Natural Language Processing

Course Overview and Introduction

Florian Valade

March 10, 2025

Présentation

Florian Valade
florian.valade@univ-eiffel.fr
Data Scientist chez Fujitsu
Chercheur au LAMA

À qui s'adresse ce cours

- **Étudiants souhaitant devenir Data Scientists ou ML Engineers**
  - Se spécialiser dans les problématiques de langage
  - Acquérir des compétences recherchées en entreprise
  - Développer une expertise en analyse de données textuelles
- **Étudiants intéressés par la recherche en NLP**
  - Comprendre les fondements théoriques
  - Explorer les architectures avancées
  - Préparer un parcours en recherche
- **Étudiants curieux de comprendre le fonctionnement des LLMs**
  - Démystifier les modèles comme GPT, BERT, LLaMA...
  - Comprendre les capacités et limites actuelles

Objectifs du cours

- **Comprendre ce qu'est le langage naturel et comment l'analyser**
  - Structure linguistique et particularités du langage humain
  - Défis spécifiques au traitement automatique
  - Évaluation des systèmes de NLP
- **Maîtriser les méthodes classiques de NLP**
  - Prétraitement et représentation du texte
  - Vectorisation et modèles statistiques
  - Classification, extraction d'information et clustering
- **Explorer les modèles de langage avancés (LLMs)**
  - Architectures basées sur l'attention (Transformers)
  - Préentraînement et fine-tuning
  - Applications et cas d'usage pratiques

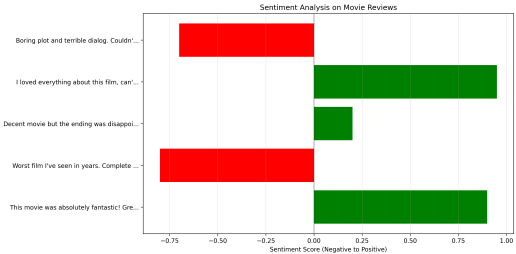
Plan du cours

Qu'est-ce que le NLP?

- Le **Traitement du Langage Naturel (NLP)** est un domaine de l'intelligence artificielle qui se concentre sur l'interaction entre les ordinateurs et le langage humain
- Il vise à permettre aux machines de **comprendre**, **interpréter** et **générer** du langage humain
- Le langage est **omniprésent** dans notre monde:
  - Livres, articles et documents
  - Contenus web et réseaux sociaux
  - Messages et conversations
  - Emails et communications professionnelles
- Le NLP nous permet d'exploiter et de donner du sens à cette ressource abondante

Applications du NLP: Compréhension de texte

- **Analyse de sentiment**
  - Détection des opinions positives/négatives
  - Modération automatique des commentaires
  - Analyse des retours clients et avis
- **Classification de texte**
  - Catégorisation des emails (spam/non-spam)
  - Organisation des documents par thème
  - Filtrage de contenu inapproprié
- **Extraction d'information**
  - Reconnaissance d'entités nommées (noms, lieux, dates)
  - Extraction de relations entre entités
  - Résumé automatique de documents

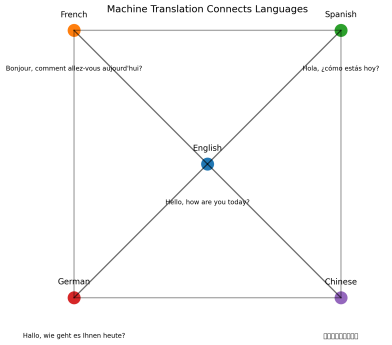


Exemple d'analyse de sentiment sur des critiques de films

Applications du NLP: Traduction et Assistants

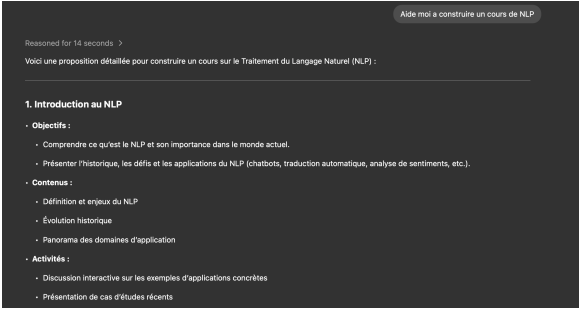
Traduction automatique

- Traduction instantanée entre langues
- Facilite la communication internationale
- Applications: Google Translate, DeepL



Chatbots et assistants virtuels

- Répondent automatiquement aux questions
- Peuvent effectuer des tâches spécifiques
- Applications: assistants vocaux, service client



Représentation des mots en informatique

- En informatique, un **mot** est représenté comme une **séquence de caractères**
- Chaque **caractère** est encodé sur un nombre fini de bits:
  - ASCII: 7 bits (128 caractères)
  - Unicode: jusqu'à 32 bits (plus de 143 000 caractères)

- Pour traiter le langage, les modèles ont besoin de **représentations numériques**
- Le texte doit être converti en nombres pour être compris par les algorithmes
- Cette conversion s'appelle la **tokenisation**

Encodage des caractères

- En informatique, chaque caractère a un **identifiant numérique unique**
  - Exemple simple: a=0, b=1, c=2, etc.
  - En réalité: tables d'encodage (ASCII, Unicode)
- **Tokenisation par caractère**:
  - Chaque caractère = un token
  - Vocabulaire très limité (128-256 tokens)
  - Séquences très longues
- **Avantages**:
  - Pas de mots inconnus
  - Solution universelle pour toutes les langues

Encodage des caractères

Caractère	ID
a	0
b	1
c	2
d	3
e	4
...	...

