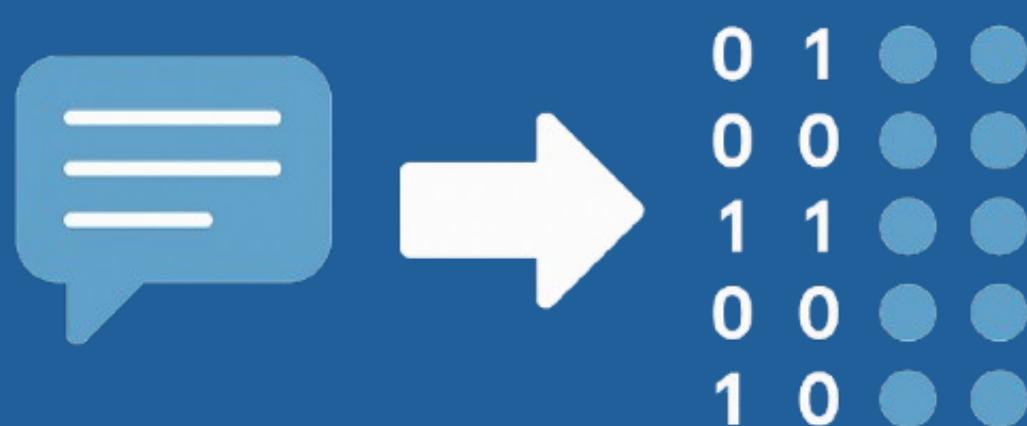


---

01  
02  
03  
04

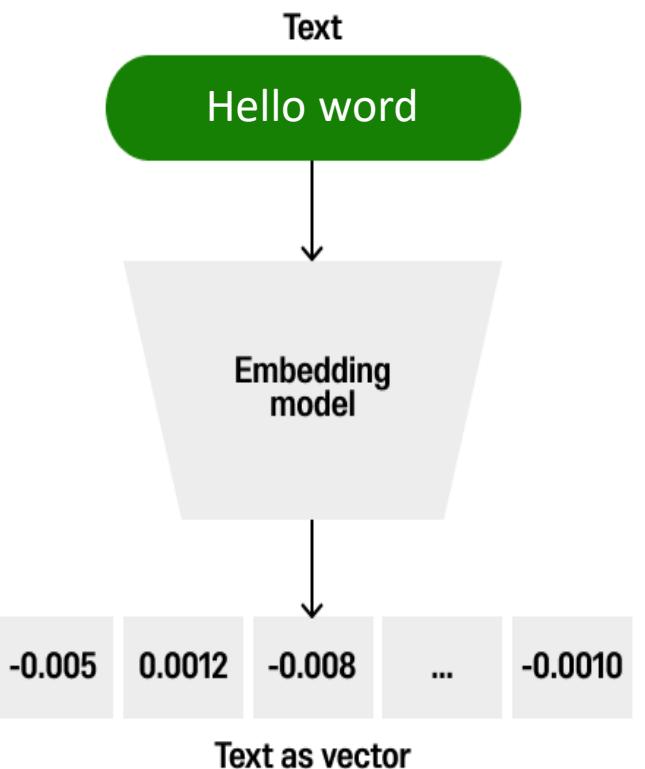
# Chapitre 1 : Représentation des textes



# Traitement du langage naturel - NLP

***'Le traitement du langage naturel est la manière dont les ordinateurs comprennent le langage humain.'***

- Pour construire un modèle d'apprentissage automatique, nous considérons uniquement des **caractéristiques (features) au format numérique**, car **nos modèles ne comprennent que les nombres**.
- Le principal défi apparaît lorsque toutes les caractéristiques sont du texte, par exemple des avis de produits, des tweets ou des commentaires. Comment entraîner un modèle d'apprentissage automatique avec des données textuelles, alors que nous savons que les algorithmes de machine learning ne fonctionnent qu'avec des entrées numériques ?



## Dans ce cas, nous appliquons des techniques :

---

**1. Prétraitement (Preprocessing)** : nettoyer et normaliser le texte brut.

- **Tokenization** : couper le texte en mots ou sous-mots
- **Lowercasing** : mettre tout en minuscule
- **Suppression des stopwords** : enlever le, la, de, et...
- **Stemming / Lemmatisation** : réduire les mots à leur racine (jouer, joue, jouaient → jouer)
- **Nettoyage** : enlever ponctuation, chiffres, HTML tags, etc.

**2. Encodage des caractéristiques (Feature Encoding)**: transformer le texte (symbolique) en vecteurs numériques utilisables par les modèles.

- **Représentations classiques** : BoW (Bag of Words), TF-IDF.
- **Word Embeddings** : Word2Vec, GloVe, FastText.

## Dans ce cas, nous appliquons des techniques :

Hi Everyone! My name is Rami Chaymae, and I smile to create this notebook and save it in

[https://www.kaggle.com/ ....](https://www.kaggle.com/)

Cleaning

Hi Everyone My name Rami Chaymae and I am smiling face with smiling eyes to create this notebook and save it in

Pre-processing

[‘Hi’ , ‘Everyone’ , ‘name’ , ‘Rami’ , ‘Chaymae’ , ‘smile’ , ‘face’ , ‘smile’ , ‘eyes’ , ‘create’ ,  
‘notebook’ , ‘save’ ]

Feature Extraction

**[0.41285857 0.0.0.69903033 0.41285857 0.41285857]**

# Prétraitement (Preprocessing) :

---

## 1. Suppression du bruit dans les textes

- **Les stop words** sont des mots qui n'ont pas une grande importance pour être utilisés. Ce sont des mots très courants dans une langue (par ex. a, an, the en anglais). Ils n'apportent généralement pas d'aide dans la plupart des tâches de NLP comme l'analyse sémantique, la classification, etc.
- **Le nettoyage des données (Data Cleaning) joue un rôle important en NLP pour éliminer le bruit présent dans les données.**
- En général, ces mots sont filtrés du texte car ils génèrent une grande quantité d'informations inutiles.
- Chaque langue possède sa propre liste de stop words. Par exemple, en anglais, des mots fréquemment utilisés comme as, the, be, are.



# Prétraitement (Preprocessing) :

---

## 1. Suppression du bruit dans les textes

```
from nltk.corpus import stopwords

en_stopwords = set(stopwords.words('english'))
print(f"Stop Words in English : \n{ en_stopwords}")

def remove_stopwords(text):
    text = [word for word in text if word not in en_stopwords]
    return text

sample = "The car is red."

print("Before removing stopwords : " , word_tokenize(sample) )
print("\n\nAfter removing stopwords : " ,
      remove_stopwords(word_tokenize(sample.lower())))
```

The car is red.

car red.

