RÉPUBLIQUE TOGOLAISE
Ministère de l'Économie Numérique et de la Transformation Digitale
Togo Data Lab

Masterclass organisée par :
Togo Data Lab
Fondements Mathématiques des Transformers et des LLMs

UNIVERSITE DE KARA

Module 1 (Suite) : Les bases mathématiques du Deep Learning pour le NLP

Présentée par : Tiebekabe Pagdame
Enseignant-chercheur - Université de Kara

Date : 14-15 juillet 2025

RÉPUBLIQUE TOGOLAISE
Ministère de l'Économie Numérique et de la Transformation Digitale
Togo Data Lab

RÉPUBLIQUE TOGOLAISE
Ministère de l'Économie Numérique et de la Transformation Digitale

atd AGENCE TOGO DIGITAL

Bienvenue à la Masterclass

Objectifs : poser les fondations mathématiques nécessaires à la compréhension des LLMs.

- Algèbre linéaire appliquée (vecteurs, matrices, tenseurs, produit scalaire, softmax)
- Probabilités et statistiques dans les modèles séquentiels
- Fonction d'activation et optimisation (fonction de perte, descente de gradient)
- Notion d'encodage : embeddings, représentation vectorielle du langage
- Mesure de similarité sémantique (cosine similarity, attention score)

Public cible

- Étudiants en Mathématiques/Informatique et Science des Données
- Étudiants à la Faculté des Sciences et de la Santé
- Chercheurs en NLP
- Professionnels du secteur

Horaires

- Date : 14-15 juillet 2025
- Heure : 9h - 12h
- Lieu : Université de Kara

Sommaire

Definitions

Transformer

Un transformer est une architecture flexible basée sur le mécanisme d'attention, capable de transformer des séquences d'entrée en sorties complexes.

LLM

Un LLM (Large Language Model) est un modèle de langage basé sur des réseaux neuronaux profonds, entraîné sur de grandes quantités de texte pour comprendre et générer du langage humain.

Pourquoi l'Algèbre Linéaire ?

- Les modèles de langage manipulent des vecteurs et des matrices à chaque étape.
- Chaque mot est représenté comme un vecteur dans un espace à d dimensions.
- L'architecture Transformer repose sur des multiplications matricielles massives.

Comprendre les fondements algébriques est essentiel pour interpréter les LLMs.

Vecteurs dans le NLP

- Un vecteur est une liste ordonnée de nombres réels :

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$$

- Représentation vectorielle d'un mot = embedding $\in \mathbb{R}^d$
- Exemple : "`chat`" $\rightarrow \vec{v}_{\text{chat}} \in \mathbb{R}^{300}$

Interprétation : chaque coordonnée encode un aspect sémantique latent (genre, animal, domestique, etc.)

Vecteurs dans le NLP

- Un vecteur est une liste ordonnée de nombres réels :

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$$

- Représentation vectorielle d'un mot = embedding $\in \mathbb{R}^d$
- Exemple : "`chat`" $\rightarrow \vec{v}_{\text{chat}} \in \mathbb{R}^{300}$

Interprétation : chaque coordonnée encode un aspect sémantique latent (genre, animal, domestique, etc.)

Matrices dans les Transformers

- Une matrice est une collection de vecteurs colonnes ou lignes :

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1d} \\ w_{21} & w_{22} & \cdots & w_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{nd} \end{bmatrix}$$

- Exemple : la matrice d'embeddings $E \in \mathbb{R}^{V \times d}$
- Chaque ligne = vecteur associé à un mot du vocabulaire

Tenseurs : une généralisation multidimensionnelle des matrices

- Un **tenseur** est une structure de données mathématique qui généralise les **scalaires** (0D), **vecteurs** (1D) et **matrices** (2D) à des dimensions supérieures.
- Notation : $T \in \mathbb{R}^{B \times T \times d}$ signifie que T est un tenseur à 3 dimensions :
  - B : taille du **batch** (nombre d'exemples traités en parallèle)
  - T : **longueur de la séquence** (ex. nombre de mots dans une phrase)
  - d : **dimension d'un embedding** (représentation vectorielle d'un mot)
- Exemple : $T[b,t,i]$ représente la i^{e} composante de l'embedding du t^{e} mot de la b^{e} phrase du batch.
- Les tenseurs permettent de représenter efficacement des données complexes comme :
  - Un lot de phrases vectorisées (texte)
  - Une séquence d'images ou de frames (vidéo)
  - Un volume médical 3D (scanner)
- **Transformers** : modèles qui manipulent ces tenseurs en appliquant des opérations linéaires (produits matriciels, projections) et non linéaires (softmax, attention).

À retenir : les tenseurs sont au coeur des architectures modernes d'apprentissage profond (transformers, CNN, RNN).

Produit Scalaire et Similarité

- **Produit scalaire** de deux vecteurs $\vec{u}, \vec{v} \in \mathbb{R}^d$:

$$\vec{u} \cdot \vec{v} = \sum_{i=1}^{d} u_i v_i$$

- C'est une mesure d'alignement ou de corrélation directionnelle entre $\vec{u}$ et $\vec{v}$.
- **Similarité cosinus** : mesure la similarité entre deux vecteurs, indépendamment de leur norme :

$$\cos(\theta) = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \|\vec{v}\|} \in [-1, 1]$$

  - $= 1$ si les vecteurs pointent dans la même direction
  - $= 0$ s'ils sont orthogonaux (pas similaires)
  - < 0 s'ils pointent dans des directions opposées
- **Exemple concret** : Soient les représentations vectorielles de deux mots :

$$\vec{u} = [1, 2], \quad \vec{v} = [2, 3]$$

$$\vec{u} \cdot \vec{v} = 1 \times 2 + 2 \times 3 = 8$$

$$\|\vec{u}\| = \sqrt{1^2 + 2^2} = \sqrt{5}, \quad \|\vec{v}\| = \sqrt{13}$$

$$\cos(\theta) = \frac{8}{\sqrt{5 \times 13}} \approx 0{,}99$$

  Les deux mots sont fortement similaires.
- **Applications** :
  - **Attention** : mesure de la pertinence entre un mot "requête" et un mot "clé"
  - **Recherche sémantique** : retrouver un document proche d'une requête

Produit Scalaire et Similarité

- **Produit scalaire** de deux vecteurs $\vec{u}, \vec{v} \in \mathbb{R}^d$:

$$\vec{u} \cdot \vec{v} = \sum_{i=1}^{d} u_i v_i$$

- C'est une mesure d'alignement ou de corrélation directionnelle entre $\vec{u}$ et $\vec{v}$.
- **Similarité cosinus** : mesure la similarité entre deux vecteurs, indépendamment de leur norme :

$$\cos(\theta) = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \|\vec{v}\|} \in [-1, 1]$$

  - $= 1$ si les vecteurs pointent dans la même direction
  - $= 0$ s'ils sont orthogonaux (pas similaires)
  - < 0 s'ils pointent dans des directions opposées

- **Exemple concret** : Soient les représentations vectorielles de deux mots :

$$\vec{u} = [1, 2], \quad \vec{v} = [2, 3]$$

$$\vec{u} \cdot \vec{v} = 1 \times 2 + 2 \times 3 = 8$$

$$\|\vec{u}\| = \sqrt{1^2 + 2^2} = \sqrt{5}, \quad \|\vec{v}\| = \sqrt{13}$$

$$\cos(\theta) = \frac{8}{\sqrt{5 \times 13}} \approx 0{,}99$$

  Les deux mots sont fortement similaires.

- **Applications** :
  - **Attention** : mesure de la pertinence entre un mot "requête" et un mot "clé"
  - **Recherche sémantique** : retrouver un document proche d'une requête

Produit Scalaire et Similarité

- **Produit scalaire** de deux vecteurs $\vec{u}, \vec{v} \in \mathbb{R}^d$:

$$\vec{u} \cdot \vec{v} = \sum_{i=1}^{d} u_i v_i$$

- C'est une mesure d'alignement ou de corrélation directionnelle entre $\vec{u}$ et $\vec{v}$.
- **Similarité cosinus** : mesure la similarité entre deux vecteurs, indépendamment de leur norme :

$$\cos(\theta) = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \|\vec{v}\|} \in [-1, 1]$$

  - $= 1$ si les vecteurs pointent dans la même direction
  - $= 0$ s'ils sont orthogonaux (pas similaires)
  - < 0 s'ils pointent dans des directions opposées
- **Exemple concret** : Soient les représentations vectorielles de deux mots :

$$\vec{u} = [1, 2], \quad \vec{v} = [2, 3]$$

$$\vec{u} \cdot \vec{v} = 1 \times 2 + 2 \times 3 = 8$$

$$\|\vec{u}\| = \sqrt{1^2 + 2^2} = \sqrt{5}, \quad \|\vec{v}\| = \sqrt{13}$$

$$\cos(\theta) = \frac{8}{\sqrt{5 \times 13}} \approx 0{,}99$$

Les deux mots sont fortement similaires.

