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 2 : Transformer : architecture et principes mathématiques

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

Dates : 15-16 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 : comprendre comment fonctionne un modèle Transformer avec ses fondements mathématiques.

- Problème du traitement séquentiel et limites des RNN/LSTM
- Mécanisme d'attention : attention scalaire, multi-tête
- Encodage positionnel (sinusoidal encoding)
- Architecture générale du Transformer (encoder/décodeur, normalisation, résidus)
- Complexité temporelle et spatiale
- Base mathématique du self-attention : matrices de requêtes (Q), clés (K) et valeurs (V)

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

Sommaire

Traitement séquentiel des données

- Les données textuelles sont naturellement séquentielles.
- Objectif : modéliser les dépendances entre les éléments de la séquence.
- Problème clé : comment capturer efficacement les relations entre mots éloignés ?

Modèles traditionnels :

- RNN (Recurrent Neural Networks)
- LSTM (Long Short-Term Memory)

Traitement séquentiel des données

- Les données textuelles sont naturellement séquentielles.
- Objectif : modéliser les dépendances entre les éléments de la séquence.
- Problème clé : comment capturer efficacement les relations entre mots éloignés ?

Modèles traditionnels :

- RNN (Recurrent Neural Networks)
- LSTM (Long Short-Term Memory)

RNN : Réseaux Récurrents

Principe

$$h_t = f(W_{hh}h_{t-1} + W_{xh}x_t + b)$$

- x_t : entrée au temps t ; h_t : état caché.
- Le même bloc de neurones est appliqué à chaque étape.
- Capture des dépendances par "mémoire" des états précédents.

Limites majeures :

- Traitement strictement séquentiel $\Rightarrow$ pas de parallélisme.
- Gradients peuvent exploser ou disparaître.

RNN : Réseaux Récurrents

Principe

$$h_t = f(W_{hh}h_{t-1} + W_{xh}x_t + b)$$

- x_t : entrée au temps t ; h_t : état caché.
- Le même bloc de neurones est appliqué à chaque étape.
- Capture des dépendances par "mémoire" des états précédents.

Limites majeures :

- Traitement strictement séquentiel $\Rightarrow$ pas de parallélisme.
- Gradients peuvent exploser ou disparaître.

LSTM : mémoire à long terme

- Introduit une mémoire cellulaire pour mieux gérer les dépendances longues.
- Structure interne avec portes (input, forget, output).

Formules clés

$$
\begin{aligned}
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \tanh(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

LSTM : mémoire à long terme

- Introduit une mémoire cellulaire pour mieux gérer les dépendances longues.
- Structure interne avec portes (input, forget, output).

Formules clés

$$
\begin{aligned}
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \tanh(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

Limites des RNN / LSTM

- **Problème 1** **Dépendances longues** : difficile de relier deux mots éloignés.
- **Problème 2** **Séquentialité** : calcul pas parallélisable (step-by-step).
- **Problème 3** **Vanishing Gradient** : les gradients se dissipent dans le temps.
- **Problème 4** **Coût temporel élevé** pour de longues séquences.

Conséquence : Difficulté à traiter de longs textes ou à comprendre le contexte global.

Limites des RNN / LSTM

- **Problème 1** **Dépendances longues** : difficile de relier deux mots éloignés.
- **Problème 2** **Séquentialité** : calcul pas parallélisable (step-by-step).
- **Problème 3** **Vanishing Gradient** : les gradients se dissipent dans le temps.
- **Problème 4** **Coût temporel élevé** pour de longues séquences.

Conséquence : Difficulté à traiter de longs textes ou à comprendre le contexte global.

Illustration du problème de dépendance longue

La dépendance entre x_1 et x_n est difficilement apprise.

Conclusion : pourquoi aller au-delà des RNN/LSTM ?

- Les modèles RNN et LSTM ont marqué un progrès important.
- Mais leurs limites structurelles freinent leur efficacité :
  - faible parallélisation
  - dépendances longues mal gérées
- **Transformer** : nouvelle architecture entièrement parallèle et fondée sur le mécanisme d'attention.