# Prétraitement (Preprocessing) :

---

## 1. Suppression du bruit dans les textes

En plus des stop words, il est souvent utile de nettoyer le texte en supprimant :

- URLs (liens web)
- Ponctuation (., !, ?, ...)
- ....

```
import re
import string

text = "Hello !!! How are you ? Consult"
print(f"Text before removing punctuation: \n {text}")

text = re.sub('[\+string.punctuation\+]', ' ', text)

print(f"\n\nText after removing punctuation: \n {text}")
```

```
text = "Voici le URL suivant : https://www.kaggle.com"
print(f"Text before removing url: \n {text}")

text = re.sub(r"http\S+", "", text)

print(f"\n\nText after removing url: \n {text}")
```

# Prétraitement (Preprocessing) :

---

## 2. Gestion des emojis

- Les emojis sont de plus en plus présents dans les textes (réseaux sociaux, avis utilisateurs, messages).
- Pour le NLP, ils peuvent introduire du bruit ou compliquer l'analyse si on ne les traite pas.



# Prétraitement (Preprocessing) :

---

## 2. Gestion des emojis

```
import demoji

text = "I am 😊 to create this notebook"

emojis = demoji.findall(text)
emojis
print(emojis)

print(f"Before Handling emoji: {text}")

for emoji in emojis:
    text = text.replace(emoji, " " + emojis[emoji].split(":")[0])

print(f"After Handling emoji: {text}")
```

I 😊 to create this  
notebook

I am smiling face with  
smiling eyes to create this  
notebook

## Prétraitement (Preprocessing) :

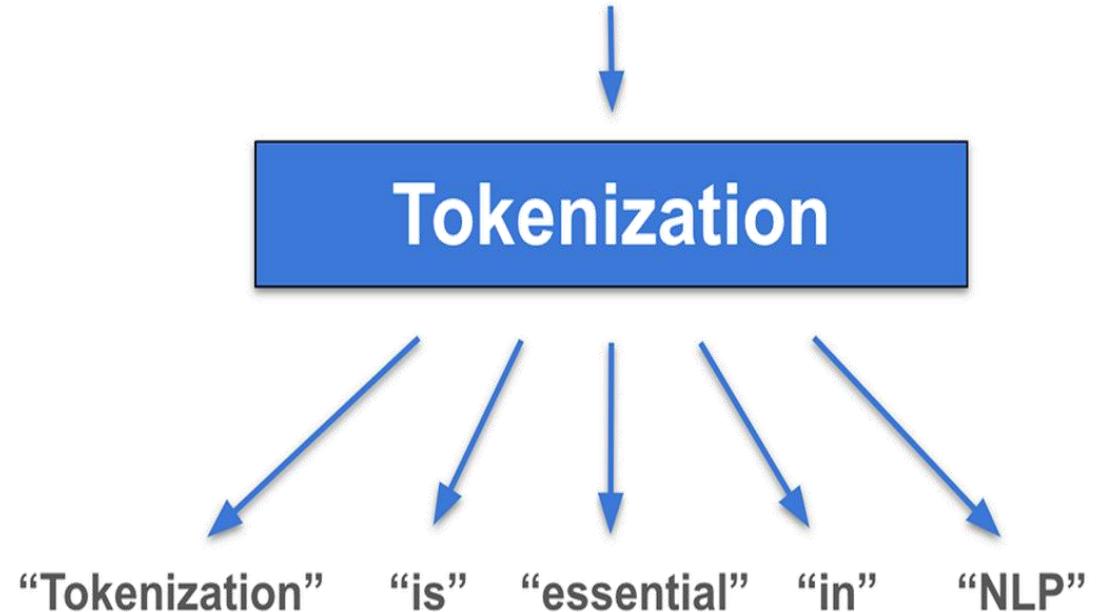
---

### 3. Tokenization

La tokenisation est le processus qui consiste à découper des données complexes comme des paragraphes en unités simples appelées tokens.

- **Tokenisation en phrases** : découper un paragraphe en une liste de phrases.
- **Tokenisation en mots** : découper une phrase en une liste de mots.

“ Tokenization is essential in NLP! ”



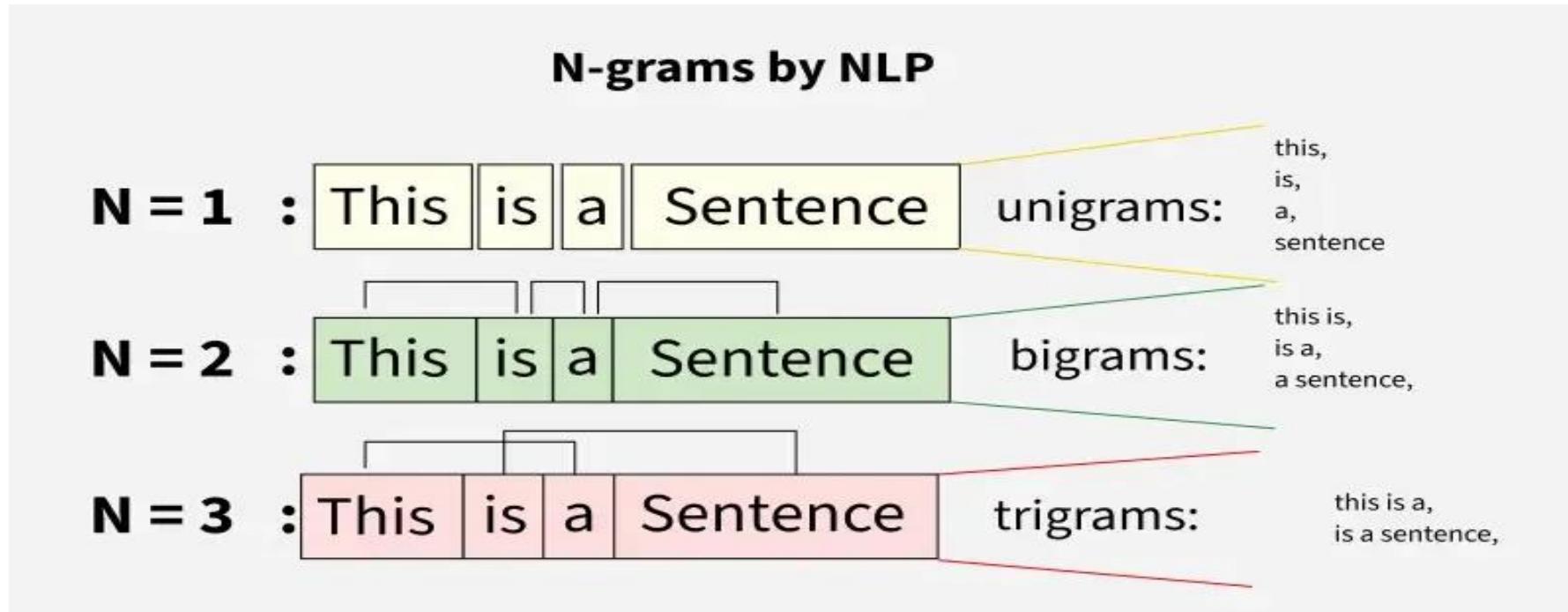
# Prétraitement (Preprocessing) :

---

## 3. Tokenization

Quelques autres termes importants liés à la tokenisation en mots sont :

- **Bigrammes** : les tokens sont constitués de deux mots consécutifs, appelés bigrammes.
- **Trigrammes** : les tokens sont constitués de trois mots consécutifs, appelés trigrammes.
- **N-grammes** : les tokens sont constitués de 'N' mots consécutifs, appelés n-grammes.



