction du modèle:", predicted_label)
```

Deuxième partie

Questions réponses les bases de l'IA

1) Qu'est-ce que le machine Learning

Le Machine Learning consiste à laisser l'ordinateur apprendre quel calcul effectuer, plutôt que de lui donner ce calcul (c'est-à-dire le programme de façon explicite). C'est en tout cas la définition du Machine Learning selon son inventeur Arthur Samuel, un mathématicien américain qui a développé un programme pouvant apprendre tout seul comment jouer aux Dames en 1959.

2)Quelles sont les différentes méthodes d'apprentissage machine

• L'apprentissage supervisé (Supervised Learning)

• L'apprentissage non supervisé (Unsupervised Learning)

• L'apprentissage par renforcement (Reinforcement

Learning)

3) Qu'est-ce que l'apprentissage supervisé

Imaginez que vous commenciez à apprendre le chinois. Pour ce faire, il vous faudra soit acheter un livre de traduction chinois- français, ou bien trouver un professeur de chinois. Le rôle du professeur ou du livre de traduction sera de superviser votre apprentissage en vous fournissant des exemples de traductions français- chinois que vous devrez mémoriser. On parle ainsi d'apprentissage supervisé lorsque l'on fournit à une machine beaucoup d'exemples qu'elle doit étudier.

4) Donner les 4 notions essentielles qu'il faut maitriser pour mettre en place l'apprentissage supervisé

Pour maîtriser l’apprentissage supervisé, il faut absolument comprendre et connaitre les 4 notions suivantes :

• Le Dataset

• Le Modèle et ses paramètres

• La Fonction Coût

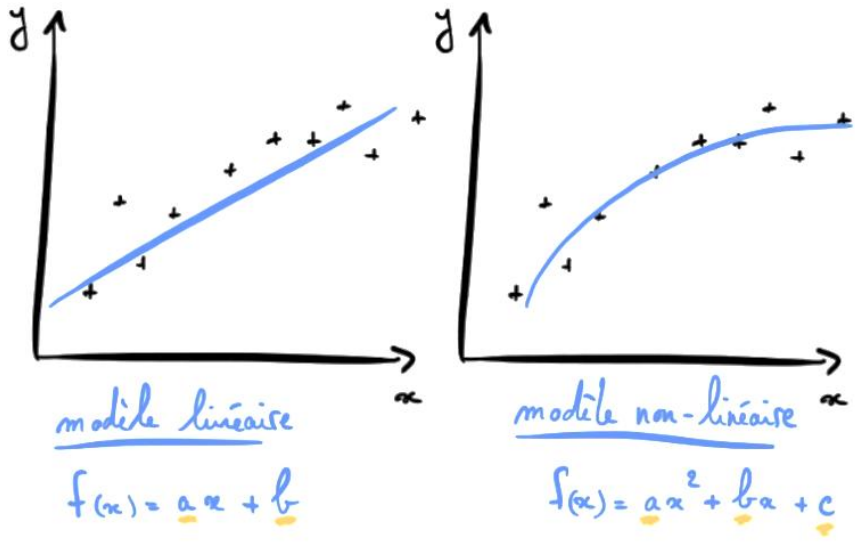
• L’Algorithme d’apprentissage

5)Qu’est-ce que le Dataset

Un "dataset" ou ensemble de données est une collection organisée de données numériques. Ces données peuvent être de divers types, comme des chiffres, du texte, des images, des vidéos, etc. Les datasets sont utilisés dans des domaines tels que l'apprentissage automatique, la recherche scientifique, l'analyse de données et la visualisation pour effectuer des analyses, des entraînements de modèles, des recherches, des études, etc. Chaque enregistrement dans un dataset contient des caractéristiques spécifiques, et les datasets peuvent varier en taille, de petits ensembles à des millions voire des milliards de données.

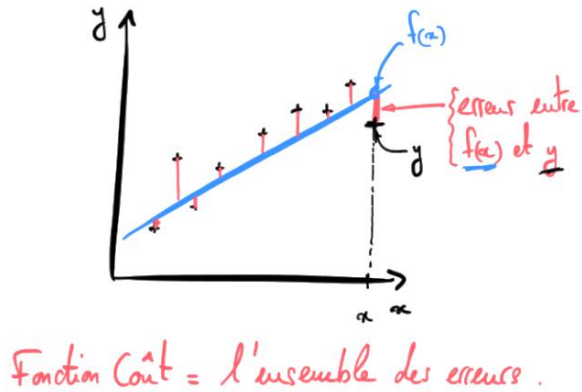
6) Qu’est-ce qu’un modèle et donner le nom de 2 modéles utilisé en Machine learning.

En Machine Learning, on développe un modèle à partir de ce Dataset. Il peut s’agir d’un modèle linéaire comme vous pouvez le voir à gauche, ou bien un modèle non-linéaire comme vous pouvez le voir à droite. Nous verrons dans ce livre comment choisir un modèle plutôt qu’un autre.

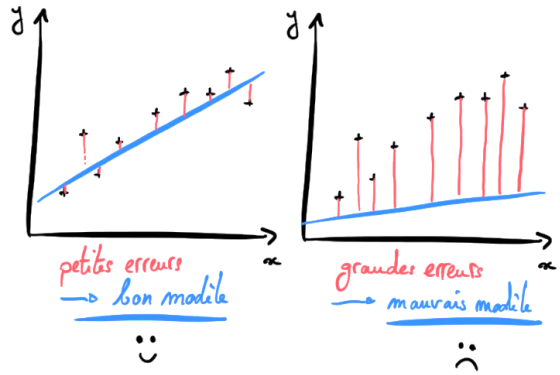


7)Qu’est-ce que la fonction coût (erreur de modèle)

Autre chose à noter est qu’un modèle nous retourne des erreurs par rapport à notre Dataset. On appelle Fonction Coût l’ensemble de ces erreurs



Allons droit au but : Avoir un bon modèle, c’est avoir un modèle qui nous donne de petites erreurs, donc une petite Fonction Coût.



8) Qu’est-ce que l’algorithme d’apprentissage ? Donnez le nom de l’algorithme le plus utilisé

