CHAPITRE ATTENTION · DEEP LEARNING PAR LA PRATIQUE

Intuition sur l'attention et les Transformers

Comprendre pas à pas comment les mots se regardent entre eux

Ce document suit, étape par étape, le mécanisme d'**attention** puis le **bloc Transformer** qui en est fait. Chaque étape est expliquée avec un exemple chiffré simple, vérifié par calcul, autour d'une question concrète : dans une phrase, à quel mot renvoie « il » ? Trois questions y reçoivent une réponse :

- pourquoi aller **au-delà des RNN** (étapes 1 et 2) ;
- comment un mot **choisit** les mots importants parmi tous les autres (étapes 3 à 7) ;
- comment ces ingrédients s'assemblent en un **bloc Transformer** (étapes 8 à 13).

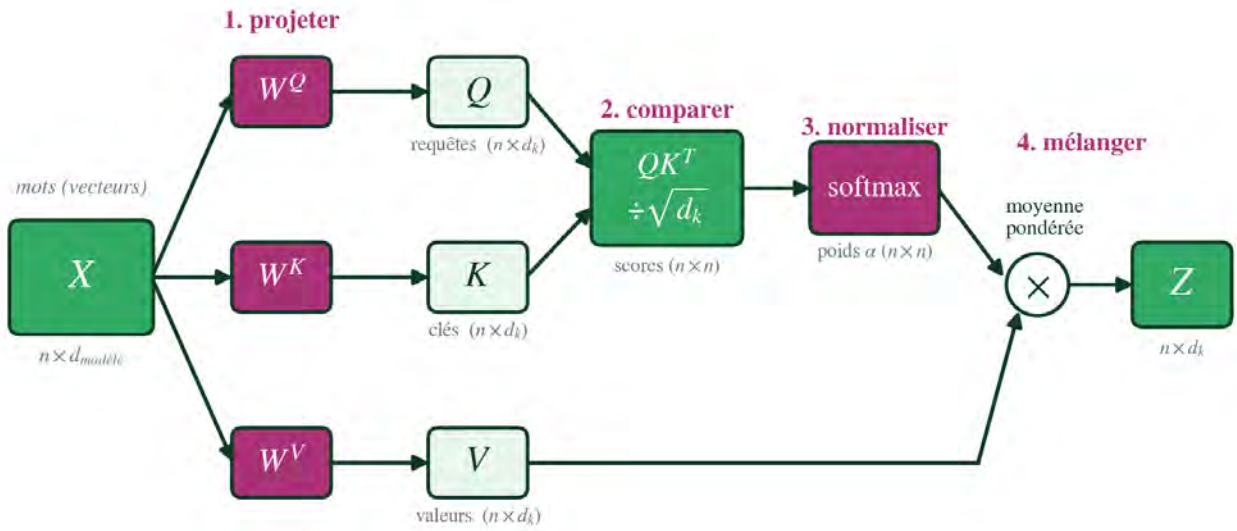

Figure 1 – Le calcul d'une attention : projeter, comparer, normaliser, mélanger

> **Notations** : n est le nombre de mots, d_modèle la taille du vecteur de chaque mot. X est la matrice n × d_modèle des mots. Q, K et V sont les requêtes, les clés et les valeurs ; d_k est la taille d'une requête ou d'une clé.

Étape 0 – Le point de départ : des mots sous forme de vecteurs

Un réseau de neurones calcule avec des nombres. Chaque mot est donc remplacé par un **vecteur** (un « embedding », vu avec Word2Vec) de d_modèle valeurs. Une phrase de n mots devient une **matrice X de n lignes** (une par mot) et d_modèle colonnes. Tout ce qui suit transforme cette matrice en une autre matrice de même forme, dans laquelle **chaque mot a été enrichi par le contexte** des autres.

Étape 1 – Pourquoi aller au-delà des RNN

Les RNN, même avec des portes (LSTM, GRU), butent sur trois limites :

Limite des RNN	Ce que cela implique	Réponse de l'attention
Lecture séquentielle	Pour calculer h_t, il faut avoir calculé h_(t−1) : impossible de traiter tous les mots en parallèle, l'entraînement est lent.	Tous les mots sont lus en même temps.
Mémoire qui s'efface	Même avec un LSTM, l'information d'un mot lointain traverse tous les pas intermédiaires : un chemin de longueur n.	Chaque mot accède directement à n'importe quel autre : un chemin de longueur 1.
Goulot d'étranglement	En traduction (encodeur-décodeur), toute la phrase source est résumée dans un seul vecteur de taille fixe.	On garde un vecteur par mot, pas un seul résumé.

