

# Texte mining par la pratique

Redha Moulla

Paris, 14 - 16 octobre 2024

# Plan de la formation

- Introduction au NLP
- Techniques statistiques pour le NLP
- Machine learning pour le NLP
- Deep learning pour le NLP
- Transformers et LLMs
- Alignement des LLMs

# Introduction au NLP

# 1956 : conférence de Dartmouth

L'histoire du NLP est intimement liée à celle de l'intelligence artificielle. Il figure même dans le programme de la conférence de Dartmouth, qui a fondé l'IA.

AI Magazine Volume 27 Number 4 (2006) (c) AAAI  
Articles

**A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence**

August 31, 1955

John McCarthy, Marvin L. Minsky,  
Nathaniel Rochester,  
and Claude E. Shannon

■ The 1956 Dartmouth summer research project on artificial intelligence was initiated by this August 31 proposal, submitted jointly by John McCarthy, Marvin L. Minsky, Nathaniel Rochester, and Claude Shannon. The original typescript document of 11 pages has been preserved in the collection of the papers of John McCarthy, located in the archives at Dartmouth College and available online at [www.cs.cmu.edu/~mcarthy/](http://www.cs.cmu.edu/~mcarthy/). The proposal, and the remaining pages give qualifications and plans for the Dartmouth summer study. In the interest of brevity, this article omits almost only the proposed itself, along with the short autobiographical comments of the proposers.

We propose that a 2-month, 10-man study of artificial intelligence be carried out during the summer of 1956 at Dartmouth College in Hanover, New Hampshire. The purpose of the study will be to test the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves. We think that a significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer.

1. Automatic Computers

If a machine can do a job, then an automatic computer can probably be designed to simulate the machine. The speeds and memory capacities of present computers are not yet sufficient to simulate many of the higher functions of the human brain, but the major obstacle is not lack of memory capacity, but rather the lack of programs taking full advantage of what we know.

2. How Can a Computer be Programmed to Use a Language

It may be speculated that a large part of human thought consists of manipulating words according to rules of reasoning and rules of conjecture. From this point of view, forming a generalization consists of advertising a new

*"We propose that a 2-month, 10-man study of artificial intelligence be carried out during the summer of 1956 at Dartmouth College in Hanover, New Hampshire. The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves. We think that a significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer."*

# Évolution du NLP

L'histoire du NLP peut être divisée en plusieurs phases clés, allant des approches symboliques aux révolutions récentes avec les modèles de langage géants (LLMs).

- **Années 1950 - 1980 : Approches symboliques**
- **Années 1990 - 2010 : Approches statistiques**
- **2010 - 2018 : Deep Learning et réseaux de neuronaux "classiques"**
- **2018 - Présent : L'ère des Large Language Models (LLMs)**

# Approches basées sur la grammaire formelle

Les approches basées sur la grammaire formelle étaient très utilisées dans les débuts du traitement automatique du langage naturel (NLP) pour analyser la structure syntaxique des phrases.

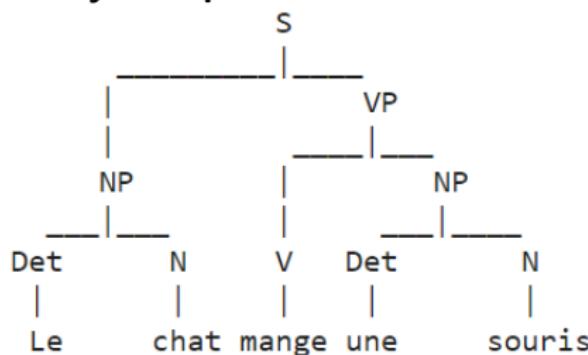
- **Grammaire formelle** : Un ensemble de règles définissant comment former des phrases grammaticalement correctes dans une langue.
- Ces approches étaient souvent utilisées pour le **parsing** (analyse syntaxique), c'est-à-dire la construction d'arbres syntaxiques représentant la structure d'une phrase.
- Le parsing peut être :
  - **Descendant (top-down)** : La phrase complète est décomposée en groupes syntaxiques successifs.
  - **Ascendant (bottom-up)** : Les mots sont progressivement regroupés pour former la structure syntaxique complète.

# Exemple de parsing de phrase

Le parsing consiste à analyser la structure syntaxique d'une phrase en construisant un arbre qui montre la relation entre les différents composants de la phrase.

Phrase : "Le chat mange une souris."

## Arbre syntaxique :



## Détails :

- **S** : Phrase complète (Sentence)
- **NP** : Groupe nominal (Noun Phrase)
- **VP** : Groupe verbal (Verb Phrase)
- **Det** : Déterminant (Determinant)
- **N** : Nom (Noun)
- **V** : Verbe (Verb)

# Chatbot ELIZA

**ELIZA** est un des premiers programmes de traitement automatique du langage naturel, développé par Joseph Weizenbaum en 1966. Il est basé sur des règles simples et des mots-clés pour imiter un thérapeute rogerien.

## Exemple de conversation avec **ELIZA** :

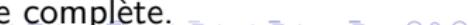
- **Utilisateur** : Je suis très triste aujourd'hui.
- **ELIZA** : Pourquoi êtes-vous triste aujourd'hui ?
- **Utilisateur** : Je me sens seul.
- **ELIZA** : Pourquoi vous sentez-vous seul ?
- **Utilisateur** : Mon père ne m'aime pas.
- **ELIZA** : Parlez-moi de votre père.

**Fonctionnement** : ELIZA identifie des mots-clés comme "je suis", "je me sens", ou "père" dans les phrases de l'utilisateur et applique des règles simples pour formuler une réponse, souvent en reformulant la phrase sous forme de question.

# Limites des approches basées sur la grammaire

Les approches basées sur des grammaires formelles ont été largement utilisées dans les débuts du traitement automatique du langage naturel (NLP). Cependant, elles présentent plusieurs limites importantes.

- **Rigidité des règles** : Les grammaires sont strictement définies par des règles linguistiques. Chaque nouvelle structure ou variation dans le langage nécessite de nouvelles règles, rendant ces approches peu flexibles.
- **Difficulté à gérer l'ambiguïté** : Le langage naturel est souvent ambigu (ex. "*Je vais voir un avocat*"), et les grammaires formelles ont du mal à traiter des ambiguïtés sémantiques ou syntaxiques complexes.
- **Manque de généralisation** : Ces approches fonctionnent bien pour des phrases ou des structures bien définies, mais elles échouent souvent à généraliser sur des phrases inattendues ou plus variées.
- **Incapacité à capturer le sens commun** : Les grammaires formelles ne peuvent pas intégrer le **sens commun** ou des connaissances du monde, ce qui limite leur capacité à comprendre le contexte et l'implicite.
- **Dépendance forte au domaine** : Les systèmes basés sur la grammaire sont souvent trop spécifiques à un domaine particulier et peinent à s'adapter à des domaines ou contextes différents sans une refonte complète.



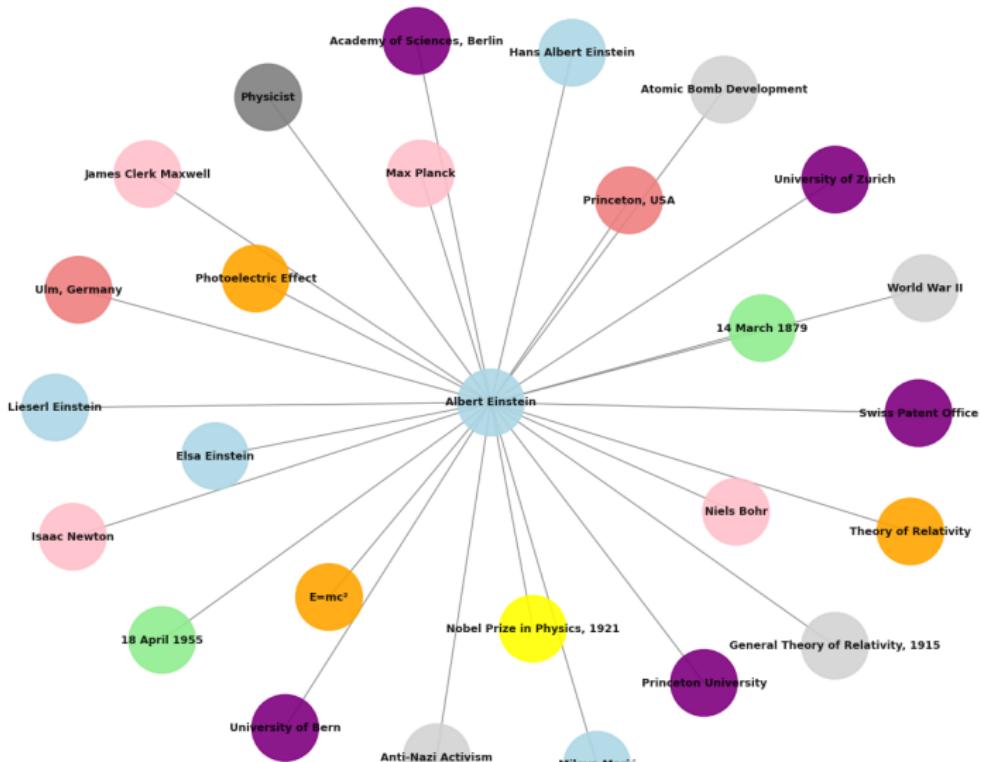
# Ontologies de domaine

Les ontologies de domaine sont des représentations formelles d'un ensemble de concepts et des relations entre ces concepts dans un domaine spécifique. Elles sont largement utilisées en traitement automatique du langage naturel (NLP) et en intelligence artificielle pour structurer la connaissance.

- **Définition** : Une ontologie de domaine est une structuration explicite des concepts et des relations dans un domaine donné, sous la forme d'un graphe sémantique.
- **Concepts clés :**
  - **Concepts (Classes)** : Les entités du domaine. Ex : Personne, Organisation, Lieu.
  - **Relations (Propriétés)** : Les liens entre les concepts. Ex : "travaille pour", "situé à".
  - **Instances** : Les occurrences spécifiques de ces concepts. Ex : "Albert Einstein" (instance de Personne), "Université de Princeton" (instance d'Organisation).
- **Utilisation en NLP :**
  - Structurer et organiser la connaissance d'un domaine.
  - Faciliter l'extraction d'information et la recherche sémantique.
  - Améliorer la désambiguïsation des termes dans des contextes complexes.

# Exemple d'ontologie

Extended Knowledge Graph - Albert Einstein



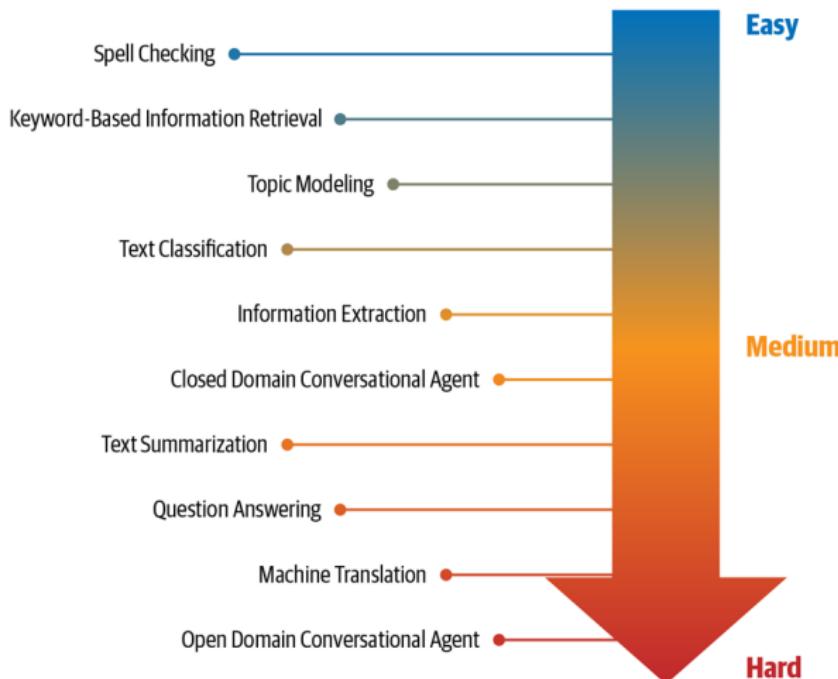
# Tâches classiques en NLP

On retrouve dans le NLP un certain nombre de tâches classiques, qui servent à la fois dans les applications pratiques que dans l'évaluation de l'état de l'art.

- Modélisation du langage (*language modeling*)
- Classification de texte
- Extraction d'informations
- Détection de thématiques (*topic modeling*)
- Résumé de texte
- Question/réponse
- Traduction automatique
- Agent conversationnel (domaine ouvert ou fermé)

# Classement des tâches en NLP par ordre de difficulté

Le classement ci-dessous est à titre indicatif et sujet à des changements. Certaines tâches, comme la traduction, paraissent aujourd'hui moins difficiles qu'il y a quelques années.



# Pourquoi le NLP est si difficile ?

Il y a plusieurs facteurs qui rendent le langage naturel difficile à traiter par la machine :

- Le langage naturel est un problème en grande dimension.
- Le langage naturel est **ambigu**.
  - “Acheter un avocat”
- Le langage naturel repose sur le **sens commun**.
  - Il fait froid *en hiver*
  - Le beurre fond dans le four
- Le langage naturel n'est pas basé sur des **règles**.
  - *En bleu adorable fleurit  
Le toit de métal du clocher. Alentour  
Plane un cri d'hirondelles, autour  
S'étend le bleu le plus touchant. Le soleil*  
*Friedrich Hölderlin*

# Pré-traitement des données textuelles

Les données textuelles sont non structurées. Les principales étapes de pré-traitement impliquent :

- Segmentation des phrases et **tokenisation**.
- Suppression de **stop words**, de la ponctuation, etc.
- **Vectorisation** : Transformation du texte en vecteurs numériques.
- **Stemming, lemmatisation**, conversion des majuscules, etc.
- **Normalisation** : traitement des accents, des caractères spéciaux, etc.
- **Suppression des entités nommées** : Filtrage des noms de lieux, personnes, organisations, etc.
- Détection du langage, **POS tagging** (part-of-speech), etc.
- Correction orthographique : Pour améliorer la qualité des textes bruts.

# Tokenisation

La tokenisation consiste à segmenter une phrase en unités plus petites appelées **tokens**, qui peuvent être des mots, des caractères ou des sous-parties de mots.

## Types de tokenisation :

- **Tokenisation par mots** : Chaque mot est un token.
  - *Exemple* : "Il fait beau à Paris." → ["Il", "fait", "beau", "à", "Paris"]
- **Tokenisation par caractères** : Chaque caractère est un token.
  - *Exemple* : "Paris" → ["P", "a", "r", "i", "s"]
- **Tokenisation par sous-mots (subword)** : Découpe les mots en sous-unités fréquentes, utile pour gérer les langues morphologiquement complexes ou les néologismes.
  - *Exemple* : "démocratisation" → ["démocrat", "isation"]

La tokenisation est une étape essentielle du pré-traitement dans le NLP. Elle permet de transformer du texte brut en unités traitables par les algorithmes de machine learning.

# Stemming and lemmatisation

- Le **stemming** est l'opération qui consiste à supprimer les suffixes et réduire les mots à leur forme de base.
  - marcher → march
  - marchait → march
  - marcha → march
  - marchassiez → march
  - marché → march
- La **lemmatisation** est l'opération qui consiste à réduire un mot à sa racine.
  - nous → nous
  - sommes → être
  - venus → venir
  - faire → faire
  - les → le
  - marchés → marché

# Vectorisation

- **One-Hot encoding:** Étant donné un vocabulaire  $V$ , chaque mot est représenté par un vecteur binaire (à 0 ou 1) de dimension  $V$ .
  - $V = \{Tom, mange, pomme, chien, ciel\}$ , où  $\dim(V) = 5$
  - Tom = [1, 0, 0, 0, 0]
  - mange = [0, 1, 0, 0, 0]
  - pomme = [0, 0, 1, 0, 0]
  - etc.
- **N-grams**
  - "machine learning est très utilisé en NLP."
  - 1-gram: {machine, learning, est, très, utilisé, en, NLP}
  - 2-gram: {machine learning, learning est, est utilisé, utilisé en, en NLP}

# Reconnaissance des entités nommées (NER)

La reconnaissance d'entités nommées (**NER**) est une tâche de NLP qui consiste à identifier et classifier des entités dans un texte, comme des personnes, des organisations, des lieux, des dates, etc.

## Exemples d'entités nommées :

- **Personne** : "Albert Einstein"
- **Organisation** : "Université de Princeton"
- **Lieu** : "Paris"
- **Date** : "15 octobre 2024"

## Catégories courantes d'entités nommées :

- **Personnes** : Noms de personnes célèbres ou non.
- **Lieux** : Pays, villes, régions.
- **Organisations** : Entreprises, institutions, ONG.
- **Dates et heures** : Dates précises, années, heures.
- **Monnaie et quantités** : Montants financiers, unités de mesure.

La NER est cruciale pour de nombreuses applications NLP, telles que l'extraction d'informations, la recherche d'informations, et la classification de documents.

# Part-of-Speech (POS) Tagging

Le **Part-of-Speech (POS) Tagging** est une tâche de NLP qui consiste à assigner à chaque mot d'une phrase son rôle grammatical (catégorie grammaticale).

## Exemples de catégories grammaticales (POS tags) :

- **Nom (N)** : Objet, personne, lieu, idée. *Ex. : chat, Paris*
- **Verbe (V)** : Action, état. *Ex. : mange, est*
- **Adjectif (Adj)** : Caractéristique, qualité. *Ex. : grand, heureux*
- **Adverbe (Adv)** : Modifie un verbe ou une phrase. *Ex. : rapidement, bien*
- **Déterminant (Det)** : Définit un nom. *Ex. : le, une*
- **Préposition (Prep)** : Indique une relation. *Ex. : dans, avec*

## Exemple de POS Tagging :

- "Le chat mange une souris."
- **Résultat** : Le/Det chat/N mange/V une/Det souris/N

Le POS Tagging est une étape clé dans de nombreuses applications de NLP, notamment pour l'analyse syntaxique, la traduction automatique et l'extraction d'information.

# Techniques statistiques pour le NLP

# Introduction aux techniques statistiques pour le NLP

Les techniques statistiques en NLP reposent sur des modèles probabilistes pour analyser et traiter les textes. Elles permettent d'extraire des informations pertinentes en se basant sur les fréquences de mots et les relations entre eux.

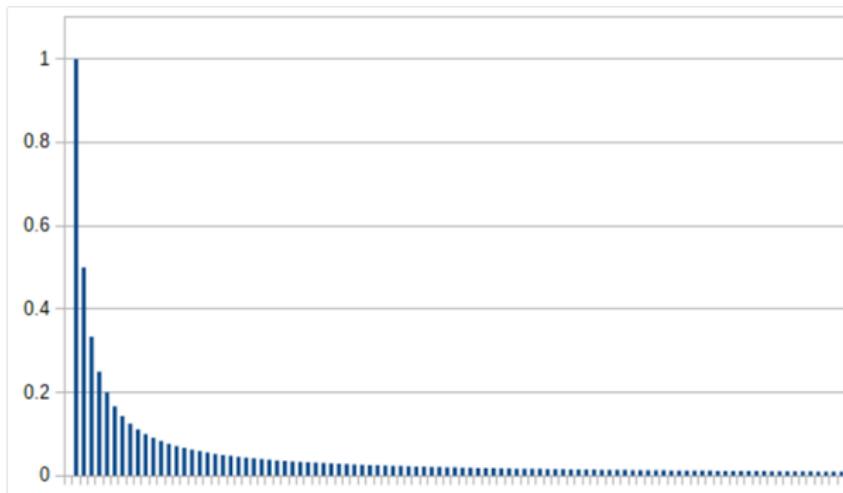
## Principes de base :

- Les modèles probabilistes estiment les probabilités des séquences de mots et des événements linguistiques.
- Les algorithmes apprennent à partir des données d'entraînement, sans règles explicites définies à l'avance.