Ainsi l’objectif central en Supervised Learning, c’est de trouver les paramètres du modèle qui minimisent la Fonction Coût. Pour cela, on utilise un algorithme d’apprentissage, l’exemple le plus courant étant l’algorithme de Gradient Descent, que vous apprendrez dans le chapitre

9) Donnez le nom de 2 types de problémes que l’on peut résoudre avec l’apprentissage supervisé

Les applications du Supervised Learning Avec le Supervised Learning on peut développer des modèles pour résoudre 2 types de problèmes : Les problèmes de Régression Les problèmes de Classification Dans les problèmes de régression, on cherche à prédire la valeur d’une variable continue, c’est-à-dire une variable qui peut prendre une infinité de valeurs. Par exemple : Prédire le prix d’un appartement (z) selon sa surface

10) Pour minimiser la fonction coût (erreur), on utilise un algorithme. Donnez le nom de cet algorithme

L'algorithme couramment utilisé en intelligence artificielle pour minimiser la fonction de coût s'appelle la "descente de gradient".

11)Expliquer brièvement l’algorithme de la descente gradients

C'est une méthode d'optimisation qui ajuste les paramètres d'un modèle pour réduire la fonction de coût, améliorant ainsi les performances du modèle. Il existe différentes variantes de la descente de gradient, notamment la descente de gradient stochastique (SGD) et d'autres méthodes d'optimisation avancées.

12) Qu’est-ce que le problème d’optimisation ? Le Gradient ? Donner la formule de « gradient Descent », (mettre les réponses, mettre le programme python de régression linéaire,

En intelligence artificielle (IA) et en apprentissage automatique (machine learning), le problème d'optimisation se pose fréquemment lorsqu'il s'agit de trouver les meilleurs paramètres d'un modèle ou d'un algorithme afin de minimiser ou maximiser une certaine fonction de coût, souvent appelée fonction objective. Le but de l'optimisation est de régler ces paramètres de manière à obtenir les performances les plus élevées pour une tâche donnée.

Le gradient est un concept clé en optimisation. Il s'agit d'un vecteur qui indique la direction et le taux de variation le plus rapide d'une fonction par rapport à ses paramètres. En d'autres termes, le gradient montre comment la fonction évolue lorsque les paramètres sont modifiés. Pour une fonction à plusieurs variables, le gradient est un vecteur contenant les dérivées partielles de la fonction par rapport à chaque paramètre.

La descente de gradient (Gradient Descent en anglais) est une méthode d'optimisation couramment utilisée pour trouver les paramètres optimaux d'un modèle. La formule de la descente de gradient est la suivante :

) $\theta i+1=\theta i-\alpha\nabla J(\theta i)$

Où :

La descente de gradient commence généralement avec une initialisation aléatoire ou déterminée des paramètres, puis elle itère en ajustant ces paramètres dans la direction opposée du gradient jusqu'à ce qu'une condition d'arrêt soit satisfaite (par exemple, un nombre maximal d'itérations ou une convergence suffisante).

Voici un exemple de programme Python pour effectuer une régression linéaire simple en utilisant la descente de gradient :