> **L'idée du Transformer** (2017) : abandonner la récurrence et tout confier à un mécanisme, l'**attention**.

Étape 2 – Un problème concret : à quoi renvoie « il » ?

Phrase	Mot à interpréter	Il renvoie à
« L'animal n'a pas traversé la rue parce qu'il était trop fatigué »	il	l'animal
« L'animal n'a pas traversé la rue parce qu'elle était trop large »	elle	la rue

Les deux phrases ne diffèrent que par un mot, et pourtant **la référence change**. Pour interpréter « il » ou « elle », le modèle doit :

- **aller chercher dans la phrase le mot pertinent**, quelle que soit sa distance ;
- savoir **quels mots ignorer** : « traversé » ou « parce que » n'aident pas à résoudre la référence.

> **La question** : comment un mot peut-il « regarder » les autres mots et choisir lesquels comptent ? C'est ce que fait l'attention.

Étape 3 – L'idée clé : une moyenne pondérée dont les poids dépendent du contenu

Pour chaque mot, l'attention fait trois choses :

- elle calcule un **score de pertinence** avec tous les mots de la phrase (lui-même compris) ;
- elle transforme ces scores en **poids positifs de somme 1** (avec la fonction softmax) ;
- le nouveau vecteur du mot est la **moyenne pondérée** des vecteurs de tous les mots.

> **sortie_i = Σ_j α_ij · v_j** avec **α_ij = softmax_j(score(i, j))**
>
> i est le mot qui regarde, j un mot regardé, α_ij l'attention de i sur j, v_j l'information portée par j.

La différence avec une couche dense est essentielle. Dans une couche dense, les poids sont fixés une fois l'entraînement terminé : ils sont les mêmes pour toutes les phrases. Ici, les poids α **sont recalculés pour chaque phrase**, à partir de son contenu. C'est ce qui permet à « il » de regarder « animal » dans une phrase et à « elle » de regarder « rue » dans l'autre.

Étape 4 – Requête, clé, valeur

Pour calculer les scores, chaque mot joue **trois rôles** distincts. L'analogie est celle d'une recherche dans une bibliothèque :

Rôle	Question	Analogie	Dans « …parce qu'il était fatigué »
Requête Q	Que cherche ce mot ?	ma recherche	« il » cherche un nom animé
Clé K	Que présente ce mot comme repère ?	l'étiquette de chaque livre	« animal » : nom, animé ; « rue » : nom, lieu
Valeur V	Quelle information ce mot apporte-t-il ?	le contenu du livre	le sens de « animal »

- Le **score** compare la requête d'un mot à la clé de chaque autre mot, par un produit scalaire : plus ils se ressemblent, plus le score est élevé.
- Le résultat est un mélange des **valeurs**, dosé par les poids : on récupère le contenu des mots dont l'étiquette correspond à la recherche.

Pourquoi trois vecteurs et pas un seul ? Parce que ce qu'un mot cherche, la façon dont il se présente et ce qu'il apporte sont trois choses différentes. Un pronom cherche un nom, mais il ne se présente pas comme un nom : il doit pouvoir poser une question sans qu'on la confonde avec sa propre étiquette.

Étape 5 – Le calcul pas à pas, sur un exemple chiffré

5.1 Mise en place

Trois mots, chacun représenté par un vecteur de dimension 2, **choisi à la main** pour l'illustration (dans un vrai modèle, ces vecteurs sont appris). Pour simplifier, on prend **Q = K = V = X** : il n'y a pas encore de projection apprise (étape 8).