Ces techniques ont permis de remplacer les approches basées sur des règles linguistiques par des modèles plus flexibles et adaptatifs.

# Principe des techniques statistiques pour le NLP

- Un document est caractérisé par la distribution des mots qui le composent.
- La distribution des mots obéit généralement à une loi de puissance (ou de Zipf). La majorité du langage que nous utilisons ordinairement est composé d'un nombre très limité de mots.



# Introduction aux modèles n-grams

Les modèles **n-grams** sont une des premières techniques statistiques utilisées pour modéliser les séquences de mots dans le langage naturel.

## Principe :

- Un modèle n-gram prédit la probabilité d'un mot en fonction des  $n - 1$  mots précédents.
- $n$ -gram peut être : un unigram (1 mot), bigram (2 mots), trigram (3 mots), etc.

## Exemple (2-gram - bigram) :

- Phrase : "Je mange une pomme."
- Bigrammes : {("Je", "mange"), ("mange", "une"), ("une", "pomme")}

# Calcul de la probabilité avec un modèle n-grams

Les modèles n-grams estiment la **probabilité** d'une phrase en se basant sur les probabilités conditionnelles des mots.

**Probabilité d'une phrase :**

$$P(w_1, w_2, w_3, \dots, w_n) \approx P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_2) \cdot \dots \cdot P(w_n|w_{n-1})$$

**Exemple (2-gram) :**

- Phrase : "Je mange une pomme."
- Probabilité de la phrase :

$$P("Je") \cdot P("mange"|"Je") \cdot P("une"|"mange") \cdot P("pomme"|"une")$$

# Applications et limites des modèles n-grams

## Applications des n-grams :

- **Modélisation du langage** : Utilisés pour estimer la probabilité des séquences de mots dans des applications comme la reconnaissance vocale ou la correction orthographique.
- **Traduction automatique** : Les n-grams permettent de prédire les séquences de mots dans les langues cibles.
- **Génération de texte** : Génération automatique de phrases probables à partir de contextes.

## Limites des n-grams :

- **Explosion combinatoire** : La taille du modèle croît exponentiellement avec  $n$ , rendant le stockage des probabilités complexe pour de grandes valeurs de  $n$ .
- **Manque de contexte à long terme** : Les n-grams ne capturent que les relations entre les  $n$  mots précédents, ignorant les dépendances à longue distance.

# Introduction à TF-IDF

**TF-IDF (Term Frequency - Inverse Document Frequency)** est une méthode utilisée pour évaluer l'importance d'un mot dans un document, relativement à l'ensemble d'un corpus.

## Objectif :

- Mesurer l'importance d'un mot dans un document tout en réduisant l'importance des mots trop fréquents dans le corpus.

## Applications :

- Extraction de caractéristiques textuelles pour des tâches de classification.
- Recherche d'information et moteurs de recherche (identifier les mots-clés).
- Résumé automatique de textes.

# Term Frequency (TF)

La **fréquence d'un terme (TF)** mesure combien de fois un mot apparaît dans un document. Plus un mot apparaît souvent dans le document, plus sa fréquence est élevée.

**Formule :**

$$TF(t, d) = \frac{\text{Nombre d'occurrences du terme } t \text{ dans le document } d}{\text{Nombre total de mots dans le document } d}$$

**Exemple :**

- Document : "Le chat mange une souris. Le chat dort."
- Nombre d'occurrences du mot "chat" = 2
- Nombre total de mots dans le document = 7
- Donc :

$$TF(\text{"chat"}, d) = \frac{2}{7} = 0.285$$

# Inverse Document Frequency (IDF)

L'**Inverse Document Frequency (IDF)** mesure l'importance d'un mot dans le corpus. Les mots qui apparaissent dans de nombreux documents ont une faible importance, car ils sont trop communs.

**Formule :**

$$IDF(t) = \log \left( \frac{\text{Nombre total de documents dans le corpus}}{\text{Nombre de documents contenant le terme } t} \right)$$

**Exemple :**

- Supposons que le mot "chat" apparaisse dans 100 documents d'un corpus de 1000 documents.
- Le nombre total de documents dans le corpus = 1000
- Le nombre de documents contenant le mot "chat" = 100
- Donc :

$$IDF("chat") = \log \left( \frac{1000}{100} \right) = \log(10) = 1$$

# Calcul de TF-IDF

La valeur **TF-IDF** combine la fréquence d'un terme dans un document (TF) et l'inverse de sa fréquence dans le corpus (IDF).

**Formule :**

$$TF-IDF(t, d) = TF(t, d) \times IDF(t)$$

**Exemple :**

- Document : "Le chat mange une souris."
- $TF("chat", d) = 0.285$ ,  $IDF("chat") = 1$
- $TF-IDF("chat", d) = 0.285 \times 1 = 0.285$

# Exemple de matrice TF-IDF

## Documents :

- Document 1 (D1) : "le chat dort."
- Document 2 (D2) : "le chien dort."
- Document 3 (D3) : "le chat mange."
- Document 4 (D4) : "le chien mange."

## Matrice TF-IDF :

Matrice TF-IDF

	D1	D2	D3	D4
chat	0.231	0.0	0.231	0.0
chien	0.0	0.231	0.0	0.231
dort	0.231	0.231	0.0	0.0
mange	0.0	0.0	0.231	0.231

# Machine learning pour le NLP

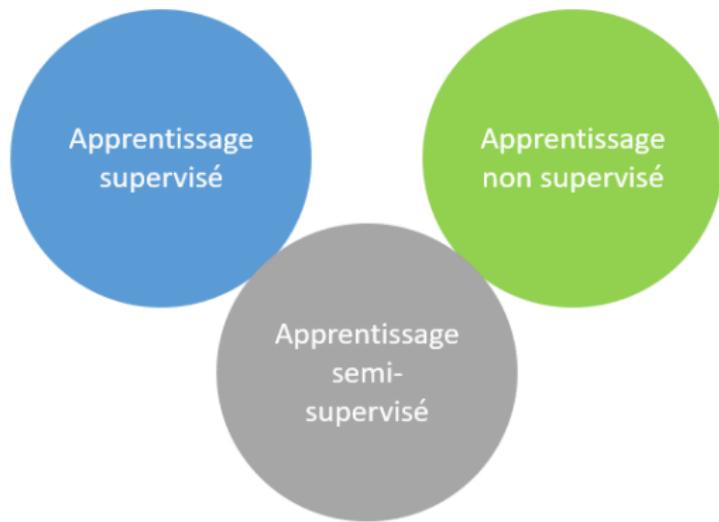
# Définition de l'apprentissage automatique

L'apprentissage automatique est une branche de l'intelligence artificielle qui consiste à doter les machines de la capacité d'apprendre à partir de données sans que celles-ci ne soient explicitement programmées pour exécuter des tâches spécifiques.

Le machine learning englobe plusieurs types d'apprentissage :

- **Supervisé** : Les algorithmes apprennent à partir de données étiquetées pour faire des prédictions ou classifications.
- **Non supervisé** : L'apprentissage est effectué sur des données non étiquetées pour trouver des structures cachées.
- **Semi-supervisé** : Combine des éléments des deux premiers types en utilisant une petite quantité de données étiquetées et une grande quantité de données non étiquetées.
- **Par renforcement** : Les modèles apprennent à prendre des décisions en maximisant une récompense à travers des interactions.

# Typologies d'apprentissage automatique



# L'apprentissage supervisé

**L'apprentissage supervisé** consiste à apprendre un modèle qui associe une étiquette (*label*) à un ensemble de caractéristiques (*features*).

- **Inputs** : un jeu de données *annotées* pour entraîner le modèle.
  - Exemple : des textes (tweets, etc.) avec les *sentiment* associés, positifs ou négatifs.
- **Output** : une étiquette pour un point de donnée inconnu par le modèle.

L'apprentissage supervisé se décline lui-même en deux grandes familles :

- **La classification** : prédire une catégorie ou une classe.
  - Exemple : prédire l'étiquette d'une image (chat, chien, etc.), le sentiment associé à un texte, le centre d'intérêt d'un client à partir de ses commentaires, etc.
- **La régression** : prédire une valeur continue (un nombre réel typiquement).
  - Exemple : prédire le prix d'un appartement, la lifetime value d'un client, etc.

# L'apprentissage non supervisé

**L'apprentissage non supervisé** se réfère à l'utilisation de modèles d'apprentissage automatique pour identifier des patterns et des structures dans des données qui ne sont pas étiquetées.

Principales typologies de l'apprentissage non supervisé :

- **Clustering** : Regroupement de points de données similaires ensemble.  
Exemple : segmentation de marché, regroupement social.
- **Réduction de dimensionnalité** : Techniques pour réduire la dimension des données tout en préservant leur structure. Exemple : visualisation de données en grande dimension.
- **Détection d'anomalies** : Déetecter des observations dont les caractéristiques sont inhabituelles par rapport à la majorité.

# Pipeline Machine Learning typique en NLP

Le processus d'application du machine learning au NLP suit généralement un pipeline en plusieurs étapes :

## Étapes du pipeline :

- **1. Pré-traitement des données** : Nettoyage, tokenisation, vectorisation.
- **2. vectorisation** : Transformation des documents en vecteurs (TF-IDF, etc.).
- **3. Modélisation** : Sélection et entraînement d'un modèle de machine learning (Naïve Bayes, SVM, etc.).
- **4. Évaluation** : Mesure de la performance (précision, F1-score, etc.).
- **5. Optimisation** : Ajustement des hyperparamètres et amélioration du modèle.

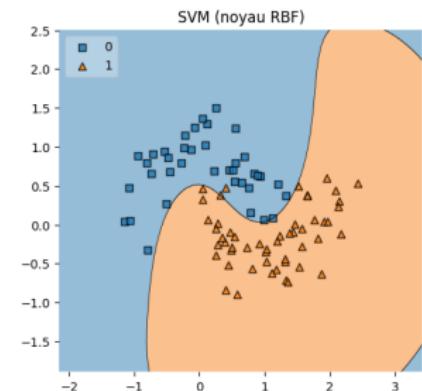
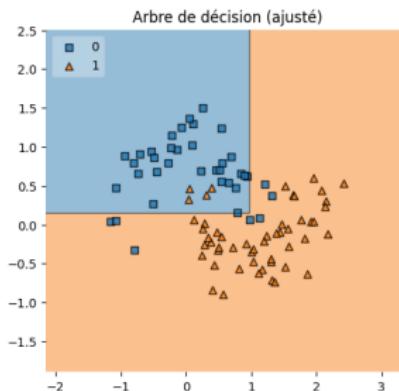
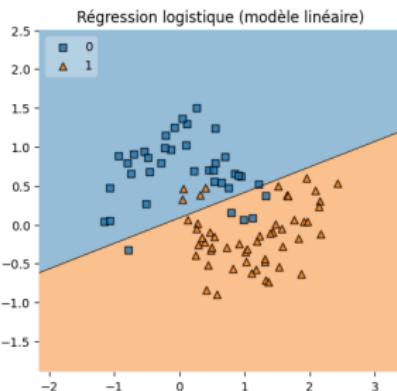


# Apprentissage supervisé

# Formalisation du problème de l'apprentissage supervisé

Comment choisir un modèle parmi tous les modèles possibles ?

- A priori sur la classe du modèle (biais inductif).
- Minimisation du risque empirique.



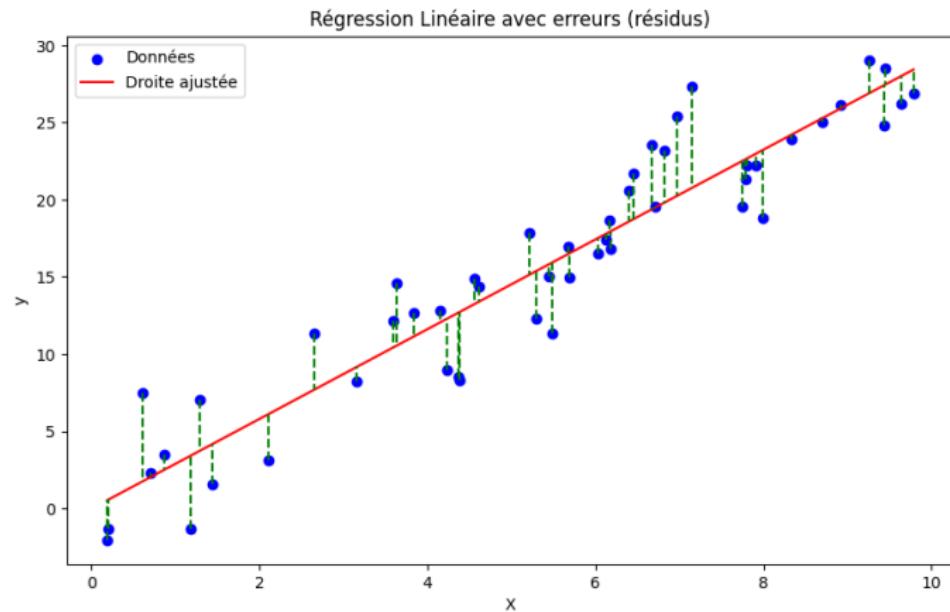
# Principes de l'apprentissage supervisé

La construction d'un modèle en apprentissage supervisé repose sur trois piliers essentiels qui guident le processus d'apprentissage et déterminent sa réussite :

- **Données** : L'ensemble d'exemples étiquetés utilisés pour l'entraînement du modèle. La qualité, la quantité et la représentativité de ces données sont cruciales pour la capacité du modèle à apprendre et à généraliser à de nouvelles observations.
- **Fonction objective** : Aussi connue sous le nom de fonction de perte ou de coût, elle quantifie l'erreur entre les prédictions du modèle et les valeurs réelles. L'objectif de l'apprentissage est de minimiser cette fonction, guidant ainsi le modèle vers une meilleure performance.
- **Hypothèses (biais inductif)** : Les hypothèses préalables sur la forme du modèle et les relations dans les données. Ces hypothèses concernent le choix de l'algorithme d'apprentissage, et toute préconception sur la distribution des données. Ces hypothèses dirigent l'espace de recherche des solutions possibles et influencent directement la capacité du modèle à apprendre et à généraliser.

# Minimisation du risque empirique

Une fois la famille de modèles est choisie, un modèle peut être déterminé en minimisant l'erreur de prédiction sur le jeu de données d'entraînement.



# Fonctions de coût

Le choix de la fonction de coût dépend de la nature du problème et des données.  
On distingue deux types de fonctions de coût.

## Régression

- L'erreur quadratique :  $L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$
- L'erreur absolue :  $L(\hat{y}, y) = |\hat{y} - y|$

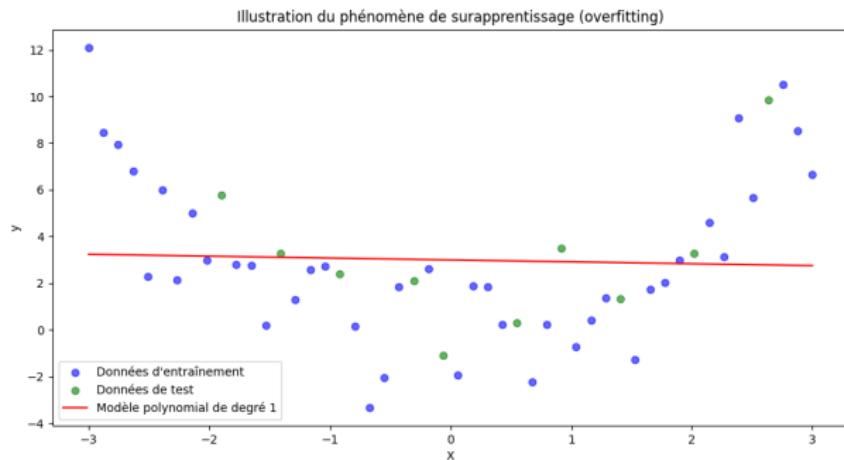
## Classification

- Entropie croisée :  $L(\hat{y}, y) = -y\log(\hat{y}) - (1 - y)\log(1 - \hat{y})$
- L'erreur Hinge :  $L(\hat{y}, y) = \max(0, 1 - \hat{y}y)$

# Sous-apprentissage

## Definition

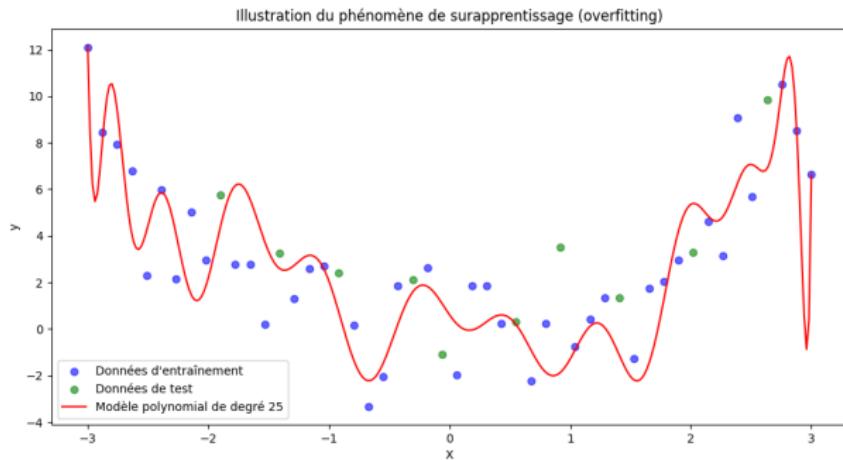
On dit qu'un modèle de machine learning est en régime de sous-apprentissage (underfitting) lorsqu'il n'arrive pas à capturer la complexité (l'information) présente dans le jeu de données d'entraînement.



# Sur-apprentissage

## Definition

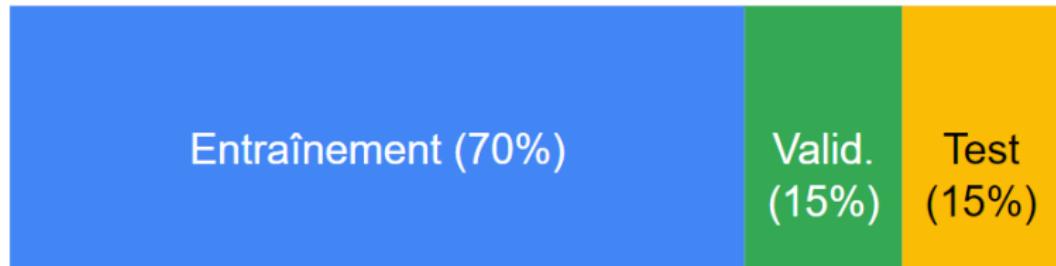
On dit qu'un modèle de machine learning est en régime de sur-apprentissage (overfitting) lorsqu'il n'arrive pas à généraliser à des données non encore observées, i.e. lorsqu'il est trop adapté aux données d'entraînement.