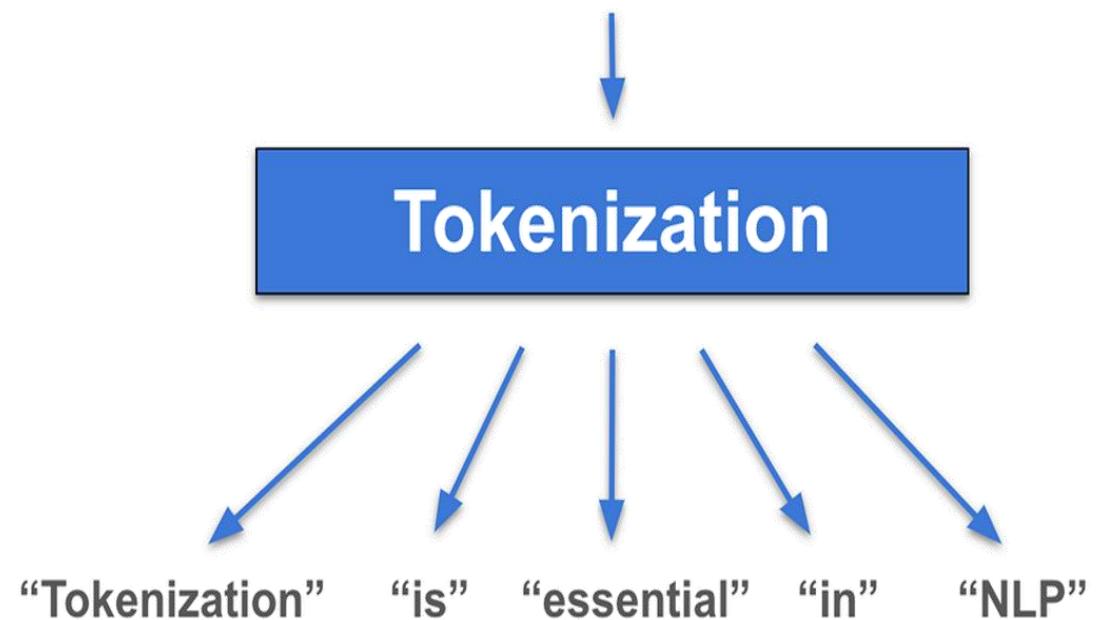
## Prétraitement (Preprocessing) :

---

### 3. Tokenization

```
from nltk.tokenize import word_tokenize  
  
sample = "Tokenization is essential in NLP"  
print(sample)  
  
word_tokenize(sample)
```

“ Tokenization is essential in NLP! ”



# Prétraitement (Preprocessing) :

## 4. Text Normalization

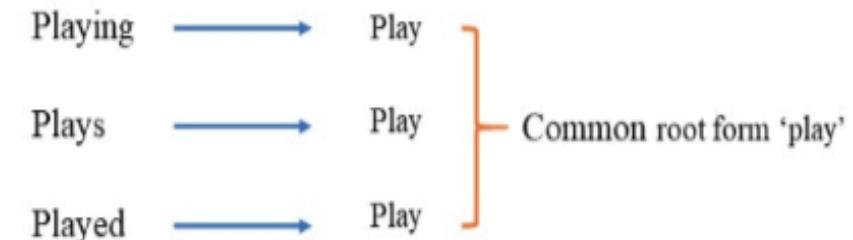
- Le **stemming** et la **lemmatisation** sont des techniques de normalisation de texte (ou parfois appelées normalisation de mots)

**Stemming**

adjustable → adjust  
formality → formaliti  
formaliti → formal  
airliner → airlin △

**Lemmatization**

was → (to) be  
better → good  
meeting → meeting



am, are, is → be  
Car cars, car's, cars' → car

Using above mapping a sentence could be normalized as follows:

the boy's cars are different colors → the boy car be differ color

## Prétraitement (Preprocessing) :

---

### 4. Text Normalization

- **Stemming :**

```
from nltk.stem.porter import PorterStemmer

stemmer = PorterStemmer()

def stemming(text):
    text = [stemmer.stem(word) for word in text]
    return text
```

# Prétraitement (Preprocessing) :

---

## 4. Text Normalization

- Lemmatisation :

```
import spacy

sp = spacy.load("en_core_web_sm")

def lemmatization(text):
    text = " ".join(text)
    token = sp(text)
    text = [word.lemma_ for word in token]
    return text

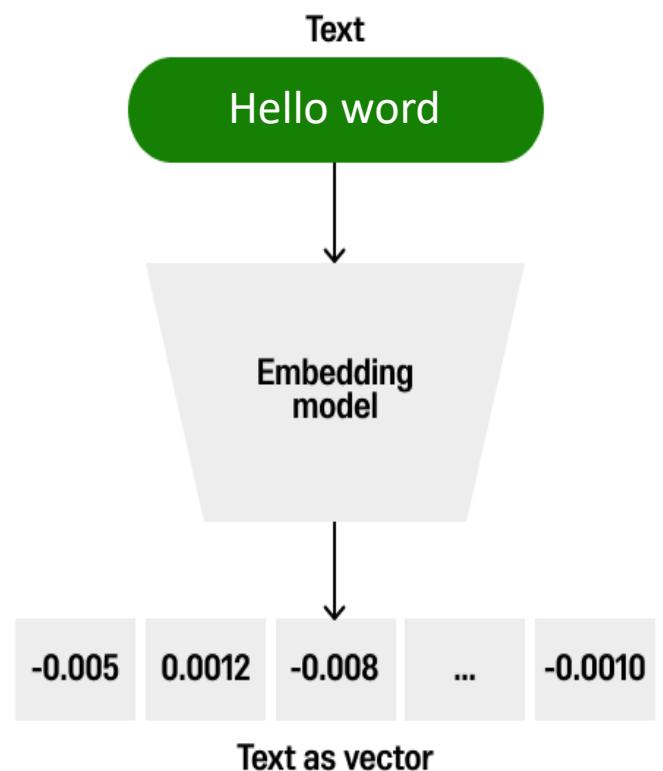
sample = "I was enjoying the cats running and studies were finished"
print(f"Before Lemmatization : {word_tokenize(sample)}")
print(f"After Lemmatization : {lemmatization(word_tokenize(sample))}")
```

## Encodage des caractéristiques (Feature Encoding):

---

La plupart des techniques traditionnelles d'apprentissage automatique (machine learning) travaillent sur des features créées soit par **Bag-of-Words (BoW)**, **TF-IDF**.