- **Applications** :
  - **Attention** : mesure de la pertinence entre un mot "requête" et un mot "clé"
  - **Recherche sémantique** : retrouver un document proche d'une requête

Produit Scalaire et Similarité

- **Produit scalaire** de deux vecteurs $\vec{u}, \vec{v} \in \mathbb{R}^d$:

$$\vec{u} \cdot \vec{v} = \sum_{i=1}^{d} u_i v_i$$

- C'est une mesure d'alignement ou de corrélation directionnelle entre $\vec{u}$ et $\vec{v}$.
- **Similarité cosinus** : mesure la similarité entre deux vecteurs, indépendamment de leur norme :

$$\cos(\theta) = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \|\vec{v}\|} \in [-1, 1]$$

  - $= 1$ si les vecteurs pointent dans la même direction
  - $= 0$ s'ils sont orthogonaux (pas similaires)
  - < 0 s'ils pointent dans des directions opposées
- **Exemple concret** : Soient les représentations vectorielles de deux mots :

$$\vec{u} = [1, 2], \quad \vec{v} = [2, 3]$$

$$\vec{u} \cdot \vec{v} = 1 \times 2 + 2 \times 3 = 8$$

$$\|\vec{u}\| = \sqrt{1^2 + 2^2} = \sqrt{5}, \quad \|\vec{v}\| = \sqrt{13}$$

$$\cos(\theta) = \frac{8}{\sqrt{5 \times 13}} \approx 0{,}99$$

  Les deux mots sont fortement similaires.
- **Applications** :
  - **Attention** : mesure de la pertinence entre un mot "requête" et un mot "clé"
  - **Recherche sémantique** : retrouver un document proche d'une requête

Fonction Softmax : transformation en distribution de probabilité

- La fonction **softmax** transforme un vecteur de scores réels $(z_1, \ldots, z_n)$ en une **distribution de probabilité** sur n éléments :

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

- **Interprétation** :
  - Amplifie les différences entre les scores (les plus grands deviennent dominants).
  - Chaque z_i est exponentié les écarts sont accentués.
  - Résultat : des poids positifs normalisés (somme = 1) comparable à une probabilité.
- **Exemple pratique** :

$$\text{Soit } \vec{z} = [2,1,0] \Rightarrow e^z = [e^2, e^1, e^0] \approx [7{,}39, 2{,}72, 1]$$

$$\text{softmax}(z) \approx \left[\frac{7{,}39}{11{,}11}, \frac{2{,}72}{11{,}11}, \frac{1}{11{,}11}\right] \approx [0{,}665, 0{,}245, 0{,}090]$$

  Le score 2 reçoit le poids le plus élevé.
- **Application concrète** :
  - Dans les **transformers**, on calcule une similarité (produit scalaire) entre les vecteurs "requête" et "clé".
  - Ces scores sont passés par une **softmax** pour en faire des **poids d'attention**.
  - Chaque mot du contexte est pondéré selon sa pertinence pour produire une représentation contextuelle.
- **Propriétés fondamentales** :
  - $0 < \text{softmax}(z_i) < 1$ pour tout i
  - $\sum_i \text{softmax}(z_i) = 1$ (distribution de probabilité)

Fonction Softmax : transformation en distribution de probabilité

- La fonction **softmax** transforme un vecteur de scores réels $(z_1, \ldots, z_n)$ en une **distribution de probabilité** sur n éléments :

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

- **Interprétation** :
  - Amplifie les différences entre les scores (les plus grands deviennent dominants).
  - Chaque z_i est exponentié les écarts sont accentués.
  - Résultat : des poids positifs normalisés (somme = 1) comparable à une probabilité.
- **Exemple pratique** :

$$\text{Soit } \vec{z} = [2,1,0] \Rightarrow e^z = [e^2, e^1, e^0] \approx [7{,}39, 2{,}72, 1]$$

$$\text{softmax}(z) \approx \left[\frac{7{,}39}{11{,}11}, \frac{2{,}72}{11{,}11}, \frac{1}{11{,}11}\right] \approx [0{,}665, 0{,}245, 0{,}090]$$

  Le score 2 reçoit le poids le plus élevé.
- **Application concrète** :
  - Dans les **transformers**, on calcule une similarité (produit scalaire) entre les vecteurs "requête" et "clé".
  - Ces scores sont passés par une **softmax** pour en faire des **poids d'attention**.
  - Chaque mot du contexte est pondéré selon sa pertinence pour produire une représentation contextuelle.
- **Propriétés fondamentales** :
  - $0 < \text{softmax}(z_i) < 1$ pour tout i
  - $\sum_i \text{softmax}(z_i) = 1$ (distribution de probabilité)

Fonction Softmax : transformation en distribution de probabilité

- La fonction **softmax** transforme un vecteur de scores réels $(z_1, \ldots, z_n)$ en une **distribution de probabilité** sur n éléments :

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

- **Interprétation** :
  - Amplifie les différences entre les scores (les plus grands deviennent dominants).
  - Chaque z_i est exponentié les écarts sont accentués.
  - Résultat : des poids positifs normalisés (somme = 1) comparable à une probabilité.
- **Exemple pratique** :

$$\text{Soit } \vec{z} = [2,1,0] \Rightarrow e^z = [e^2, e^1, e^0] \approx [7{,}39, 2{,}72, 1]$$

$$\text{softmax}(z) \approx \left[\frac{7{,}39}{11{,}11}, \frac{2{,}72}{11{,}11}, \frac{1}{11{,}11}\right] \approx [0{,}665, 0{,}245, 0{,}090]$$

  Le score 2 reçoit le poids le plus élevé.
- **Application concrète** :
  - Dans les **transformers**, on calcule une similarité (produit scalaire) entre les vecteurs "requête" et "clé".
  - Ces scores sont passés par une **softmax** pour en faire des **poids d'attention**.
  - Chaque mot du contexte est pondéré selon sa pertinence pour produire une représentation contextuelle.
- **Propriétés fondamentales** :
  - $0 < \text{softmax}(z_i) < 1$ pour tout i
  - $\sum_i \text{softmax}(z_i) = 1$ (distribution de probabilité)

Fonction Softmax : transformation en distribution de probabilité

- La fonction **softmax** transforme un vecteur de scores réels $(z_1, \ldots, z_n)$ en une **distribution de probabilité** sur n éléments :

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

- **Interprétation** :
  - Amplifie les différences entre les scores (les plus grands deviennent dominants).
  - Chaque z_i est exponentié les écarts sont accentués.
  - Résultat : des poids positifs normalisés (somme = 1) comparable à une probabilité.
- **Exemple pratique** :

$$\text{Soit } \vec{z} = [2,1,0] \Rightarrow e^z = [e^2, e^1, e^0] \approx [7{,}39, 2{,}72, 1]$$

$$\text{softmax}(z) \approx \left[\frac{7{,}39}{11{,}11}, \frac{2{,}72}{11{,}11}, \frac{1}{11{,}11}\right] \approx [0{,}665, 0{,}245, 0{,}090]$$

  Le score 2 reçoit le poids le plus élevé.
- **Application concrète** :
  - Dans les **transformers**, on calcule une similarité (produit scalaire) entre les vecteurs "requête" et "clé".
  - Ces scores sont passés par une **softmax** pour en faire des **poids d'attention**.
  - Chaque mot du contexte est pondéré selon sa pertinence pour produire une représentation contextuelle.
- **Propriétés fondamentales** :
  - $0 < \text{softmax}(z_i) < 1$ pour tout i
  - $\sum_i \text{softmax}(z_i) = 1$ (distribution de probabilité)

Fonction Softmax : transformation en distribution de probabilité

- La fonction **softmax** transforme un vecteur de scores réels $(z_1, \ldots, z_n)$ en une **distribution de probabilité** sur n éléments :

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

- **Interprétation** :
  - Amplifie les différences entre les scores (les plus grands deviennent dominants).
  - Chaque z_i est exponentié les écarts sont accentués.
  - Résultat : des poids positifs normalisés (somme = 1) comparable à une probabilité.
- **Exemple pratique** :

$$\text{Soit } \vec{z} = [2,1,0] \Rightarrow e^z = [e^2, e^1, e^0] \approx [7{,}39, 2{,}72, 1]$$

$$\text{softmax}(z) \approx \left[\frac{7{,}39}{11{,}11}, \frac{2{,}72}{11{,}11}, \frac{1}{11{,}11}\right] \approx [0{,}665, 0{,}245, 0{,}090]$$

  Le score 2 reçoit le poids le plus élevé.
- **Application concrète** :
  - Dans les **transformers**, on calcule une similarité (produit scalaire) entre les vecteurs "requête" et "clé".
  - Ces scores sont passés par une **softmax** pour en faire des **poids d'attention**.
  - Chaque mot du contexte est pondéré selon sa pertinence pour produire une représentation contextuelle.
- **Propriétés fondamentales** :
  - $0 < \text{softmax}(z_i) < 1$ pour tout i
  - $\sum_i \text{softmax}(z_i) = 1$ (distribution de probabilité)

Briques de base prêtes

- Les vecteurs représentent les mots.
- Les matrices transforment et projettent ces vecteurs.
- Les tenseurs manipulent les phrases complètes.
- Le produit scalaire mesure la similarité.
- Le softmax normalise pour pondérer les informations.

Prochaine étape : découvrir comment ces concepts construisent le coeur du Transformer.

Pourquoi les Probabilités ?

- Le langage humain est **incertain**, **variable** et souvent **ambigu**.
- Il peut être modélisé comme une **distribution de probabilité** sur des séquences de mots.
- Objectif des modèles de langage :

$$P(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} P(w_t \mid w_1, \ldots, w_{t-1})$$

- Chaque mot dépend du **contexte précédent**, d'où l'intérêt des modèles séquentiels.

Un LLM est un estimateur probabiliste entraîné à prédire les mots dans de longues séquences.

Pourquoi les Probabilités ?

- Le langage humain est **incertain**, **variable** et souvent **ambigu**.
- Il peut être modélisé comme une **distribution de probabilité** sur des séquences de mots.
- Objectif des modèles de langage :

$$P(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} P(w_t \mid w_1, \ldots, w_{t-1})$$

- Chaque mot dépend du **contexte précédent**, d'où l'intérêt des modèles séquentiels.

Un LLM est un estimateur probabiliste entraîné à prédire les mots dans de longues séquences.