# Sélection de modèle

Pour sélectionner le modèle le plus pertinent par rapport à une métrique donnée, on applique la méthodologie suivante :

- On partitionne le jeu de données disponible en trois parties : un jeu d'entraînement, un jeu de validation et un jeu de test.
- On entraîne  $M$  modèles sur le jeu d'entraînement.
- On évalue les performances respectives des  $M$  modèles sur le jeu de validation et on sélectionne le meilleur.
- Le modèle sélectionné est ensuite évalué sur le jeu de test. Idéalement, le jeu de test est ainsi utilisé une seule fois.



# Validation croisée K-fold

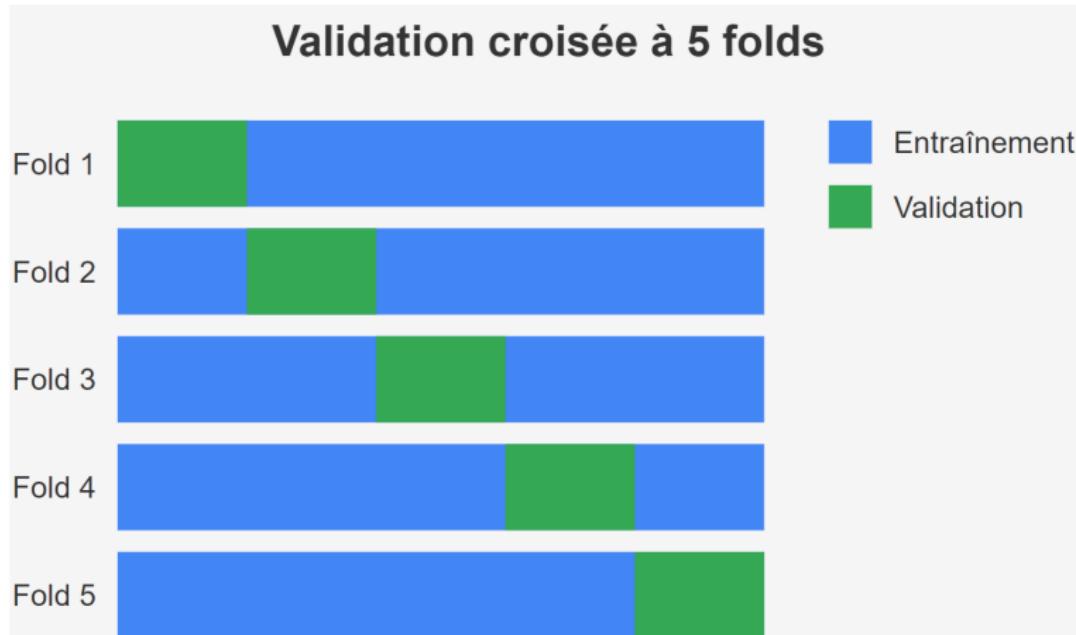
La validation croisée est une méthode plus robuste pour évaluer les performances des modèles.

La validation croisée K-fold s'effectue selon la méthodologie suivante :

- On partitionne le jeu de données  $\mathcal{D}$  en  $K$  parties ayant approximativement la même taille.
- Pour chaque partie  $\mathcal{D}_k$ , on entraîne le modèle sur l'ensemble des données restantes. Ce modèle est ensuite évalué sur  $\mathcal{D}_k$ .
- On considère la moyennes des performances du modèles sur les différents  $\mathcal{D}_k$ .

La validation croisée K-fold permet ainsi de prendre en compte la variabilité qu'il peut y avoir dans les données. Par ailleurs, on peut également avoir une estimation de la variance du modèle.

# Illustration de la validation croisée



# Métriques de performance : régression

On dispose d'un certain nombre de métriques pour évaluer les performances des modèles de machine learning. Celles-ci peuvent être divisées en deux catégories.

## Régression

- L'erreur quadratique moyenne (MSE) : elle est définie comme la moyenne des carrés des écarts entre les prédictions et les valeurs observées.

$$MSE = \frac{1}{n} \sum_{i=1}^n (f(x_i) - y_i)^2$$

- La racine carrée de l'erreur quadratique moyenne (RMSE) :

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (f(x_i) - y_i)^2}$$

## Métriques de performance : classification 1/2

**Accuracy** : L'accuracy est la métrique de base qui permet d'évaluer les performances d'un modèle de classification. Elle est définie comme :

$$\text{Accuracy} = \frac{\text{Nombre de prédictions correctes}}{\text{Nombre total de prédictions}}$$

**Matrice de confusion** : La matrice de confusion est une représentation permettant d'offrir plus de finesse par rapport à l'accuracy, notamment quand le jeu de données est déséquilibré (présence de classes majoritaires). Elle compare les prédictions du modèle avec les valeurs réelles et est structurée comme suit :

		Valeur Prédite	
		Positif	Négatif
Valeur Réelle	Positif	Vrai Positif (VP)	Faux Négatif (FN)
	Négatif	Faux Positif (FP)	Vrai Négatif (VN)

# Métriques de performance : classification 1/2

A partir de la matrice de confusion, on peut dériver d'autres métriques :

- Précision : elle est définie comme la proportion des prédictions correctes parmi toutes les prédictions positives :

$$\text{Précision} = \frac{VP}{VP + FP}$$

- Rappel (recall) : il représente la proportion des vrais positifs correctement prédits par le modèle.

$$\text{Rappel} = \frac{VP}{VP + FN}$$

- Score F1 (F1-score) : Le score F1 est défini comme la moyenne harmonique de la précision et du rappel.

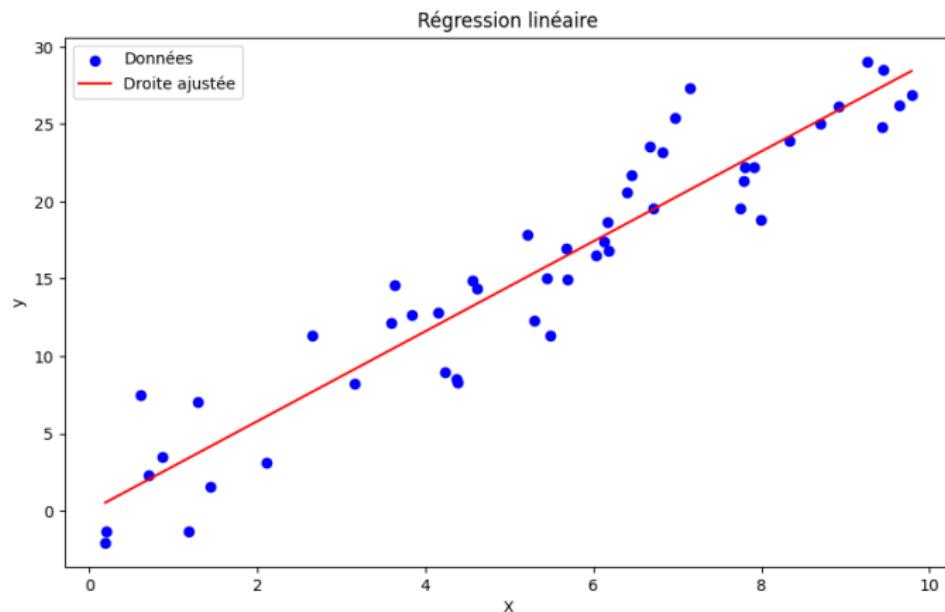
$$\text{Score F1} = 2 \frac{\text{Précision} \times \text{Rappel}}{\text{Précision} + \text{Rappel}}$$

# Modèles classiques de machine learning

# Régression linéaire simple

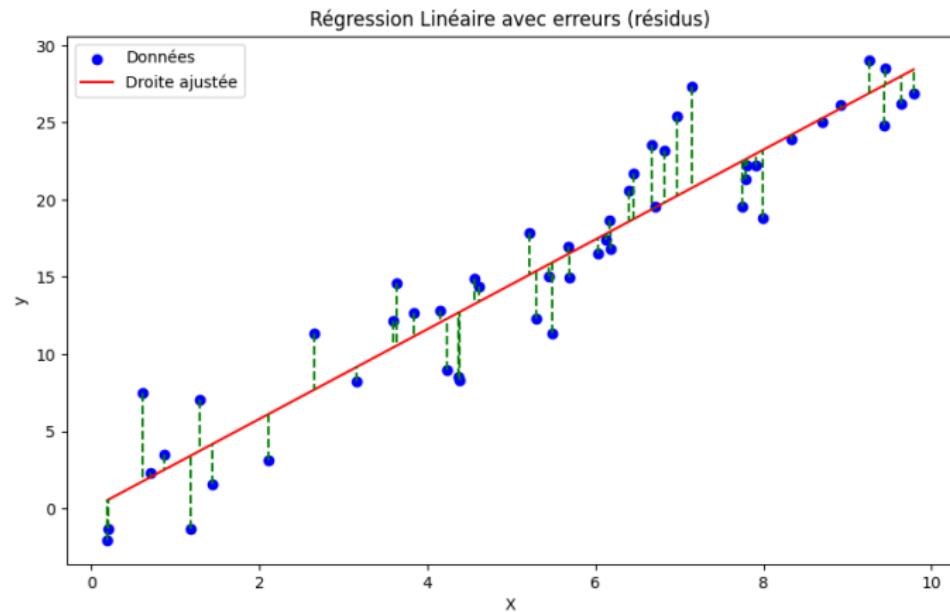
Soit un ensemble de  $n$  observations  $x_1, x_2, \dots, x_n$  avec les labels correspondants  $y_1, y_2, \dots, y_n$ , on cherche le modèle linéaire qui ajuste le mieux ces données.

$$\hat{y} = \beta_0 + \beta_1 x$$



# Minimisation du risque empirique 1/2

L'erreur de prédiction pour la  $i$  ième observation est :  $e_i = y_i - \hat{y}_i$ . où  $\hat{y}_i = \beta_0 + \beta_1 x_i$ .



## Minimisation du risque empirique 2/2

Déterminer un modèle de régression linéaire simple revient à minimiser les erreurs de prédiction (risque empirique) :

$$\sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Autrement dit, chercher les coefficients  $\beta_j$  qui minimisent le risque empirique :

$$\arg \min_{\beta_0, \beta_1} \left( \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i)^2 \right)$$

# Régression linéaire multiple

On considère  $n$  observations  $X^1, X^2, \dots, X^n$  où chaque observation  $X^i$  est désormais un vecteur ayant  $p$  composantes ( $p$  variables explicatives).

$$X^i = \begin{pmatrix} x_1^i \\ x_2^i \\ \vdots \\ x_p^i \end{pmatrix}$$

La régression linéaire s'écrit alors :

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p$$

Les coefficients  $\beta_0, \beta_1, \dots, \beta_p$  sont déterminés par la méthode des moindres carrés :

$$\arg \min_{\beta_0, \beta_1, \dots, \beta_p} \sum_{i=1}^n \left( y^i - (\beta_0 + \sum_{j=1}^p \beta_j x_j^i) \right)^2$$

# Régression logistique : introduction

La régression logistique est une technique d'analyse statistique utilisée pour modéliser la probabilité d'une variable dépendante binaire. C'est un cas particulier de modèle linéaire généralisé qui est utilisé pour des problèmes de classification.

Principes de la régression logistique :

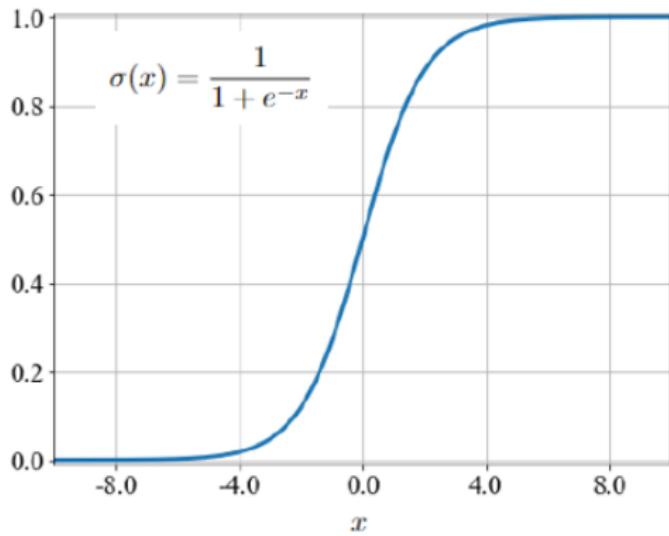
- **Variable dépendante** : On cherche la probabilité que la variable dépendante ( $y$ ) appartienne à une classe (0 ou 1, vrai ou faux, succès ou échec). Autrement dit, on cherche à modéliser  $P(y = 1)$  en fonction des variables dépendantes (explicatives)  $x$ .
- **Odds ratio** : Plus concrètement, on cherche à exprimer la côte anglaise (odd ratio) en fonction des variables dépendantes ( $x$ ).

$$\ln \frac{p(\mathbf{x})}{1 - p(\mathbf{x})} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p$$

# Régression logistique : fonction sigmoïde

Après quelques simplifications, on peut écrire la probabilité  $p(x)$  (la probabilité pour que  $y$  soit un succès par exemple) :

$$p(\mathbf{x}) = \frac{1}{1 + e^{-(\beta^T \mathbf{x})}}$$



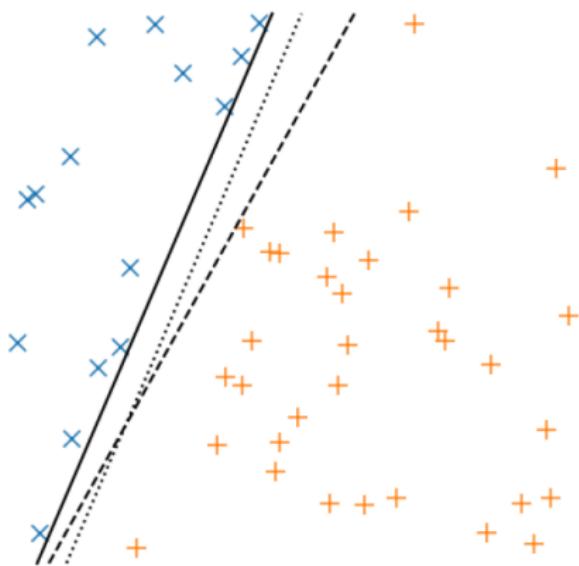
# Calcul des coefficients de la régression logistique

Les coefficients de la régression logistique peuvent être calculés en minimisant le risque empirique par rapport à une fonction de coût sous forme d'entropie croisée :

$$\arg \min_{\beta_0, \beta_1, \dots, \beta_p} \left( - \sum_{i=1}^n y_i \ln(\hat{y}_i) + (1 - y_i) \ln(1 - \hat{y}_i) \right)$$

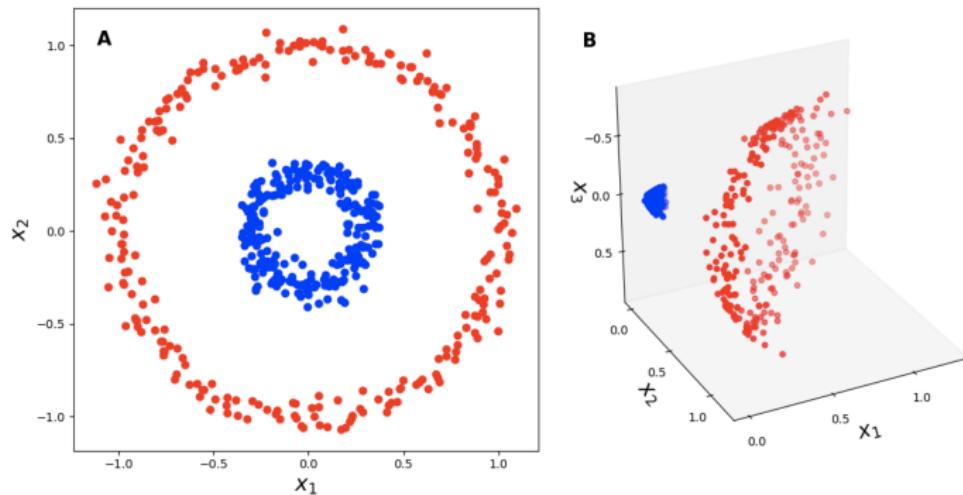
# Machines à vecteurs de support (SVM)

Considérons un problème de classification binaire. On recherche l'hyperplan séparateur qui maximise la marge  $\gamma$  entre les deux classes. La marge  $\gamma$  étant définie comme la distance entre cet hyperplan et les observations les plus proches.



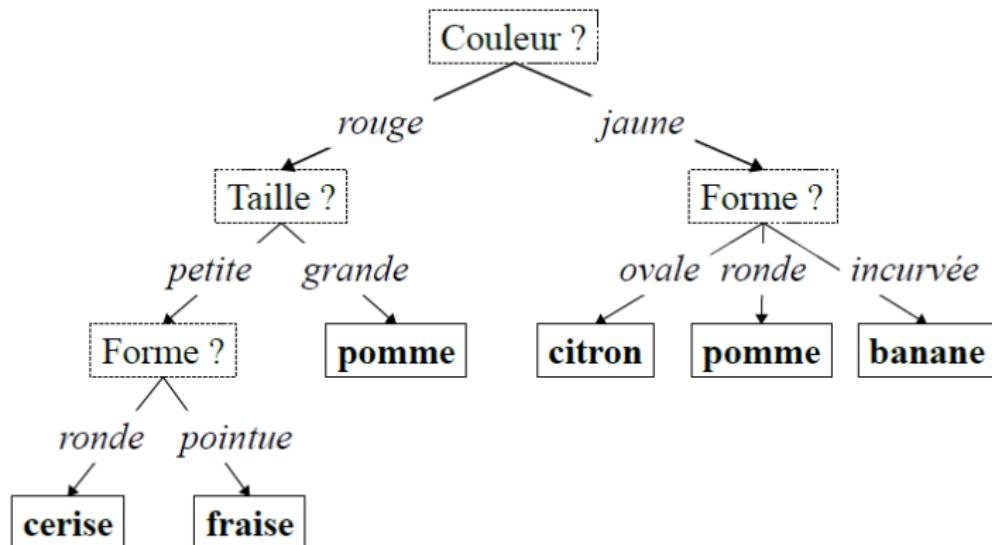
# SVM à noyau

On considère un problème de classification non linéaire. L'astuce du noyau consiste à augmenter la dimension du problème, pour le résoudre ensuite avec un séparateur linéaire dans le nouvel espace.



# Arbres de décision

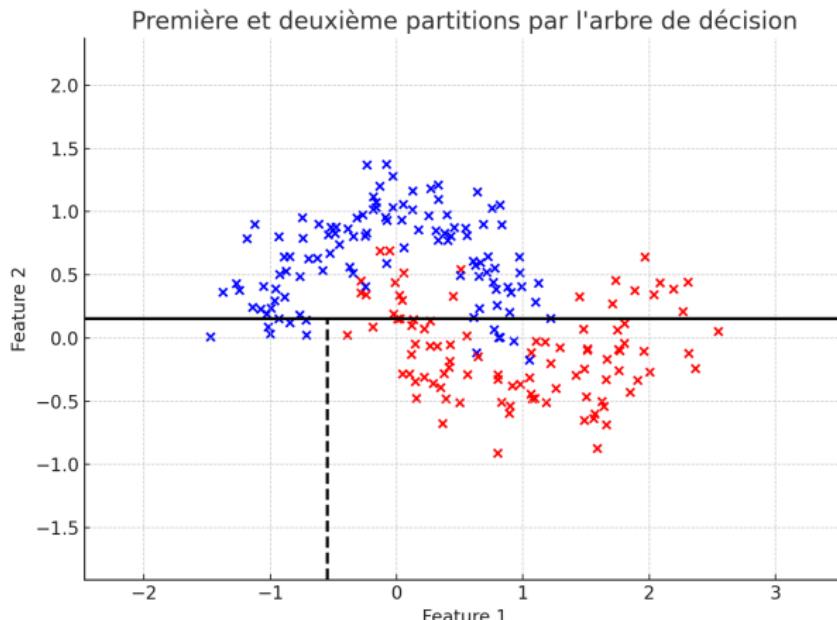
Les arbres de décisions sont des modèles dont le processus de décision est hiérarchique et prend la forme d'un arbre.