```
Import numpy as np


# Données d'entraînement

X = np.array([1, 2, 3, 4, 5])

Y = np.array([2, 4, 5, 4, 5])
```

```
# Initialisation des paramètres
Theta0 = 0.0 # Intercept
Theta1 = 0.0 # Coefficient

# Hyperparamètres
Alpha = 0.01     # Taux d'apprentissage
num_iterations = 1000

# Boucle d'entraînement
for _ in range(num_iterations):
        # Calcul des prédictions
        y_pred = theta0 + theta1 * X

        # Calcul du gradient
        gradient_theta0 = (1/len(X)) * np.sum(y_pred - y)
        gradient_theta1 = (1/len(X)) * np.sum((y_pred - y) * X)

        # Mise à jour des paramètres
        theta0 -= alpha * gradient_theta0
        theta1 -= alpha * gradient_theta1

# Affichage des paramètres appris
print("Intercept (theta0):", theta0)
print("Coefficient (theta1):", theta1)
```

Ce code effectue une régression linéaire en utilisant la descente de gradient pour trouver les paramètres qui minimisent l'erreur quadratique moyenne entre les prédictions du modèle et les données d'entraînement.

13) Pour le programme régression linéaire, voir l’exemple page 38

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make_regression

from sklearn.linear_model import SGDRegressor

  np.random.seed(0)

x, y = make_regression(n_samples=100, n_features=1, noise=10)

plt.scatter(x, y)
```

Question n°14) Pour le programme régression linéaire à plusieurs variables, voir l’exemple page 43

```
from sklearn.preprocessing import PolynomialFeatures

np.random.seed(0)
# création du Dataset
x, y = make_regression(n_samples=100, n_features=1, noise=10)
y = y**2 # y ne varie plus linéairement selon x !
# On ajoute des variables polynômiales dans notre dataset
poly_features = PolynomialFeatures(degree=2, include_bias=False)
x = poly_features.fit_transform(x)
plt.scatter(x[:,0], y)
x.shape # la dimension de x : 100 lignes et 2 colonnes


# On entraine le modele comme avant ! rien ne change !
model = SGDRegressor(max_iter=1000, eta0=0.001)
model.fit(x,y)
```

```
print('Coeff R2 =', model.score(x, y))
plt.scatter(x[:,0], y, marker='o')
plt.scatter(x[:,0], model.predict(x), c='red', marker='+')
```

Question n°15) Expliquer brièvement le problème de régression logistique (pages 48-49)

Dans ce genre de problème, on aura un Dataset contenant une variable target $\boldsymbol{y}$ pouvant prendre 2 valeurs seulement, par exemple 0 ou 1

- • si $y = 0$, alors l'email n'est pas un spam
- • si $y = 1$, alors l'email est un spam

On dit également que l'on a **2 classes**, c'est une classification **binaire**.
Pour ces problèmes, on ajoute au modèle une **frontière de décision** qui permet de classer un email dans la $classe$ 0 ou la $classe$ 1.