Distributions de Probabilité sur un Vocabulaire

- Soit V un vocabulaire fini de taille $|V|$ (ex. : les 50 000 mots les plus fréquents).
- On modélise un mot $w \in V$ comme une **réalisation** d'une variable aléatoire W.
- La loi de probabilité de W est donnée par :

$$P(W = w_i), \quad \text{avec} \sum_{i=1}^{|V|} P(w_i) = 1$$

- **Interprétation** : $P(w_i)$ est la probabilité d'observer le mot w_i dans un contexte donné.
- Dans un corpus, on peut approximer cette loi par une distribution empirique :

$$\hat{P}(w_i) = \frac{\text{Nombre d'apparitions de } w_i}{\text{Nombre total de mots observés}}$$

- Cette estimation est la base de nombreux modèles statistiques de langage (n-gram, unigramme, etc.).

But d'un modèle de langage : approximer cette distribution, conditionnée par le contexte.

Distributions de Probabilité sur un Vocabulaire

- Soit V un vocabulaire fini de taille $|V|$ (ex. : les 50 000 mots les plus fréquents).
- On modélise un mot $w \in V$ comme une **réalisation** d'une variable aléatoire W.
- La loi de probabilité de W est donnée par :

$$P(W = w_i), \quad \text{avec} \sum_{i=1}^{|V|} P(w_i) = 1$$

- **Interprétation** : $P(w_i)$ est la probabilité d'observer le mot w_i dans un contexte donné.
- Dans un corpus, on peut approximer cette loi par une distribution empirique :

$$\hat{P}(w_i) = \frac{\text{Nombre d'apparitions de } w_i}{\text{Nombre total de mots observés}}$$

- Cette estimation est la base de nombreux modèles statistiques de langage (n-gram, unigramme, etc.).

But d'un modèle de langage : approximer cette distribution, conditionnée par le contexte.

Modèles de Langage : Hypothèse de Markov

- **Objectif fondamental** : estimer la probabilité jointe d'une séquence de mots :

$$P(w_1, w_2, \ldots, w_T)$$

- **Décomposition par la règle de la chaîne (ou règle du produit)** :

$$P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t \mid w_1, w_2, \ldots, w_{t-1})$$

- **Hypothèse de Markov d'ordre** $n-1$: on suppose que le mot w_t ne dépend que des $n-1$ derniers mots :

$$P(w_t \mid w_1, \ldots, w_{t-1}) \approx P(w_t \mid w_{t-n+1}, \ldots, w_{t-1})$$

- **Exemple : modèle bigramme ($n = 2$)** :

$$P(w_1, w_2, \ldots, w_T) \approx P(w_1) \prod_{t=2}^{T} P(w_t \mid w_{t-1})$$

- Cette hypothèse permet de réduire la complexité du modèle tout en restant raisonnable pour de courtes séquences.

Limite : fenêtre de contexte fixe $\Rightarrow$ incapacité à capturer des dépendances à long terme.

Modèles de Langage : Hypothèse de Markov

- **Objectif fondamental** : estimer la probabilité jointe d'une séquence de mots :

$$P(w_1, w_2, \ldots, w_T)$$

- **Décomposition par la règle de la chaîne (ou règle du produit)** :

$$P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t \mid w_1, w_2, \ldots, w_{t-1})$$

- **Hypothèse de Markov d'ordre** $n-1$: on suppose que le mot w_t ne dépend que des $n-1$ derniers mots :

$$P(w_t \mid w_1, \ldots, w_{t-1}) \approx P(w_t \mid w_{t-n+1}, \ldots, w_{t-1})$$

- **Exemple : modèle bigramme ($n = 2$)** :

$$P(w_1, w_2, \ldots, w_T) \approx P(w_1) \prod_{t=2}^{T} P(w_t \mid w_{t-1})$$

- Cette hypothèse permet de réduire la complexité du modèle tout en restant raisonnable pour de courtes séquences.

Limite : fenêtre de contexte fixe $\Rightarrow$ incapacité à capturer des dépendances à long terme.

Estimation des Probabilités : Maximum de Vraisemblance (MLE)

- Soit un corpus C contenant N mots, formant une séquence $w_1, w_2, \ldots, w_N$ sur un vocabulaire fini V.
- Objectif : estimer la probabilité conditionnelle $P(w_t \mid w_{t-1})$ à partir des données observées dans C.
- **Estimateur du Maximum de Vraisemblance (MLE)** pour un modèle bigramme :

$$\hat{P}(w_t \mid w_{t-1}) = \frac{\text{Count}(w_{t-1}, w_t)}{\text{Count}(w_{t-1})}$$

où :

  - $\text{Count}(w_{t-1}, w_t)$ est le nombre de fois que w_t suit w_{t-1} dans le corpus.
  - $\text{Count}(w_{t-1}) = \sum_{w \in V} \text{Count}(w_{t-1}, w)$ est le nombre total de fois où w_{t-1} apparaît comme mot précédent.

- Cette estimation repose sur la fréquence relative observée dans le corpus : on suppose que la fréquence observée converge vers la probabilité vraie lorsque la taille du corpus tend vers l'infini.

Problème : certaines transitions (w_{t-1}, w_t) peuvent ne jamais apparaître $\Rightarrow$ **probabilités nulles**.

Solution : *Techniques de lissage* pour éviter les zéros :

- Lissage additif (Laplace) : $\hat{P}(w_t \mid w_{t-1}) = \dfrac{\text{Count}(w_{t-1}, w_t) + \alpha}{\text{Count}(w_{t-1}) + \alpha |V|}$
- Lissage de Kneser-Ney (plus performant pour les grands corpus)

Estimation des Probabilités : Maximum de Vraisemblance (MLE)

- Soit un corpus C contenant N mots, formant une séquence $w_1, w_2, \ldots, w_N$ sur un vocabulaire fini V.
- Objectif : estimer la probabilité conditionnelle $P(w_t \mid w_{t-1})$ à partir des données observées dans C.
- **Estimateur du Maximum de Vraisemblance (MLE)** pour un modèle bigramme :

$$\hat{P}(w_t \mid w_{t-1}) = \frac{\text{Count}(w_{t-1}, w_t)}{\text{Count}(w_{t-1})}$$

où :

  - $\text{Count}(w_{t-1}, w_t)$ est le nombre de fois que w_t suit w_{t-1} dans le corpus.
  - $\text{Count}(w_{t-1}) = \sum_{w \in V} \text{Count}(w_{t-1}, w)$ est le nombre total de fois où w_{t-1} apparaît comme mot précédent.

- Cette estimation repose sur la fréquence relative observée dans le corpus : on suppose que la fréquence observée converge vers la probabilité vraie lorsque la taille du corpus tend vers l'infini.

Problème : certaines transitions (w_{t-1}, w_t) peuvent ne jamais apparaître $\Rightarrow$ **probabilités nulles**.

Solution : *Techniques de lissage* pour éviter les zéros :

- Lissage additif (Laplace) : $\hat{P}(w_t \mid w_{t-1}) = \dfrac{\text{Count}(w_{t-1}, w_t) + \alpha}{\text{Count}(w_{t-1}) + \alpha |V|}$
- Lissage de Kneser-Ney (plus performant pour les grands corpus)

Entropie : Quantifier l'Incertitude

- Soit une variable aléatoire W à valeurs dans un vocabulaire fini $V = \{w_1, w_2, \ldots, w_{|V|}\}$.
- Sa loi de probabilité P est donnée par : $P(w_i) \geq 0$ et $\sum_{i=1}^{|V|} P(w_i) = 1$.
- **L'entropie de Shannon** est définie par :

$$H(P) = -\sum_{i=1}^{|V|} P(w_i) \log P(w_i)$$

- Elle mesure l'incertitude moyenne (en bits si $\log$ en base 2) :
  - $H(P)$ est minimale (= 0) si P est une distribution **dirac** (certitude absolue).
  - $H(P)$ est maximale si P est **uniforme** : $P(w_i) = \frac{1}{|V|}$ pour tout i.
- **Interprétation** :
  - **Faible entropie** : la distribution est concentrée ⇒ le modèle est *confiant*.
  - **Haute entropie** : la distribution est diffuse ⇒ le modèle est *incertain*.
- **Application** : utilisée pour évaluer la "connaissance" d'un modèle de langage sur une tâche donnée, par exemple via la perplexité :

$$\text{Perplexité} = 2^{H(P)} \quad \text{(mesure de difficulté moyenne à prédire un mot)}$$

Perplexité : Qualité d'un Modèle de Langage

- Soit une séquence de mots $w_1, w_2, \ldots, w_T$ issue d'un vocabulaire V, et un modèle de langage P qui assigne des probabilités conditionnelles $P(w_t \mid w_1, \ldots, w_{t-1})$.
- **Vraisemblance logarithmique moyenne** :

$$\ell(P) = \frac{1}{T} \sum_{t=1}^{T} \log P(w_t \mid w_{<t})$$

- **Perplexité (PPL)** : mesure exponentielle de l'entropie moyenne sur la séquence :

$$\text{PPL}(P) = \exp\left(-\ell(P)\right) = \exp\left(-\frac{1}{T} \sum_{t=1}^{T} \log P(w_t \mid w_{<t})\right)$$

- **Interprétation** :
  - La perplexité est le nombre effectif de mots également probables que le modèle considère en moyenne à chaque position.
  - Plus $P(w_t \mid w_{<t})$ est concentré sur le bon mot, plus $\log P(w_t \mid w_{<t})$ est grand, donc la perplexité est faible.
  - **PPL minimale** = 1 le modèle prédit toujours correctement avec probabilité 1.
- **Comparaison de modèles** :
  - Les **modèles n-gram** ont des PPL élevées (souvent > 100 sur des corpus ouverts).
  - Les **Transformers** (BERT, GPT, etc.) ont des PPL beaucoup plus faibles (souvent < 20), grâce à leur capacité de généralisation et de capture du contexte long.