Mot	Vecteur	Intuition
animal	(2,0 ; 0,4)	plutôt « animé »
rue	(0,2 ; 2,0)	plutôt « lieu »
il	(1,8 ; 0,6)	proche de « animé »

5.2 Étape 1 : les scores

Le score de i sur j est le produit scalaire du vecteur de i par celui de j. Par exemple, pour « il » et « animal » : 1,8 × 2,0 + 0,6 × 0,4 = 3,60 + 0,24 = **3,84**. Pour toute la phrase, c'est la matrice $X X^T$:

Qui regarde ↓ Regardé →	animal	rue	il
animal	4,16	1,20	3,84
rue	1,20	4,04	1,56
il	3,84	1,56	3,60

5.3 Étape 2 : la mise à l'échelle

On divise tous les scores par $\sqrt{d_k} = \sqrt{2} \approx 1{,}414$ (la raison est expliquée à l'étape 7). Pour la ligne de « il » : (3,84 ; 1,56 ; 3,60) devient **(2,715 ; 1,103 ; 2,546)**.

5.4 Étape 3 : la softmax

La softmax transforme chaque ligne en poids positifs de somme 1, en mettant en avant les scores les plus grands. Pour la ligne de « il » : e^2,715 = 15,1 ; e^1,103 = 3,01 ; e^2,546 = 12,8, de somme 30,9. Les poids sont donc 15,1 / 30,9 = 0,489, 3,01 / 30,9 = 0,098 et 12,8 / 30,9 = 0,413. Pour toute la phrase :

Qui regarde ↓ Regardé →	animal	rue	il
animal	0,521	0,064	0,415
rue	0,103	0,765	0,132
il	0,489	0,098	0,413

5.5 Étape 4 : le mélange des valeurs

Le nouveau vecteur de « il » est la moyenne pondérée des valeurs :

> 0,489 × (2,0 ; 0,4) + 0,098 × (0,2 ; 2,0) + 0,413 × (1,8 ; 0,6) = **(1,742 ; 0,639)**

Mot	Vecteur avant	Vecteur après attention
animal	(2,0 ; 0,4)	(1,801 ; 0,586)
rue	(0,2 ; 2,0)	(0,597 ; 1,650)
il	(1,8 ; 0,6)	(1,742 ; 0,639)

Chaque mot est devenu un **mélange** des mots de la phrase, dans des proportions qui dépendent de leur ressemblance. « il » contient maintenant 49 % de « animal », 41 % de lui-même et seulement 10 % de « rue ».

Étape 6 – Lire la carte d'attention

La matrice des poids se représente comme une carte : chaque **ligne** est la répartition du regard d'un mot (la somme vaut 1), chaque **colonne** un mot regardé. Voici la carte de la phrase avec « il », et celle de la phrase avec « elle », où le vecteur de « elle » est (0,5 ; 1,8), plutôt « lieu » :