Question n°16) Quel est le nom et l'équation de la fonction logistique adapté à ce genre de problème ?

En bref, la fonction logistique (ou fonction sigmoïde) est utilisée dans les problèmes de classification binaire, comme la classification d'emails en spam ou non-spam. Son équation est la suivante :

$$

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

$$

Elle prend en entrée une valeur z, qui est une combinaison linéaire des caractéristiques de l'échantillon, et renvoie une valeur entre 0 et 1, interprétée comme la probabilité que l'échantillon appartienne à la classe 1 (par exemple, spam) par opposition à la classe 0 (par exemple, non-spam). Plus la valeur de σ(z) est proche de 1, plus la probabilité d'appartenir à la classe 1 est élevée, et plus elle est proche de 0, plus la probabilité d'appartenir à la classe 0 est élevée. Cette fonction est

largement utilisée dans la régression logistique et les réseaux de neurones pour résoudre des problèmes de classification binaire.

Question n°17) Pour le programme de classification voir pages 53-54

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make_classification

from sklearn.linear_model import SGDClassifier

# Génération de données aléatoires: 100 exemples, 2 classes, 2 features x0 et x1

np.random.seed(1)

X, y = make_classification(n_samples=100,n_features=2, n_redundant=0, n_informative=1,

n_clusters_per_class=1)

# Visualisation des données

plt.figure(num=None, figsize=(8, 6))

plt.scatter(x[:,0], x[:, 1], marker = 'o', c=y, edgecolors='k')

plt.xlabel('X0')

plt.ylabel('X1')

x.shape
```

Question n°18) Quel est l’algorithme de Nearest Neighbour ?

L'algorithme des plus proches voisins (k-NN, pour k-Nearest Neighbors en anglais) est un algorithme de classification et de régression. Il fonctionne comme suit :

1. **Étape d'apprentissage** :
    - L'algorithme mémorise l'ensemble des données d'entraînement, qui est constitué de paires (caractéristiques, étiquette).
    - Il ne construit pas de modèle mathématique complexe lors de cette phase. Au lieu de cela, il se souvient simplement des données d'entraînement.

2. **Étape de prédiction (classification)** :
    - Lorsqu'un nouvel exemple doit être classifié, l'algorithme calcule la distance entre cet exemple et tous les exemples d'entraînement.
    - Il sélectionne les "k" exemples d'entraînement les plus proches en termes de distance, où "k" est un nombre prédéfini (un hyperparamètre).
    - Il effectue un vote majoritaire parmi les "k" exemples les plus proches pour déterminer la classe à attribuer à l'exemple de test. La classe la plus fréquemment présente parmi les voisins est attribuée à l'exemple de test.
3. **Étape de prédiction (régression)** :

Pour la régression, au lieu de prédire une classe, l'algorithme prédit une valeur numérique en prenant la moyenne des étiquettes des "k" exemples d'entraînement les plus proches.

L'algorithme k-NN est simple à comprendre et à implémenter. Cependant, il a des limitations, notamment sa sensibilité à la valeur de "k" et sa dépendance à la structure des données. Le choix de la mesure de distance (par exemple, la distance euclidienne) peut également influencer ses performances. Il est souvent utilisé pour des ensembles de données de petite à moyenne taille et peut être sensible à la dimensionnalité élevée.

Question n°19) Voir le programme page 56-59-60 pour l'algo de K-NN

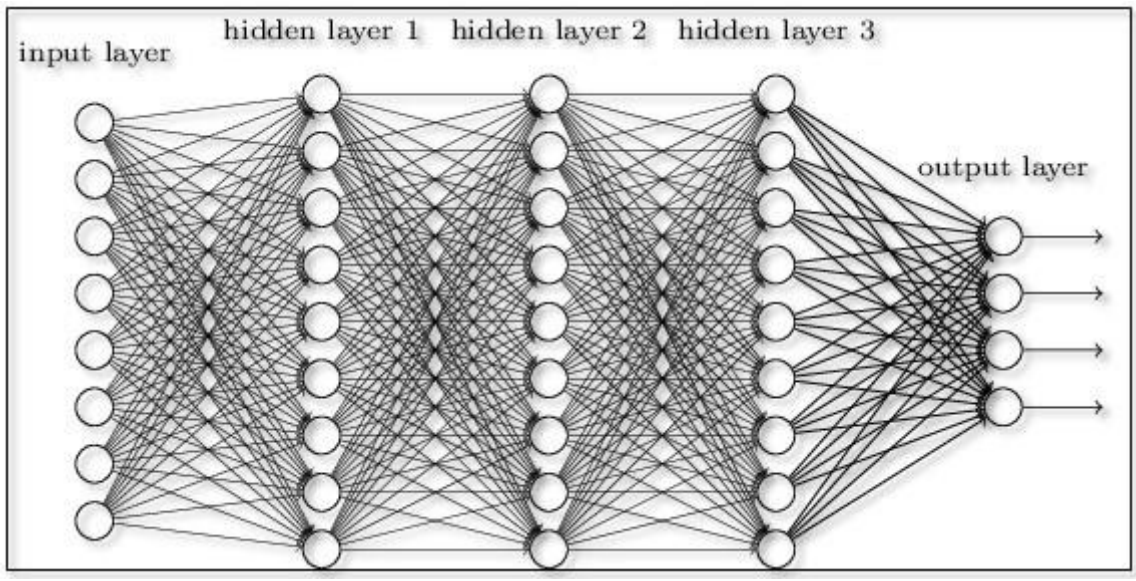
```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load_digits