La perplexité est une mesure importante pour l'évaluation quantitative des modèles de langage.

Perplexité : Qualité d'un Modèle de Langage

- Soit une séquence de mots $w_1, w_2, \ldots, w_T$ issue d'un vocabulaire V, et un modèle de langage P qui assigne des probabilités conditionnelles $P(w_t \mid w_1, \ldots, w_{t-1})$.
- **Vraisemblance logarithmique moyenne** :

$$\ell(P) = \frac{1}{T} \sum_{t=1}^{T} \log P(w_t \mid w_{<t})$$

- **Perplexité (PPL)** : mesure exponentielle de l'entropie moyenne sur la séquence :

$$\text{PPL}(P) = \exp\left(-\ell(P)\right) = \exp\left(-\frac{1}{T} \sum_{t=1}^{T} \log P(w_t \mid w_{<t})\right)$$

- **Interprétation** :
  - La perplexité est le nombre effectif de mots également probables que le modèle considère en moyenne à chaque position.
  - Plus $P(w_t \mid w_{<t})$ est concentré sur le bon mot, plus $\log P(w_t \mid w_{<t})$ est grand, donc la perplexité est faible.
  - **PPL minimale** = 1 le modèle prédit toujours correctement avec probabilité 1.
- **Comparaison de modèles** :
  - Les **modèles n-gram** ont des PPL élevées (souvent > 100 sur des corpus ouverts).
  - Les **Transformers** (BERT, GPT, etc.) ont des PPL beaucoup plus faibles (souvent < 20), grâce à leur capacité de généralisation et de capture du contexte long.

La perplexité est une mesure importante pour l'évaluation quantitative des modèles de langage.

Vers les Transformers

- Les **grands modèles de langage** (LLMs) cherchent à approximer les probabilités conditionnelles :

$$P(w_t \mid w_1, \ldots, w_{t-1}) = P(w_t \mid w_{<t})$$

à l'aide d'architectures de **réseaux de neurones profonds**, en particulier les Transformers.

- Chaque mot w_t est représenté par un vecteur dense $\mathbf{x}_t \in \mathbb{R}^d$ obtenu via un **embedding** :

$$\mathbf{x}_t = \text{Embed}(w_t)$$

- La séquence d'entrées $\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}$ est transformée via un **encodeur Transformer** (auto-attention, normalisation, etc.) en un vecteur de contexte $h_t \in \mathbb{R}^d$:

$$h_t = f_\theta(\mathbf{x}_1, \ldots, \mathbf{x}_{t-1})$$

- Le modèle calcule alors une distribution de probabilité sur le vocabulaire V par la fonction softmax :

$$P(w_t = v_i \mid w_{<t}) = \frac{\exp(\langle \mathbf{w}_i, h_t \rangle + b_i)}{\sum_{j=1}^{|V|} \exp(\langle \mathbf{w}_j, h_t \rangle + b_j)}$$

où $\mathbf{w}_i$ est le vecteur de poids associé au mot v_i dans la matrice $W \in \mathbb{R}^{|V| \times d}$.

- En notation matricielle compacte :

$$\mathbf{p}_t = \text{softmax}(W h_t + b)$$

où $\mathbf{p}_t \in \mathbb{R}^{|V|}$ est la distribution prédite.

Les Transformers combinent une modélisation séquentielle, une attention contextuelle et une sortie probabiliste sur le vocabulaire.

Vers les Transformers

- Les **grands modèles de langage** (LLMs) cherchent à approximer les probabilités conditionnelles :

$$P(w_t \mid w_1, \ldots, w_{t-1}) = P(w_t \mid w_{<t})$$

à l'aide d'architectures de **réseaux de neurones profonds**, en particulier les Transformers.

- Chaque mot w_t est représenté par un vecteur dense $\mathbf{x}_t \in \mathbb{R}^d$ obtenu via un **embedding** :

$$\mathbf{x}_t = \text{Embed}(w_t)$$

- La séquence d'entrées $\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}$ est transformée via un **encodeur Transformer** (auto-attention, normalisation, etc.) en un vecteur de contexte $h_t \in \mathbb{R}^d$:

$$h_t = f_\theta(\mathbf{x}_1, \ldots, \mathbf{x}_{t-1})$$

- Le modèle calcule alors une distribution de probabilité sur le vocabulaire V par la fonction softmax :

$$P(w_t = v_i \mid w_{<t}) = \frac{\exp(\langle \mathbf{w}_i, h_t \rangle + b_i)}{\sum_{j=1}^{|V|} \exp(\langle \mathbf{w}_j, h_t \rangle + b_j)}$$

où $\mathbf{w}_i$ est le vecteur de poids associé au mot v_i dans la matrice $W \in \mathbb{R}^{|V| \times d}$.

- En notation matricielle compacte :

$$\mathbf{p}_t = \text{softmax}(W h_t + b)$$

où $\mathbf{p}_t \in \mathbb{R}^{|V|}$ est la distribution prédite.

Les Transformers combinent une modélisation séquentielle, une attention contextuelle et une sortie probabiliste sur le vocabulaire.

Résumé et Transition

- Le langage naturel est modélisé comme un **processus stochastique séquentiel**, où chaque mot dépend (au moins partiellement) des précédents.
- L'**hypothèse de Markov** (modèles n-gram) permet une estimation simplifiée des probabilités conditionnelles :

$$P(w_t \mid w_{<t}) \approx P(w_t \mid w_{t-n+1}, \ldots, w_{t-1})$$

- Des **mesures d'évaluation** comme l'entropie (incertitude) et la perplexité (qualité de prédiction) quantifient la performance des modèles de langage.
- Les **LLMs** (Large Language Models) étendent cette approche en modélisant $P(w_t \mid w_{<t})$ à l'aide de **réseaux neuronaux profonds**, capables d'intégrer un large contexte.

Prochain module : **Mécanisme d'Attention** et **Architecture Transformer**.

Pourquoi les Fonctions d'Activation ?

- Un réseau de neurones est une composition de transformations :

$$h^{(1)} = W^{(1)}x + b^{(1)}, \quad h^{(2)} = W^{(2)}h^{(1)} + b^{(2)}, \quad \dots$$

- Si l'on n'utilise **que des transformations linéaires**, alors toute composition reste linéaire :

$$h^{(n)} = W^{(n)} \cdots W^{(1)}x + \text{biais} \Rightarrow \text{Pas de gain en expressivité}$$

- Pour apprendre des fonctions non-linéaires complexes, on insère des **fonctions d'activation non linéaires** f entre chaque couche :

$$h^{(l)} = f(W^{(l)}h^{(l-1)} + b^{(l)}), \quad l = 1, \dots, L$$

- Cela permet au réseau d'approximer des fonctions non linéaires arbitraires (Théorème d'universalité).

Les fonctions d'activation sont essentielles pour la puissance d'expression des réseaux neuronaux.

Pourquoi les Fonctions d'Activation ?

- Un réseau de neurones est une composition de transformations :

$$h^{(1)} = W^{(1)}x + b^{(1)}, \quad h^{(2)} = W^{(2)}h^{(1)} + b^{(2)}, \quad \dots$$

- Si l'on n'utilise **que des transformations linéaires**, alors toute composition reste linéaire :

$$h^{(n)} = W^{(n)} \cdots W^{(1)}x + \text{biais} \Rightarrow \text{Pas de gain en expressivité}$$

- Pour apprendre des fonctions non-linéaires complexes, on insère des **fonctions d'activation non linéaires** f entre chaque couche :

$$h^{(l)} = f(W^{(l)}h^{(l-1)} + b^{(l)}), \quad l = 1, \dots, L$$

- Cela permet au réseau d'approximer des fonctions non linéaires arbitraires (Théorème d'universalité).

Les fonctions d'activation sont essentielles pour la puissance d'expression des réseaux neuronaux.

Exemples de Fonctions d'Activation (avec Propriétés)

- **Sigmoïde logistique** :

$$f(x) = \frac{1}{1+e^{-x}} \quad \text{(valeurs dans } (0,1)\text{)}$$

  - Dérivable, utilisée pour modéliser une probabilité
  - **Problème** : saturation pour $x \ll 0$ ou $x \gg 0$ (vanishing gradient)

- **Tangente hyperbolique (tanh)** :

$$f(x) = \tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \quad \text{(valeurs dans } (-1,1)\text{)}$$

  - Centrée en zéro (meilleure symétrie que sigmoïde)
  - Même problème de saturation

- **ReLU (Rectified Linear Unit)** :

$$f(x) = \max(0,x)$$

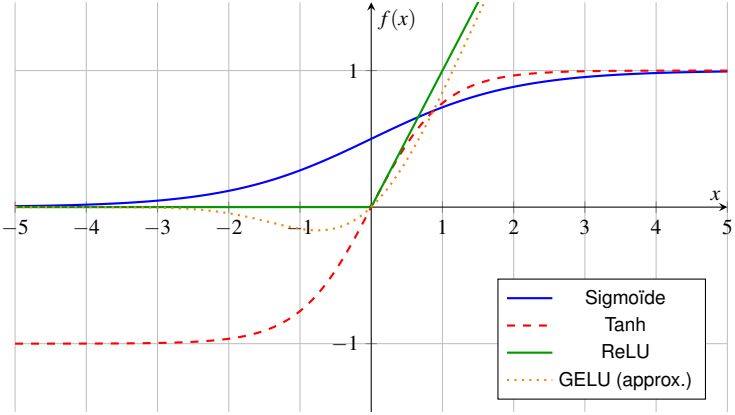
  - Non linéaire, mais simple à dériver (dérivée : 1 si $x > 0$, sinon 0)
  - Rapide, favorise la convergence
  - **Problème** : neurones morts si $x < 0$ fréquemment

- **GELU (Gaussian Error Linear Unit)** :

$$f(x) = x \cdot \Phi(x), \quad \Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{t^2}{2}}\, dt$$

  - Approximation douce de ReLU : pondère x par sa probabilité d'être positif
  - Utilisée dans BERT, GPT (meilleure expressivité empirique)

Visualisation des Fonctions d'Activation



L'approximation de GELU utilisée ici est celle de HENDRYCKS et GIMPEL (2016) :

$$\mathrm{GELU}(x) \approx 0.5x\left(1+\tanh\left(\sqrt{\frac{2}{\pi}}\left(x+0.044715x^3\right)\right)\right)$$

Chaque fonction d'activation introduit une non-linéarité contrôlée pour enrichir la capacité d'expression du réseau.