Correspondance entre caractères et identifiants

Méthodes de tokenisation 2: Mots et sous-mots

- **Tokenisation par mot**:
  - Chaque mot = un token
  - Vocabulaire très grand (100K-1M tokens)
  - Problème des mots inconnus (OOV)
  - Difficulté avec les variations morphologiques
- **Tokenisation par sous-mots**:
  - Compromis entre caractères et mots
  - Vocabulaire raisonnable (10K-50K tokens)
  - Capture les morphèmes et parties de mots
  - Solution privilégiée par les modèles actuels

Tokenisation par mot

Mot	ID
L'intelligence	1
artificielle	2
transforme	3
notre	4
monde.	5

Tokenisation par sous-mots (BPE)

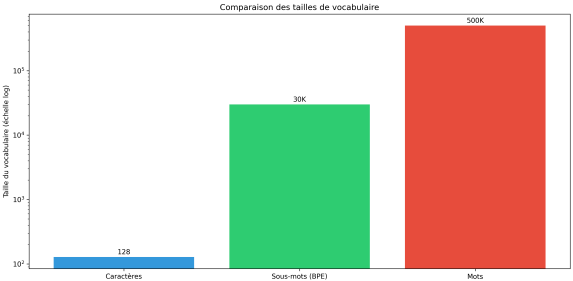
Sous-mot	ID
L'	1
intel	2
ligence	3
arti	4
fici	5
elle	6
trans	7
forme	8
notre	9
monde	10
.	11

Tokenisations par mot et sous-mot de la phrase d'exemple

Comparaison des tailles de vocabulaire

Compromis entre taille et expressivité:

- **Vocabulaire petit** (caractères):
  - Avantage: pas de mots inconnus
  - Inconvénient: séquences très longues
- **Vocabulaire grand** (mots):
  - Avantage: capture directement les unités sémantiques
  - Inconvénient: nombreux mots inconnus
- **Vocabulaire intermédiaire** (sous-mots):
  - Équilibre entre les deux approches
  - Solution adoptée par la plupart des modèles modernes



Comparaison des tailles de vocabulaire selon la méthode

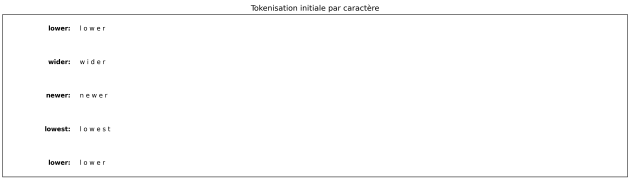
Algorithme BPE (Byte Pair Encoding)

Principe du BPE:

1. Commencer avec un vocabulaire de caractères
2. Identifier les paires de tokens les plus fréquentes
3. Fusionner ces paires pour créer un nouveau token
4. Répéter jusqu'à atteindre la taille de vocabulaire souhaitée

Avantages:

- Adapté au corpus d'entraînement
- Gère efficacement les mots rares
- Capture les morphèmes et affixes
- Utilisé dans GPT, BERT, T5, etc.



Processus de fusion BPE

Étape	Fusion	Nouveau token
1	l + o	lo
2	lo + w	low
3	e + r	er
4	low + er	lower
5	low + e	lowe
6	s + t	st

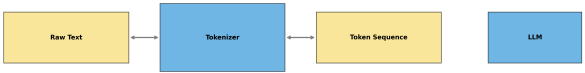
Exemple de fusion progressive avec BPE

Démonstration: Tokenisation en pratique

Démonstration sur Jupyter Notebook

Tokenizers: Un composant distinct du modèle

- Un même modèle peut utiliser différents tokenizers
- Un même tokenizer peut être utilisé par de multiples modèles
- Le choix du tokenizer influence directement les performances du modèle



Le tokenizer est la passerelle entre le langage humain et le langage machine

Points clés

Le tokenizer est une étape de prétraitement **cruciale** et **indépendante** qui conditionne la qualité des représentations du langage dans tous les modèles de NLP.

Au-delà du BPE: Une diversité d'approches pour la tokenisation

Alternatives au BPE:

- **WordPiece** (Google, BERT)
  - Fusion basée sur la probabilité des paires
  - Optimisation de la vraisemblance du corpus
- **Unigram Language Model** (SentencePiece)
  - Approche probabiliste (EM algorithm)
  - Permet plusieurs segmentations possibles
- **Byte-level BPE** (GPT-2, RoBERTa)
  - Opère sur les bytes plutôt que les caractères
  - Vocabulaire universel garanti

Tokenizers spécialisés:

- **CharacterBERT**: Modèle basé sur les caractères
- **SentencePiece**: Solution pour langues non-segmentées (japonais, chinois)
- **CANINE**: Tokenizer-free architecture

Tokens Spéciaux: Au-delà des Mots et Sous-mots

Tokens de contrôle et structure:

- **Début/Fin**:
  - <|endoftext|> (GPT)
  - <s>, </s> (RoBERTa)
- **Traitement de séquence**:
  - <pad> (padding)
  - <unk> (token inconnu)
  - <mask> (pour masked language modeling)
- **Dialogue et conversation**:
  - <|im_start|>, <|im_end|> (ChatGPT)
  - <human>, <assistant> (Claude)
- **Raisonnement**:
  - <think>, </think>
  - <reasoning>, </reasoning>

Tokens pour contenus spécifiques:

- **Programmation**:
  - <code>, </code>
  - <python>, <json>, etc.