Les techniques plus récentes incluent **Word2Vec**, **GloVe**, et **l'apprentissage des features directement pendant le processus d'entraînement d'un réseau de neurones**.

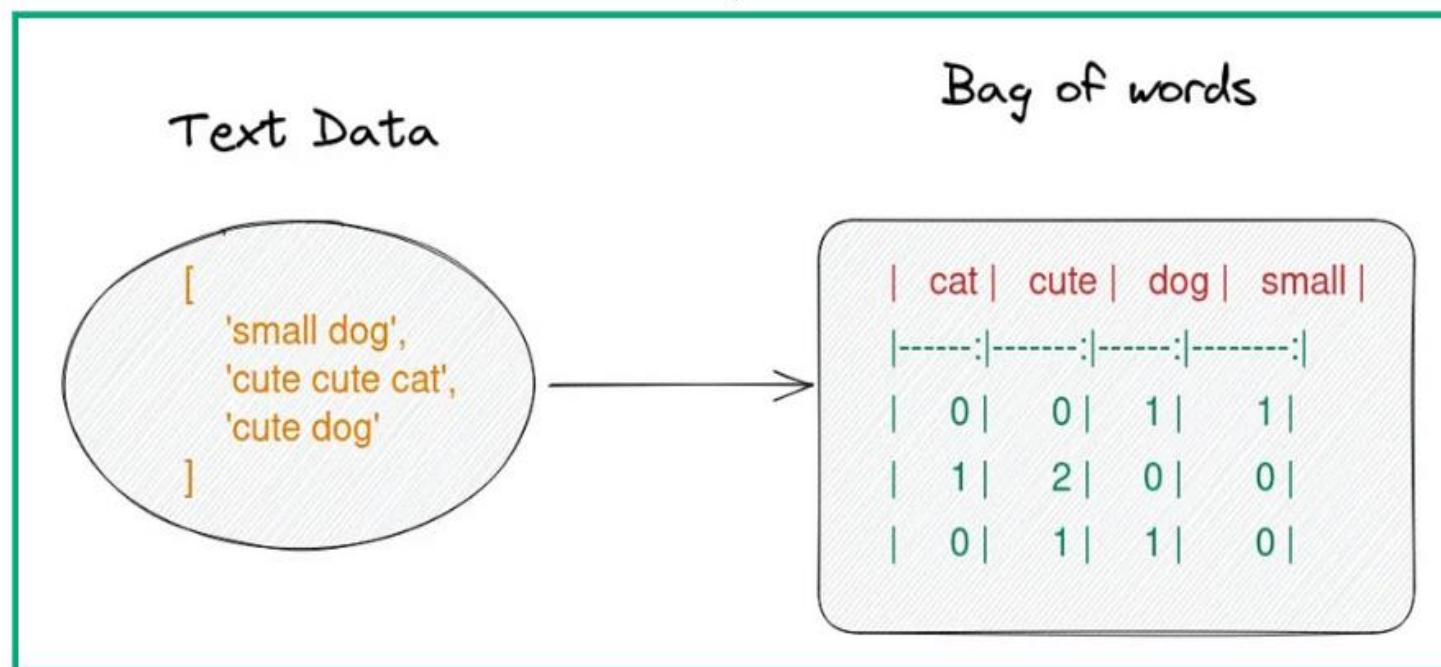


# Encodage des caractéristiques (Feature Encoding):

---

## 1. Bag-of-Words (BoW):

Le modèle **Bag of Words (BoW)** est une méthode courante en traitement du langage naturel (NLP) utilisée pour représenter des documents textuels de longueurs variables sous forme de vecteurs de longueur fixe contenant les fréquences des mots.



# Encodage des caractéristiques (Feature Encoding):

---

## 1. Bag-of-Words (BoW):

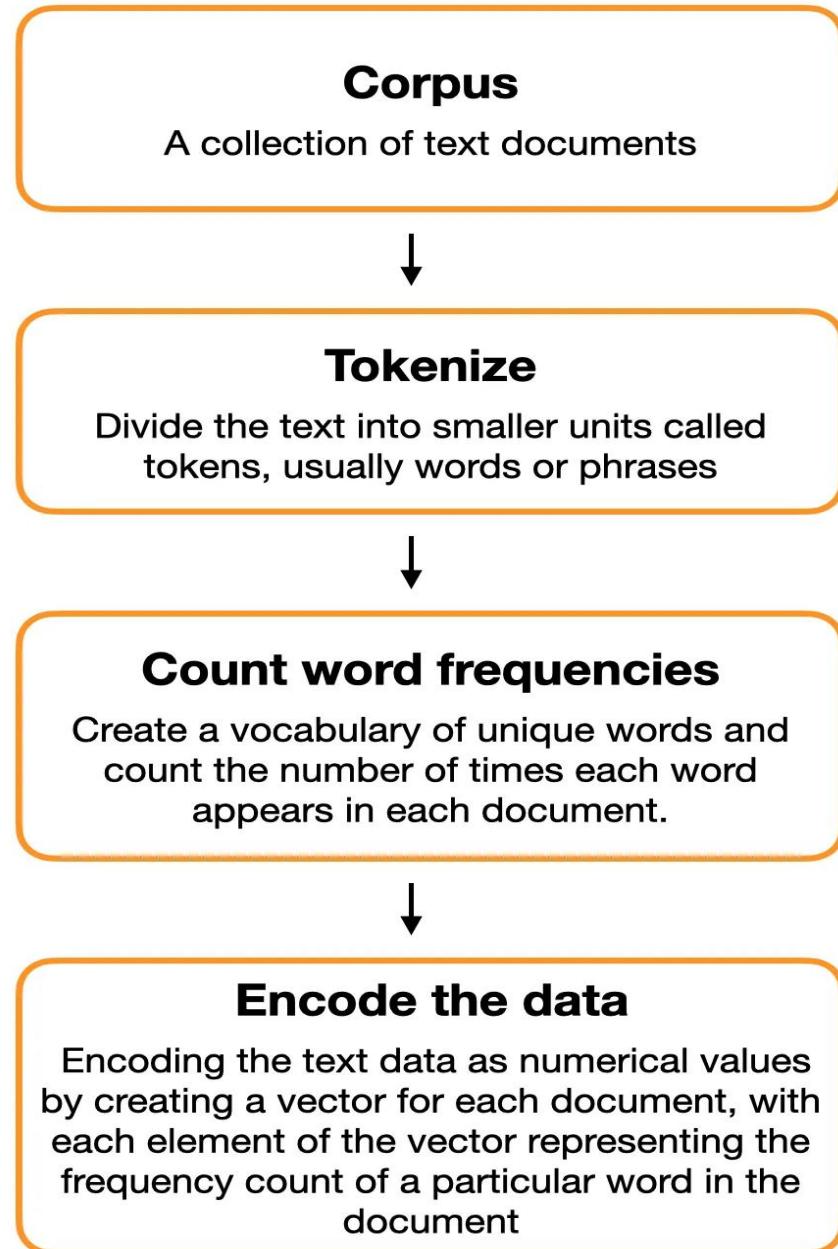
Le processus de conversion d'un texte en Bag of Words comprend :