Fonction de Perte (Loss Function)

- En apprentissage supervisé, une **fonction de perte** $\mathcal{L}(y, \hat{y})$ quantifie l'erreur entre la sortie prédite $\hat{y}$ et la vérité de terrain y.
- Dans le contexte du traitement du langage naturel (NLP), on utilise couramment la **perte d'entropie croisée** :

$$\mathcal{L}(y, \hat{y}) = -\sum_{i=1}^{|V|} y_i \log \hat{y}_i$$

où :

- $y \in \{0,1\}^{|V|}$ est un vecteur one-hot indiquant le mot cible (réel),
- $\hat{y} = \text{softmax}(z)$ est la distribution prédite sur le vocabulaire V, où $z \in \mathbb{R}^{|V|}$ est le vecteur de scores logit produits par le modèle.

- Cette perte correspond à l'opposé du log de la probabilité prédite pour la classe correcte :

$$\mathcal{L}(y, \hat{y}) = -\log \hat{y}_j \quad \text{où } j = \arg\max_i y_i$$

Objectif : Minimiser la perte moyenne sur le corpus :

$$\mathbb{E}_{(x,y)\sim\mathcal{D}}[\mathcal{L}(y, \hat{y}(x))]$$

Fonction de Perte (Loss Function)

- En apprentissage supervisé, une **fonction de perte** $\mathcal{L}(y, \hat{y})$ quantifie l'erreur entre la sortie prédite $\hat{y}$ et la vérité de terrain y.
- Dans le contexte du traitement du langage naturel (NLP), on utilise couramment la **perte d'entropie croisée** :

$$\mathcal{L}(y, \hat{y}) = -\sum_{i=1}^{|V|} y_i \log \hat{y}_i$$

où :

- $y \in \{0,1\}^{|V|}$ est un vecteur one-hot indiquant le mot cible (réel),
- $\hat{y} = \text{softmax}(z)$ est la distribution prédite sur le vocabulaire V, où $z \in \mathbb{R}^{|V|}$ est le vecteur de scores logit produits par le modèle.

- Cette perte correspond à l'opposé du log de la probabilité prédite pour la classe correcte :

$$\mathcal{L}(y, \hat{y}) = -\log \hat{y}_j \quad \text{où } j = \arg\max_i y_i$$

Objectif : Minimiser la perte moyenne sur le corpus :

$$\mathbb{E}_{(x,y)\sim\mathcal{D}}[\mathcal{L}(y, \hat{y}(x))]$$

Descente de Gradient : Principe

- Soit $\mathcal{L}(\theta)$ une fonction de coût (ou fonction objectif) à minimiser, où $\theta \in \mathbb{R}^d$ est le vecteur des paramètres du modèle.
- La **descente de gradient** est une méthode d'optimisation itérative basée sur le calcul du gradient :

$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \nabla_\theta \mathcal{L}(\theta^{(t)})$$

où :

- $\theta^{(t)}$ est la valeur des paramètres à l'itération t,
- $\eta > 0$ est le **taux d'apprentissage** (learning rate),
- $\nabla_\theta \mathcal{L}(\theta)$ est le **gradient** de la fonction de perte par rapport à θ :

$$\nabla_\theta \mathcal{L}(\theta) = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \theta_1} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial \theta_d} \end{bmatrix} \in \mathbb{R}^d$$

- Interprétation géométrique : $\nabla_\theta \mathcal{L}(\theta)$ indique la direction de la plus forte croissance locale de $\mathcal{L}$; on se déplace donc dans la direction opposée pour la réduire.

Chaque mise à jour réduit la perte localement, en espérant converger vers un minimum (local ou global).

Descente de Gradient : Principe

- Soit $\mathcal{L}(\theta)$ une fonction de coût (ou fonction objectif) à minimiser, où $\theta \in \mathbb{R}^d$ est le vecteur des paramètres du modèle.
- La **descente de gradient** est une méthode d'optimisation itérative basée sur le calcul du gradient :

$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \nabla_\theta \mathcal{L}(\theta^{(t)})$$

où :

- $\theta^{(t)}$ est la valeur des paramètres à l'itération t,
- $\eta > 0$ est le **taux d'apprentissage** (learning rate),
- $\nabla_\theta \mathcal{L}(\theta)$ est le **gradient** de la fonction de perte par rapport à θ :

$$\nabla_\theta \mathcal{L}(\theta) = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \theta_1} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial \theta_d} \end{bmatrix} \in \mathbb{R}^d$$

- Interprétation géométrique : $\nabla_\theta \mathcal{L}(\theta)$ indique la direction de la plus forte croissance locale de $\mathcal{L}$; on se déplace donc dans la direction opposée pour la réduire.

Chaque mise à jour réduit la perte localement, en espérant converger vers un minimum (local ou global).

Backpropagation : Calcul du Gradient

- **Objectif** : calculer les dérivées partielles de la fonction de perte $\mathcal{L}$ par rapport aux paramètres θ du réseau.
- Utilise la **règle de la chaîne** pour des fonctions composées :

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial z} \cdot \frac{\partial z}{\partial h} \cdot \frac{\partial h}{\partial \theta}$$

où typiquement $h = f(Wx+b)$ est une activation, et $z = g(h)$ est une couche suivante.

- Exemple : pour une couche linéaire suivie d'une activation f :

$$h = f(Wx+b) \quad \Rightarrow \quad \frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}}{\partial h} \cdot f'(Wx+b) \cdot x^\top$$

- La rétropropagation consiste à :
  1. calculer l'erreur à la sortie du réseau,
  2. propager cette erreur vers l'arrière couche par couche,
  3. accumuler les gradients locaux pour obtenir les gradients globaux.
- Implémentée efficacement via les bibliothèques d'**auto-différentiation** comme `PyTorch` ou `TensorFlow`.

La rétropropagation permet un entraînement efficace des réseaux profonds comme les Transformers.

Backpropagation : Calcul du Gradient

- **Objectif** : calculer les dérivées partielles de la fonction de perte $\mathcal{L}$ par rapport aux paramètres θ du réseau.
- Utilise la **règle de la chaîne** pour des fonctions composées :

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial z} \cdot \frac{\partial z}{\partial h} \cdot \frac{\partial h}{\partial \theta}$$

où typiquement $h = f(Wx+b)$ est une activation, et $z = g(h)$ est une couche suivante.

- Exemple : pour une couche linéaire suivie d'une activation f :

$$h = f(Wx+b) \quad \Rightarrow \quad \frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}}{\partial h} \cdot f'(Wx+b) \cdot x^\top$$

- La rétropropagation consiste à :
  1. calculer l'erreur à la sortie du réseau,
  2. propager cette erreur vers l'arrière couche par couche,
  3. accumuler les gradients locaux pour obtenir les gradients globaux.
- Implémentée efficacement via les bibliothèques d'**auto-différentiation** comme `PyTorch` ou `TensorFlow`.

La rétropropagation permet un entraînement efficace des réseaux profonds comme les Transformers.

Méthodes d'Optimisation Avancées

- **SGD (Stochastic Gradient Descent)** :
  - Met à jour les paramètres à partir du gradient évalué sur un **mini-lot** de données.
  - Formule de mise à jour :
$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta \mathcal{L}_{\text{batch}}(\theta_t)$$
- **Momentum** :
  - Accélère la convergence en cumulant les gradients passés :
$$v_t = \beta v_{t-1} + (1-\beta)\nabla_\theta \mathcal{L}(\theta_t), \quad \theta_{t+1} = \theta_t - \eta v_t$$
  - $\beta \in [0,1]$ est le coefficient de friction (typiquement 0.9).
- **Adam (Adaptive Moment Estimation)** :
  - Combine **Momentum** (1er moment) et normalisation adaptative (2ème moment).
  - Moyennes mobiles :
$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla_\theta \mathcal{L}(\theta_t)$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\left(\nabla_\theta \mathcal{L}(\theta_t)\right)^2$$
  - Biais corrigés :
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$
  - Mise à jour finale :
$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$$
  - $\beta_1 = 0{,}9$, $\beta_2 = 0{,}999$, $\varepsilon \approx 10^{-8}$ sont les valeurs standard.
- **Remarque** : Adam est largement utilisé dans les LLMs (Transformers, BERT, GPT) pour sa stabilité et sa rapidité de convergence.

Pourquoi vectoriser les mots ?

- Les modèles d'apprentissage profond opèrent dans des espaces vectoriels : les entrées doivent appartenir à $\mathbb{R}^d$.
- Or, les mots sont des objets discrets, non numériques : `"chien"`, `"maison"`, etc.
- On introduit une application de plongement (embedding) :

$$\phi : \mathcal{V} \to \mathbb{R}^d$$

où $\mathcal{V}$ est le vocabulaire (ensemble fini ou large de symboles linguistiques), et $\phi(w)$ est une représentation dense du mot w dans $\mathbb{R}^d$.