# Entraînement des arbres de décision

Les arbres de décisions sont généralement entraînés à l'aide de la technique CART (Classification And Regression Trees).

Etant donné un ensemble d'observations  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ , les arbres de décision partitionnent cet espace en plusieurs régions  $R_1, R_2, \dots, R_m$ .



# Critères d'optimisation

Le critère d'optimisation dépend de la tâche en question.

- **Classification**

- Indice de Gini simplifié :  $\sum_{k=1}^K p_{mk}(1 - p_{mk})$
- Entropie croisée :  $-\sum_{k=1}^K p_{mk} \ln(p_{mk})$

- **Régression**

$$\sum_{i=n}^n (y_i - f(x_i))^2$$

# Forêts aléatoires (random forests)

La technique des forêts aléatoires (random forests) consiste à appliquer une approche de type bagging sur les arbres de décision.

L'algorithme random forests suit cette procédure :

- Tirer par bootstrap  $B$  échantillons de tailles  $n$  à partir de l'ensemble  $D$ .
- Pour chaque échantillon tiré, construire un arbre en répétant les étapes suivantes jusqu'à atteindre  $n_{min}$ .
  - Tirer d'une manière aléatoire  $m$  variables parmi les  $p$  variables.
  - Sélectionner la meilleure variable avec le meilleur point de partitionnement.
  - Partitionner le noeud en deux sous-branches.
- Agréger les arbres construits.

Les prédictions sont agrégées selon qu'il s'agisse de régression ou de classification :

- Régression : moyenne  $f^B(x) = \frac{1}{B} \sum_{b=1}^B T_b(x)$
- Classification : vote majoritaire.

## Remarques

- L'algorithme de random forests intègre nativement une forme de validation croisée. Les performances mesurées sur  $\bigcup_{b_i \neq b_k}$  (out of bag ou OOB) sont souvent proches de celles que l'on pourrait mesurer avec une validation croisée.
- Le nombre de variable tirées pour chaque noeud est généralement donné par  $\sqrt{p}$  pour la classification et  $\frac{p}{3}$  pour la régression. Cet hyperparamètre dépend cependant du problème considéré.
- Lorsque le nombre de variable est élevé alors que le nombre de variables réellement partinente est faible, la probabilité que les  $p$  variables sélectionnées pour chaque partitionnement incluent des variables pertinentes devient faible, et les performances du modèle en termes de généralisation peuvent se détériorer considérablement.
- L'algorithme random forests permet de restituer des informations sur l'importance des variable (feature importance).

# Classification Bayésienne

La classification bayésienne repose sur l'utilisation de la probabilité conditionnelle et le théorème de Bayes pour prédire la catégorie d'une nouvelle observation. Elle est particulièrement efficace dans les situations où la dimensionnalité des données est élevée ou lorsque les données sont incomplètes.

La formule fondamentale de la classification bayésienne est le calcul de la probabilité a posteriori pour chaque classe  $C_k$  donnée une observation  $\mathbf{x}$  :

$$P(C_k|\mathbf{x}) = \frac{P(\mathbf{x}|C_k) \cdot P(C_k)}{P(\mathbf{x})}$$

où :

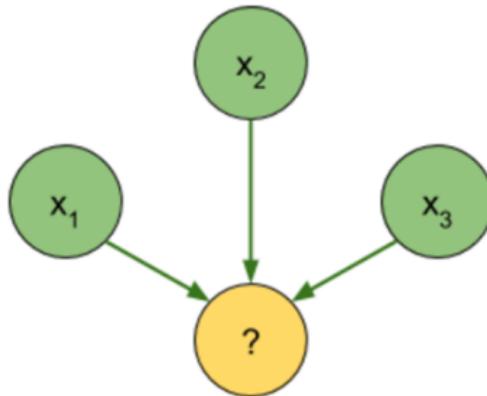
- $P(C_k|\mathbf{x})$  est la probabilité a posteriori de la classe  $C_k$  étant donné l'observation  $\mathbf{x}$ .
- $P(\mathbf{x}|C_k)$  est la vraisemblance de l'observation  $\mathbf{x}$  dans la classe  $C_k$ .
- $P(C_k)$  est la probabilité a priori de la classe  $C_k$ .
- $P(\mathbf{x})$  est la probabilité marginale de l'observation  $\mathbf{x}$ , souvent calculée comme la somme des vraisemblances de  $\mathbf{x}$  sur toutes les classes pondérée par leur probabilité a priori.

# Hypothèse d'indépendance dans Naive Bayes

L'hypothèse d'indépendance de Naive Bayes suppose que la présence (ou l'absence) d'une caractéristique particulière est indépendante de la présence (ou l'absence) de toute autre caractéristique, donnée la classe.

Cela simplifie le calcul de  $P(x|C_k)$  car :

$$P(x_1, \dots, x_n|C_k) = P(x_1|C_k) \times \dots \times P(x_n|C_k)$$



# Apprentissage non supervisé

# Apprentissage non supervisé

Dans l'apprentissage non supervisé, on considère  $n$  observations sans labels. On s'intéresse fondamentalement à la probabilité jointe de ces observations.

On peut distinguer deux grandes catégories d'apprentissage non supervisé :

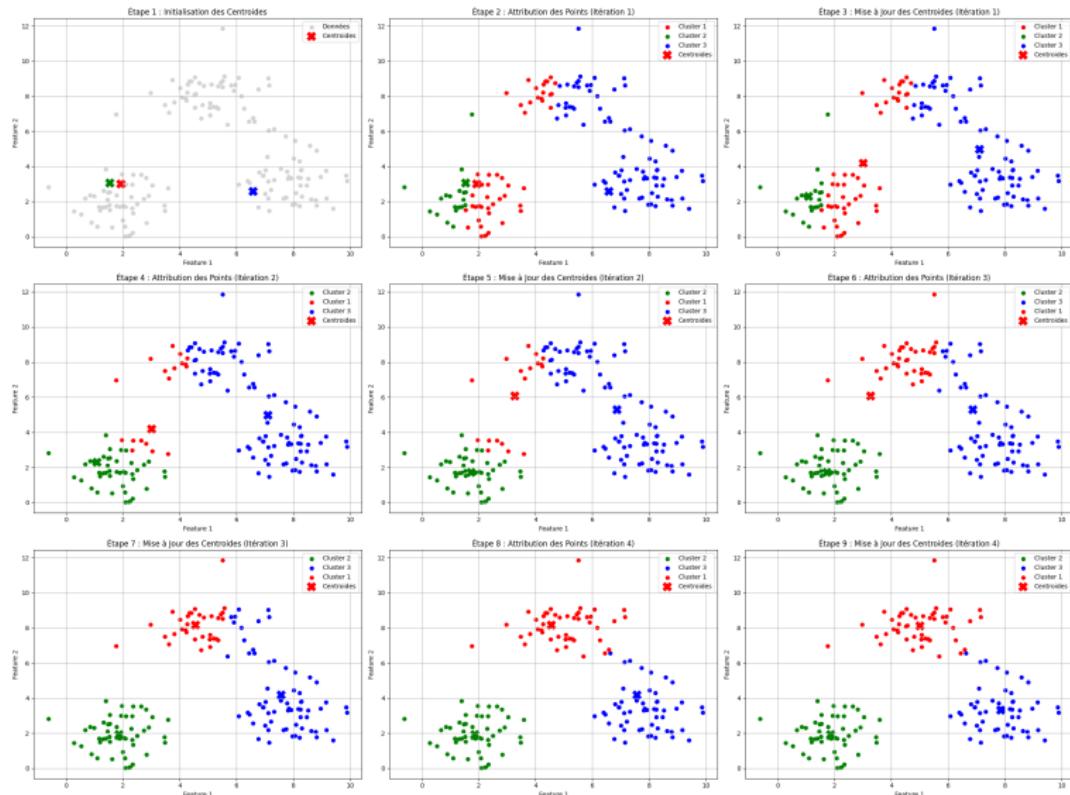
- **Clustering (partitionnement)** : cela consiste à partitionner les  $n$  observations en  $K$  groupes pertinents (généralement le critère de pertinence a une signification d'un point de vue métier).
- **Réduction de dimension** : il s'agit de trouver une représentation des données originelles dans un nouvelles espace de plus petite dimension. Cela peut être effectué à différentes fins : visualisation des données, compression des données, amélioration des performances du modèles (modèles plus robuste, plus explicables, etc.).
- **Détection d'anomalie** : il s'agit de détecter des observations qui présentent un profil (des features) inhabituels par rapport au profil moyen de la majorité des observations.

## K-means

L'algorithme de Lloyd tente de regrouper les données en clusters en minimisant les distances entre les points d'un même cluster tout en maximisant les distances entre points appartenant à différents clusters.



# Algorithme de Lloyd

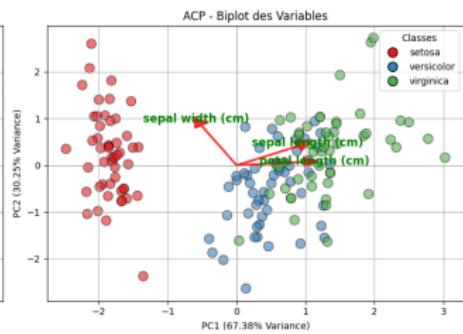
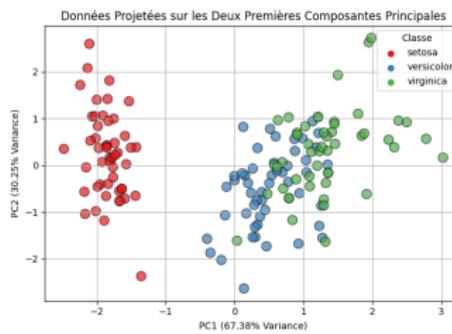
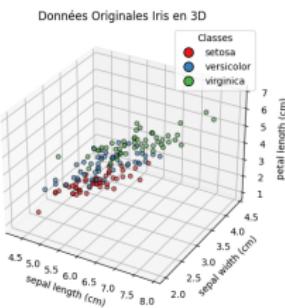


## Remarques

- L'algorithme des k-means étant basé sur une distance euclidienne, il est nécessaire de normaliser les données avant de l'exécuter.
- L'algorithme des k-means est très sensible aux données aberrantes (outliers). Il faut donc considérer les données d'une manière attentive. Cependant, cela permet également d'utiliser l'algorithme des k-means pour la détection automatique des outliers.
- Les centroïdes étant initialisés d'une manière aléatoire, les clusters obtenus ne sont pas stables ; les clusters peuvent changer d'une exécution à l'autre. Il existe cependant une variante plus stable, appelée k-means++, qui permet de sélectionner les centroïdes d'une manière semi-aléatoire.
- Il est possible de partitionner les données avec une métrique plus générale que la distance euclidienne. On peut définir un algorithme k-means à noyau sur un espace de Hilbert pour aller au-delà de la métrique euclidienne.
- **K-means n'est pas adapté aux données en grande dimension.**

# Analyse en Composantes Principales (ACP)

L'Analyse en Composantes Principales (ACP) est une technique statistique de réduction de dimensionnalité. Elle transforme les données en un nouveau système de coordonnées où la plus grande variance est capturée sur les premiers axes, appelés composantes principales.



# Formulation de l'ACP

Soit  $X$  une matrice de données de dimension  $n \times p$  ( $n$  observations,  $p$  variables), centrée (moyenne nulle). L'ACP cherche à trouver les vecteurs propres et les valeurs propres de la matrice de covariance  $C = \frac{1}{n-1} X^T X$ .

La matrice de covariance  $C$  peut être décomposée comme suit :

$$C = VLV^T$$

où  $V = [u_1, u_2, \dots, u_p]$  est la matrice des vecteurs propres et  $L$  est une matrice diagonale des valeurs propres  $\lambda_k$ .

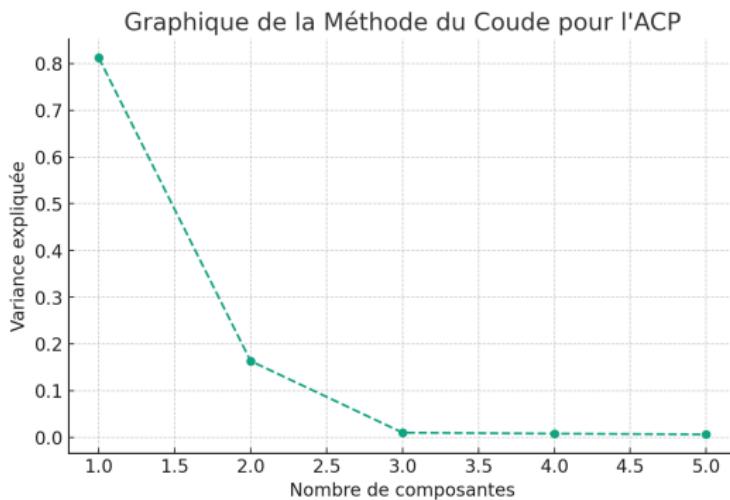
La contribution de chaque composante principale à la variance totale est donnée par :

$$\frac{\lambda_k}{\sum_{i=1}^p \lambda_i}$$

# Choix du nombre de composantes principales

Le nombre de composantes à retenir est déterminé en fonction du pourcentage de variance totale que l'on souhaite expliquer.

On utilise généralement la méthode du coude (Scree plot). Il s'agit d'un graphique montrant la proportion de la variance expliquée en fonction du nombre de composantes.



# Classification Bayésienne

La classification bayésienne repose sur l'utilisation de la probabilité conditionnelle et le théorème de Bayes pour prédire la catégorie d'une nouvelle observation. Elle est particulièrement efficace dans les situations où la dimensionnalité des données est élevée ou lorsque les données sont incomplètes.

La formule fondamentale de la classification bayésienne est le calcul de la probabilité a posteriori pour chaque classe  $C_k$  donnée une observation  $\mathbf{x}$  :

$$P(C_k|\mathbf{x}) = \frac{P(\mathbf{x}|C_k) \cdot P(C_k)}{P(\mathbf{x})}$$

où :

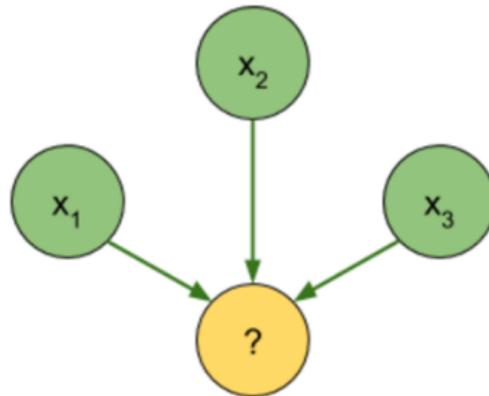
- $P(C_k|\mathbf{x})$  est la probabilité a posteriori de la classe  $C_k$  étant donné l'observation  $\mathbf{x}$ .
- $P(\mathbf{x}|C_k)$  est la vraisemblance de l'observation  $\mathbf{x}$  dans la classe  $C_k$ .
- $P(C_k)$  est la probabilité a priori de la classe  $C_k$ .
- $P(\mathbf{x})$  est la probabilité marginale de l'observation  $\mathbf{x}$ , souvent calculée comme la somme des vraisemblances de  $\mathbf{x}$  sur toutes les classes pondérée par leur probabilité a priori.

# Hypothèse d'indépendance dans Naive Bayes

L'hypothèse d'indépendance de Naive Bayes suppose que la présence (ou l'absence) d'une caractéristique particulière est indépendante de la présence (ou l'absence) de toute autre caractéristique, donnée la classe.

Cela simplifie le calcul de  $P(x|C_k)$  car :

$$P(x_1, \dots, x_n|C_k) = P(x_1|C_k) \times \dots \times P(x_n|C_k)$$



# Introduction au deep learning

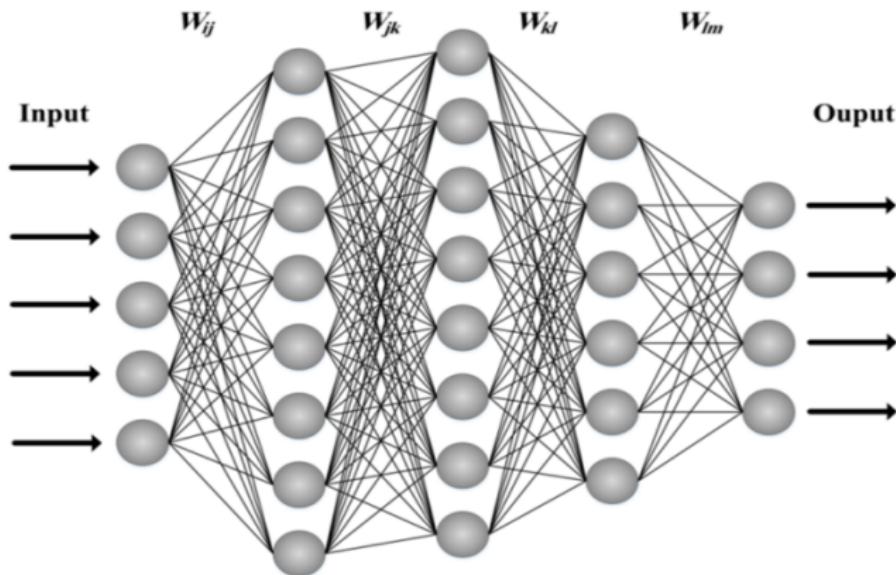
# Introduction aux réseaux de neurones

**Définition :** Les réseaux de neurones sont des modèles computationnels inspirés par le fonctionnement des neurones dans le cerveau humain. Ils sont capables d'apprendre des tâches complexes en modélisant des relations non linéaires entre les entrées et les sorties.

## Caractéristiques :

- **Extraction automatique des features** : Capacité d'adaptation et d'extraction des features à partir des données sans programmation explicite.
- **Modélisation non linéaire** : Aptitude à capturer des relations complexes dans les données.
- **Flexibilité** : Applicables à un large éventail de tâches et de typologies de données (images, langage naturel, données graphiques, etc.).

# Illustration d'un réseau de neurones classique



# Le neurone artificiel

**Modèle Mathématique** : Chaque neurone artificiel effectue une somme pondérée de ses entrées, ajoute un biais, et passe le résultat à travers une fonction d'activation pour obtenir la sortie.