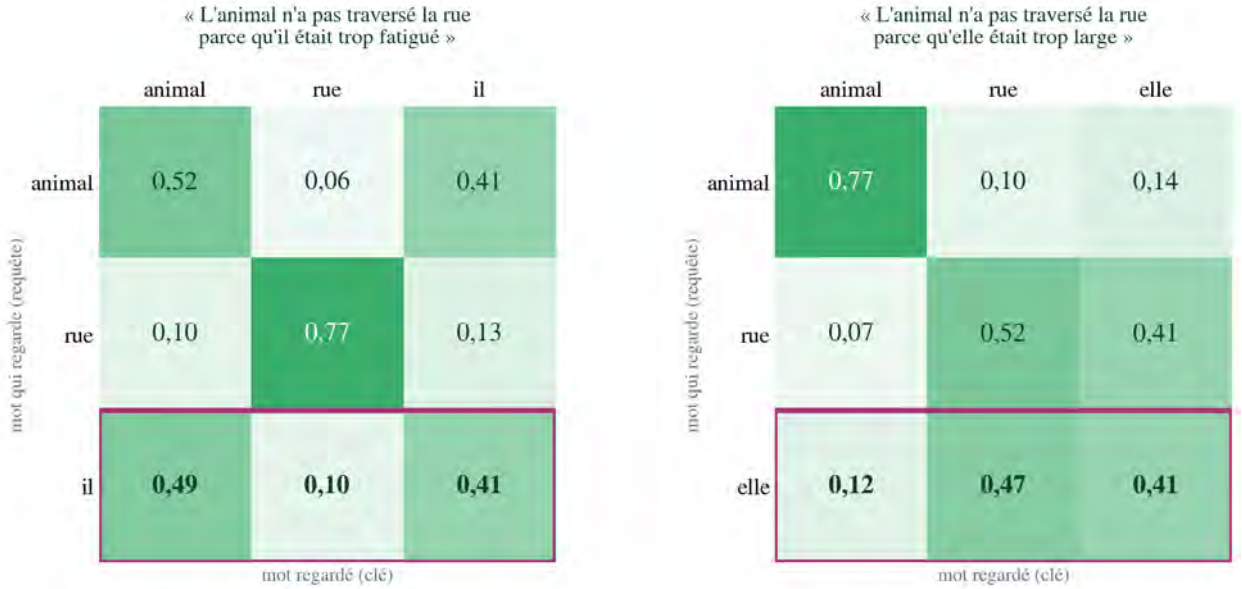

Figure 2 – Cartes d'attention : « il » regarde « animal », « elle » regarde « rue »

- Dans la première phrase, « il » regarde surtout « animal » (0,49). Dans la seconde, « elle » regarde surtout « rue » (0,47) : **le même mécanisme choisit des mots différents** selon le contenu.
- Un mot se regarde **lui-même** (la diagonale) : un mot est toujours très proche de lui-même, il garde donc une part de son propre sens.
- Les vecteurs sont ici choisis à la main : dans un modèle entraîné, les poids ne seraient pas aussi nets, mais ils sont appris pour aider la tâche.

Étape 7 – Pourquoi diviser par √d_k

Le produit scalaire de deux vecteurs aléatoires de dimension d a un écart-type d'environ √**d** : plus la dimension est grande, plus les scores sont grands. Or avec de grands scores, la softmax devient presque **tout ou rien** : un seul mot reçoit presque tout le poids, et les gradients deviennent minuscules. Diviser par √d_k ramène les scores à une échelle comparable, quelle que soit la dimension. Mesure sur des vecteurs aléatoires (loi normale), poids maximal d'une softmax sur 10 clés :

Dimension d_k	Écart-type des scores	Poids maximal moyen, sans échelle	Poids maximal moyen, avec échelle
4	2,0	0,50	0,31
16	4,0	0,72	0,31
64	8,0	0,86	0,32
256	16,1	0,93	0,32

Sans échelle, pour d_k = 256, le mot le plus regardé reçoit en moyenne 93 % du poids : l'attention devient un simple choix du meilleur mot. Avec l'échelle, elle reste nuancée (32 %), quelle que soit la dimension.

Étape 8 – Les projections apprises

Dans l'exemple de l'étape 5, nous avions pris Q = K = V = X. Dans un vrai modèle, trois **matrices apprises** transforment chaque vecteur : **Q = X W^Q**, **K = X W^K**, **V = X W^V** (voir la figure 1). Elles sont indispensables, pour deux raisons :

- **Les trois rôles sont différents** (étape 4) : un même mot doit pouvoir chercher une chose, se présenter d'une autre façon et apporter une troisième information.
- **Sans projection, un mot se ressemble toujours le plus à lui-même** : le score d'un mot avec lui-même est $|x|^2$, toujours élevé. Les projections permettent à « il » de poser une requête qui ressemble à la clé de « animal » sans ressembler à sa propre clé.

Pour d_modèle = 8 et d_k = 8, chaque matrice contient 8 × 8 = 64 poids, soit 192 pour les trois. Ces poids sont appris par rétropropagation, comme tous les autres.

Étape 9 – L'attention multi-têtes

Une seule attention ne mesure qu'**un type de relation** à la fois. Or un mot entretient plusieurs relations : une référence (« il » avec « animal »), un accord (sujet et verbe), un voisinage. On lance donc **h attentions en parallèle**, les **têtes**, chacune avec ses propres W^Q, W^K, W^V, sur des vecteurs plus courts : d_k = d_modèle / h.

- Chaque tête produit une matrice n × d_k.
- On **concatène** les h sorties pour retrouver une matrice n × d_modèle.
- On applique une dernière matrice **W^O** (d_modèle × d_modèle), qui mélange les têtes.