- **Propriétés attendues de ϕ** :
  - *Sémantique* : des mots similaires doivent être proches en norme euclidienne ou cosinus :

$$\text{sim}(w_i, w_j) := \cos(\phi(w_i), \phi(w_j)) \approx 1.$$

  - *Syntaxe* : les relations grammaticales doivent également émerger via les opérations linéaires (ex : genre, pluriel, temps).
  - *Capacité de généralisation* : ϕ doit permettre à un réseau de généraliser à des contextes linguistiques nouveaux.

Le but de la vectorisation est d'encoder structurellement les propriétés linguistiques dans un espace métrique exploitable par les modèles neuronaux.

Pourquoi vectoriser les mots ?

- Les modèles d'apprentissage profond opèrent dans des espaces vectoriels : les entrées doivent appartenir à $\mathbb{R}^d$.
- Or, les mots sont des objets discrets, non numériques : `"chien"`, `"maison"`, etc.
- On introduit une application de plongement (embedding) :

$$\phi : \mathcal{V} \to \mathbb{R}^d$$

où $\mathcal{V}$ est le vocabulaire (ensemble fini ou large de symboles linguistiques), et $\phi(w)$ est une représentation dense du mot w dans $\mathbb{R}^d$.

- **Propriétés attendues de ϕ** :
  - *Sémantique* : des mots similaires doivent être proches en norme euclidienne ou cosinus :

$$\text{sim}(w_i, w_j) := \cos(\phi(w_i), \phi(w_j)) \approx 1.$$

  - *Syntaxe* : les relations grammaticales doivent également émerger via les opérations linéaires (ex : genre, pluriel, temps).
  - *Capacité de généralisation* : ϕ doit permettre à un réseau de généraliser à des contextes linguistiques nouveaux.

Le but de la vectorisation est d'encoder structurellement les propriétés linguistiques dans un espace métrique exploitable par les modèles neuronaux.

Encodage One-Hot

- Soit un vocabulaire $V = \{w_1, w_2, \ldots, w_{|V|}\}$ contenant $|V|$ mots distincts.
- L'encodage **one-hot** est une fonction d'injection

$$\phi : V \to \{0,1\}^{|V|} \subset \mathbb{R}^{|V|}$$

telle que pour tout $i \in \{1, \ldots, |V|\}$,

$$\phi(w_i) = \mathbf{e}_i = (0, \ldots, 0, \underbrace{1}_{i\text{-ième}}, 0, \ldots, 0)$$

où $\mathbf{e}_i$ est le i-ème vecteur de la base canonique de $\mathbb{R}^{|V|}$.

- **Conséquences mathématiques** :
  - $\forall i \neq j$, $\langle \phi(w_i), \phi(w_j) \rangle = 0$: tous les vecteurs sont orthogonaux.
  - $\|\phi(w_i)\|_2 = 1$: chaque vecteur a norme unitaire.
- **Limitations majeures** :
  - **Sparsité extrême** : chaque vecteur a une seule coordonnée non nulle ⇒ inefficacité mémoire.
  - **Absence de sémantique** : "`chien`" et "`chat`" sont aussi éloignés que "`chien`" et "`banane`".
  - **Pas de généralisation** : le modèle ne peut rien inférer sur un mot qu'il n'a pas vu pendant l'apprentissage.

L'encodage one-hot est une représentation symbolique sans structure géométrique exploitable par les modèles neuronaux.

Encodage One-Hot

- Soit un vocabulaire $V = \{w_1, w_2, \ldots, w_{|V|}\}$ contenant $|V|$ mots distincts.
- L'encodage **one-hot** est une fonction d'injection

$$\phi : V \to \{0,1\}^{|V|} \subset \mathbb{R}^{|V|}$$

telle que pour tout $i \in \{1, \ldots, |V|\}$,

$$\phi(w_i) = \mathbf{e}_i = (0, \ldots, 0, \underbrace{1}_{i\text{-ième}}, 0, \ldots, 0)$$

où $\mathbf{e}_i$ est le i-ème vecteur de la base canonique de $\mathbb{R}^{|V|}$.

- **Conséquences mathématiques** :
  - $\forall i \neq j$, $\langle \phi(w_i), \phi(w_j) \rangle = 0$: tous les vecteurs sont orthogonaux.
  - $\|\phi(w_i)\|_2 = 1$: chaque vecteur a norme unitaire.
- **Limitations majeures** :
  - **Sparsité extrême** : chaque vecteur a une seule coordonnée non nulle $\Rightarrow$ inefficacité mémoire.
  - **Absence de sémantique** : `"chien"` et `"chat"` sont aussi éloignés que `"chien"` et `"banane"`.
  - **Pas de généralisation** : le modèle ne peut rien inférer sur un mot qu'il n'a pas vu pendant l'apprentissage.

L'encodage one-hot est une représentation symbolique sans structure géométrique exploitable par les modèles neuronaux.

Vers des Représentations Denses

Objectif : Apprendre une représentation vectorielle dense de chaque mot dans un vocabulaire V, c'est-à-dire une application

$$\phi : V \to \mathbb{R}^d,$$

où la dimension d est bien plus petite que la taille du vocabulaire, i.e. $d \ll |V|$.

Hypothèse distributionnelle (Principe de Harris, 1954) :

"Un mot est défini par le contexte dans lequel il apparaît."

Formellement, cela signifie que la représentation d'un mot doit capturer la distribution de ses contextes d'apparition. Autrement dit, deux mots apparaissant dans des contextes similaires doivent avoir des représentations proches dans $\mathbb{R}^d$.

Contexte et objectif d'apprentissage : Pour un mot cible $w \in V$, on note $C(w)$ l'ensemble de ses contextes observés dans un corpus. Chaque contexte est typiquement constitué des mots voisins dans une fenêtre glissante. Le but est d'apprendre $\phi(w)$ tel que la probabilité conditionnelle

$$P(c \mid w) \quad \text{ou} \quad P(w \mid c)$$

soit bien modélisée par la similarité dans l'espace vectoriel.

Vers des représentations denses

Principales approches :

1. **Word2Vec** (Mikolov et al., 2013) :
   - *CBOW* (Continuous Bag of Words) : modélise $P(w \mid \text{contexte})$ en prédisant le mot cible à partir de son contexte.
   - *Skip-Gram* : modélise $P(\text{contexte} \mid w)$ en prédisant les mots de contexte à partir du mot cible.

   Ces modèles optimisent une fonction de log-vraisemblance sur le corpus, souvent avec des approximations comme le *negative sampling* ou le *hierarchical softmax*.

2. **GloVe** (Pennington et al., 2014) : Apprend des vecteurs en factorisant une matrice de co-occurrences globales entre mots et contextes. Le modèle repose sur la relation approximative :

$$\phi(w_i)^\top \psi(w_j) + b_i + \tilde{b}_j \approx \log(X_{ij}),$$

   où X_{ij} est le nombre de co-occurrences du mot w_i avec le contexte w_j, et ϕ, ψ sont des représentations du mot et du contexte respectivement.

Word2Vec Skip-Gram

- **Objectif** : apprendre une fonction d'embedding $\phi : V \to \mathbb{R}^d$ telle que les mots apparaissant dans des contextes similaires aient des représentations proches.
- **Skip-Gram** : pour un mot cible w_t, prédire les mots du contexte $C_t = \{w_{t-c}, \ldots, w_{t+c}\} \setminus \{w_t\}$ dans une fenêtre de taille $2c$.
- On cherche à maximiser la probabilité conjointe :

$$\prod_{-c \leq k \leq c, k \neq 0} P(w_{t+k} \mid w_t)$$

- Cette probabilité est modélisée via une couche softmax :

$$P(w_o \mid w_i) = \frac{\exp(\vec{v}_{w_i}^\top \vec{u}_{w_o})}{\sum_{w \in V} \exp(\vec{v}_{w_i}^\top \vec{u}_w)}$$

où :

- $\vec{v}_{w_i} \in \mathbb{R}^d$: vecteur d'entrée (embedding du mot cible)
- $\vec{u}_{w_o} \in \mathbb{R}^d$: vecteur de sortie (embedding du mot contexte)

- **Problème** : le coût computationnel de la normalisation est $O(|V|)$.
- **Solution pratique** : *Negative Sampling* (Mikolov et al., 2013) :

$$\log \sigma(\vec{v}_{w_i}^\top \vec{u}_{w_o}) + \sum_{j=1}^{K} \mathbb{E}_{w_j \sim P_n} \left[\log \sigma(-\vec{v}_{w_i}^\top \vec{u}_{w_j}) \right]$$

- K : nombre d'exemples négatifs
- P_n : distribution de bruit (souvent $P_n(w) \propto f(w)^{3/4}$)
- $\sigma(x) = \frac{1}{1+e^{-x}}$: fonction sigmoïde

Word2Vec encode les régularités sémantiques dans l'espace vectoriel par apprentissage non supervisé.

Word2Vec Skip-Gram

- **Objectif** : apprendre une fonction d'embedding $\phi : V \to \mathbb{R}^d$ telle que les mots apparaissant dans des contextes similaires aient des représentations proches.
- **Skip-Gram** : pour un mot cible w_t, prédire les mots du contexte $C_t = \{w_{t-c}, \ldots, w_{t+c}\} \setminus \{w_t\}$ dans une fenêtre de taille $2c$.
- On cherche à maximiser la probabilité conjointe :

$$\prod_{-c \leq k \leq c, k \neq 0} P(w_{t+k} \mid w_t)$$

- Cette probabilité est modélisée via une couche softmax :

$$P(w_o \mid w_i) = \frac{\exp(\vec{v}_{w_i}^\top \vec{u}_{w_o})}{\sum_{w \in V} \exp(\vec{v}_{w_i}^\top \vec{u}_w)}$$

où :

  - $\vec{v}_{w_i} \in \mathbb{R}^d$: vecteur d'entrée (embedding du mot cible)
  - $\vec{u}_{wo} \in \mathbb{R}^d$: vecteur de sortie (embedding du mot contexte)

- **Problème** : le coût computationnel de la normalisation est $O(|V|)$.
- **Solution pratique** : *Negative Sampling* (Mikolov et al., 2013) :

$$\log \sigma(\vec{v}_{w_i}^\top \vec{u}_{w_o}) + \sum_{j=1}^{K} \mathbb{E}_{w_j \sim P_n} \left[\log \sigma(-\vec{v}_{w_i}^\top \vec{u}_{w_j}) \right]$$

  - K : nombre d'exemples négatifs
  - P_n : distribution de bruit (souvent $P_n(w) \propto f(w)^{3/4}$)
  - $\sigma(x) = \frac{1}{1+e^{-x}}$: fonction sigmoïde

Word2Vec encode les régularités sémantiques dans l'espace vectoriel par apprentissage non supervisé.