Pourquoi le mécanisme d'attention ?

- Limite des RNN : difficulté à se souvenir d'éléments distants.
- Intuition : **à chaque étape, "regarder" tout le contexte**.
- L'attention pondère les éléments de la séquence selon leur importance contextuelle.

Motivation : donner à chaque mot la capacité de se connecter aux autres.

Pourquoi le mécanisme d'attention ?

- Limite des RNN : difficulté à se souvenir d'éléments distants.
- Intuition : **à chaque étape, "regarder" tout le contexte**.
- L'attention pondère les éléments de la séquence selon leur importance contextuelle.

Motivation : donner à chaque mot la capacité de se connecter aux autres.

Clés, requêtes, valeurs (Q, K, V)

- Chaque mot est transformé en :
  - une requête q
  - une clé k
  - une valeur v
- Ces vecteurs sont issus de matrices d'embedding projetées :

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V$$

- But : calculer une représentation contextuelle pondérée de V en fonction de la similarité entre Q et K.

Attention scalaire (Scaled Dot-Product)

Formule mathématique

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

- $QK^\top$: produit scalaire mesurant la similarité.
- $\sqrt{d_k}$: facteur de normalisation pour stabiliser les gradients.
- Softmax : transforme en distribution de poids attentionnels.
- Produit final : combinaison pondérée des vecteurs V.

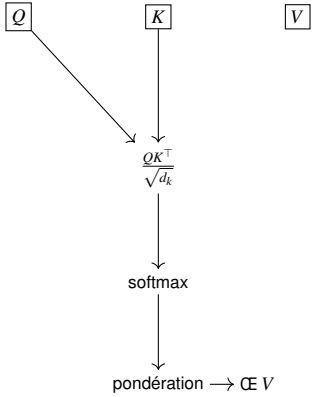
Attention scalaire (Scaled Dot-Product)

Formule mathématique

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

- $QK^\top$: produit scalaire mesurant la similarité.
- $\sqrt{d_k}$: facteur de normalisation pour stabiliser les gradients.
- Softmax : transforme en distribution de poids attentionnels.
- Produit final : combinaison pondérée des vecteurs V.

Illustration : attention scalaire



Attention multi-tête

Idée : plusieurs têtes d'attention apprennent à capter différentes relations.

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,\ldots,\text{head}_h)W^O$$

avec :

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

- Chaque tête projette Q, K, V différemment.
- Permet de capturer plusieurs perspectives sur le contexte.
- Concaténation puis projection finale W^O.

Attention multi-tête

Idée : plusieurs têtes d'attention apprennent à capter différentes relations.

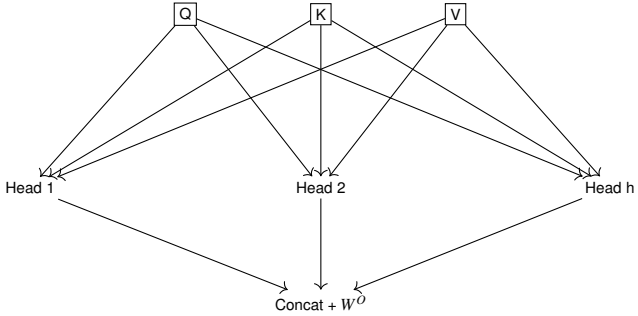
$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,\ldots,\text{head}_h)W^O$$

avec :

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

- Chaque tête projette Q, K, V différemment.
- Permet de capturer plusieurs perspectives sur le contexte.
- Concaténation puis projection finale W^O.

Illustration : attention multi-tête



Résumé

- **Attention** = mécanisme de pondération par similarité contextuelle.
- **Scaled dot-product** = base mathématique du focus contextuel.
- **Multi-head** = enrichit la modélisation par vues parallèles.
- L'attention est **parallélisable**, **globale**, et **différentiable**.