from sklearn.neighbors import KneighborsClassifier

# importons une base de données de chiffre

digits = load_digits()

X = digits.data

y = digits.target

print('dimension de X:', X.shape)

# visualisons un de ces chiffres

plt.imshow(digits['images'][0], cmap = 'Greys_r')
```

```
# Entraînement du modele
model = KNeighborsClassifier()
model.fit(X, y)
model.score(X, y)
#Test du modele
test = digits['images'][100].reshape(1, -1)
plt.imshow(digits['images'][100], cmap = 'Greys_r')
model.predict(test)
```

Question n°20) Qu’est-ce que les réseaux de neurones (page 62)

Les Réseaux de Neurones sont des modèles bien plus complexes que tous les autres modèles de Machine Learning dans le sens où ils représentent des fonctions mathématiques avec des millions de coefficients (les paramètres). Rappelez-vous, pour la régression linéaire nous n’avions que 2 coefficients $a \; et \; b$… Avec une telle puissance, il est possible d’entraîner la machine sur des tâches bien plus avancées : La reconnaissance d’objets et reconnaissance faciale L’analyse de sentiments L’analyse du langage naturel La création artistique Etc Cependant, développer une fonction aussi complexe à un coût. Pour parvenir, il faut souvent fournir : Un Dataset beaucoup plus grand (des millions de données) Un temps d’apprentissage plus long (parfois plusieurs jours) Une plus grande puissance de calcul.

Question n°21) Décrie les réseaux de neurones en détailles avec schéma (page 63)

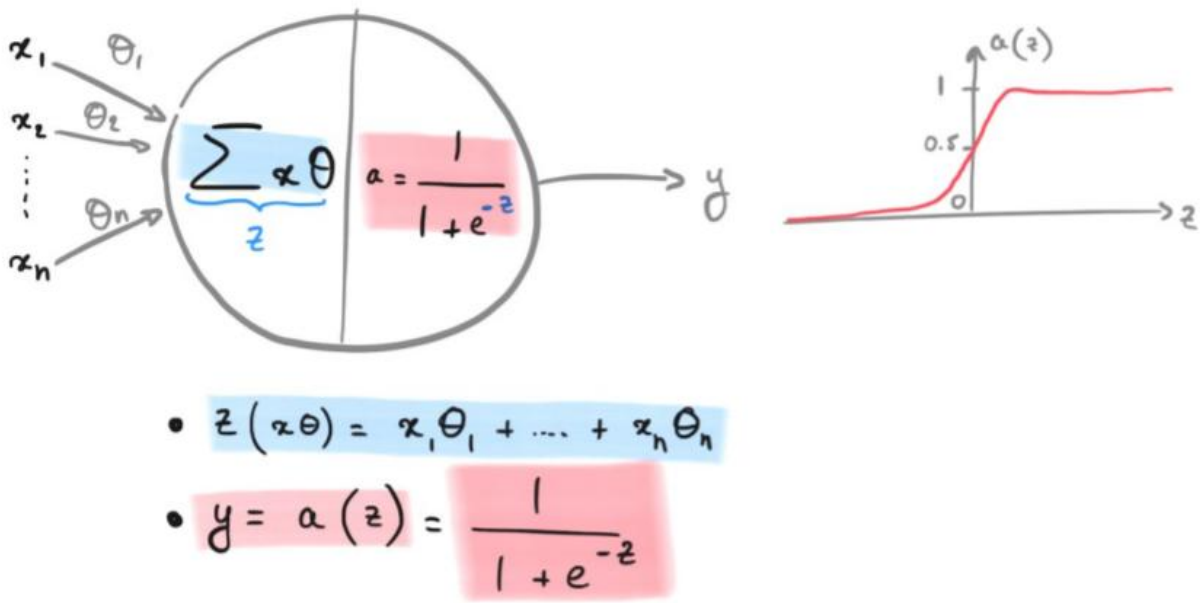


Question n°22) Qu’est-ce qu’un perceptron ?

Vous remarquez un niveau d’entrées (input layer) à gauche, un niveau de sorties (output layer) à droite, et plusieurs niveaux cachés entre deux. Les petits ronds sont appelés les neurones et représentent des fonctions d’activation. Pour un réseau de neurone basique, la fonction Logistique est utilisée comme fonction d’activation.

Question n°23) Décrie en détailles le fonctionnement d’un perceptron (schéma, etc) (page 64)

Un perceptron est un modèle de neurone artificiel qui prend des valeurs d'entrée, les pondère, les somme, puis applique une fonction d'activation pour produire une sortie. Les poids et le biais sont ajustés pendant l'apprentissage pour permettre au perceptron d'apprendre à effectuer des classifications ou des prédictions précises. Il s'agit d'une unité de base utilisée dans les réseaux de neurones plus complexes pour la résolution de tâches d'apprentissage automatique.