Word2Vec Skip-Gram

- **Objectif** : apprendre une fonction d'embedding $\phi : V \to \mathbb{R}^d$ telle que les mots apparaissant dans des contextes similaires aient des représentations proches.
- **Skip-Gram** : pour un mot cible w_t, prédire les mots du contexte $C_t = \{w_{t-c}, \ldots, w_{t+c}\} \setminus \{w_t\}$ dans une fenêtre de taille $2c$.
- On cherche à maximiser la probabilité conjointe :

$$\prod_{-c \leq k \leq c, k \neq 0} P(w_{t+k} \mid w_t)$$

- Cette probabilité est modélisée via une couche softmax :

$$P(w_o \mid w_i) = \frac{\exp(\vec{v}_{w_i}^\top \vec{u}_{w_o})}{\sum_{w \in V} \exp(\vec{v}_{w_i}^\top \vec{u}_w)}$$

où :

  - $\vec{v}_{w_i} \in \mathbb{R}^d$: vecteur d'entrée (embedding du mot cible)
  - $\vec{u}_{wo} \in \mathbb{R}^d$: vecteur de sortie (embedding du mot contexte)

- **Problème** : le coût computationnel de la normalisation est $O(|V|)$.
- **Solution pratique** : *Negative Sampling* (Mikolov et al., 2013) :

$$\log \sigma(\vec{v}_{w_i}^\top \vec{u}_{w_o}) + \sum_{j=1}^{K} \mathbb{E}_{w_j \sim P_n} \left[\log \sigma(-\vec{v}_{w_i}^\top \vec{u}_{w_j}) \right]$$

  - K : nombre d'exemples négatifs
  - P_n : distribution de bruit (souvent $P_n(w) \propto f(w)^{3/4}$)
  - $\sigma(x) = \frac{1}{1+e^{-x}}$: fonction sigmoïde

Word2Vec encode les régularités sémantiques dans l'espace vectoriel par apprentissage non supervisé.

GloVe Global Vectors

- **Objectif** : apprendre des vecteurs de mots $\vec{v}_i, \vec{v}_j \in \mathbb{R}^d$ tels que leur produit scalaire encode les statistiques globales de co-occurrence des mots dans un corpus.
- On construit une matrice de co-occurrence $X \in \mathbb{R}^{|V| \times |V|}$:

$$X_{ij} = \text{nombre de fois que le mot } w_j \text{ apparaît dans le contexte de } w_i.$$

- Hypothèse centrale : il existe une relation linéaire entre les représentations vectorielles et $\log(X_{ij})$:

$$\vec{v}_i^\top \vec{v}_j + b_i + b_j \approx \log(X_{ij})$$

où b_i, b_j sont des biais scalaires associés aux mots w_i et w_j.

- **Fonction de coût** à minimiser :

$$J = \sum_{i,j=1}^{|V|} f(X_{ij}) \left(\vec{v}_i^\top \vec{v}_j + b_i + b_j - \log X_{ij} \right)^2$$

avec f une fonction de pondération qui réduit l'influence des grandes fréquences :

$$f(x) = \begin{cases} \left(\frac{x}{x_{\max}} \right)^\alpha & \text{si } x < x_{\max} \\ 1 & \text{sinon} \end{cases}$$

Typiquement, $\alpha = 0{,}75$ et $x_{\max} = 100$.

- GloVe capture des régularités sémantiques et syntaxiques à partir des co-occurrences globales, contrairement à Word2Vec basé sur des fenêtres locales.

GloVe tire profit de la structure statistique globale du corpus pour produire des embeddings informatifs.

GloVe Global Vectors

- **Objectif** : apprendre des vecteurs de mots $\vec{v}_i, \vec{v}_j \in \mathbb{R}^d$ tels que leur produit scalaire encode les statistiques globales de co-occurrence des mots dans un corpus.
- On construit une matrice de co-occurrence $X \in \mathbb{R}^{|V| \times |V|}$:

$$X_{ij} = \text{nombre de fois que le mot } w_j \text{ apparaît dans le contexte de } w_i.$$

- Hypothèse centrale : il existe une relation linéaire entre les représentations vectorielles et $\log(X_{ij})$:

$$\vec{v}_i^\top \vec{v}_j + b_i + b_j \approx \log(X_{ij})$$

où b_i, b_j sont des biais scalaires associés aux mots w_i et w_j.

- **Fonction de coût** à minimiser :

$$J = \sum_{i,j=1}^{|V|} f(X_{ij}) \left(\vec{v}_i^\top \vec{v}_j + b_i + b_j - \log X_{ij} \right)^2$$

avec f une fonction de pondération qui réduit l'influence des grandes fréquences :

$$f(x) = \begin{cases} \left(\frac{x}{x_{\max}} \right)^\alpha & \text{si } x < x_{\max} \\ 1 & \text{sinon} \end{cases}$$

Typiquement, $\alpha = 0{,}75$ et $x_{\max} = 100$.

- GloVe capture des régularités sémantiques et syntaxiques à partir des co-occurrences globales, contrairement à Word2Vec basé sur des fenêtres locales.

GloVe tire profit de la structure statistique globale du corpus pour produire des embeddings informatifs.

GloVe Global Vectors

- **Objectif** : apprendre des vecteurs de mots $\vec{v}_i, \vec{v}_j \in \mathbb{R}^d$ tels que leur produit scalaire encode les statistiques globales de co-occurrence des mots dans un corpus.
- On construit une matrice de co-occurrence $X \in \mathbb{R}^{|V| \times |V|}$:

$$X_{ij} = \text{nombre de fois que le mot } w_j \text{ apparaît dans le contexte de } w_i.$$

- Hypothèse centrale : il existe une relation linéaire entre les représentations vectorielles et $\log(X_{ij})$:

$$\vec{v}_i^\top \vec{v}_j + b_i + b_j \approx \log(X_{ij})$$

où b_i, b_j sont des biais scalaires associés aux mots w_i et w_j.

- **Fonction de coût** à minimiser :

$$J = \sum_{i,j=1}^{|V|} f(X_{ij}) \left(\vec{v}_i^\top \vec{v}_j + b_i + b_j - \log X_{ij} \right)^2$$

avec f une fonction de pondération qui réduit l'influence des grandes fréquences :

$$f(x) = \begin{cases} \left(\frac{x}{x_{\max}} \right)^\alpha & \text{si } x < x_{\max} \\ 1 & \text{sinon} \end{cases}$$

Typiquement, $\alpha = 0{,}75$ et $x_{\max} = 100$.

- GloVe capture des régularités sémantiques et syntaxiques à partir des co-occurrences globales, contrairement à Word2Vec basé sur des fenêtres locales.

GloVe tire profit de la structure statistique globale du corpus pour produire des embeddings informatifs.

Exemples de sémantique vectorielle

$$\vec{v}_{\text{roi}} - \vec{v}_{\text{homme}} + \vec{v}_{\text{femme}} \approx \vec{v}_{\text{reine}}$$

- Les relations sémantiques (genre, pluriel, métier) deviennent des **directions vectorielles**.
- Ces propriétés sont émergentes dans l'espace d'embedding.

Embeddings dans les Transformers

- Dans les modèles de type Transformer (Vaswani et al., 2017), les vecteurs d'entrée sont constitués de la combinaison de deux types d'embeddings :
  1. **Token embeddings** : chaque unité linguistique (mot, sous-mot ou caractère) est associée à un vecteur appris $\vec{v}_{\text{token}_i} \in \mathbb{R}^d$. Ces unités proviennent souvent d'un découpage par :
     - BPE (Byte Pair Encoding),
     - WordPiece,
     - SentencePiece.
  2. **Positional embeddings** : ajoutent l'information de la position i dans la séquence. Deux types :
     - **Sinusoïdaux** (non appris) : définis par
     $$\text{PE}_{(i,2k)} = \sin\left(\frac{i}{10000^{2k/d}}\right), \quad \text{PE}_{(i,2k+1)} = \cos\left(\frac{i}{10000^{2k/d}}\right)$$
     où d est la dimension d'embedding.
     - **Appris** : vecteurs $\vec{v}_{\text{position}_i} \in \mathbb{R}^d$ optimisés pendant l'entraînement.
- Le vecteur d'entrée final pour la position i est :
$$\vec{e}_i = \vec{v}_{\text{token}_i} + \vec{v}_{\text{position}_i}$$
- Ce vecteur $\vec{e}_i$ est ensuite passé au premier bloc du Transformer (couche d'attention).

Cette représentation vectorielle riche permet au modèle de tenir compte à la fois du contenu et de la position dans la séquence.