Importance des tokens spéciaux

Les tokens spéciaux permettent de:

- Structurer l'information pour le modèle
- Contrôler le comportement de génération
- Améliorer les performances sur des tâches spécifiques
- Communiquer des instructions implicites au modèle

Introduction à la Classification de Sentiment

- La **classification de sentiment** est l'une des applications les plus courantes du NLP
- Elle vise à déterminer l'**opinion** ou l'**émotion** exprimée dans un texte
- Applications:
  - Analyse des avis clients et commentaires sur les produits
  - Surveillance de la réputation de marque sur les réseaux sociaux
  - Étude de la réception d'événements ou de décisions politiques
  - Prédiction des tendances de marché basées sur l'opinion publique
- Différents niveaux de granularité:
  - **Binaire**: positif vs. négatif
  - **Multi-classe**: positif, neutre, négatif, etc.
  - **Analyse fine**: détection de nuances, sarcasme, ironie
  - **Analyse par aspect**: sentiment envers différents aspects d'un produit

Exemples de critiques cinéma - Classification de sentiment

Comment classifieriez-vous le sentiment de ces critiques de films?

Critique 1

A rating of ""1"" does not begin to express how dull, depressing and relentlessly bad this movie is.

Critique 2

My kids picked this out at the video store...it's great to hear Liza as Dorothy cause she sounds just like her mom. But there are too many bad songs, and the animation is pretty crude compared to other cartoons of that time.

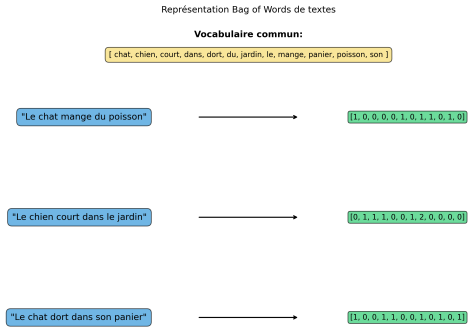
Critique 3

Malcolm McDowell has not had too many good movies lately and this is no different. Especially designed for people who like Yellow filters on their movies.

Bag of Words: Représentation Simplifiée du Texte

Principe du Bag of Words (BoW)

- Représentation simple du texte comme un "sac de mots"
- **L'ordre des mots** est ignoré, seule la **fréquence** compte
- Chaque document devient un vecteur dans l'espace des mots
- Algorithme:
  1. Créer un **vocabulaire** de tous les mots uniques
  2. Pour chaque document, compter les occurrences de chaque mot
  3. Obtenir un vecteur de fréquences par document
- Application: recherche d'information, classification de texte, etc.



Représentation vectorielle de 3 phrases avec BoW

Limites du Bag of Words et Introduction au TF-IDF

Limites du Bag of Words

- Perte de la structure grammaticale et syntaxique
- Tous les mots ont la même importance
- Les mots fréquents (le, la, et, de...) dominent la représentation
- Ne capture pas les relations sémantiques entre les mots
- Problème de dimensionnalité élevée (vocabulaire large)

TF-IDF: Une amélioration importante

- **TF** (Term Frequency): fréquence du terme dans le document
- **IDF** (Inverse Document Frequency): rareté du terme dans le corpus
- **Intuition**: les mots importants apparaissent souvent dans un document mais rarement dans les autres

Avantages du TF-IDF

- Réduit l'importance des mots trop fréquents
- Met en valeur les mots discriminants
- Plus efficace pour de nombreuses tâches de NLP

Formule Mathématique du TF-IDF

Term Frequency (TF)

$$\text{TF}(t, d) = \frac{\text{nombre d'occurrences de } t \text{ dans } d}{\text{nombre total de termes dans } d}$$

Inverse Document Frequency (IDF)

$$\text{IDF}(t) = \log \frac{\text{nombre total de documents}}{\text{nombre de documents contenant } t}$$

TF-IDF

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

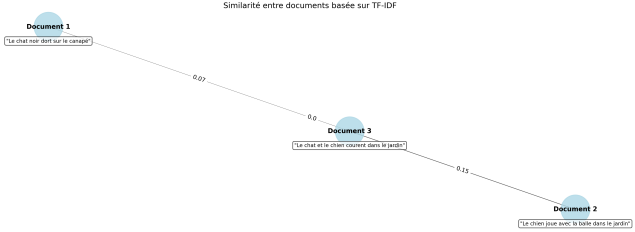
TF-IDF: Calcul et Applications

Calcul du TF-IDF

- Soit un corpus de 3 documents:
  1. "Le chat noir dort sur le canapé"
  2. "Le chien joue avec la balle dans le jardin"
  3. "Le chat et le chien courent dans le jardin"
- Pour le mot "chat":
  - TF(chat, doc1) = 1/7
  - TF(chat, doc2) = 0
  - TF(chat, doc3) = 1/8
  - IDF(chat) = log(3/2) = 0.41
- TF-IDF(chat, doc1) = $1/7 \times 0.41 = 0.059$
- TF-IDF(chat, doc3) = $1/8 \times 0.41 = 0.051$