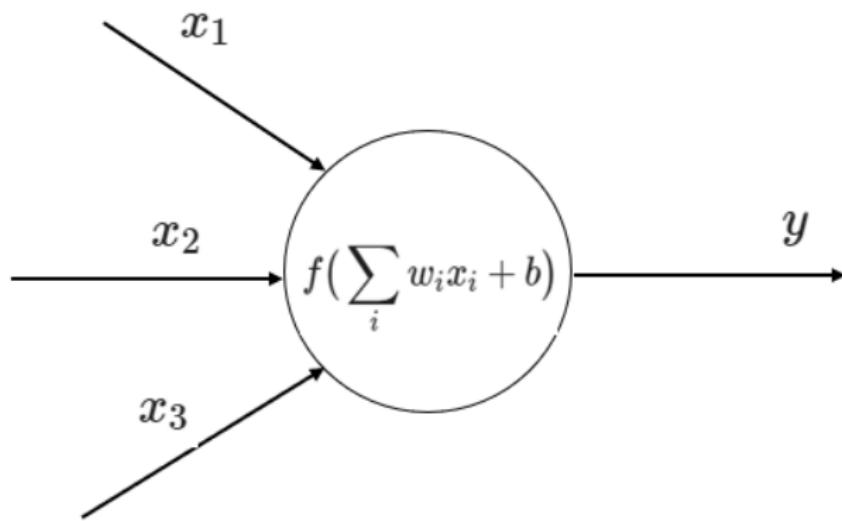
$$z_i = \sum_{j=1}^m w_{ij}x_j + b_i$$

$$a_i = f(z_i)$$

où :

- $x_j$  représente l'entrée du neurone,
- $w_{ij}$  est le poids associé à l'entrée  $x_j$ ,
- $b_i$  est le biais du neurone,
- $f$  est la fonction d'activation,
- $z_i$  est le potentiel d'action pré-synaptique,
- $a_i$  est la sortie activée du neurone.

## Illustration d'un neurone artificiel



# Fonctions d'activation

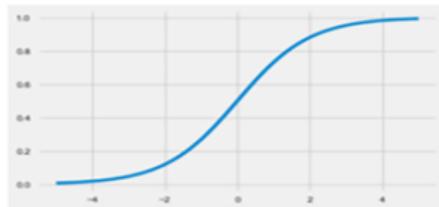
Les fonctions d'activation permettent aux modèles d'apprendre des relations plus complexes en capturant les non-linéarités dans les données.

- **Sigmoïde** :  $\sigma(z) = \frac{1}{1+e^{-z}}$ , plage de sortie (0, 1), utilisée pour la probabilité dans la classification binaire.
- **Tangente Hyperbolique (tanh)** :  $\tanh(z)$ , plage de sortie (-1, 1), version centrée et normalisée de la sigmoïde.
- **ReLU (Unité Linéaire Rectifiée)** :  $f(z) = \max(0, z)$ , non saturante, favorise la convergence rapide et permet d'éviter le problème de disparition des gradients.
- **Leaky ReLU** :  $f(z) = \max(\alpha z, z)$ , variante de ReLU qui permet un petit gradient lorsque  $z < 0$ .
- **Softmax** : Utilisée pour la couche de sortie des problèmes de classification multi-classes.

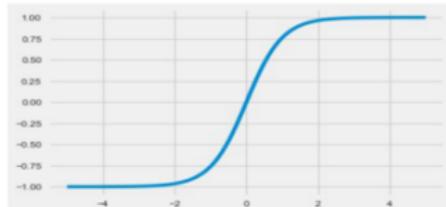
$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

# Illustration des fonctions d'activation

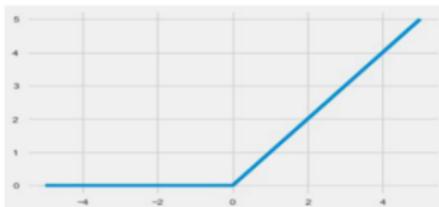
Sigmoïde



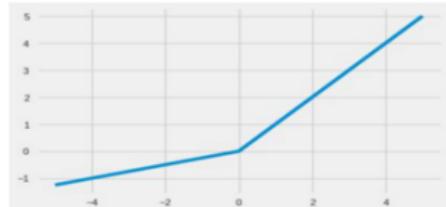
Tanh



ReLU



ReLU paramétrique



# Entraînement d'un réseau de neurones artificiel

**Principe d'Entraînement :** L'entraînement d'un réseau de neurones consiste à ajuster ses poids pour minimiser une fonction de coût qui mesure l'erreur entre les prédictions et les vraies valeurs.

## Descente de gradient stochastique (SGD) :

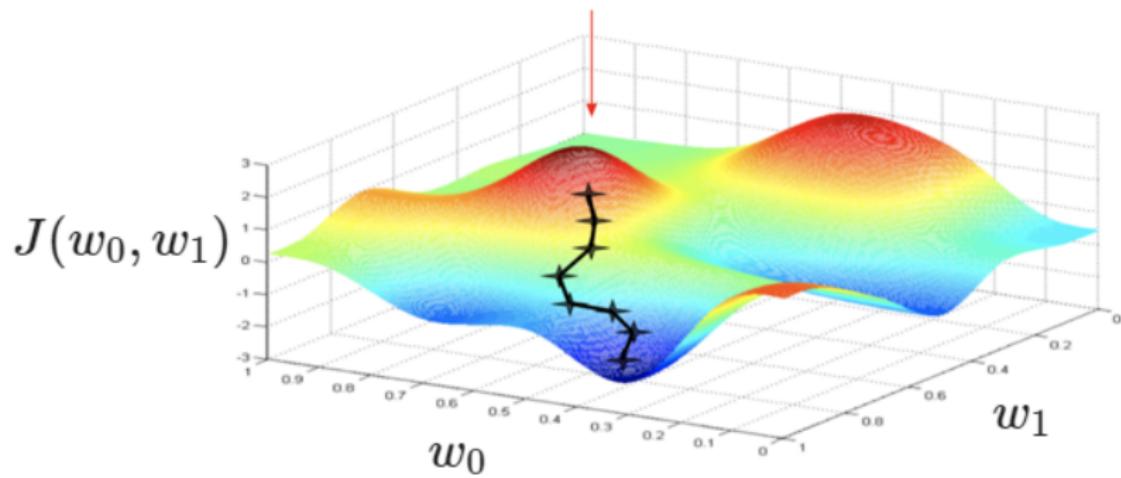
- Méthode d'optimisation utilisée pour mettre à jour les poids du réseau de manière itérative.
- À chaque itération, un sous-ensemble (batch) de données est utilisé pour calculer le gradient de la fonction de coût.
- Les poids sont mis à jour dans la direction opposée du gradient pour réduire l'erreur.

$$w_{new} = w_{old} - \eta \cdot \nabla_w J(w)$$

où :

- $w_{old}$  et  $w_{new}$  sont les valeurs des poids avant et après la mise à jour,
- $\eta$  est le taux d'apprentissage,
- $\nabla_w J(w)$  est le gradient de la fonction de coût par rapport aux poids.

# Illustration graphique de la SGD



# Entraînement des réseaux de neurones profonds

Dans les réseaux de neurones profonds, l'erreur calculée à la sortie du réseau dépend indirectement des poids des couches cachées. Ce lien indirect rend difficile de savoir comment ajuster ces poids pour réduire l'erreur.

## Implications pour l'Entraînement :

- **Propagation de l'erreur** : Sans un mécanisme pour propager l'erreur de la sortie vers les couches antérieures, il est impossible de déterminer l'impact de chaque poids sur l'erreur finale.
- **Complexité de l'ajustement des poids** : Chaque poids dans les couches cachées affecte l'erreur de sortie de manière complexe, nécessitant une méthode précise pour leur ajustement.



# Rétropropagation du gradient

Il est facile de calculer les gradients correspondant à la dernière couche  $\frac{\partial J}{\partial W^{(n)}}$  car l'erreur  $J$  dépend immédiatement de  $W^{(n)}$ .



Examinons maintenant le cas de la couche  $n - 1$  :

$$\frac{\partial J}{\partial W^{(n-1)}} = \frac{\partial J}{\partial W^{(n)}} \frac{\partial W^{(n)}}{\partial O_n} \frac{\partial O_n}{\partial W^{(n-1)}}$$

Or

$$\frac{\partial O_n}{\partial W^{(n-1)}} = \frac{\partial O_n}{\partial O_{n-1}} \frac{\partial O_{n-1}}{\partial W^{(n-1)}}$$

# Représentations distribuées

# Introduction aux Embeddings

## • Qu'est-ce qu'un Embedding ?

- Un embedding est une représentation vectorielle d'un mot qui capture le contexte du mot dans un document, des relations sémantiques et syntaxiques avec d'autres mots.
- Ils transforment des mots en vecteurs de nombres pour que les algorithmes de machine learning puissent les traiter efficacement.

## • Pourquoi sont-ils importants ?

- Les embeddings permettent de capturer non seulement l'identité d'un mot mais aussi ses aspects sémantiques et contextuels.
- Ils facilitent des tâches telles que la classification de texte, la traduction automatique, et la détection des sentiments.

## • Évolution des embeddings

- Historiquement, les mots étaient représentés comme des indices ou des vecteurs one-hot, où chaque mot est indépendant des autres.
- Les embeddings modernes, tels que Word2Vec et GloVe, représentent les mots dans des espaces vectoriels continus où les mots de sens similaires sont proches les uns des autres.

# Différence entre One-hot Encoding et Embeddings

## • One-hot Encoding

- Chaque mot est représenté par un vecteur avec un '1' dans la position qui lui est propre et des '0' partout ailleurs. Ce type de représentation est simple mais très inefficace en termes d'espace et ne capture pas les relations entre les mots.
- Exemple : pour un vocabulaire de dimension 100 000 :

Véhicule = [0, 0, 1, 0, 0, 0, 0, 0, ..., 0, 0]

Voiture = [0, 0, 0, 0, 0, 1, 0, 0, 0, ..., 0, 0]

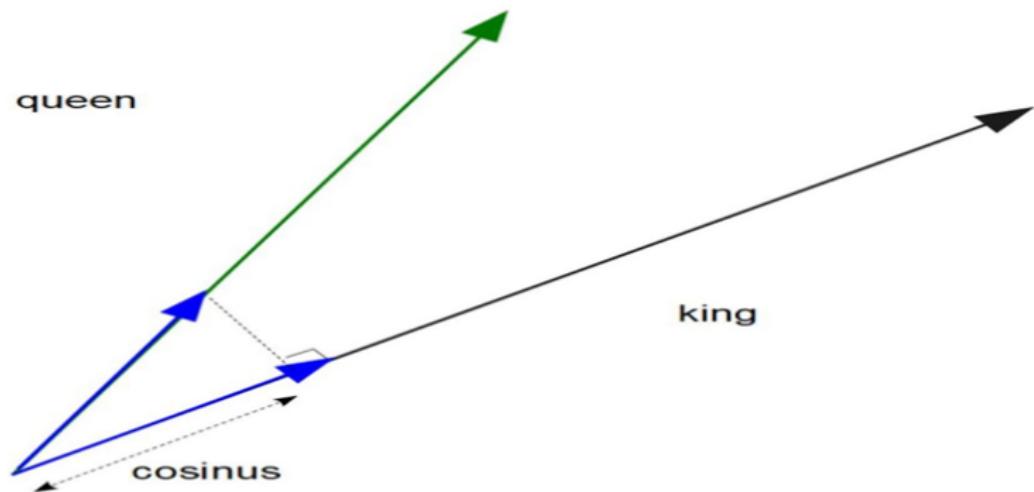
Cette représentation ne permet pas de prendre en compte la dimension sémantique

## • Embeddings

- Les embeddings représentent les mots comme des vecteurs denses de nombres flottants (généralement entre 50 et 300 dimensions).
- Cette représentation est beaucoup plus riche et peut capturer des relations complexes entre les mots, comme la similarité sémantique.

# Similarité sémantique

Nous sommes intéressés par des représentations de mots qui capturent la distance sémantique, sous forme de produit scalaire par exemple (ou distance cosinus).



# Représentations distribuées : Principes

*“You shall know a word by the company it keeps”*

— J.R. Firth (1957)

Les mots sont similaires s'ils apparaissent fréquemment dans le même contexte.

- Il conduit son *véhicule* pour rentrer à la maison.
- Il conduit sa *voiture* pour rentrer chez lui.

# Construction d'une représentation distribuée

Considérons le corpus suivant à titre d'exemple : **cnn in crop analysis**

**cnn and svm are widely used.**

**linear\_regression performed along with svm**

**linear\_regression for crop and farm**

**svm being used for farm monitoring**

**Do cnn, svm and linear\_regression appear in the same context?**

Le vocabulaire est alors :

[cnn, in, crop, analysis, and, svm, are, widely, used,  
linear\_regression, performed, along, with, for, farm, being,  
monitoring, do, appear, the, same, context]

$$\dim(\text{vocabulaire}) = 22$$

# Similarité sémantique

La matrice de co-occurrence représente la fréquence à laquelle les mots apparaissent ensemble deux à deux.

cnn in crop ...

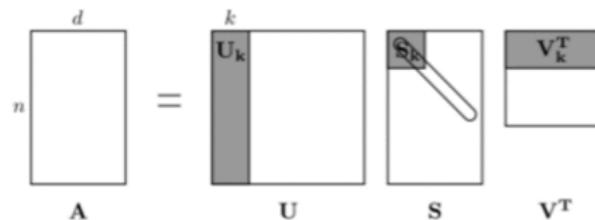
```
cnn [[0, 2, 1, 1, 1, 2, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1],  
in [2, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1],  
crop [1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
... [1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[2, 1, 0, 0, 1, 0, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],  
[1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[1, 0, 0, 0, 1, 2, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0],  
[1, 1, 1, 0, 1, 2, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1],  
[0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 2, 0, 1, 1, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0],  
[1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1],  
[1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1],  
[1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1],  
[1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1],  
[1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1],  
[1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0]]]
```

# Réduction de la dimension

La décomposition en valeurs singulières est une technique permettant de réduire la dimension des données (similaire à l'ACP).

Étant donné une matrice  $A \in \mathbb{R}^{n \times d}$

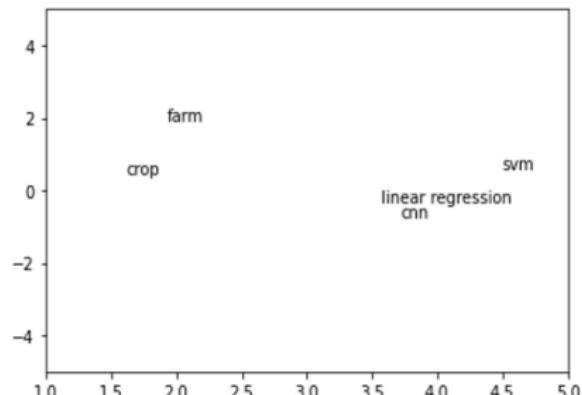
$$A = UDV^T \quad \text{où} \quad U \in \mathbb{R}^{n \times r}, D \in \mathbb{R}^{r \times r}, V \in \mathbb{R}^{d \times r}.$$



$U$  est la matrice contenant les représentations (vecteurs de mots)

# Visualisation des représentations distribuées

```
cnn :[ 3.72113518, -0.73233585],  
ln [ 2.7940387 , -1.40864784],  
crop [ 1.61178082,  0.44786021],  
... [ 0.77784994, -0.31230389],  
[ 2.12245048,  1.29571556],  
[ 4.49293777,  0.58090417],  
[ 1.33312748,  0.66758743],  
[ 1.33312748,  0.66758743],  
[ 2.25882809,  1.80738544],  
[ 3.56593605, -0.31006976],  
[ 0.95394179,  0.079159 ],  
[ 0.95394179,  0.079159 ],  
[ 0.95394179,  0.079159 ],  
[ 0.95394179,  0.079159 ],  
[ 1.92921553,  1.94783497],  
[ 1.92921553,  1.94783497],  
[ 1.12301312,  1.42126096],  
[ 1.12301312,  1.42126096],  
[ 2.26025282, -1.31571175],  
[ 2.26025282, -1.31571175],  
[ 2.26025282, -1.31571175],  
[ 2.26025282, -1.31571175],  
[ 2.26025282, -1.31571175]]
```



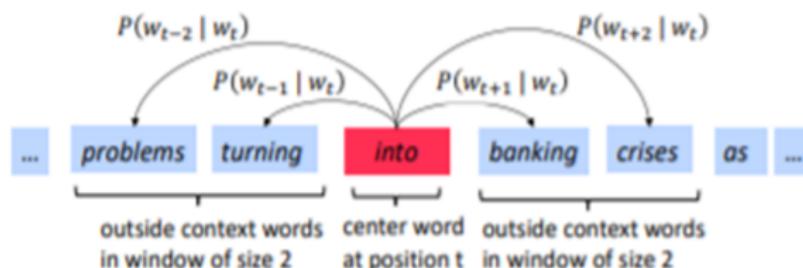
## Word2Vec : Principles

- **Principes de base :**

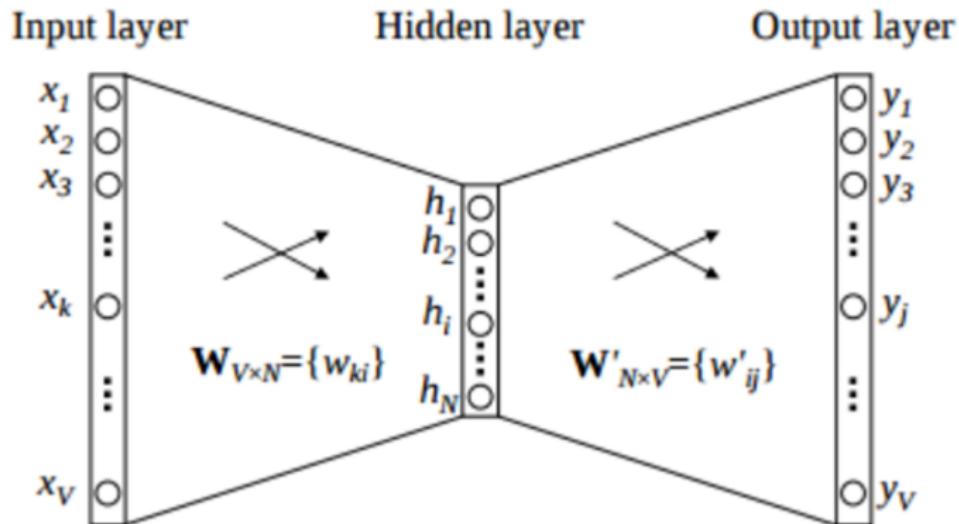
- Word2Vec est basé sur l'hypothèse distributionnelle : les mots qui apparaissent dans des contextes similaires ont des significations similaires.
  - Utilise un réseau de neurones peu profond pour apprendre des embeddings de mots à partir de grands corpus.

- Deux architectures principales :

- ① *Continuous Bag of Words (CBOW)* : Prédit le mot cible à partir du contexte.
  - ② *Skip-Gram* : Prédit le contexte à partir du mot cible.



# Architecture du modèle CBOW



# Modèle CBOW : Formulation mathématique

- **Objectif** : Prédire le mot cible  $w_t$  en fonction du contexte  $C$ .
- **Fonction de prédiction** :

$$P(w_t|C) = \frac{e^{\mathbf{v}_{w_t}^T \cdot \mathbf{h}}}{\sum_{w \in W} e^{\mathbf{v}_w^T \cdot \mathbf{h}}} \quad (1)$$

- **Où** :
  - $\mathbf{v}_w$  est le vecteur de l'embedding du mot  $w$ .
  - $\mathbf{h}$  est le vecteur du contexte, la moyenne des embeddings des mots du contexte.
  - $W$  est l'ensemble de tous les mots du vocabulaire.
- **Optimisation** : Minimisation de la fonction de coût, souvent une forme de cross-entropy ou log-likelihood négatif.