Embeddings dans les Transformers

- Dans les modèles de type Transformer (Vaswani et al., 2017), les vecteurs d'entrée sont constitués de la combinaison de deux types d'embeddings :
  1. **Token embeddings** : chaque unité linguistique (mot, sous-mot ou caractère) est associée à un vecteur appris $\vec{v}_{\text{token}_i} \in \mathbb{R}^d$. Ces unités proviennent souvent d'un découpage par :
     - BPE (Byte Pair Encoding),
     - WordPiece,
     - SentencePiece.
  2. **Positional embeddings** : ajoutent l'information de la position i dans la séquence. Deux types :
     - **Sinusoïdaux** (non appris) : définis par
     $$\text{PE}_{(i,2k)} = \sin\left(\frac{i}{10000^{2k/d}}\right), \quad \text{PE}_{(i,2k+1)} = \cos\left(\frac{i}{10000^{2k/d}}\right)$$
     où d est la dimension d'embedding.
     - **Appris** : vecteurs $\vec{v}_{\text{position}_i} \in \mathbb{R}^d$ optimisés pendant l'entraînement.
- Le vecteur d'entrée final pour la position i est :
$$\vec{e}_i = \vec{v}_{\text{token}_i} + \vec{v}_{\text{position}_i}$$
- Ce vecteur $\vec{e}_i$ est ensuite passé au premier bloc du Transformer (couche d'attention).

Cette représentation vectorielle riche permet au modèle de tenir compte à la fois du contenu et de la position dans la séquence.

Résumé et Transition

- Les mots sont transformés en vecteurs à l'aide d'embeddings.
- Les modèles pré-Transformer (Word2Vec, GloVe) apprennent des représentations statiques.
- Les LLMs utilisent des embeddings dynamiques apprises pendant l'entraînement.
- Position et contexte sont intégrés dans les représentations.

Prochaine étape : comprendre comment ces représentations sont traitées par l'auto-attention.

Mesure de Similarité entre Représentations

- En traitement automatique des langues (NLP), chaque mot (ou phrase) est représenté par un vecteur dense $\vec{v} \in \mathbb{R}^d$.
- Objectif : quantifier la proximité sémantique entre deux représentations vectorielles $\vec{u}, \vec{v} \in \mathbb{R}^d$.
- Deux approches principales :

1. **Similarité cosinus** : mesure l'angle entre deux vecteurs (indépendamment de leur norme) :

$$\text{sim}_{\cos}(\vec{u}, \vec{v}) = \frac{\vec{u}^\top \vec{v}}{\|\vec{u}\| \cdot \|\vec{v}\|} \in [-1, 1]$$

   - Si $\text{sim}_{\cos} \approx 1$: vecteurs très proches directionnellement $\Rightarrow$ forte similarité.
   - Si $\text{sim}_{\cos} \approx 0$: vecteurs orthogonaux $\Rightarrow$ pas de lien.
   - Si $\text{sim}_{\cos} < 0$: directions opposées.

2. **Score d'attention (Transformers)** : défini pour une paire de vecteurs *requête* $\vec{q}$ et *clé* $\vec{k}$:

$$\text{score}(\vec{q}, \vec{k}) = \frac{\vec{q}^\top \vec{k}}{\sqrt{d}} \quad \text{(produit scalaire normalisé)}$$

   - Plus le produit scalaire est élevé, plus l'attention accordée est forte.
   - Le facteur $\sqrt{d}$ évite des valeurs trop grandes pour des grandes dimensions.

La similarité vectorielle est au coeur de la compréhension sémantique dans les LLMs.

Mesure de Similarité entre Représentations

- En traitement automatique des langues (NLP), chaque mot (ou phrase) est représenté par un vecteur dense $\vec{v} \in \mathbb{R}^d$.
- Objectif : quantifier la proximité sémantique entre deux représentations vectorielles $\vec{u}, \vec{v} \in \mathbb{R}^d$.
- Deux approches principales :

1. **Similarité cosinus** : mesure l'angle entre deux vecteurs (indépendamment de leur norme) :

$$\text{sim}_{\cos}(\vec{u}, \vec{v}) = \frac{\vec{u}^\top \vec{v}}{\|\vec{u}\| \cdot \|\vec{v}\|} \in [-1, 1]$$

   - Si $\text{sim}_{\cos} \approx 1$: vecteurs très proches directionnellement $\Rightarrow$ forte similarité.
   - Si $\text{sim}_{\cos} \approx 0$: vecteurs orthogonaux $\Rightarrow$ pas de lien.
   - Si $\text{sim}_{\cos} < 0$: directions opposées.

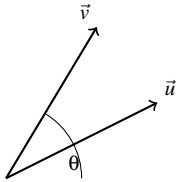
2. **Score d'attention (Transformers)** : défini pour une paire de vecteurs *requête* $\vec{q}$ et *clé* $\vec{k}$:

$$\text{score}(\vec{q}, \vec{k}) = \frac{\vec{q}^\top \vec{k}}{\sqrt{d}} \quad \text{(produit scalaire normalisé)}$$

   - Plus le produit scalaire est élevé, plus l'attention accordée est forte.
   - Le facteur $\sqrt{d}$ évite des valeurs trop grandes pour des grandes dimensions.

La similarité vectorielle est au coeur de la compréhension sémantique dans les LLMs.

Visualisation Géométrique de la Similarité Cosinus



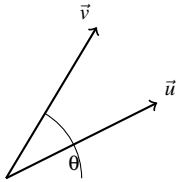
- L'angle θ entre les vecteurs $\vec{u}$ et $\vec{v}$ détermine leur similarité directionnelle.
- La similarité cosinus est définie par :

$$\cos(\theta) = \frac{\vec{u}^\top \vec{v}}{\|\vec{u}\| \cdot \|\vec{v}\|}$$

- $\cos(\theta)$ est maximal (=1) quand $\vec{u}$ et $\vec{v}$ pointent dans la même direction.

L'orientation dans l'espace vectoriel encode des relations sémantiques entre mots.

Visualisation Géométrique de la Similarité Cosinus



- L'angle θ entre les vecteurs $\vec{u}$ et $\vec{v}$ détermine leur similarité directionnelle.
- La similarité cosinus est définie par :

$$\cos(\theta) = \frac{\vec{u}^\top \vec{v}}{\|\vec{u}\| \cdot \|\vec{v}\|}$$

- $\cos(\theta)$ est maximal (=1) quand $\vec{u}$ et $\vec{v}$ pointent dans la même direction.

L'orientation dans l'espace vectoriel encode des relations sémantiques entre mots.

Similarité Cosinus et Attention dans les Transformers

- Chaque token x_i est encodé comme vecteur $\vec{x}_i \in \mathbb{R}^d$.
- Ces vecteurs sont transformés en :
  - Requête $\vec{q}_i = W^Q \vec{x}_i$
  - Clé $\vec{k}_j = W^K \vec{x}_j$
- Le score d'attention entre x_i et x_j est :

$$\text{score}(i,j) = \frac{\vec{q}_i^\top \vec{k}_j}{\sqrt{d}}$$

  - Produit scalaire directionnel : lié à la similarité cosinus.
  - Normalisation par $\sqrt{d}$: contrôle l'amplitude des gradients.
- Application du **softmax** :

$$\alpha_{ij} = \frac{\exp(\text{score}(i,j))}{\sum_{k=1}^{n} \exp(\text{score}(i,k))}$$

- α_{ij} représente la part d'attention que le token x_i accorde au token x_j.

L'auto-attention apprend quelles parties de la séquence sont importantes pour chaque token.

Exemple Numérique Simplifié : Attention

- Soit une requête $Q = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ et deux clés :

$$K_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad K_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

- Calcul des scores d'attention (produits scalaires sans normalisation) :

$$s_1 = Q^\top K_1 = 1 \cdot 1 + 0 \cdot 0 = 1$$

$$s_2 = Q^\top K_2 = 1 \cdot 0 + 0 \cdot 1 = 0$$

- Application de la fonction softmax :

$$\text{softmax}(s_1, s_2) = \left(\frac{e^{s_1}}{e^{s_1} + e^{s_2}}, \frac{e^{s_2}}{e^{s_1} + e^{s_2}} \right)$$

$$= \left(\frac{e^1}{e^1 + e^0}, \frac{e^0}{e^1 + e^0} \right) = \left(\frac{e}{e+1}, \frac{1}{e+1} \right) \approx (0.731, 0.269)$$

- Interprétation : la requête Q est beaucoup plus proche de K_1 que de K_2, donc le poids d'attention est fortement concentré sur K_1.

Le mécanisme d'attention discrimine les tokens selon leur similarité directionnelle.

Exemple Numérique Simplifié : Attention

- Soit une requête $Q = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ et deux clés :

$$K_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad K_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

- Calcul des scores d'attention (produits scalaires sans normalisation) :

$$s_1 = Q^\top K_1 = 1 \cdot 1 + 0 \cdot 0 = 1$$

$$s_2 = Q^\top K_2 = 1 \cdot 0 + 0 \cdot 1 = 0$$

- Application de la fonction softmax :

$$\text{softmax}(s_1, s_2) = \left(\frac{e^{s_1}}{e^{s_1} + e^{s_2}}, \frac{e^{s_2}}{e^{s_1} + e^{s_2}} \right)$$

$$= \left(\frac{e^1}{e^1 + e^0}, \frac{e^0}{e^1 + e^0} \right) = \left(\frac{e}{e+1}, \frac{1}{e+1} \right) \approx (0.731, 0.269)$$

- Interprétation : la requête Q est beaucoup plus proche de K_1 que de K_2, donc le poids d'attention est fortement concentré sur K_1.

Le mécanisme d'attention discrimine les tokens selon leur similarité directionnelle.

Résumé

- La similarité cosinus permet de comparer des mots/vecteurs.
- Elle est la base des mécanismes d'attention.
- L'Attention Score est une mesure pondérée utilisée pour extraire les dépendances dans le texte.

À suivre : Module 2 Mécanisme complet des Transformers.