Calcul des valeurs TF-IDF pour un corpus d'exemple

Mot	TF Doc1	TF Doc2	TF Doc3	IDF	TF-IDF Doc1	TF-IDF Doc2	TF-IDF Doc3
chat	0.143	0.000	0.125	0.41	0.059	0.000	0.051
chien	0.000	0.125	0.125	0.41	0.000	0.051	0.051
noir	0.143	0.000	0.000	1.10	0.157	0.000	0.000
dort	0.143	0.000	0.000	1.10	0.157	0.000	0.000
canapé	0.143	0.000	0.000	1.10	0.157	0.000	0.000
joue	0.000	0.125	0.000	1.10	0.000	0.138	0.000
balle	0.000	0.125	0.000	1.10	0.000	0.138	0.000
jardin	0.000	0.125	0.125	0.41	0.000	0.051	0.051
courent	0.000	0.000	0.125	1.10	0.000	0.000	0.138

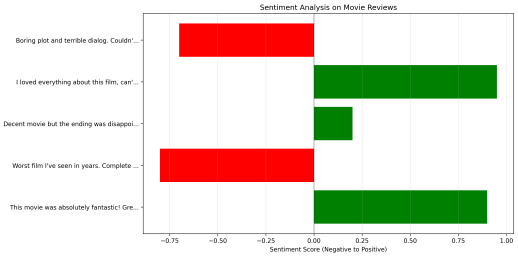


Calcul et application du TF-IDF à un corpus de documents

Classification de Sentiment avec Apprentissage Supervisé

Pipeline de classification classique

1. **Prétraitement** du texte
   - Tokenisation, suppression des stop words
   - Normalisation (minuscules, lemmatisation...)
2. **Extraction de caractéristiques**
   - Vectorisation (BoW ou TF-IDF)
   - Éventuellement, ajout de caractéristiques spécifiques (présence d'émoticons, nombre d'adjectifs...)
3. **Entraînement d'un classifieur**
   - Naïve Bayes, Régression Logistique, SVM...
   - Évaluation avec validation croisée
4. **Prédiction** sur de nouvelles données



Résultats d'analyse de sentiment sur des critiques

Avantages

- Modèles légers et interprétables
- Rapides à entraîner et déployer
- Efficaces pour des tâches simples

Exemple Concret: Classification de Sentiment

Notre phrase d'exemple

"Ce film était vraiment terrible, je ne le recommande à personne!"

- Nous allons suivre toutes les étapes pour classifier cette critique de film:
  1. **Prétraitement**: normalisation, lemmatisation, stop words, tokenisation
  2. **Vectorisation**: représentation en Bag of Words
  3. **Classification**: entraînement d'un Perceptron Multi-couches
  4. **Évaluation**: prédiction du sentiment
- Sur un corpus réel, ces étapes seraient appliquées à des milliers d'exemples

Prétraitement de la Phrase

1. **Normalisation en minuscules**:
   - "ce film était vraiment terrible, je ne le recommande à personne!"
2. **Lemmatisation**:
   - "ce film être vraiment terrible, je ne le recommander à personne!"
3. **Suppression des stop words**:
   - ~~ce~~ film ~~être vraiment~~ terrible, ~~je ne le~~ recommander ~~à~~ personne~~!~~
   - $\rightarrow$ "film terrible recommander personne"
4. **Tokenisation par mot**:
   - ["film", "terrible", "recommander", "personne"] $\rightarrow$ $[4, 9, 8, 7]$

Stop words en français

- le, la, les, un, une, des
- de, du, à, au, aux
- je, tu, il, elle, nous, vous
- est, sont, être, avoir
- et, ou, mais, donc, car
- très, peu, plus, vraiment
- ...

Vectorisation avec Bag of Words

1. **Notre exemple tokenisé**:
   - ["film", "terrible", "recommander", "personne"] $\rightarrow [4, 9, 8, 7]$
2. **Vocabulaire du corpus**:
   - ["aimer", "bon", "excellent", "film", "mauvais", "nul", "personne", "recommander", "terrible", "voir", ...]
   - *Indices*: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ...]
3. **Vecteur Bag of Words**:
   - $[0, 0, 0, 1, 0, 0, 1, 1, 1, 0, ...]$

Note sur les dimensions

Dans un cas réel, ce vecteur aurait typiquement **plusieurs milliers de dimensions**, correspondant à la taille du vocabulaire. Pour notre exemple simplifié, nous travaillons avec un petit vocabulaire.

Choix du Classifieur pour la Classification de Sentiment

Classifieurs traditionnels en NLP:

- **Naïve Bayes**
  - Simple, rapide, efficace pour textes courts
  - Hypothèse d'indépendance des mots
- **SVM (Support Vector Machine)**
  - Très performant pour les espaces à haute dimension
  - Bonne généralisation, robuste
- **Régression Logistique**
  - Probabilités interprétables
  - Bon équilibre complexité/performance

Pourquoi un Perceptron?