Pourquoi l'encodage positionnel ?

- Le Transformer n'a pas de structure séquentielle intrinsèque (pas de boucle comme RNN).
- Chaque token est traité en parallèle perte de l'ordre.
- Solution : **injecter une information de position dans les embeddings**.

Deux approches principales :

- Encodage sinusoïdal (fixe, déterministe)
- Encodage appris (learned positional embeddings)

Pourquoi l'encodage positionnel ?

- Le Transformer n'a pas de structure séquentielle intrinsèque (pas de boucle comme RNN).
- Chaque token est traité en parallèle perte de l'ordre.
- Solution : **injecter une information de position dans les embeddings**.

Deux approches principales :

- Encodage sinusoïdal (fixe, déterministe)
- Encodage appris (learned positional embeddings)

Encodage sinusoïdal Formule

Pour une position pos et une dimension i :

$$\mathrm{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

$$\mathrm{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

- d_{model} : taille des embeddings.
- $\mathrm{PE}_{pos} \in \mathbb{R}^{d_{model}}$
- Fréquences différentes par dimension encodage multi-échelle.

Encodage sinusoïdal Formule

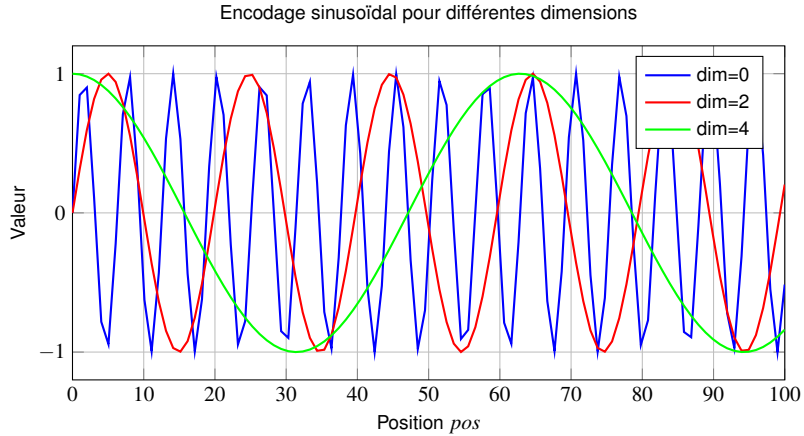
Pour une position pos et une dimension i :

$$\mathrm{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

$$\mathrm{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

- d_{model} : taille des embeddings.
- $\mathrm{PE}_{pos} \in \mathbb{R}^{d_{model}}$
- Fréquences différentes par dimension encodage multi-échelle.

Visualisation : positions encodées par sinus/cosinus



Propriétés de l'encodage sinusoïdal

- **Généralisable** à des séquences plus longues que celles vues en entraînement.
- **Différentiable et périodique** : bien adapté aux modèles neuronaux.
- **Codage relatif** possible :

$$\text{PE}_{pos+k} \text{ peut être exprimé comme fonction linéaire de } \text{PE}_{pos}$$

- **Aucune mémoire apprise** : pas de paramètres pas d'overfitting.

Ajout de l'encodage aux embeddings

- Soit x_{pos} l'embedding d'un mot à la position pos
- Ajout de l'information de position :

$$z_{pos} = x_{pos} + \mathrm{PE}_{pos}$$

- Le modèle reçoit une représentation enrichie par la position.
- Compatible avec les couches d'attention qui restent non positionnées par défaut.

Résumé

- L'encodage sinusoïdal permet d'introduire l'ordre dans un modèle parallèle.
- Il encode l'information de position dans des fonctions continues et périodiques.
- Alternative : encodage appris, mais moins généralisable.

Avantage clef : pas de paramètres, robustesse aux séquences longues.

Structure globale du Transformer

Deux grandes composantes

- **Encodeur (Encoder)** : encode la séquence source.
- **Décodeur (Decoder)** : génère la séquence cible de manière autoregressive.