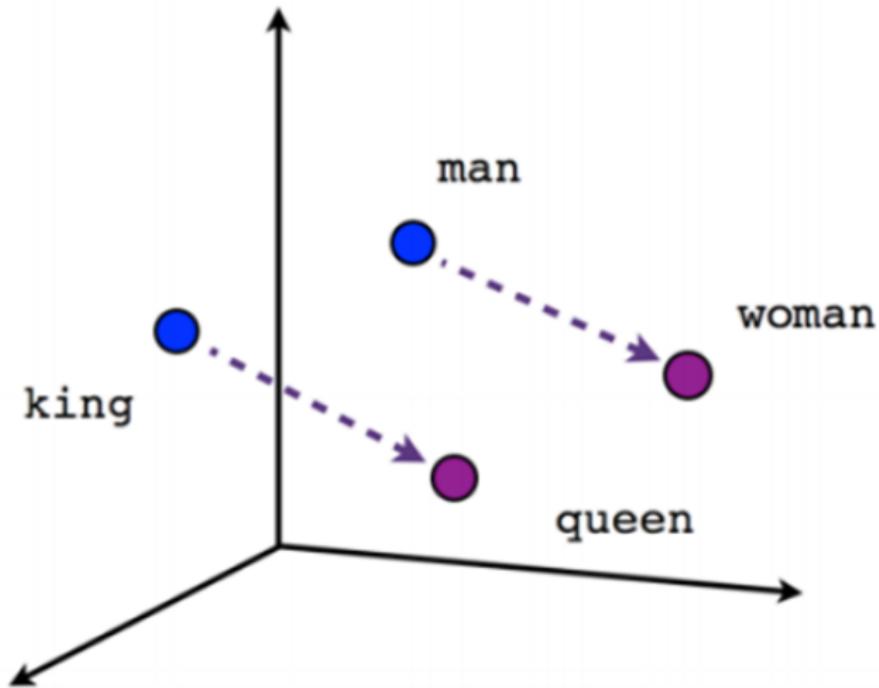
# Modèle Skip-Gram : Formulation mathématiques

- **Objectif** : Prédire les mots de contexte à partir du mot cible  $w_t$ .
- **Fonction de prédiction** :

$$P(C|w_t) = \prod_{w_c \in C} \frac{e^{\mathbf{v}_{w_c}^T \cdot \mathbf{h}}}{\sum_{w \in W} e^{\mathbf{v}_w^T \cdot \mathbf{h}}} \quad (2)$$

- **Où** :
  - $\mathbf{v}_{w_c}$  est le vecteur de l'embedding du mot de contexte  $w_c$ .
  - $\mathbf{h}$  est le vecteur de l'embedding du mot cible  $w_t$ .
- **Optimisation** : Minimisation de la fonction de coût, souvent une forme de cross-entropy ou log-likelihood négatif.

# Représentations Word2Vec



# Applications pratiques de Word2Vec

- **Analogie de mots :**

- Word2Vec est célèbre pour capturer des relations complexes, comme "homme est à femme ce que roi est à reine".
- Permet de résoudre des analogies en utilisant des opérations arithmétiques simples sur les vecteurs de mots.

- **Clustering sémantique :**

- Les embeddings peuvent être utilisés pour regrouper des mots sémantiquement similaires, facilitant l'analyse de textes volumineux.

- **Amélioration des systèmes de recommandation :**

- Les vecteurs de mots de Word2Vec peuvent être utilisés pour améliorer la précision des recommandations en comprenant mieux les préférences des utilisateurs.

# Autres représentations distribuées

Il existe d'autres représentations distribuées :

- Glove (2014) : proposé par une équipe de Stanford, il combine à la fois les techniques modernes de deep learning et les techniques statistiques (co-occurrences entre les mots). Il permet d'avoir des représentations des mots plus globales que Word2Vec.
- Fasttext (à partir de 2016) : la librairie a été créée par Facebook. Le modèle est entraîné sur des subwords (n-grams de caractères). Il est ainsi plus efficace pour traiter les mots inconnus (out of vocabulary).
- Représentations contextuelles (Transformers), etc.

# Réseaux de neurones récurrents

# Perspective historique des RNN

Les Réseaux de Neurones Récurrents (RNN) ont évolué au fil du temps, marquant des étapes importantes dans le domaine du machine learning et du NLP.

## Développements clés :

- **1980s : Naissance des RNN** - Introduction des concepts de base des RNN par John Hopfield et David Rumelhart, posant les fondations des réseaux à mémoire.
- **1990s : Popularisation des RNN** - Jordan et Elman développent des architectures permettant une meilleure gestion des séquences temporelles et des dépendances.
- **Début des années 2000** - Identification des problèmes de disparition et d'explosion du gradient, limitant l'efficacité des RNN sur de longues séquences.
- **1997 : Avènement des LSTM** - Introduction des LSTM par Hochreiter et Schmidhuber, offrant une solution aux problèmes de mémoire à long terme.
- **2010s : GRU** - Les GRU simplifient la structure des LSTM. Les RNN sont de plus en plus utilisés dans des applications complexes comme la traduction automatique et la reconnaissance vocale.

# Introduction aux RNN

Les Réseaux de Neurones Récurrents (RNN) sont une classe de réseaux de neurones artificiels spécialement conçus pour traiter des séquences de données, tels que des séries temporelles ou des séquences textuelles.

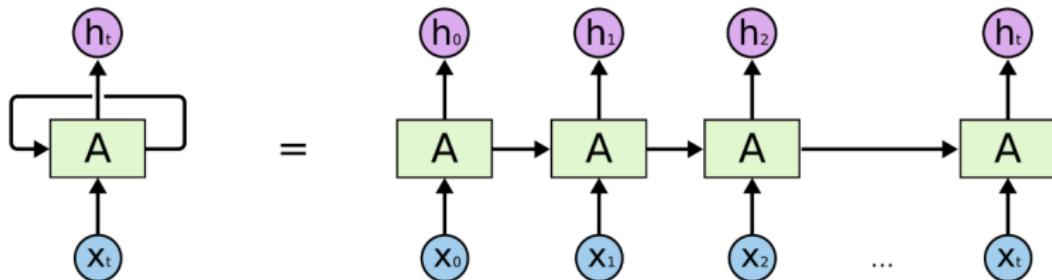
## Caractéristiques:

- **Mémoire à court terme** : Les RNN ont la capacité de se "souvenir" d'informations passées grâce à leurs connexions récurrentes.
- **Traitement de séquences** : Ils sont particulièrement adaptés pour des tâches où les données sont séquentielles et où le contexte est important (ex: langage naturel, musique).
- **Modélisation de dépendances temporelles** : Les RNN peuvent capturer des dépendances temporelles et contextuelles dans les données.

# Architecture des RNN

Cette architecture représente un RNN à la fois dans sa forme condensée et dépliée à travers le temps.

- À gauche, le RNN est présenté de manière condensée avec:
  - Une entrée  $x_t$ .
  - Une sortie d'état caché  $h_t$ .
  - Un bloc  $A$  représentant la cellule RNN qui effectue les calculs à partir de l'état caché précédent et de l'entrée courante pour produire le nouvel état caché.
- À droite, l'architecture est déroulée pour montrer la séquence complète:
  - Chaque bloc  $A$  est le même RNN à différents instants temporels.
  - Le bloc  $A$  prend une entrée  $x_t$  et produit un état caché  $h_t$ , qui est alors passé à la même cellule au pas de temps suivant.
  - L'état initial  $h_0$  est souvent initialisé à zéro ou une petite valeur aléatoire.



# Problème de disparition du gradient

Le problème de disparition du gradient survient lors de la rétropropagation dans les RNN traditionnels. Les gradients des paramètres peuvent devenir très petits, rendant l'apprentissage des dépendances à long terme difficile.

Mathématiquement, pendant la rétropropagation, si les valeurs de  $\frac{\partial h_t}{\partial W}$  sont petites, le gradient total peut tendre vers zéro à mesure que  $T$  augmente.

## Solutions au problème de disparition du gradient :

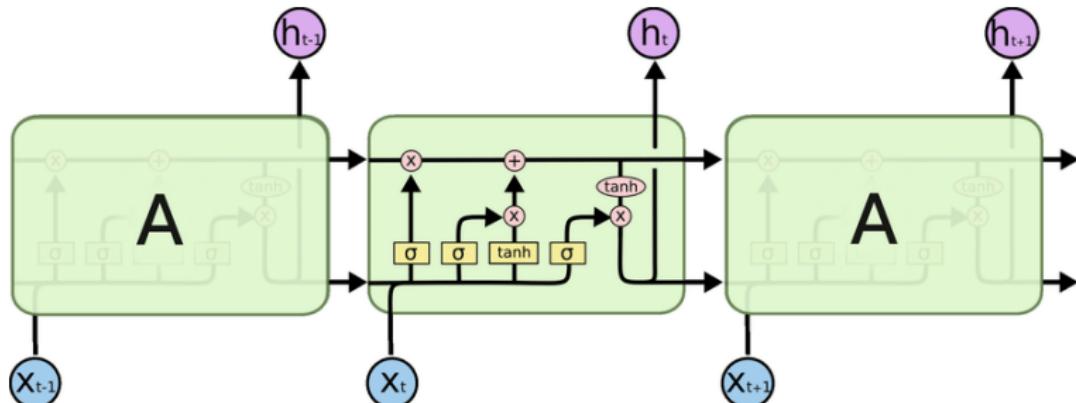
Pour atténuer le problème de disparition du gradient, différentes solutions ont été proposées :

- Utilisation de fonctions d'activation comme ReLU au lieu de sigmoïde ou tanh.
- Introduction d'architectures RNN avancées comme LSTM et GRU.
- Techniques de clipping de gradient pour éviter l'explosion des gradients.

# RNN avec Long Short-Term Memory (LSTM)

Les LSTM sont une variante des RNN conçus pour mieux capturer les dépendances à long terme. Ils introduisent des 'cellules mémoire' avec des portes de régulation :

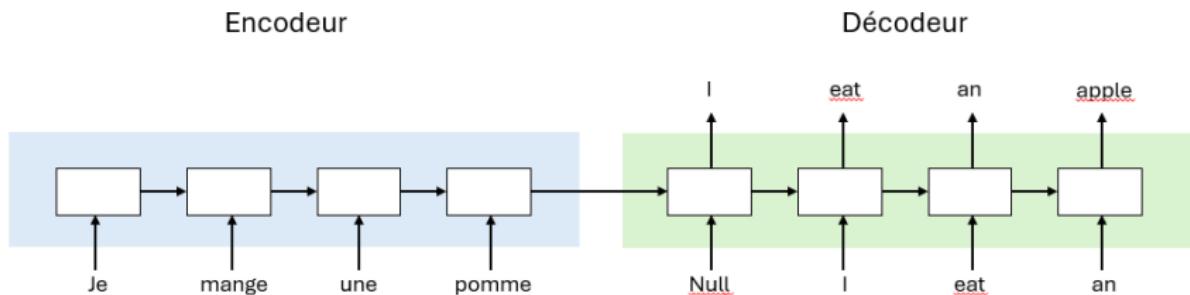
- Porte d'oubli : contrôle la quantité d'informations à retenir de l'état précédent.
- Porte d'entrée : contrôle la quantité d'informations à ajouter de l'entrée actuelle.
- Porte de sortie : détermine la quantité d'informations à transmettre à l'état suivant.



# Modèles Seq2Seq

L'architecture seq-to-seq comprend deux composants principaux :

- **Encodeur** : Un réseau de neurones qui lit et encode la séquence d'entrée en un vecteur de contexte (une représentation dense de la séquence).
- **Décodeur** : Un autre réseau de neurones qui lit le vecteur de contexte pour générer la séquence de sortie, un élément à la fois.



# Importance des modèles Seq-to-Seq

Les modèles seq-to-seq ont révolutionné la façon dont les machines comprennent et génèrent le langage naturel, avec des impacts significatifs dans plusieurs domaines :

- **Traduction automatique** : Ils constituent la base des systèmes de traduction automatique de pointe, permettant une traduction fluide et naturelle entre les langues.
- **Résumé automatique** : Permettent de générer des résumés concis et pertinents de longs documents ou articles.
- **Agents conversationnel** : Facilitent la création de systèmes de dialogue intelligents capables de mener des conversations naturelles avec les utilisateurs.
- **Reconnaissance et génération de la parole** : Appliqués à la conversion de la parole en texte et vice-versa, améliorant l'accessibilité et l'interaction homme-machine.

# Défis des modèles Seq-to-Seq

Malgré leur succès, les modèles seq-to-seq rencontrent plusieurs défis :

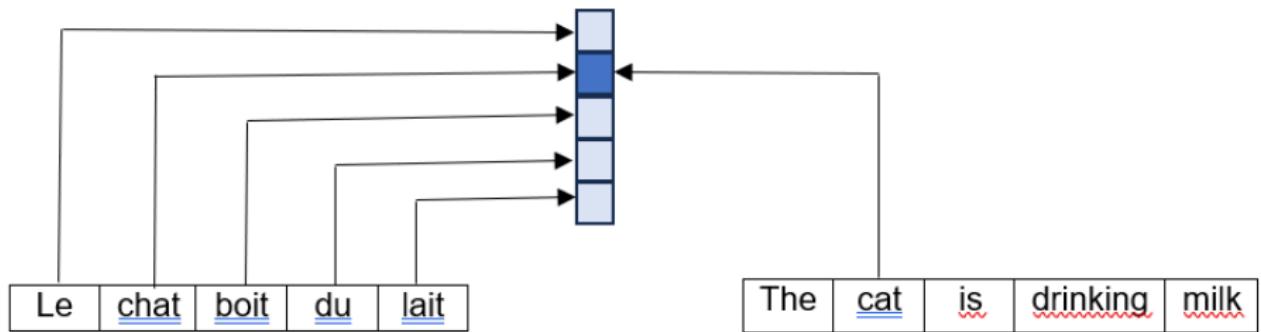
- **Gestion de la Longueur** : Difficulté à maintenir la performance sur de très longues séquences due à la dilution de l'information.
- **Complexité Computationnelle** : Les opérations récurrentes sont coûteuses en termes de calcul et de mémoire.
- **Nécessité d'Attention** : Introduction de mécanismes d'attention pour améliorer la focalisation sur les parties pertinentes de l'entrée.
- **Dépendance au Contexte** : Les modèles peuvent avoir du mal à comprendre le contexte complet ou les nuances subtiles du langage.

# Introduction au mécanisme d'Attention

Le mécanisme d'attention est une amélioration clé apportée aux modèles seq-to-seq traditionnels, permettant au modèle de se "concentrer" sur différentes parties de la séquence d'entrée lors de la génération de la séquence de sortie.

- **Objectif** : Surmonter les limitations des encodeurs seq-to-seq qui compressent toute l'information d'une séquence d'entrée dans un vecteur de contexte fixe.
- **Avantage** : Améliore la capacité du modèle à gérer de longues séquences d'entrée, en rendant le processus de génération de la séquence de sortie plus dynamique et contextuellement informé.

# Illustration du mécanisme d'Attention



# Types d'Attention

Il existe plusieurs variantes du mécanisme d'attention, chacune avec ses propres caractéristiques et applications :

- **Attention globale** : Le décodeur considère tous les états cachés de l'encodeur simultanément pour générer le vecteur de contexte.
- **Attention locale** : Le décodeur ne considère qu'une sous-séquence des états cachés de l'encodeur autour d'une position centrale pour chaque étape de décodage.
- **Self Attention** : Utilisée dans les transformateurs, cette forme d'attention permet à chaque position dans une séquence d'entrée de considérer toutes les positions et de s'auto-pondérer.

Chaque type d'attention est adapté à différentes tâches et configurations de modèle.

## Attention Is All You Need

**Ashish Vaswani\***  
Google Brain  
avaswani@google.com

**Noam Shazeer\***  
Google Brain  
noam@google.com

**Niki Parmar\***  
Google Research  
nikip@google.com

**Jakob Uszkoreit\***  
Google Research  
usz@google.com

**Llion Jones\***  
Google Research  
llion@google.com

**Aidan N. Gomez\* †**  
University of Toronto  
aidan@cs.toronto.edu

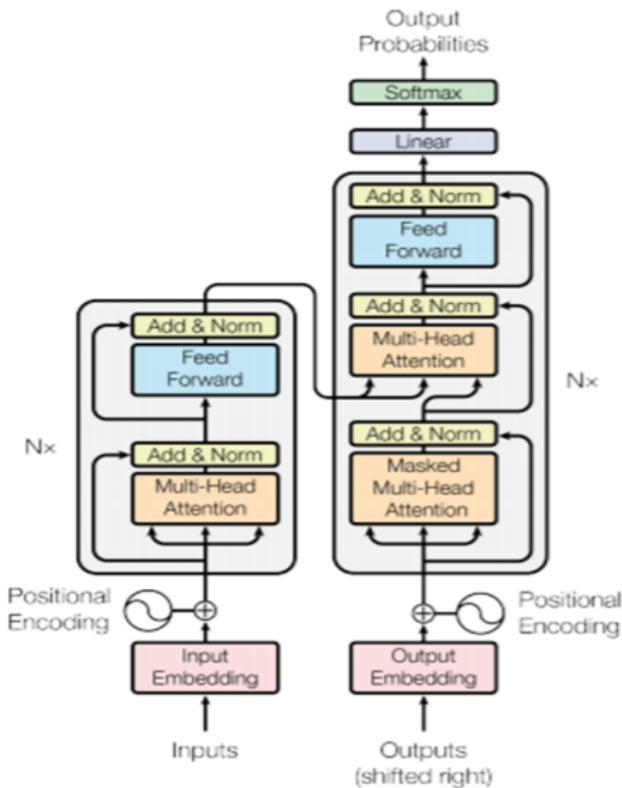
**Lukasz Kaiser\***  
Google Brain  
lukasz.kaiser@google.com

**Illia Polosukhin\* ‡**  
illia.polosukhin@gmail.com

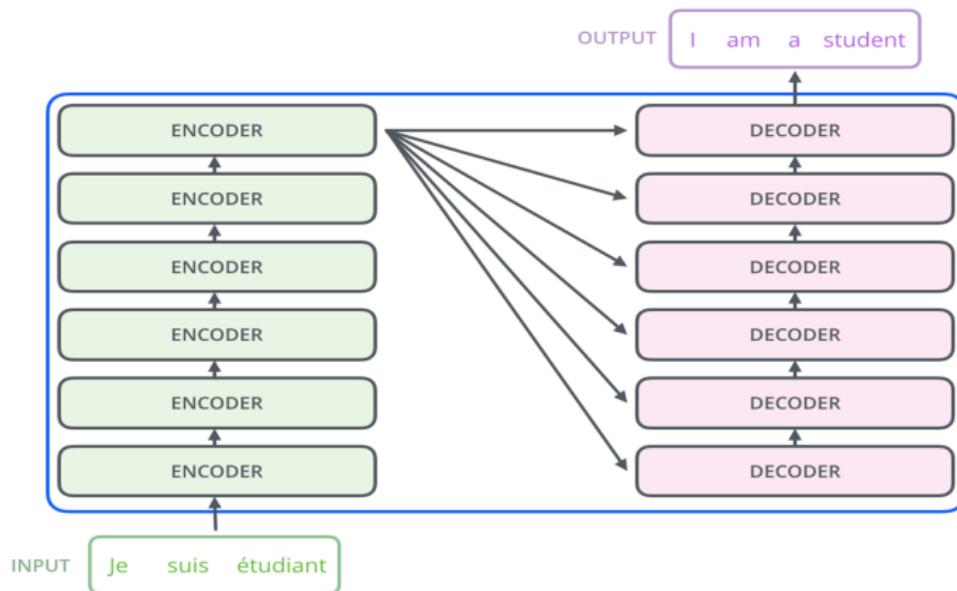
### Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to