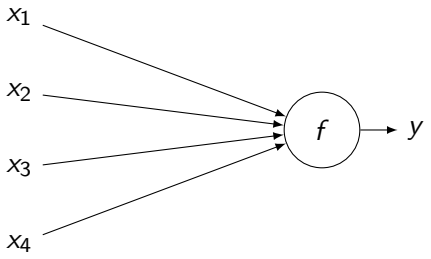
Pour notre exemple, nous utiliserons un **Perceptron Multi-couches** (MLP):

- Base des réseaux de neurones modernes
- Introduit les concepts fondamentaux des LLMs
- Permet de comprendre l'apprentissage par gradient
- Évolutif vers des architectures plus complexes

Le Perceptron: Unité de Base des Réseaux de Neurones

Du Perceptron simple au MLP

- **Neurone/Perceptron**: unité de calcul élémentaire
  - Prend plusieurs entrées pondérées
  - Somme + fonction d'activation
  - $y = f(\sum_i w_i x_i + b)$
- **Rôle dans les réseaux de neurones**:
  - Unité fondamentale de traitement de l'information
  - Peut apprendre des frontières linéaires
- **Limites du perceptron simple**:
  - Ne peut modéliser que des relations linéaires
  - Nécessite des fonctions d'activation non-linéaires
  - Doit être organisé en réseaux complexes pour des tâches avancées



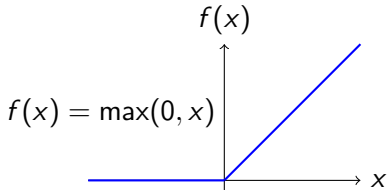
Perceptron vs Neurone Biologique

Bien que simplifié, le perceptron s'inspire du neurone biologique: il reçoit des entrées (comme les dendrites), les pondère (comme les synapses), les agrège et produit une sortie (comme l'axone).

Fonctions d'Activation: Introduire la Non-Linéarité

Pourquoi des fonctions d'activation?

- **Fonctions d'activation**: introduisent la non-linéarité
  - Transforment la somme pondérée en sortie
  - Permettent d'apprendre des motifs complexes
- **Types de fonctions d'activation**:
  - Sigmoïde: $\sigma(x) = \frac{1}{1+e^{-x}}$ (de 0 à 1)
  - Tanh: $\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$ (de -1 à 1)
- **ReLU** (Rectified Linear Unit):
  - $f(x) = \max(0, x)$ - Simple mais efficace
  - Utilisée dans la majorité des réseaux modernes



ReLU

Rectified Linear Unit

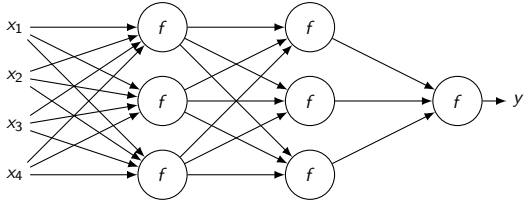
Pourquoi la non-linéarité?

Sans fonctions d'activation non-linéaires, un réseau à plusieurs couches serait équivalent à un simple modèle linéaire, incapable d'apprendre des motifs complexes.

Architecture des Réseaux de Neurones

Organisation en couches

- **Couche de neurones**: ensemble de neurones en parallèle
  - Couche d'entrée: données brutes
  - Couche(s) cachée(s): représentations intermédiaires
  - Couche de sortie: prédictions
- **MLP (Multi-Layer Perceptron)**: réseau à plusieurs couches
  - Capable d'apprendre des frontières non linéaires
  - Pour sentiment: 2 neurones de sortie (positif/négatif)

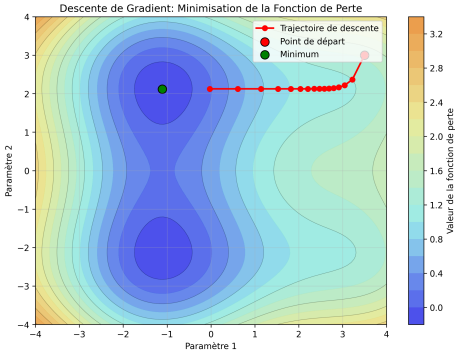


Architecture d'un MLP avec deux couches cachées

Apprentissage d'un Réseau de Neurones

Principes clés de l'apprentissage

1. **Phase de propagation avant (forward)**:
   - Les entrées traversent le réseau de gauche à droite
   - Chaque couche applique sa transformation
   - On obtient une prédiction en sortie
2. **Calcul de l'erreur**:
   - Comparaison entre prédiction et vérité terrain
   - Utilisation d'une fonction de perte (loss)
3. **Rétropropagation (backpropagation)**:
   - Calcul des gradients de l'erreur
   - Propagation des gradients de droite à gauche
4. **Mise à jour des poids**:
   - Utilisation de la descente de gradient
   - $w_{new} = w_{old} - \alpha \cdot \nabla_w L$



Descente de gradient: minimisation de la fonction de perte

Fonctions de Perte pour la Classification

Entropie croisée (Cross-Entropy)

- **Définition mathématique**:

$$L = -\sum_{i=1}^{C} y_i \log(\hat{y}_i) \quad (1)$$

où y_i est la vérité terrain, $\hat{y}_i$ la prédiction pour la classe i

- **Propriétés**:
  - Pénalise fortement les mauvaises prédictions confiantes
  - Récompense les bonnes prédictions confiantes
  - Idéale pour la classification multi-classes