Empilement modulaire

- Chaque composant est un empilement de N blocs identiques.
- Chaque bloc contient : attention, normalisation, résidu, couche feed-forward.

Blocs de l'encodeur (Encoder)

- Composé de N blocs (typiquement $N = 6$ dans le Transformer original).
- Chaque bloc contient :
  1. **Multi-head self-attention**
  2. **Add & Layer Normalization**
  3. **Feed Forward Neural Network (FFN)**
  4. **Add & Layer Normalization (à nouveau)**

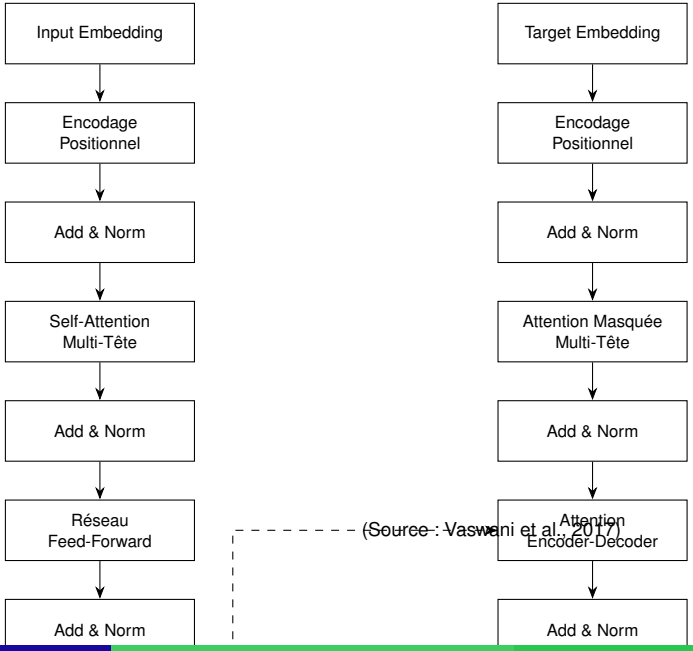
Entrée : embeddings enrichis par encodage positionnel. **Sortie :** représentation contextuelle de chaque token.

Blocs du décodeur (Decoder)

- Chaque bloc du décodeur contient :
  1. **Masked multi-head self-attention** (empêche de voir le futur)
  2. **Add & Layer Norm**
  3. **Multi-head attention sur la sortie de l'encodeur**
  4. **Add & Layer Norm**
  5. **Feed Forward NN + Add & Layer Norm**

Objectif : intégrer le contexte cible + source à chaque étape de décodage.

Architecture complète du Transformer



(Source : Vaswani et al., 2017)

Connexions résiduelles (Residual Connections)

- Chaque sous-couche est encadrée par une connexion résiduelle :

$$\text{Output} = \text{LayerNorm}(x + \text{SubLayer}(x))$$

- Avantages :
  - Favorise la propagation du gradient (évite vanishing).
  - Stabilise l'apprentissage profond.
- Introduites par He et al. (ResNet).

Normalisation de couche (Layer Normalization)

- Appliquée après chaque ajout résiduel.
- Permet de centrer et réduire les activations par dimension :

$$\text{LayerNorm}(x) = \frac{x-\mu}{\sigma+\varepsilon} \cdot \gamma + \beta$$

- μ, σ : moyenne et écart-type sur les dimensions.
- γ, β : paramètres appris.

But : améliorer la stabilité et la vitesse de convergence.

Résumé

- Le Transformer repose sur une architecture modulaire (encodeur/décodeur).
- Chaque bloc contient attention, normalisation, résidu, FFN.
- Connexions résiduelles et normalisation sont essentielles pour l'optimisation.

Clé de son efficacité : traitement parallèle + structure hiérarchique.

Pourquoi étudier la complexité ?