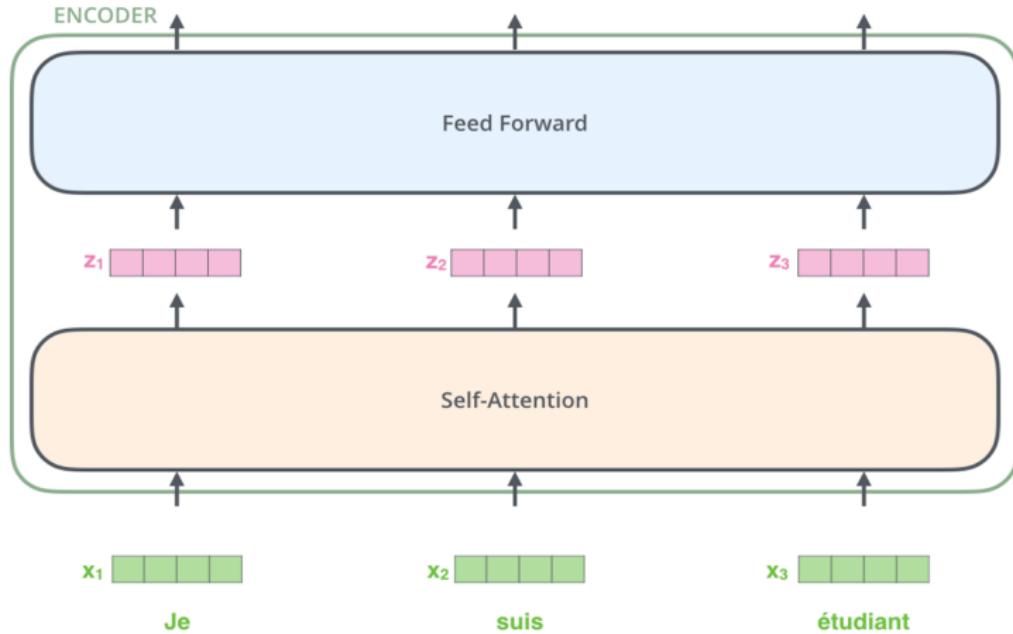
# Transformer originel



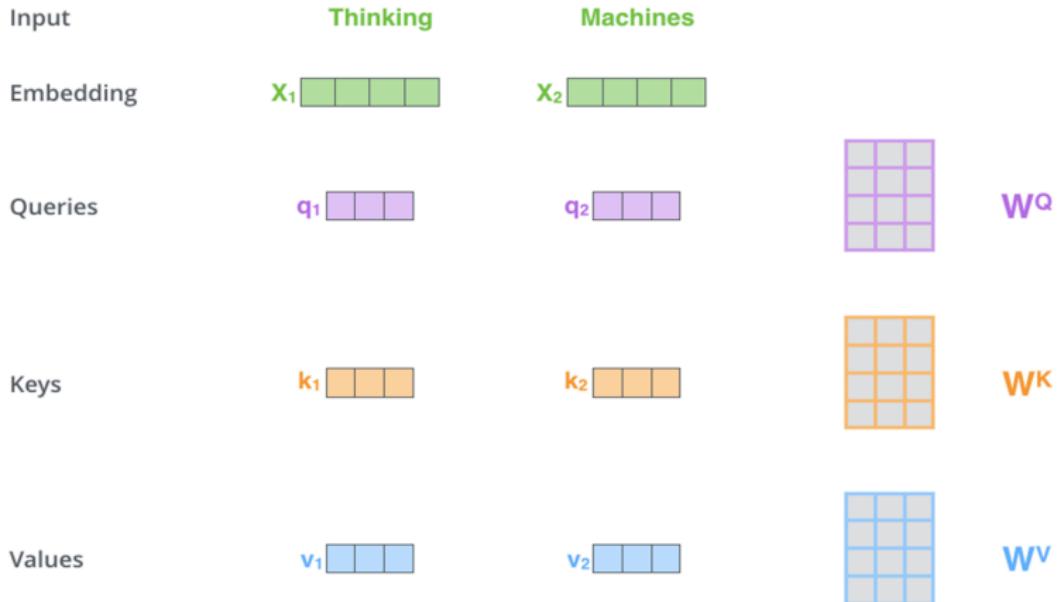
# Mécanisme de cross-attention



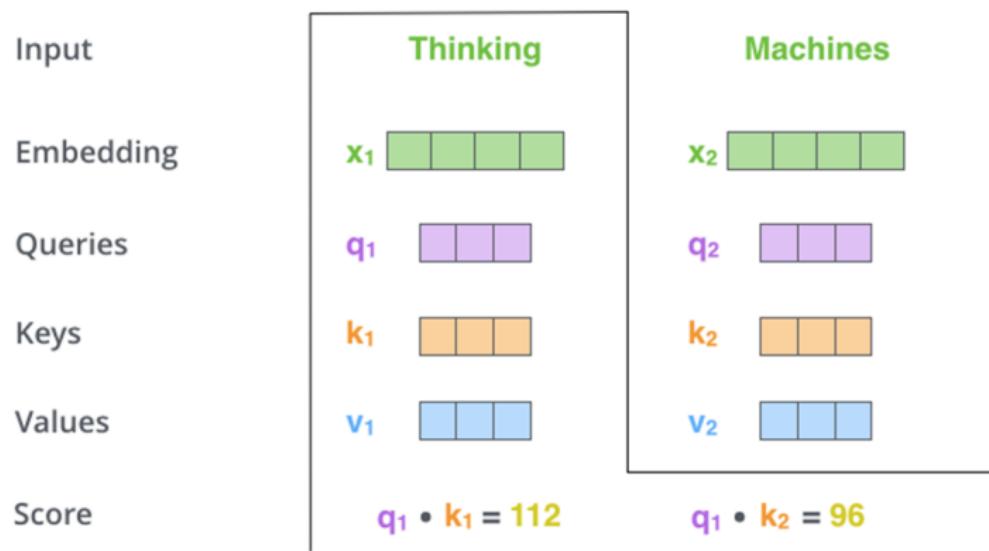
## Mécanisme de self-attention



# Fonctionnement du mécanisme de self-attention



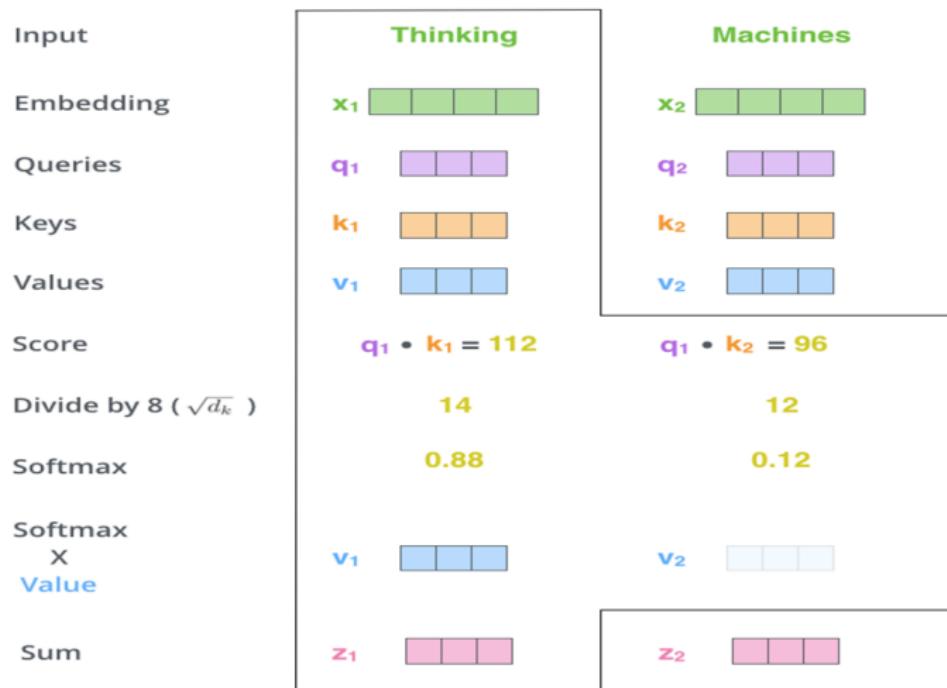
# Calcul des scores de self-attention



# Calcul des scores de self-attention

Input	Thinking		Machines	
Embedding	$x_1$	[4 green boxes]	$x_2$	[4 green boxes]
Queries	$q_1$	[3 purple boxes]	$q_2$	[3 purple boxes]
Keys	$k_1$	[3 orange boxes]	$k_2$	[3 orange boxes]
Values	$v_1$	[3 blue boxes]	$v_2$	[3 blue boxes]
Score	$q_1 \bullet k_1 = 112$		$q_1 \bullet k_2 = 96$	
Divide by 8 ( $\sqrt{d_k}$ )	14		12	
Softmax	0.88		0.12	

# Calcul de la nouvelle représentation



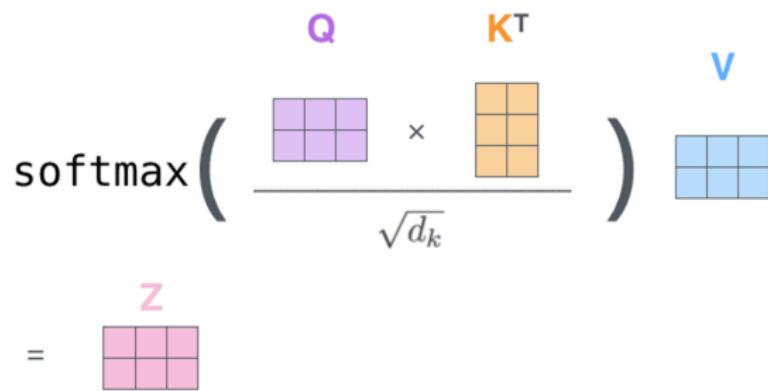
# Formulation matricielle

$$\mathbf{X} \times \mathbf{W}^Q = \mathbf{Q}$$

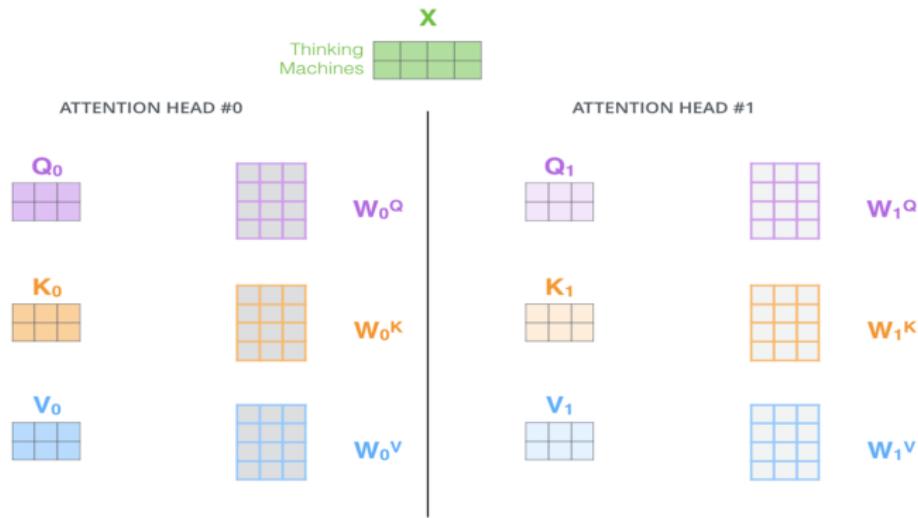
$$\mathbf{X} \times \mathbf{W}^K = \mathbf{K}$$

$$\mathbf{X} \times \mathbf{W}^V = \mathbf{V}$$

# Formulation matricielle du mécanisme de self attention

$$\text{softmax} \left( \frac{\begin{matrix} \mathbf{Q} & \times & \mathbf{K}^T \\ \begin{matrix} \text{---} \end{matrix} & \quad & \begin{matrix} \text{---} \end{matrix} \end{matrix}}{\sqrt{d_k}} \right) \mathbf{V}$$
$$= \mathbf{Z}$$


# Multihead attention

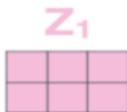


# Multihead attention

ATTENTION  
HEAD #0



ATTENTION  
HEAD #1



...

ATTENTION  
HEAD #7



$\times$



# Positional Encoding dans les Transformers

## Pourquoi le Positional Encoding ?

- Les transformers n'ont pas de mécanisme intégré pour comprendre la position des tokens dans une séquence.
- Contrairement aux RNN ou CNN, ils traitent les tokens en parallèle sans ordre naturel.
- Le Positional Encoding est introduit pour ajouter une information de position dans la séquence.

## Comment fonctionne-t-il ?

- Chaque position dans la séquence est encodée à l'aide de fonctions trigonométriques (sinus et cosinus).
- Les vecteurs d'encodage de position sont ajoutés aux embeddings des tokens.
- Formules utilisées :

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \quad (3)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \quad (4)$$

# Tokénisation dans les LLM

## Qu'est-ce que la tokénisation ?

- La tokénisation consiste à diviser un texte en unités plus petites appelées **tokens**.
- Les tokens peuvent être des mots, des sous-mots ou même des caractères.
- La tokénisation est une étape essentielle avant d'introduire le texte dans un modèle de langage comme les LLM.

## Pourquoi est-elle importante ?

- Les modèles ne peuvent pas traiter directement du texte brut, ils travaillent avec des séquences de tokens.
- Une bonne tokénisation influence la qualité des prédictions en capturant correctement le sens des phrases.

# Approches de Tokénisation dans les LLM

## Types de Tokénisation :

- **Word-level tokenization** : Chaque mot est un token. Inconvénient : vocabulaire très large.
- **Subword-level tokenization** : Divise les mots en sous-unités fréquentes. Exemples : BPE (Byte Pair Encoding), WordPiece.
- **Character-level tokenization** : Chaque caractère est un token. Utilisé pour les langues à alphabet complexe.

## Pourquoi les sous-mots sont-ils populaires ?

- **Équilibre entre taille du vocabulaire et généralisation** : Gère bien les mots rares ou nouveaux.
- **Flexibilité linguistique** : Utilisé dans les LLM comme GPT, BERT pour traiter des textes multilingues.

# Byte Pair Encoding (BPE)

## Qu'est-ce que le BPE ?

BPE est une méthode de **tokénisation par sous-mots**. Elle est utilisée pour diviser les mots en sous-unités fréquentes. Cette approche permet de réduire la taille du vocabulaire tout en préservant la capacité de généralisation.

## Exemple d'application du BPE :

- Texte initial : low, lower
- Étape 1 : Diviser chaque mot en caractères :  
l o w, l o w e r
- Étape 2 : Identifier la paire de caractères la plus fréquente.  
La paire o w est la plus fréquente.
- Étape 3 : Combiner les caractères fréquents :  
l ow, l ow e r
- Étape 4 : Répéter avec la paire suivante la plus fréquente :  
La paire l ow est maintenant la plus fréquente.  
Résultat : low, low e r

## Avantages de BPE :

- Gère les mots rares ou inconnus en les décomposant en sous-mots fréquents.
- Diminue la taille du vocabulaire tout en capturant la structure des mots.

# Performances du tranformer originel

Entraîné sur WMT 2014 English-German dataset, comprenant près de 4.5 millions de phrases sentence, le WMT 2014 English-French dataset, comprenant près de 36 millions de phrases.

- 8 NVIDIA P10 GPUs
- **Base** : 12 heures
- **Large** : 3.5 jours

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	<b>41.29</b>	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	<b><math>3.3 \cdot 10^{18}</math></b>	
Transformer (big)	<b>28.4</b>	<b>41.8</b>	$2.3 \cdot 10^{19}$	

# BERT

BERT (Bidirectional Encoder Representations from Transformers) représente une avancée majeure dans le NLP, introduisant une approche novatrice pour la modélisation de langage.

- **Contextualisation profonde** : Première architecture à utiliser pleinement la bidirectionnalité pour comprendre le contexte des mots, permettant une compréhension plus fine du langage.
- **Pré-entraînement et fine-tuning** : Méthodologie en deux étapes offrant une flexibilité pour l'adaptation à diverses tâches de TALN sans architecture spécifique à la tâche.
- **Impact sur le NLP** : Avant BERT, les approches de modélisation de langage étaient limitées par la compréhension unidirectionnelle ou des représentations statiques des mots. BERT a changé la donne en permettant des représentations contextuelles dynamiques.
- **Performances révolutionnaires** : À son introduction, BERT a établi de nouveaux standards de performance sur une gamme de benchmarks de NLP, y compris GLUE, SQuAD, etc.

# Pré-entraînement de BERT

Objectifs de pré-entraînement :

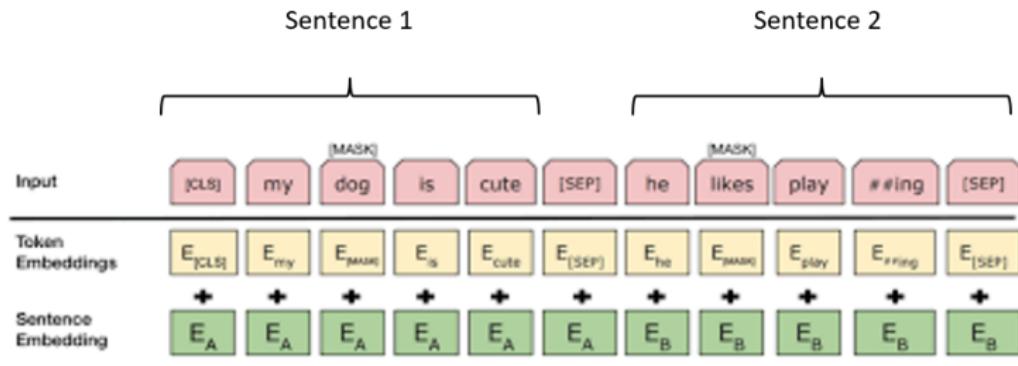
- **Masquage de token (MLM)** : 15% des tokens masqués aléatoirement, prédits par le modèle.

$$L_{\text{MLM}} = - \log P(\text{token masqué} | \text{contexte}) \quad (5)$$

- **Prédiction de la prochaine phrase (NSP)** : Déterminer si une phrase B suit logiquement une phrase A.

$$L_{\text{NSP}} = - [y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})] \quad (6)$$

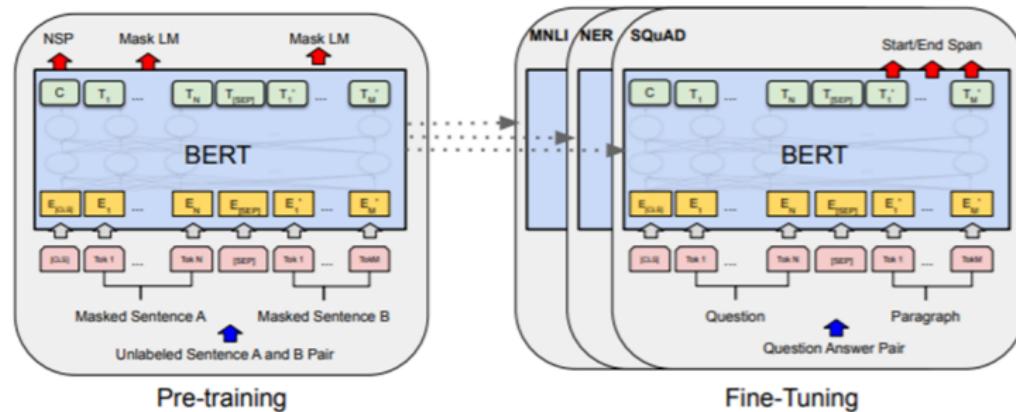
- Combinaison des pertes pour l'optimisation.