Exemple pour notre cas

Pour notre critique de film négative:

- Vérité terrain: [0, 1] (négatif)
- Mauvaise prédiction: [0.9, 0.1]
- Perte: $-(0 \times \log(0.9) + 1 \times \log(0.1)) \approx 2.3$
- Bonne prédiction: [0.1, 0.9]
- Perte: $-(0 \times \log(0.1) + 1 \times \log(0.9)) \approx 0.1$

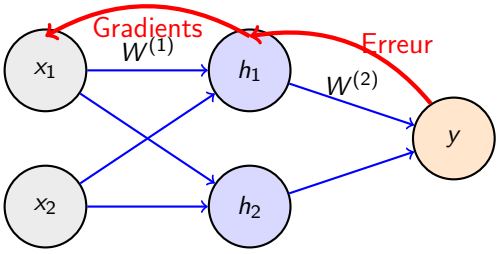
Calcul du Gradient par Rétropropagation

Principe de la rétropropagation

- **Objectif**: calculer $\frac{\partial L}{\partial w_{ij}^{(l)}}$ pour chaque poids
- **Solution**: appliquer la règle de dérivation en chaîne

Mécanisme de la rétropropagation

- **Étape 1**: Calculer l'erreur à la couche de sortie
  - $\delta^{(L)} = \frac{\partial L}{\partial y_{pred}} \cdot f'(z^{(L)})$
- **Étape 2**: Propager l'erreur vers les couches inférieures
  - $\delta^{(l)} = (W^{(l+1)})^T \delta^{(l+1)} \cdot f'(z^{(l)})$
- **Étape 3**: Calculer les gradients pour chaque poids
  - $\frac{\partial L}{\partial W^{(l)}} = a^{(l-1)} \cdot (\delta^{(l)})^T$



Efficacité de la rétropropagation

La rétropropagation permet de calculer tous les gradients en une seule passe arrière, réduisant significativement la complexité de calcul.

Mise à Jour des Poids: Optimisation par Descente de Gradient

Algorithme de descente de gradient

- **Principe**: mise à jour itérative des paramètres
- **Formule générale**:

$$W^{(l)} \leftarrow W^{(l)} - \eta \cdot \frac{\partial L}{\partial W^{(l)}} \qquad (2)$$

où η est le taux d'apprentissage

- **Variantes de l'algorithme**:
  - **Batch**: calcul sur l'ensemble du dataset
  - **Mini-batch**: calcul sur un sous-ensemble
  - **Stochastique (SGD)**: calcul sur un exemple à la fois

Importance du taux d'apprentissage

- **Trop petit**: convergence très lente
- **Trop grand**: oscillations ou divergence
- **Idéal**: adaptatif selon la forme de la fonction de perte

Optimiseurs avancés

- **Adam**: adapte le taux d'apprentissage par

Résumé du Processus Complet de Classification

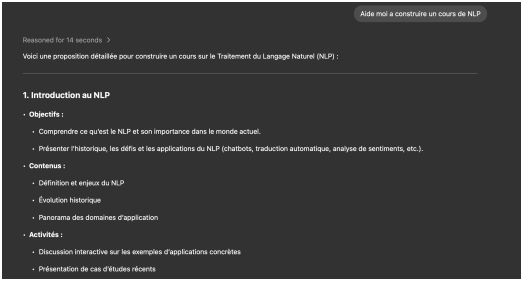
1. **Préparation des données**
   - Prétraitement: normalisation, lemmatisation, stop words
   - Tokenisation: découpage en unités lexicales (mots)
   - Vectorisation: transformation en représentation numérique (BoW)
2. **Conception du modèle**
   - Architecture MLP: couche d'entrée (taille du vocabulaire), couche(s) cachée(s), couche de sortie (2 neurones)
   - Choix des hyperparamètres: nombre de couches, nombre de neurones, fonction d'activation
3. **Entraînement**
   - Propagation avant: calcul des prédictions
   - Calcul de la perte: entropie croisée
   - Rétropropagation: calcul des gradients
   - Optimisation: mise à jour des poids par descente de gradient
4. **Évaluation et prédiction**
   - Métriques: précision, rappel, F1-score
   - Application à de nouvelles données

Génération de Texte

Modèles N-gram et approches statistiques

Introduction à la Génération de Texte

- La **génération de texte** est une tâche fondamentale du NLP
- Elle vise à produire du texte nouveau et cohérent
- Applications:
  - Complétion automatique de phrases
  - Génération de résumés
  - Création de contenu (articles, poésie)
  - Chatbots et assistants conversationnels
- Approches principales:
  - **Modèles statistiques**: N-grams
  - **Réseaux de neurones**: RNN, LSTM, Transformers
  - **Modèles hybrides**: combinaison des approches

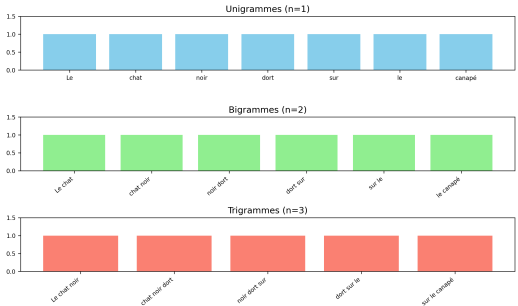


Exemple de génération de texte dans un assistant conversationnel

Qu'est-ce qu'un Modèle N-gram?