- Les modèles Transformer sont performants mais coûteux.
- Comprendre la complexité permet :
  - d'identifier les limites pratiques ;
  - de motiver les améliorations architecturales ;
  - d'évaluer l'adéquation à des cas réels.

Notations utilisées

- n : longueur de la séquence d'entrée
- d : dimension de l'espace des représentations
- h : nombre de têtes d'attention
- b : taille du batch

On s'intéressera à :

- **Temps de calcul (temporel)** : nombre d'opérations à effectuer
- **Mémoire requise (spatial)** : taille mémoire pour stocker les intermédiaires

Complexité de l'attention standard

Produit de matrices clés-valeurs :

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- $Q, K, V \in \mathbb{R}^{n \times d}$
- Produit QK^T coûte $O(n^2 d)$
- Puis multiplication par V : $O(n^2 d)$

Donc :

$$\text{Complexité temporelle et spatiale} = O(n^2 d)$$

Comparaison avec RNN/LSTM

RNN / LSTM

Complexité temporelle $= O(nd^2)$, séquentielle (pas parallélisable)

Mais mémoire plus faible : $O(nd)$

Transformer

Complexité temporelle $= O(n^2 d)$, hautement parallélisable

Mémoire : $O(n^2)$ pour les matrices d'attention

Conclusion : plus rapide pour traitement parallèle, mais consomme beaucoup de mémoire pour grandes séquences.

Comparaison avec RNN/LSTM

RNN / LSTM

$$\text{Complexité temporelle} = O(nd^2), \quad \text{séquentielle (pas parallélisable)}$$

Mais mémoire plus faible : $O(nd)$

Transformer

$$\text{Complexité temporelle} = O(n^2 d), \quad \text{hautement parallélisable}$$

Mémoire : $O(n^2)$ pour les matrices d'attention

Conclusion : plus rapide pour traitement parallèle, mais consomme beaucoup de mémoire pour grandes séquences.

Comparaison avec RNN/LSTM

RNN / LSTM

$$\text{Complexité temporelle} = O(nd^2), \quad \text{séquentielle (pas parallélisable)}$$

Mais mémoire plus faible : $O(nd)$

Transformer

$$\text{Complexité temporelle} = O(n^2 d), \quad \text{hautement parallélisable}$$

Mémoire : $O(n^2)$ pour les matrices d'attention

Conclusion : plus rapide pour traitement parallèle, mais consomme beaucoup de mémoire pour grandes séquences.

Limitations pratiques

- Les Transformers standards deviennent inefficaces lorsque n est grand (> 2048).
- Les modèles LLM nécessitent souvent du padding et du masking, ce qui augmente encore le coût.
- Nécessité de réduire $O(n^2)$ à $O(n \log n)$ ou $O(n)$.

Problème : le goulot d'étranglement est la self-attention $\Rightarrow$ optimisation cruciale.

Limitations pratiques

- Les Transformers standards deviennent inefficaces lorsque n est grand (> 2048).
- Les modèles LLM nécessitent souvent du padding et du masking, ce qui augmente encore le coût.
- Nécessité de réduire $O(n^2)$ à $O(n \log n)$ ou $O(n)$.

Problème : le goulot d'étranglement est la self-attention $\Rightarrow$ optimisation cruciale.

Approches pour réduire la complexité

- **Linformer** : approximation basse-rang de la matrice d'attention. $\mathcal{O}(n)$
- **Performer** : kernelized attention. $\mathcal{O}(n)$
- **Longformer, BigBird** : attention locale + globales. $\mathcal{O}(n)$
- **Reformer** : hashing + réversibilité. $\mathcal{O}(n \log n)$

Objectif : rendre les Transformers utilisables sur des séquences longues (texte, génomique, vidéo...).

Résumé

- Transformer : complexité quadratique $O(n^2 d)$ en temps et en espace.
- Avantage : parallélisme. Inconvénient : coût mémoire.
- Des variantes modernes réduisent cette complexité tout en gardant la performance.