- **Tokenisation** : diviser le texte en unités plus petites appelées tokens, généralement des mots ou des expressions.
- **Comptage des fréquences des mots** : créer un vocabulaire de tous les mots uniques du corpus textuel et compter combien de fois chaque mot apparaît dans chaque document.
- **Encodage des données** : représenter les données textuelles sous forme de valeurs numériques en créant un vecteur pour chaque document, où chaque élément du vecteur correspond au nombre d'occurrences d'un mot particulier dans ce document.

La représentation numérique ainsi obtenue, sous forme de vecteur de fréquences de mots, est appelée modèle **Bag of Words**.

# Bag of Words Model explanation

## Example



*Corpus:*  
The dog is happy. The child makes the dog happy. The dog makes the child happy

*Tokenization:*  
D1: [The] [dog] [is] [happy]  
D2: [The] [child] [makes] [the] [dog] [happy]  
D3: [The] [dog] [makes] [the] [child] [happy]

Documents	Counting word frequencies
D1	the: 1, dog: 1, is: 1, happy: 1
D2	the: 2, dog: 1, makes: 1, child: 1, happy: 1
D3	the: 2, child: 1, makes: 1, dog: 1, happy: 1

Encode	child	dog	happy	is	makes	the	BoW Vector representations
D1	0	1	1	1	0	1	[0,1,1,1,0,1]
D2	1	1	1	0	1	2	[1,1,1,0,1,2]
D3	1	1	1	0	1	2	[1,1,1,0,1,2]

# Encodage des caractéristiques (Feature Encoding):

---

## 1. Bag-of-Words (BoW):

```
from sklearn.feature_extraction.text import CountVectorizer

# Define the text data
text_data = ["The dog is happy.", "The shild makes the dog happy", "The dog makes the shild happy"]

# Créer une instance de la classe CountVectorizer
vectorizer = CountVectorizer()

# Ajuster le vectoriseur aux données du texte
vectorizer.fit(text_data)

print("Vocabulaire :", vectorizer.get_feature_names_out())

# Transformer les données textuelles en une représentation BoW
bow_matrix = vectorizer.transform(text_data)

print(bow_matrix.toarray())
```

# Encodage des caractéristiques (Feature Encoding):

---

## 1. Bag-of-Words (BoW):

Limites du modèle Bag of Words :

- Perte d'information sémantique** : Le modèle Bag of Words ignore l'ordre et le contexte des mots dans le texte. Il traite chaque mot comme une entité indépendante, sans tenir compte des relations entre eux. En conséquence, il échoue à capturer le sens sémantique du texte.

Document D1	the: 2, dog: 1, makes: 1, child: 1, happy: 1					
Document D2	the: 2, child: 1, makes: 1, dog: 1, happy: 1					
D1	1	1	1	1	2	[1,1,1,1,2]
D2	1	1	1	1	2	[1,1,1,1,2]

# Encodage des caractéristiques (Feature Encoding):

---

## 1. Bag-of-Words (BoW):

Limites du modèle Bag of Words :

2. **Information sémantique limitée** : Le modèle Bag of Words capture uniquement la présence ou l'absence d'un mot dans un document, sans tenir compte de son sens ni du contexte.
3. **Haute dimensionnalité** : Si un corpus contient un grand nombre de mots uniques, la représentation BoW aura une dimensionnalité très élevée.
4. **Ignorer les nouveaux mots** : Comme la longueur du vecteur BoW est fixée par la taille du vocabulaire initial, il est difficile d'introduire de nouveaux mots. Cela nécessiterait de recalculer les vecteurs pour tous les documents. Pour éviter cela, les nouveaux mots sont soit ignorés, soit remplacés par un jeton spécial appelé UNK (unknown).

## **Encodage des caractéristiques (Feature Encoding):**

---

### **2. TF-IDF (Term Frequency-Inverse Document Frequency) :**

TF-IDF (Term Frequency-Inverse Document Frequency) est une méthode statistique utilisée en traitement automatique du langage naturel (NLP) et en **recherche d'information** pour évaluer l'importance d'un mot dans un document par rapport à une collection plus large de documents. TF-IDF combine deux composants :

- **Fréquence du terme (TF – Term Frequency)** : mesure la fréquence d'apparition d'un mot dans un document. Une fréquence plus élevée suggère une plus grande importance. Si un terme apparaît fréquemment dans un document, il est probablement pertinent par rapport au contenu de ce document.
- **Fréquence inverse des documents (IDF – Inverse Document Frequency)** : réduit le poids des mots courants présents dans plusieurs documents tout en augmentant le poids des mots rares. Si un terme apparaît dans peu de documents, il est plus susceptible d'être significatif et spécifique.

## Encodage des caractéristiques (Feature Encoding):

---

### 2. TF-IDF (Term Frequency-Inverse Document Frequency) :

**Note:** Ces étapes sont déjà implémentées dans **scikit-learn**. Si vous souhaitez explorer **toutes les relations mathématiques** du TF et de l'IDF, consultez ce lien <https://en.wikipedia.org/wiki/Tf-idf> :

- **Fréquence du terme (TF – Term Frequency) :**

$$\text{TF}(t, d) = f_{t,d}$$

- $f_{t,d}$  = nombre d'occurrences du terme  $t$  dans le document  $d$
- **sklearn** utilise la **fréquence brute**, puis applique la normalisation du vecteur  
(`norm='l2'` par défaut).