Question n°24) Qu’est-ce que le Deep Learning ? (Pages 64-65)

le Deep Learning est une sous-catégorie de l'apprentissage automatique (machine learning) qui se concentre sur l'utilisation de réseaux de neurones artificiels profonds (d'où le "deep" dans le nom) pour résoudre des tâches complexes d'apprentissage automatique. Ces réseaux de neurones profonds sont capables d'apprendre des représentations hiérarchiques des données, ce qui leur permet de modéliser des relations complexes et de réaliser des prédictions ou des classifications précises dans divers domaines, tels que la vision par ordinateur, le traitement du langage naturel et bien d'autres. Le Deep Learning a connu un grand succès ces dernières années grâce à l'augmentation de la puissance de calcul, de la quantité de données disponibles et des avancées algorithmiques, ce qui a permis de résoudre des problèmes qui étaient auparavant difficiles à aborder.

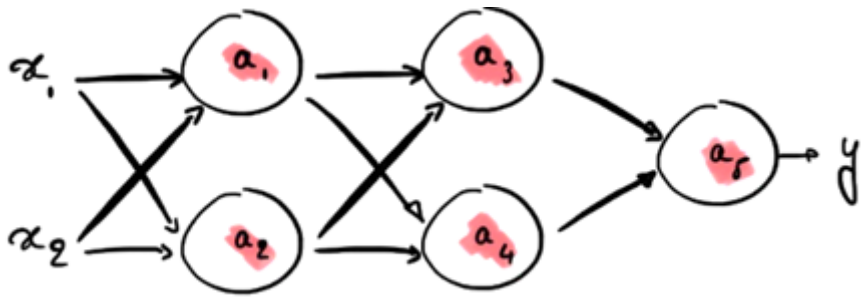
Question n°25) Décrie le fonctionnement d’un réseau profond (page 65)

On réunit les neurones en colonne (on dit qu'on les réunit en couche, en layer). Au sein de leur colonne, les neurones ne sont pas connectés entre eux.

On connecte toutes les sorties des neurones d'une colonne à gauche aux entrées de tous les neurones de la colonne

De droite qui suit. On peut ainsi construire un réseau avec autant de couches et de neurones que l'on veut. Plus il y a de couches, plus on dit que le réseau est profond (deep) et plus le modèle devient riche, mais aussi difficile à entraîner. C'est ça, le Deep Learning.

Question n°26) Donner le schéma d'un réseau de neurone à 5 neurones (3 couches) (page 65)



Question n°27) Qu'est ce qu'une couche cachée dans un réseau de neurone (page 65)

une couche cachée dans un réseau de neurones est une couche intermédiaire entre la couche d'entrée et la couche de sortie, responsable de l'extraction et de la transformation des caractéristiques des données d'entrée. Les neurones dans cette couche sont connectés aux neurones des couches précédentes et suivantes, et les poids de ces connexions sont ajustés lors de l'apprentissage du réseau pour permettre au réseau de modéliser des relations complexes dans les données.