Intuition du Self-Attention

- Le self-attention permet à chaque élément d'une séquence de **s'auto-contexualiser** :

Quel est le poids de chaque mot dans mon contexte ?

- On calcule une combinaison pondérée de tous les vecteurs de la séquence.
- Ce poids est obtenu par la similarité entre une **requête** q et toutes les **clés** k_i.

Formulation matricielle

Tenseur d'entrée

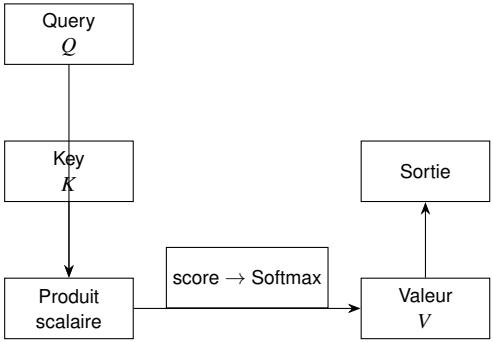
Soit $X \in \mathbb{R}^{n \times d_{\text{model}}}$ la séquence d'entrée de n vecteurs de dimension d_{model}.

Projections linéaires

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V$$

où $W^Q, W^K, W^V \in \mathbb{R}^{d_{\text{model}} \times d_k}$

Schéma d'attention scalaire simple



Score d'attention entre mots

Score de compatibilité

$$\text{score}(q_i, k_j) = \frac{q_i \cdot k_j}{\sqrt{d_k}}$$

- Produit scalaire pour mesurer la similarité.
- Division par $\sqrt{d_k}$ pour stabiliser les gradients (variance).
- Ce score est ensuite normalisé par softmax.

Formule du Self-Attention

Formule vectorielle

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- $Q \in \mathbb{R}^{n\times d_k}$, $K \in \mathbb{R}^{n\times d_k}$, $V \in \mathbb{R}^{n\times d_v}$
- $QK^T \in \mathbb{R}^{n\times n}$: scores entre chaque paire de positions.
- Softmax agit ligne par ligne pour pondérer les valeurs V.

Exemple simple de self-attention

Soit une séquence de 3 vecteurs d'entrée x_1, x_2, x_3.

1. On projette chaque x_i en q_i, k_i, v_i
2. On calcule $q_1 \cdot k_j$ pour $j = 1, 2, 3$
3. On applique softmax : $[\alpha_1, \alpha_2, \alpha_3]$
4. On effectue : $z_1 = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3$

$$z_i = \sum_{j=1}^{n} \text{softmax}_j(q_i \cdot k_j) \cdot v_j$$

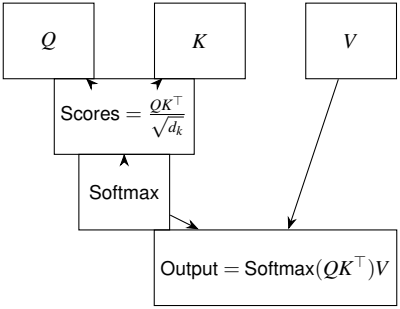
Exemple simple de self-attention

Soit une séquence de 3 vecteurs d'entrée x_1, x_2, x_3.

1. On projette chaque x_i en q_i, k_i, v_i
2. On calcule $q_1 \cdot k_j$ pour $j = 1, 2, 3$
3. On applique softmax : $[\alpha_1, \alpha_2, \alpha_3]$
4. On effectue : $z_1 = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3$

$$z_i = \sum_{j=1}^{n} \text{softmax}_j(q_i \cdot k_j) \cdot v_j$$

Résumé visuel du self-attention



Source : Vaswani et al. (2017)

Remarques importantes

- Les matrices W^Q, W^K, W^V sont **apprises** durant l'entraînement.
- Le mécanisme est **invariant à la position** nécessite un encodage positionnel.
- Chaque sortie z_i contient un résumé contextuel basé sur l'ensemble des tokens.