- **Fréquence inverse des documents (IDF – Inverse Document Frequency) :**

$$\text{IDF}(t) = \log \frac{1 + N}{1 + df(t)} + 1$$

- $N$  = nombre total de documents
- $df(t)$  = nombre de documents contenant le terme  $t$

## Encodage des caractéristiques (Feature Encoding):

---

### 2. TF-IDF (Term Frequency-Inverse Document Frequency) :

**Note:** Ces étapes sont déjà implémentées dans **scikit-learn**. Si vous souhaitez explorer **toutes les relations mathématiques** du TF et de l'IDF, consultez ce lien <https://en.wikipedia.org/wiki/Tf-idf> :

- **TF-IDF final**

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

- Ensuite, **normalisation L2 par défaut** :

$$v_{norm} = \frac{v}{\|v\|_2} = \frac{v}{\sqrt{\sum_i v_i^2}}$$

# Encodage des caractéristiques (Feature Encoding):

---

## 2. TF-IDF (Term Frequency-Inverse Document Frequency) :

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Définir les documents
text_data = ["I love dogs", "Dogs love me"]

# Créer un vecteur TF-IDF sans suppression de stopwords ni normalisation
vectorizer = TfidfVectorizer()

# Ajuster et transformer les données
tfidf_matrix = vectorizer.fit_transform(text_data)

# Afficher le vocabulaire
print("Vocabulaire :", vectorizer.get_feature_names_out())

# Afficher la matrice TF-IDF
print(tfidf_matrix.toarray())
```

## Encodage des caractéristiques (Feature Encoding):

---

### 2. TF-IDF (Term Frequency-Inverse Document Frequency) :

Bien que le TF-IDF soit utile, il présente certaines limites :

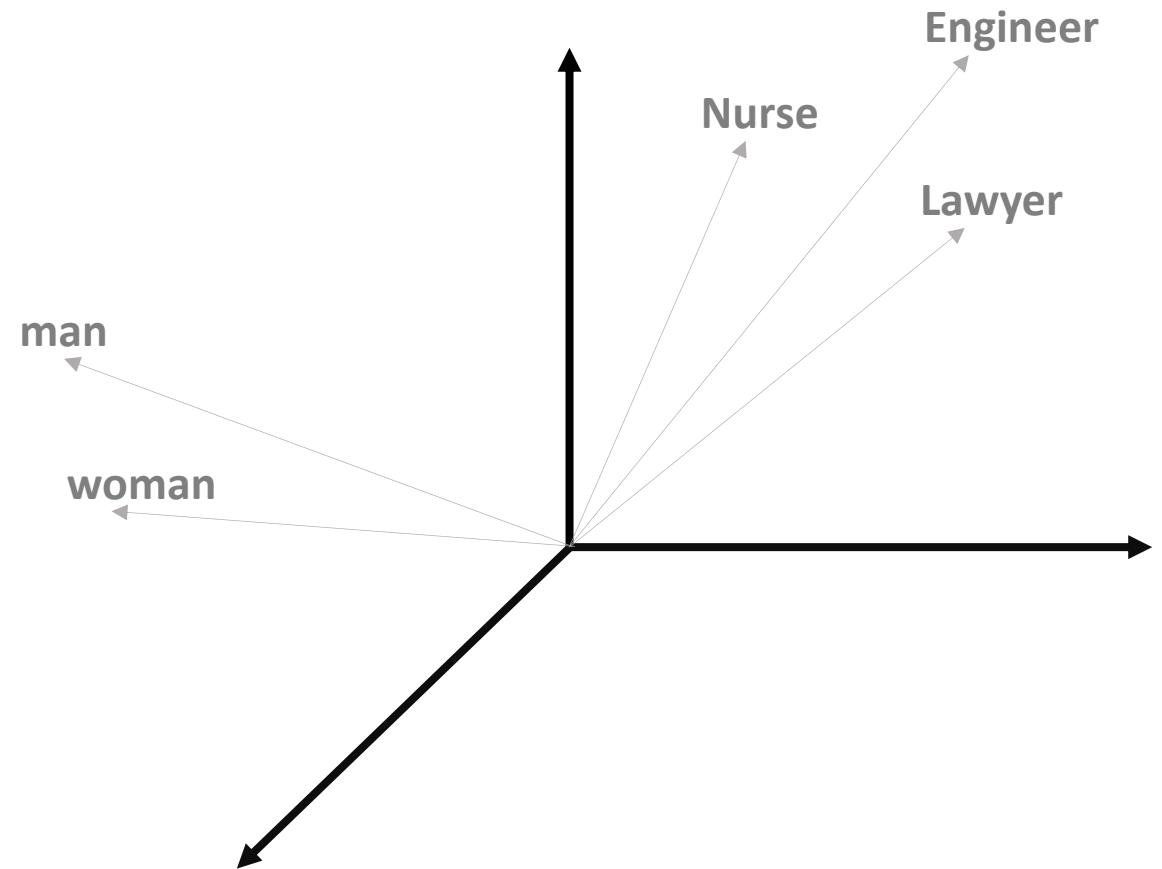
- **Ne prend pas en compte les relations entre les mots** : il traite chaque mot individuellement et ne reconnaît pas le contexte.