Principe des modèles N-gram

- Modèle **statistique** basé sur les probabilités de séquences
- Un n-gram est une **séquence de n éléments** consécutifs
  - **Unigramme (n=1)**: mots individuels
  - **Bigramme (n=2)**: paires de mots consécutifs
  - **Trigramme (n=3)**: triplets de mots consécutifs
- **Hypothèse markovienne**: la probabilité d'un mot dépend uniquement des n-1 mots précédents
  - Simplification qui rend le calcul possible
  - Limitation: ne capture pas les dépendances à long terme



Exemples d'unigrammes, bigrammes et trigrammes

Construction d'un Modèle N-gram (1/2)

Étapes de construction

1. **Collecte d'un corpus** représentatif
2. **Prétraitement** des données textuelles
3. **Extraction des n-grams** de taille n
4. **Comptage des occurrences** de chaque n-gram
5. **Calcul des probabilités conditionnelles**

Formule pour bigrammes:

$$P(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})} \quad (3)$$

Formule générale pour n-grammes:

$$P(w_i|w_{i-n+1}^{i-1}) = \frac{count(w_{i-n+1}^{i})}{count(w_{i-n+1}^{i-1})} \quad (4)$$

Construction d'un Modèle N-gram (2/2)

Exemple de construction d'un modèle bigramme

Corpus: "Le chat noir dort. Le chat blanc joue."

Autres bigrammes:

- (noir, dort): 1 occurrence
- (blanc, joue): 1 occurrence

Bigrammes extraits:

- (Le, chat): 2 occurrences
- (chat, noir): 1 occurrence
- (chat, blanc): 1 occurrence

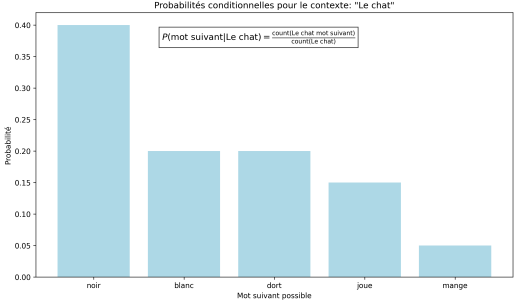
Probabilités:

- $P(chat|Le) = \frac{2}{2} = 1.0$
- $P(noir|chat) = \frac{1}{2} = 0.5$
- $P(blanc|chat) = \frac{1}{2} = 0.5$

Génération de Texte avec N-grams

Algorithme de génération

1. **Choisir un mot/contexte initial**
   - Aléatoirement ou spécifié par l'utilisateur
2. **Pour chaque étape de génération**:
   - Identifier les derniers n-1 mots (contexte)
   - Consulter la table de probabilités conditionnelles
   - Sélectionner le mot suivant selon ces probabilités
   - Ajouter ce mot à la séquence générée
3. **Répéter** jusqu'à un critère d'arrêt:
   - Longueur maximale atteinte
   - Token de fin de phrase/texte généré
   - Impossible de continuer (contexte inconnu)



Probabilités conditionnelles pour générer le mot suivant

Problèmes et Améliorations des Modèles N-gram

Problèmes fondamentaux

- **Données parcimonieuses (sparsity)**
  - De nombreux n-grams valides n'apparaissent jamais dans le corpus
  - Probabilité zéro pour les n-grams non observés
- **Contexte limité**
  - Capture uniquement les dépendances à courte distance
  - Perte de cohérence sur de longs textes
- **Explosion combinatoire**
  - Nombre de n-grams croît exponentiellement avec n
  - Stockage et calcul deviennent prohibitifs

Techniques d'amélioration

- **Lissage (smoothing)**
  - Lissage de Laplace (add-one)
  - Lissage de Good-Turing
  - Lissage de Kneser-Ney
- **Backoff et interpolation**
  - Utiliser des n-grams plus courts quand les plus longs ne sont pas disponibles
  - Combiner les probabilités de différents ordres de n-grams
- **Modèles de classe**
  - Regrouper les mots en classes sémantiques
  - Réduire la parcimonie des données

Exemple Concret de Génération de Texte

Texte d'entraînement (extrait)

Le soleil brille dans le ciel bleu. Les oiseaux chantent dans les arbres. Le vent souffle doucement sur les feuilles. Les enfants jouent dans le parc.

Modèle bigramme (extraits)

- $P(brille|soleil) = 1.0$
- $P(dans|brille) = 1.0$
- $P(le|dans) = 1.0$
- $P(ciel|le) = 0.25$
- $P(parc|le) = 0.25$
- $P(vent|le) = 0.25$
- $P(souffle|vent) = 1.0$

Génération à partir du mot "Le"

- Contexte: "Le"
- Choix possibles: "soleil", "vent", "ciel", "parc" (tous avec p=0.25)
- Supposons que "soleil" est sélectionné
- Nouveau contexte: "soleil"
- Mot suivant: "brille" (p=1.0)
- Et ainsi de suite...

Texte généré possible: "Le soleil brille dans le parc. Les enfants jouent dans les arbres."