Exemple : d_modèle = 8 et h = 2 têtes (d_k = 4)	Calcul	Poids
W^Q, W^K, W^V (8 × 8 chacune, toutes têtes réunies)	3 × 64	192
W^O (8 × 8)	64	64
Total	$4 \cdot d^2$	256

Le nombre de paramètres **ne dépend pas du nombre de têtes** : chaque tête travaille sur d_modèle / h dimensions. Dans les modèles réels, certaines têtes se spécialisent (référence, accord, position), mais ce n'est pas garanti : c'est une tendance observée, pas une règle.

Étape 10 – L'ordre des mots : l'encodage positionnel

L'attention compare des mots sans tenir compte de leur place : mélanger les mots **ne change pas les poids** (à la permutation près). « le chien mord l'homme » et « l'homme mord le chien » seraient indiscernables. La solution consiste à **ajouter à chaque embedding un vecteur de position**, différent pour chaque place :

> **PE(pos, 2i) = sin(pos / 10000^(2i/d))** et **PE(pos, 2i+1) = cos(pos / 10000^(2i/d))**

Exemple avec d = 4 (deux fréquences) :

Position	dim 0 (sin, rapide)	dim 1 (cos, rapide)	dim 2 (sin, lente)	dim 3 (cos, lente)
0	0,000	1,000	0,000	1,000
1	0,841	0,540	0,010	1,000
2	0,909	−0,416	0,020	1,000
3	0,141	−0,990	0,030	1,000

Chaque position a une **signature unique** : les premières dimensions varient vite, les suivantes lentement, comme les aiguilles d'une horloge. Beaucoup de modèles récents apprennent plutôt les positions, ou utilisent des variantes (positions rotatives).

Étape 11 – Le bloc Transformer

11.1 L'assemblage

Un bloc enchaîne deux sous-couches, chacune entourée d'une **connexion résiduelle** et d'une **normalisation** :

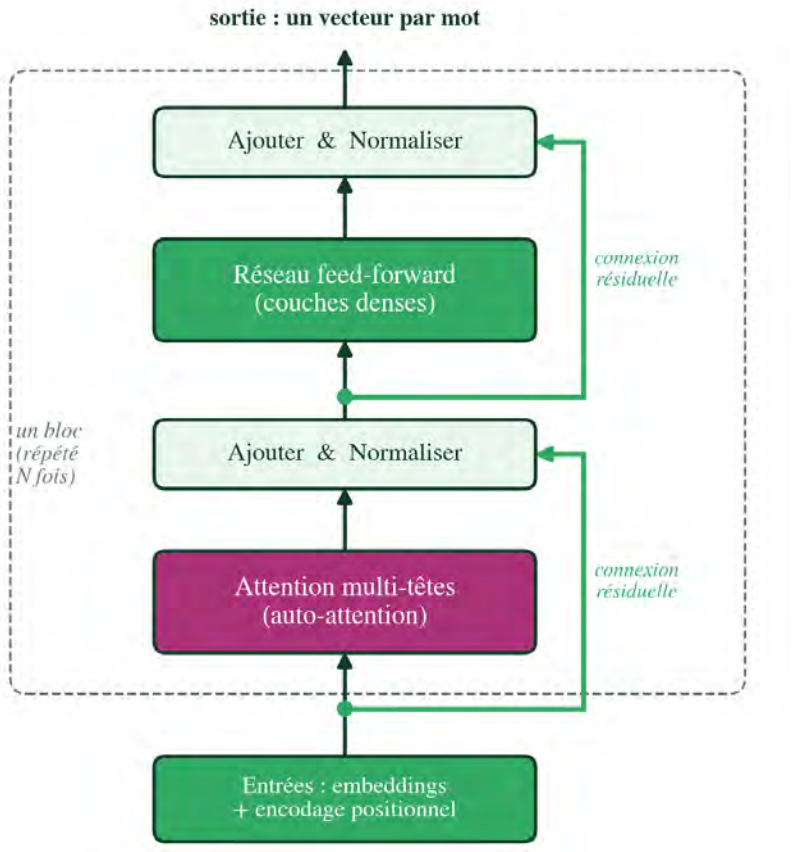

Figure 3 – Un bloc Transformer (encodeur), répété N fois

Partie	Rôle
Attention multi-têtes	les mots échangent de l'information entre eux
Ajouter & Normaliser	recopie l'entrée (connexion résiduelle) et stabilise les valeurs (normalisation)
Feed-forward	transforme chaque mot séparément, avec le même petit réseau pour tous les mots
Répété N fois	les blocs s'empilent : 6 dans le Transformer d'origine, 12 dans BERT-base