- **Sensible à la taille des documents** : les documents plus longs ont tendance à avoir des fréquences de termes plus élevées, ce qui peut fausser les résultats si la normalisation n'est pas correctement appliquée.
- **Moins efficace avec les synonymes** : il ne reconnaît pas que des mots comme « voiture» et « véhicule» peuvent avoir le même sens.

## Encodage des caractéristiques (Feature Encoding):

---

### 3. Word2Vec :

Word2Vec crée une représentation de chaque mot présent dans notre vocabulaire sous forme de vecteur. Les mots utilisés dans des contextes similaires ou ayant des relations sémantiques sont efficacement capturés grâce à leur proximité dans l'espace vectoriel – en d'autres termes, des mots similaires auront des vecteurs de mots similaires !



## Encodage des caractéristiques (Feature Encoding):

---

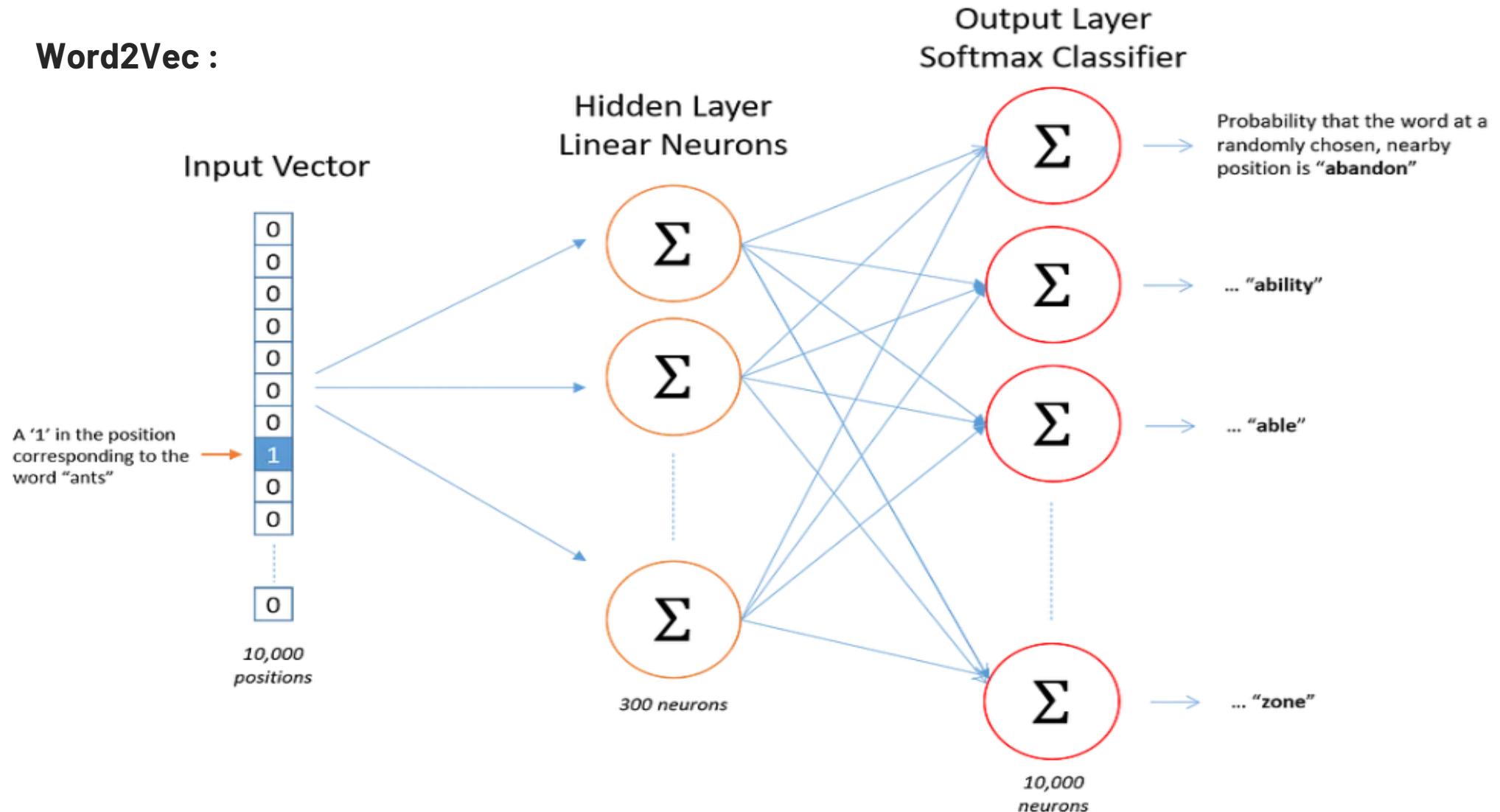
### 3. Word2Vec :

Word2Vec est essentiellement un réseau de **neurones peu profond à 2 couches, entraîné**.

- L'entrée contient tous les documents/textes de notre ensemble d'entraînement. Pour que le réseau puisse traiter ces textes, ils sont représentés sous forme d'encodage one-hot des mots.
- Le nombre de neurones présents dans la couche cachée est égal à la taille de l'embedding que nous souhaitons. Ainsi, si nous voulons que tous nos mots soient représentés par des vecteurs de longueur 300, alors la couche cachée contiendra 300 neurones.
- La couche de sortie contient les probabilités d'un mot cible (étant donné une entrée du modèle, quel mot est attendu) pour une entrée particulière.