N-grams vs. Méthodes Modernes de Génération

Avantages des N-grams

- **Simplicité** conceptuelle et d'implémentation
- **Efficacité** computationnelle (après entraînement)
- **Interprétabilité** des probabilités
- **Peu de données** nécessaires pour des applications simples
- **Domaines spécialisés**: encore utiles pour certaines applications restreintes

Limites face aux méthodes modernes

- **Dépendances à long terme** non capturées
- **Manque de généralisation** sémantique
- **Cohérence** limitée sur de longs textes
- **Compréhension** superficielle du langage
- **Créativité** limitée (reproduit le corpus)

Place des N-grams aujourd'hui

Les N-grams restent une **baseline** importante et sont encore utilisés dans des **systèmes hybrides**, notamment en combinaison avec des modèles neuronaux plus avancés.

Implémentation d'un modèle N-gram en Python

Au-delà des Approches Traditionnelles

Limites des méthodes bag-of-words

- Ne capturent pas la **sémantique** du langage
- Difficultés avec la **négation**: "pas bon" vs "bon"
- Ne gèrent pas bien le langage **figuratif** (sarcasme, ironie)
- Sensibles au **domaine**: modèle entraîné sur des critiques de films peut mal fonctionner sur des critiques de restaurants
- **Dimensionnalité** élevée avec de grands vocabulaires

Évolutions vers les méthodes modernes

- **Word Embeddings** (Word2Vec, GloVe)
  - Représentations denses et continues des mots
  - Capture des relations sémantiques
- **Réseaux de neurones récurrents**
  - LSTM et GRU pour capturer les séquences
- **Architectures Transformer**
  - BERT, RoBERTa pour l'analyse contextuelle
  - GPT pour la génération et classification
- **Méthodes multimodales**
  - Analyse conjointe du texte et d'autres signaux

Évolution des Modèles de Langage à Grande Échelle

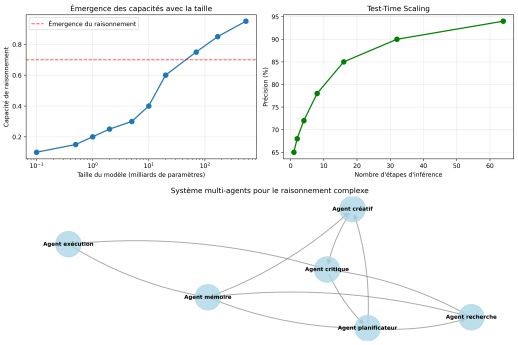
- Les **Large Language Models (LLMs)** représentent une avancée majeure en NLP
- **Évolution rapide** depuis 2018:
  - GPT-1 (0.1B paramètres) $\rightarrow$ GPT-4 ($>$1T paramètres estimés)
  - Augmentation de la taille des modèles = capacités émergentes
- **Capacités actuelles**:
  - Génération de texte cohérent et contextuel
  - Adaptation à différentes tâches sans fine-tuning spécifique
  - Compréhension complexe et résolution de problèmes
  - Génération de code, traduction avancée, créativité
- Ces modèles transforment comment nous **interagissons avec l'information** et **automatisons les tâches cognitives**

Capacités de Raisonnement Avancées

Émergence du raisonnement complexe

- **Test-Time Scaling**: Amélioration des performances avec plus de temps d'inférence
  - Chain-of-Thought (CoT): raisonnement étape par étape
  - Tree-of-Thoughts (ToT): exploration de multiples voies de raisonnement
  - Self-consistency: génération de plusieurs solutions et vote majoritaire

- **Émergence avec l'échelle**
  - Les capacités émergent non-linéairement avec la taille du modèle
  - Point critique où le raisonnement "apparaît"



Émergence des capacités et test-time scaling

Systèmes d'Agents: Intelligence Collective

- **Systèmes multi-agents**:
  - Plusieurs LLMs spécialisés collaborant sur des tâches complexes
  - Chaque agent possède un rôle spécifique: planification, critique, recherche, etc.
  - Communication inter-agents pour résoudre des problèmes multi-étapes

- **Avantages**:
  - **Auto-correction**: les agents peuvent se critiquer mutuellement
  - **Diversité de perspectives**: différentes approches pour un même problème
  - **Spécialisation**: agents optimisés pour certaines tâches spécifiques
  - **Réduction des hallucinations**: vérification croisée des informations

- Applications: recherche scientifique, résolution de problèmes complexes, diagnostic médical, création de contenu

Défis Actuels et Directions Futures

Défis

- **Hallucinations**: génération d'informations incorrectes
- **Biais et équité**: reproduction de biais présents dans les données
- **Alignement**: garantir que les modèles agissent selon les valeurs humaines
- **Efficacité énergétique**: réduire l'empreinte environnementale
- **Sécurité**: prévenir les usages malveillants

Directions de recherche

- **Modèles plus petits mais spécialisés**
- **Apprentissage continu** à partir de l'interaction humaine
- **Intégration de connaissances externes** (outils, bases de données)
- **Amélioration de l'interprétabilité** des modèles
- **Multilinguisme** et accessibilité mondiale
- **Multimodalité**: vision, audio, texte combinés

Le futur du NLP se dirige vers des systèmes hybrides intégrant LLMs, bases de connaissances, et outils spécialisés