11.2 La connexion résiduelle

Au lieu de remplacer l'entrée x par F(x), le bloc calcule **x + F(x)** : il apprend une **correction** à l'entrée, qu'il ne réécrit pas. La dérivée de x + F(x) vaut **1 + F′(x)** : le terme 1 laisse passer le gradient sans l'affaiblir. C'est la même idée que la mise à jour additive du LSTM : l'information et le gradient disposent d'une « autoroute », ce qui permet d'empiler des dizaines de blocs.

11.3 La normalisation par couche

Pour chaque mot, on normalise ses d valeurs : (x − moyenne) / écart-type, puis on applique deux paramètres appris (un facteur et un décalage). Exemple avec x = (2 ; 4 ; 6 ; 8) :

Étape	Résultat
x	(2 ; 4 ; 6 ; 8)
moyenne ; écart-type	5 ; 2,236
x normalisé	(−1,342 ; −0,447 ; 0,447 ; 1,342)

Le résultat a une moyenne de 0 et un écart-type de 1, ce qui **stabilise l'entraînement**.

11.4 Le réseau feed-forward et le nombre de paramètres

Le feed-forward est appliqué **à chaque mot séparément** : deux couches denses, d → 4d → d, avec une activation (ReLU ou GELU) entre les deux. L'attention fait communiquer les mots ; le feed-forward **transforme** l'information de chacun. Un bloc compte environ **12 d² paramètres**, dont deux tiers dans le feed-forward :

Partie du bloc	Formule	d = 8	d = 128
Attention (W^Q, W^K, W^V, W^O)	4 d²	256	65 536
Feed-forward (d → 4d → d)	8 d²	512	131 072
Total (sans biais ni normalisation)	12 d²	768	196 608

Vérification avec PyTorch : une couche `nn.TransformerEncoderLayer(d_model=8, nhead=2, dim_feedforward=32)` compte **872 paramètres**, dont 288 pour l'attention, 552 pour le feed-forward et 32 pour les deux normalisations (les biais s'ajoutent aux 768 du tableau). Avec 12 blocs et d = 768 (BERT-base), les blocs pèsent environ **85 millions** de paramètres.

11.5 Empiler les blocs

La sortie d'un bloc (une matrice n × d_modèle, de même forme que l'entrée) devient l'entrée du suivant. À mesure qu'on monte dans les blocs, la représentation de chaque mot s'enrichit du contexte à des échelles de plus en plus larges, comme les couches successives d'un CNN voient des zones de plus en plus grandes.

Étape 12 – RNN contre Transformer : ce qu'on gagne, ce qu'on paie

Critère	RNN (LSTM, GRU)	Transformer
Traitement des mots	séquentiel, un mot après l'autre	en parallèle, tous les mots à la fois
Chemin entre deux mots éloignés	n pas	1 pas (accès direct par l'attention)
Calcul par couche (longueur n, dimension d)	de l'ordre de $n \cdot d^2$	de l'ordre de $n^2 \cdot d + n \cdot d^2$
Mémoire selon la longueur n	linéaire	quadratique : une matrice d'attention n × n
Ordre des mots	naturel (lecture dans l'ordre)	à ajouter (encodage positionnel)

> **Le coût quadratique en pratique** : 1 000 mots donnent une matrice d'attention de 1 million de poids par tête et par couche ; 10 000 mots, 100 millions. C'est la limite principale des Transformers sur les très longs textes.

Étape 13 – Le masque causal et les grandes familles de modèles

Pour **générer** du texte mot à mot, un mot ne doit pas voir les mots qui le suivent (il ne les connaît pas encore). On applique donc un **masque causal** : les scores des positions futures sont remplacés par $-\infty$, et la softmax leur donne un poids nul.