Question n°28) Donner une nouvelle définition d'un réseau de neurone (page 67)

Un Réseau de Neurone n’est en fait qu’une énorme composée de milliers de fonctions mathématiques, et aujourd’hui les neuroscientifiques ont démontré que le fonctionnement du cerveau dépasse de loin l’architecture « simpliste » des réseaux de neurones.

Question n°29) Rappeler les 4 éléments essentielles (étapes) pour résoudre un problème de supervised learning (apprentissage supervisé) (page 67)

Pour le Dataset, pas de problème, il suffit de disposer d’un tableau ($\boldsymbol{X}$, $\boldsymbol{y}$) comme pour les autres problèmes. Les features ($\boldsymbol{x1}$, $\boldsymbol{x2}$, $\boldsymbol{x3}$, …) sont distribuées à l’entrée du réseau (dans le premier layer) Pour programmer le Modèle, il faut emboîter les fonctions des différent niveaux d’activation les unes dans les autres comme je l’ai montré plus haut pour l’exemple des 3 neurones. C’est ce qu’on appelle Forward Propagation (faire le chemin des entrée $\boldsymbol{X}$ vers la sortie $\boldsymbol{y}$). Pour exprimer la Fonction Coût et son gradient, c’est mathématiquement délicat. Il faut calculer la contribution de chaque neurone dans l’erreur finale. Tout ce que vous avez à savoir, c’est que cela est possible avec une technique appelée Back Propagation (faire le chemin dans le sens inverse : $\boldsymbol{y}$ vers $\boldsymbol{X}$). Enfin, pour minimiser la Fonction Coût, il suffit d’utiliser Gradient Descent en utilisant les gradients calculés avec Back Propagation. Le Gradient Descent en lui-même n’est pas différent de celui que nous avons vu dans le chapitre 2 (bonne nouvelle).

Question n°30) Voir le programme de réseau de neurones d’identification des espèces d’Iris (pages 68)

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load_iris

from sklearn.neural_network import MLPClassifier

# charger les données

iris = load_iris()
```

```
X = iris.data
y = iris.target
X.shape # notre Dataset comprend 150 exemples et 4 variables
# Visualisation des donées
colormap=np.array(['Red','green','blue'])
plt.scatter(X[:,3], X[:,1], c = colormap[y])
    hidden_layer_sizes=(10, 10, 10)
# Création du modele
model = MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=1000)
model.fit(X, y)
model.score(X, y)
```