- À la fin du processus d'entraînement, les poids de la couche cachée sont considérés comme les embeddings des mots. Intuitivement, on peut voir cela comme chaque mot possédant un ensemble de n poids (300 dans l'exemple ci-dessus) qui « pondèrent » leurs différentes caractéristiques (par analogie avec l'explication donnée plus tôt).

# Encodage des caractéristiques (Feature Encoding):

## 3. Word2Vec :



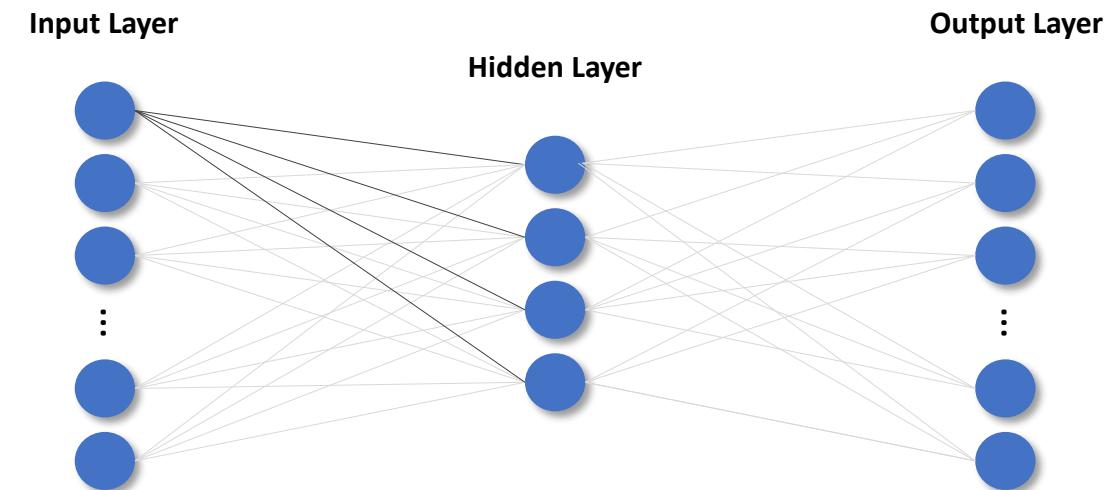
## Encodage des caractéristiques (Feature Encoding):

---

### 3. Word2Vec :

#### Corpus textuel

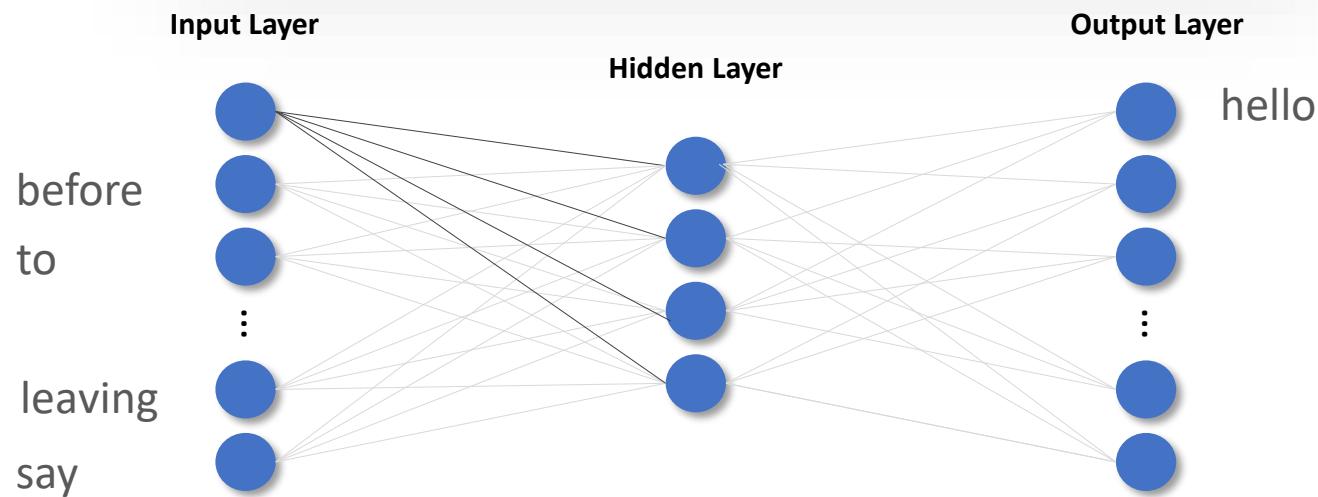
1. "I just wanted to say hello before leaving."
2. "The tall man stood in the middle of the room."
3. "She saw a woman sitting by the window."



## Encodage des caractéristiques (Feature Encoding):

### 3. Word2Vec :

1. "I just wanted to say hello before leaving."

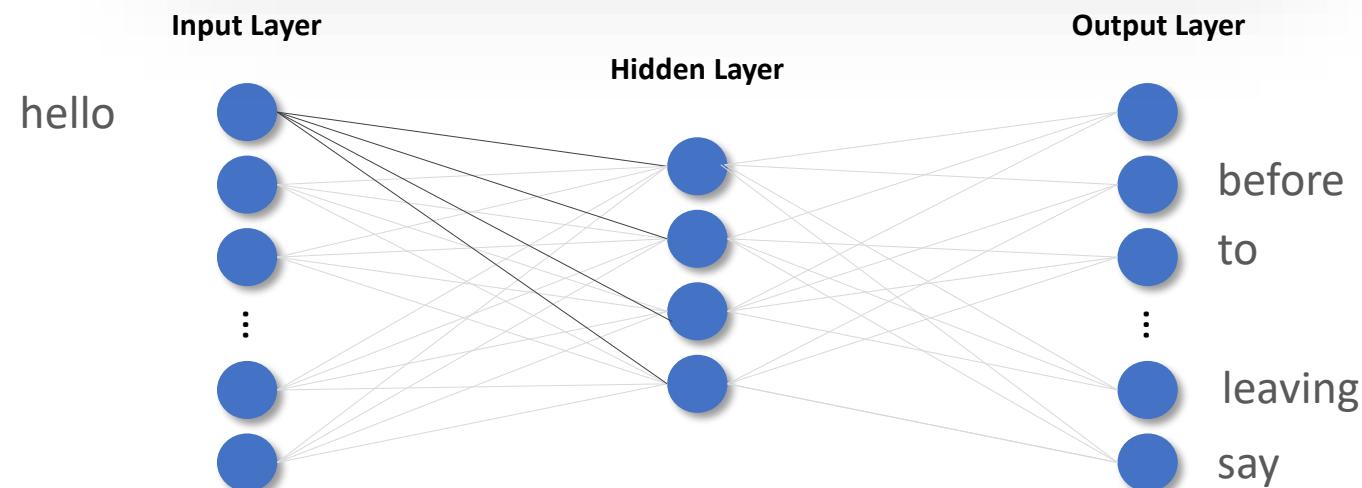


CBOW

## Encodage des caractéristiques (Feature Encoding):

### 3. Word2Vec :

1. "I just wanted to say hello before leaving."

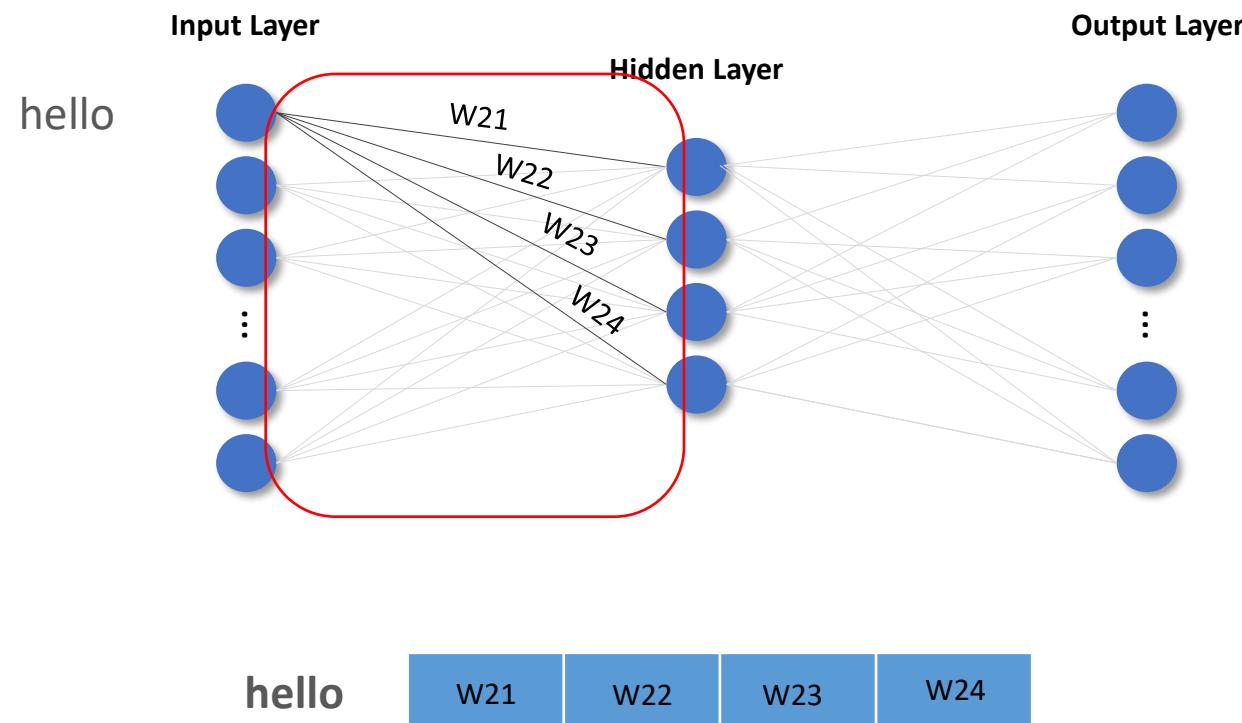


Skip-gram

# Encodage des caractéristiques (Feature Encoding):

---

## 3. Word2Vec :



## Encodage des caractéristiques (Feature Encoding):

---

### 3. Word2Vec :

```
from gensim.models import Word2Vec
# Define the text data
text_data = ["This is a positive sentence.",
             "This is a neutral sentence.",
             "This is a negative sentence."]
X_train_tokens = [text.split() for text in text_data]
w2v_model = Word2Vec(X_train_tokens, vector_size=200, window=5, min_count=1, workers=4)
w2v_model.wv.most_similar("positive")
```