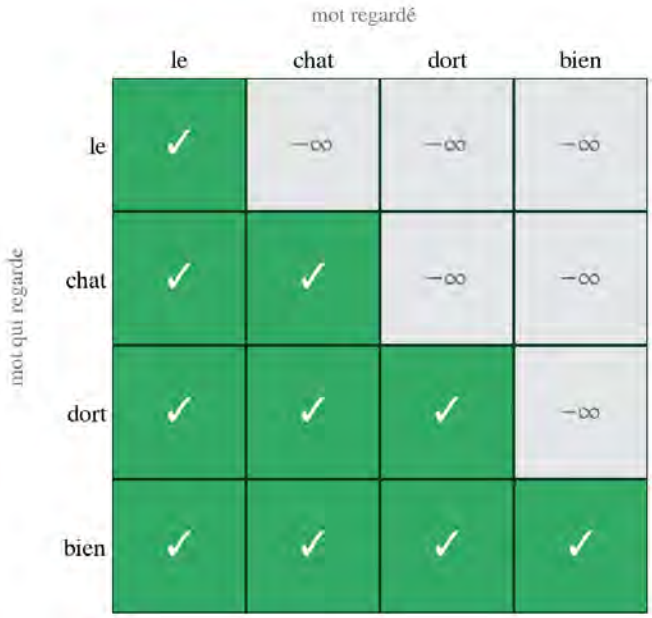

Figure 4 – Masque causal : chaque mot ne regarde que lui-même et les mots précédents

Famille	Attention	Exemples	Usage
Encodeur	bidirectionnelle : chaque mot voit tous les autres	BERT	comprendre, classer
Décodeur	masque causal : chaque mot ne voit que le passé	GPT	générer du texte
Encodeur-décodeur	les deux, plus une attention croisée (requêtes du décodeur, clés et valeurs de l'encodeur)	Transformer d'origine, T5	traduire, résumer

Étape 14 – Pour le voir par soi-même

Ce code (NumPy) reproduit exactement l'exemple de l'étape 5 :

```
import numpy as np

def softmax(s):
    e = np.exp(s - s.max(axis=-1, keepdims=True))
    return e / e.sum(axis=-1, keepdims=True)

X = np.array([[2.0, 0.4], [0.2, 2.0], [1.8, 0.6]])   # animal, rue, il
scores = X @ X.T / np.sqrt(X.shape[1])               # Q K^T / racine(d_k), avec Q =
K = X
poids = softmax(scores)                              # une ligne par mot
sortie = poids @ X                                   # moyenne pondérée des valeurs
print(poids.round(3))
print(sortie.round(3))
```

Il affiche les matrices des étapes 5.4 et 5.5 :

```
[[0.521 0.064 0.415]
 [0.103 0.765 0.132]
 [0.489 0.098 0.413]]
[[1.801 0.586]
 [0.597 1.65 ]
 [1.742 0.639]]
```

Avec PyTorch, les couches toutes faites ajoutent les projections apprises, les têtes multiples, les biais, la normalisation et le feed-forward :

```
import torch, torch.nn as nn

attn = nn.MultiheadAttention(embed_dim=8, num_heads=2, batch_first=True)
x = torch.randn(1, 5, 8)                # (lot, mots, d_modèle)
sortie, poids = attn(x, x, x)           # auto-attention : Q = K = V = x
print(sortie.shape)                     # (1, 5, 8)
print(poids.shape)                      # (1, 5, 5) : moyenne des têtes

bloc = nn.TransformerEncoderLayer(8, 2, dim_feedforward=32, batch_first=True)
print(sum(p.numel() for p in bloc.parameters()))   # 872
```

À retenir

1. Les RNN lisent les mots un par un et oublient : l'**attention** donne à chaque mot un **accès direct** à tous les autres.

2. Chaque mot a une **requête**, une **clé** et une **valeur** : les scores comparent requêtes et clés, la sortie mélange les valeurs.

3. Le calcul est **softmax(Q K^T / √d_k) V**. La division par √d_k évite des poids « tout ou rien ».

4. Les poids sont **recalculés pour chaque phrase** : c'est ce qui permet à « il » de regarder « animal » et à « elle » de regarder « rue ».

5. Plusieurs **têtes** suivent plusieurs relations ; l'**encodage positionnel** rend l'ordre des mots visible.

6. Un bloc Transformer associe **attention et feed-forward**, chacun entouré d'une connexion résiduelle et d'une normalisation, et empilé N fois.

7. On gagne le **parallélisme** et les dépendances à distance ; on paie un **coût quadratique** en longueur de séquence.