# Fine-tuning de BERT pour des tâches spécifiques

Après pré-entraînement, BERT est affiné pour des tâches spécifiques:

- Ajout d'une couche de sortie spécifique à la tâche (classification, NER, QA).
- Fine-tuning de tous les paramètres du modèle pré-entraîné sur le corpus de la tâche.



# Performances de BERT

BERT a été pré-entraîné sur le BookCorpus (800 millions de mots) et English Wikipedia (2,500 millions de mots).

Deux modèles :

- **BERT Base** : 12 couches avec 110 millions de paramètres.
- **BERT Large** : 24 couches avec 340 millions de paramètres.

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Average
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
BERT <sub>BASE</sub>	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT <sub>LARGE</sub>	<b>86.7/85.9</b>	<b>72.1</b>	<b>92.7</b>	<b>94.9</b>	<b>60.5</b>	<b>86.5</b>	<b>89.3</b>	<b>70.1</b>	<b>82.1</b>

# Introduction aux modèles autorégressifs - GPT-3

GPT-3 (Generative Pre-trained Transformer 3) est le modèle de langue développé par OpenAI, représentant la troisième génération de la série GPT. Avec 175 milliards de paramètres, GPT-3 pousse les limites de la génération de texte et de la compréhension du langage naturel.

- **Capacités générales** : GPT-3 excelle dans une variété de tâches de TALN sans fine-tuning spécifique, grâce à sa puissance de modélisation du langage.
- **Architecture autorégressive** : Utilise un modèle Transformer pour prédire le mot suivant dans un texte, apprenant des patterns complexes sur des données massives.
- **Approche few-shot learning** : Capable d'adapter ses réponses à des tâches spécifiques avec peu ou pas d'exemples d'entraînement.
- **Impact sur IA et le NLP** : GPT-3 a significativement avancé les capacités des systèmes d'IA dans la compréhension et la génération de langage naturel, ouvrant de nouvelles voies pour l'application de l'intelligence artificielle.

# Architecture de GPT-3

GPT-3 repose sur une architecture Transformer améliorée, optimisée pour traiter efficacement de grandes quantités d'informations et générer des réponses cohérentes et contextuellement pertinentes.

- **Taille du modèle :** Avec 175 milliards de paramètres, GPT-3 est l'un des modèles de langue les plus grands et les plus complexes à ce jour.
- **Mécanisme d'attention :** L'attention multi-têtes permet au modèle de pondérer différemment les parties d'un input, améliorant la compréhension du contexte.
- **Optimisation et entraînement :** Techniques d'optimisation avancées pour gérer la taille du modèle et l'efficacité de l'entraînement.
- **Formulation :**

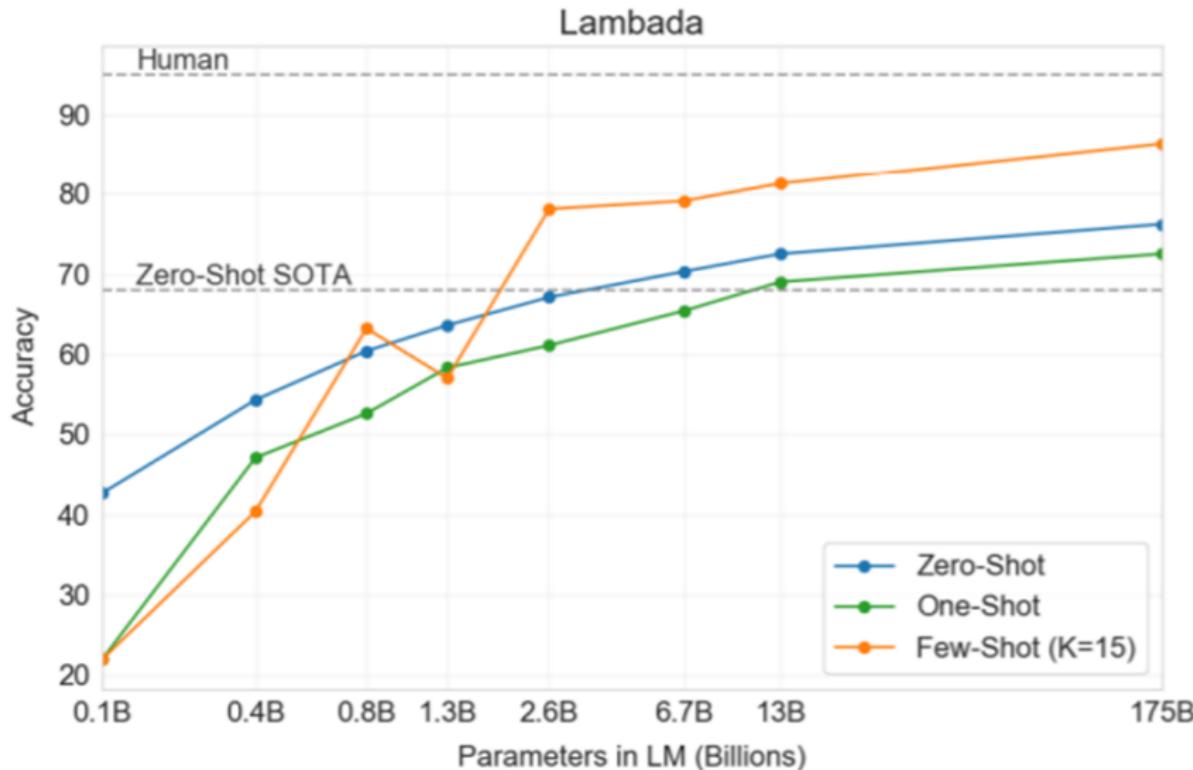
$$P(w_t|w_1, \dots, w_{t-1}) = \text{softmax}(\text{Transformer}(w_1, \dots, w_{t-1})). \quad (7)$$

# Few-Shot Learning avec GPT-3

L'une des innovations les plus remarquables de GPT-3 réside dans sa capacité à effectuer du few-shot learning, permettant au modèle de comprendre et d'exécuter des tâches spécifiques avec très peu d'exemples.

- **Définition :** Le few-shot learning désigne la capacité d'un modèle à apprendre une nouvelle tâche à partir d'un très petit nombre d'exemples d'entraînement, souvent seulement quelques-uns.
- **Mécanisme dans GPT-3 :**
  - GPT-3 utilise des prompts contenant quelques exemples de la tâche désirée pour guider le modèle sur ce qui est attendu, avant de présenter la question ou la tâche à résoudre.
  - Le modèle généralise ensuite à partir de ces exemples pour générer des réponses ou des solutions aux problèmes posés, montrant une compréhension étonnante de la tâche avec un minimum de guidance.
- **Exemples d'application :**
  - Classification de texte, génération de résumés, réponse à des questions spécifiques, et plus, avec seulement quelques exemples pour chaque tâche.
- **Impact :** Cette capacité émergente réduit considérablement le besoin de vastes ensembles de données d'entraînement spécifiques à la tâche, ouvrant la voie à des applications plus flexibles et accessibles de modèles de langue.

# Performances de GPT-3 en few-shot learning



# ChatGPT

ChatGPT, développé par OpenAI, est un modèle de langage basé sur l'architecture GPT (Generative Pre-trained Transformer) optimisé pour comprendre et générer des dialogues naturels. L'entraînement de ChatGPT se décline selon les étapes suivantes :

- ① Pré-entraînement sur un corpus volumineux :** Comme GPT-3, ChatGPT est d'abord pré-entraîné sur un vaste ensemble de données textuelles, englobant un large éventail de la littérature disponible sur Internet, pour apprendre une compréhension générale du langage.
- ② Fine-tuning supervisé :** Ensuite, ChatGPT est affiné sur des dialogues spécifiques pour améliorer ses compétences conversationnelles. Cette étape utilise des paires question-réponse et des conversations pour enseigner au modèle des structures de dialogue et des réponses contextuellement appropriées.
- ③ Reinforcement Learning from Human Feedback (RLHF):** Utilisation de techniques de renforcement pour ajuster les réponses du modèle basées sur les préférences et les corrections fournies par des évaluateurs humains, raffinant davantage la pertinence et la naturalité des réponses.

# Fine-Tuning supervisé (FST)

Le fine-tuning supervisé joue un rôle essentiel dans l'adaptation de ChatGPT à des tâches conversationnelles spécifiques, améliorant sa capacité à fournir des réponses pertinentes et contextuellement adaptées. **Méthodologie:**

- ① Collecte d'un ensemble de données de dialogues de haute qualité, comprenant des échanges humains et des interactions annotées pour capturer divers contextes conversationnels.
- ② Transformation de ces dialogues en paires de prompts et réponses pour créer des exemples d'entraînement qui guident le modèle dans l'apprentissage de structures conversationnelles et de réponses appropriées.
- ③ Utilisation d'un processus d'entraînement supervisé où le modèle apprend à prédire la réponse la plus probable à un prompt donné, en ajustant ses paramètres internes pour minimiser la divergence entre les réponses générées et les réponses attendues (annotations).

# Alignement des LLM avec RLHF

- La génération de texte par les LLM est essentielle dans de nombreux domaines tels que la traduction automatique, la résumé automatique, et la création de contenu.
- Cependant, la qualité du texte généré peut être variable et dépend souvent des données d'entraînement disponibles.
- L'ajout du feedback humain permet d'améliorer la qualité du texte généré en fournissant une supervision supplémentaire.
- RLHF combine l'apprentissage par renforcement avec le feedback humain pour guider l'apprentissage des LLM de manière plus précise et efficace.
- L'objectif est d'entraîner les LLM à générer du texte de meilleure qualité en s'appuyant sur le savoir-faire humain pour corriger les erreurs et améliorer les performances.

# Approche RLHF pour l'entraînement des LLM

- Collecte des feedbacks :
  - Les LLM génèrent du texte qui est évalué par des humains.
  - Les humains fournissent des annotations telles que des corrections, des scores de qualité ou des préférences.
- Calcul de la récompense :
  - Le feedback humain est utilisé pour calculer une récompense.
  - Différentes métriques peuvent être utilisées pour quantifier la qualité du texte généré.
  - La récompense peut être une fonction de ces métriques et des préférences humaines.
- Entraînement des LLM :
  - L'objectif est d'entraîner les LLM à maximiser la récompense définie par le feedback humain.
  - Les LLM sont entraînés à générer du texte de meilleure qualité en fonction de cette récompense.

# Construction des feedbacks humains

- Les feedbacks humains sont construits sur la base des réponses générées par les LLM.
- Ces réponses sont ensuite évaluées par les humains, qui fournissent des annotations telles que des corrections, des scores de qualité, ou des préférences.
- Les feedbacks humains sont utilisés comme données d'entraînement pour apprendre au modèle de récompense à évaluer la qualité du texte généré.
- Les caractéristiques du texte généré, telles que la similarité avec des textes de référence ou la pertinence du contenu, peuvent être utilisées comme des caractéristiques pour l'entraînement du modèle.
- L'objectif est de créer un modèle de récompense capable d'évaluer de manière précise et fiable la qualité du texte généré par les LLM, selon les critères humains.

# Entraînement des LLM avec PPO

- **Initialisation** : Les paramètres du LLM sont initialisés à partir d'un modèle pré-entraîné sur de vastes ensembles de données textuelles.
- **Génération de données** : Le LLM génère une séquence de texte en fonction de son état actuel et de la politique actuelle. Les séquences de texte générées sont évaluées à l'aide du modèle de récompense pour calculer les récompenses associées à chaque état.
- **Mise à jour de la politique** : Une fonction de loss est calculée à l'aide des séquences de texte générées. Les gradients sont calculées par rapport aux paramètres du LLM. Les paramètres sont ensuite mis à jour pour maximiser les récompenses futures.
- **Itération** : Les étapes précédentes sont répétées de manière itérative jusqu'à ce que la politique converge vers une performance satisfaisante.

# Entraînement des LLM avec PPO

- **Initialisation** : Les paramètres du LLM sont représentés par  $\theta$ , initialisés à partir d'un modèle pré-entraîné.
- **Génération de données** : Le LLM génère une séquence de texte  $s_t$  en fonction de son état actuel et de la politique actuelle, paramétrée par  $\theta$ .
- **Mise à jour de la politique** : La probabilité d'action pour chaque état  $s_t$  est donnée par  $\pi_\theta(a_t|s_t)$ . La fonction de perte utilisée dans PPO est définie comme suit :

$$\mathcal{L}(\theta) = \mathbb{E}_t \left[ \min \left( \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A_t, \text{clip} \left( \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, 1 - \epsilon, 1 + \epsilon \right) \right) \right]$$

- où  $A_t$  est l'avantage avancé estimé pour l'action  $a_t$  et  $\epsilon$  est un paramètre de clipping.

# Introduction à la quantization des modèles

## • Qu'est-ce que la quantization ?

- La quantization est une technique qui réduit la précision des poids et/ou des activations d'un modèle de deep learning, typiquement de 32 bits à 16 bits ou 8 bits.
- L'objectif est de diminuer la taille du modèle et d'accélérer le temps d'inférence sans sacrifier de manière significative la précision.

## • Pourquoi utiliser la quantization ?

- Réduction de la taille du modèle, facilitant le déploiement sur des dispositifs à ressources limitées.
- Accélération des inférences en réduisant le coût computationnel.
- Réduction de la consommation d'énergie, particulièrement bénéfique pour les applications mobiles et embarquées.

# Types de quantization

- **Quantization statique**

- Les poids et les activations sont quantifiés lors de l'entraînement du modèle.
- Nécessite un jeu de calibration pour déterminer les plages de valeurs.

- **Quantization dynamique**

- Les poids sont quantifiés lors de l'entraînement, mais les activations sont quantifiées à la volée pendant l'inférence.
- Plus flexible et peut s'adapter à des variations dans les données d'entrée.

- **Quantization à la post-formation**

- Appliquer la quantization sur un modèle pré-entraîné sans réentraîner le modèle.
- Utilisée quand l'accès aux données d'entraînement n'est pas possible ou pour simplifier le processus.

# Impact de la quantization sur les performances

- **Précision du modèle**

- La quantization peut entraîner une perte de précision, mais les techniques modernes permettent de minimiser cette perte.
- L'équilibre entre la réduction de la taille du modèle et la précision est crucial.

- **Temps d'inférence et utilisation de la mémoire**

- Réduction significative du temps d'inférence en raison du modèle plus léger.
- Diminution de l'utilisation de la mémoire, facilitant le déploiement sur des dispositifs à ressources limitées.

- **Consommation d'énergie**

- Les modèles quantifiés consomment moins d'énergie, ce qui est bénéfique pour les dispositifs embarqués et les applications mobiles.

# Introduction à l'approximation bas rang

- **Qu'est-ce que l'approximation bas rang ?**

- L'approximation bas rang est une technique utilisée pour réduire la dimensionnalité d'une matrice tout en conservant le plus d'information possible.
- Elle est couramment utilisée pour compresser des modèles de deep learning en réduisant la taille des matrices de poids.

- **Pourquoi utiliser l'approximation bas rang ?**

- Réduction de la complexité computationnelle et de la mémoire requise.
- Accélération des inférences et réduction des temps d'entraînement.
- Amélioration de l'efficacité du stockage et du déploiement des modèles.

# LoRA (Low-Rank Adaptation)

- **Qu'est-ce que LoRA ?**

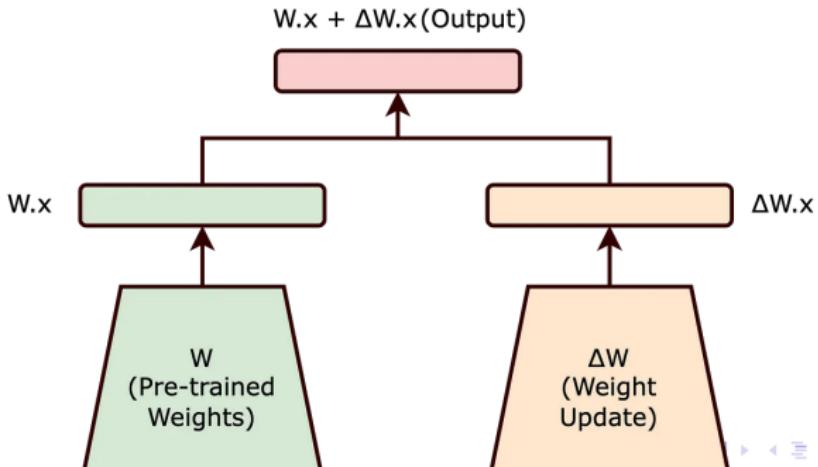
- LoRA est une technique utilisée pour adapter les grands modèles de langage en réduisant leur complexité tout en maintenant leur performance.
- Elle repose sur l'idée d'approximer les matrices de poids de ces modèles en utilisant des matrices de bas rang.

- **Pourquoi utiliser LoRA ?**

- Réduction de la complexité computationnelle et des besoins en mémoire.
- Accélération des temps d'entraînement et d'inférence.
- Facilitation du déploiement des modèles sur des dispositifs à ressources limitées.

# Principe de LoRA

- **Décomposition en matrices de bas rang** Une matrice de poids  $W$  est approximée par le produit de deux matrices de bas rang  $A$  et  $B$  :  $W \approx AB$ . Les matrices  $A$  et  $B$  ont des dimensions beaucoup plus petites que  $W$ , ce qui réduit la complexité du modèle.
- **Entraînement avec LoRA** Pendant l'entraînement, seules les matrices  $A$  et  $B$  sont mises à jour, tandis que  $W$  reste fixe. Cette approche permet de réduire la consommation de mémoire et d'accélérer le processus d'entraînement.



# LLMs propriétaires vs Open Source

- **Modèles propriétaires**

- ChatGPT, Claude, Gemini...
- Poids non disponibles au public.
- Performances élevées, accès limité par API payante
- Protection intellectuelle stricte (code non accessible)
- Support technique assuré, documentation avancée
- Dépendance aux fournisseurs et coûts potentiellement élevés

- **Modèles open source**

- Llama, Mistral, Gemma...
- Poids disponibles au public
- Transparence du code, flexibilité d'adaptation
- Performances parfois inférieures, mais amélioration continue
- Gratuit, mais coût de l'infrastructure à gérer
- Exposition aux failles potentielles de sécurité, support communautaire

**Conclusion :** Choisir entre contrôle, flexibilité et performances optimales.

# Récapitulatif

## 1. Approches symboliques vs connexionnistes

- **Symboliques:** Basées sur des règles codées par des experts (moteurs de règles).
- **Connexionnistes:** Apprentissage à partir de données (réseaux de neurones, deep learning).

## 2. Approches statistiques

- **n-grams:** Modèle de langage le plus simple (1-gram, bigram,...).
- **TF-IDF:** Modèle qui est utilisé notamment pour les moteurs de recherche.

## 3. Machine Learning

- **Modèles supervisés:** régression logistique, SVM, Random Forests, Naive Bayes.
- **Modèles non supervisés:** Clustering (K-means, réduction de dimension (ACP)).

## 4. Représentation des mots

- **Non sémantiques:** Bag of Words, TF-IDF.
- **Sémantiques:** Word2Vec, GloVe (représentation vectorielle des mots).

## 5. Deep Learning

- **Caractéristiques:** Apprentissage profond des représentations (extraction automatique des features).
- **RNN/LSTM:** Modélisation des séquences temporelles (comme le langage).
- **Transformers:** Modèles avancés pour le traitement du langage (BERT, GPT).

# Merci de votre attention

[redha.moulla@axia-conseil.com](mailto:redha.moulla@axia-conseil.com)