ETUDE DE L'ARCHITECTURE DES TRANSFORMERS ET DÉMONSTRATION SUR LA TRADUCTION ANGLAIS-ITALIEN

Introduction :

L'avènement des transformers a marqué une révolution significative dans le domaine du machine learning au cours des dernières années. Ces architectures neuronales, initialement introduites par Vaswani et al. en 2017, ont rapidement conquis le monde de l'intelligence artificielle en offrant des performances exceptionnelles dans une variété de tâches. Leur succès fulgurant s'explique en grande partie par leur capacité à capturer les dépendances à longue distance, permettant ainsi une meilleure compréhension des relations complexes au sein des données. Les transformers ont notamment brillé dans le traitement du langage naturel, la traduction automatique et d'autres domaines, dépassant souvent les modèles préexistants. Leur architecture modulaire, basée sur l'attention, a non seulement permis des avancées significatives en termes de précision, mais a également ouvert la voie à des modèles plus vastes et plus complexes. Dans cette ère de l'apprentissage automatique, les transformers se positionnent comme des piliers incontournables, propulsant le domaine vers de nouveaux sommets.

I.Partie 1 : Etude Théorique

1.Limitation des réseaux de neurons :

Avant l'avènement des transformers, les réseaux de neurones traditionnels, en particulier les réseaux de neurones récurrents (RNNs) et les long short-term memory networks (LSTMs), étaient largement utilisés pour modéliser des séquences de données, telles que des séquences temporelles ou du texte. Cependant, ces approches étaient confrontées à des limitations majeures qui entravent leur capacité à traiter efficacement des tâches complexes.

Les RNNs, en raison de leur structure séquentielle, rencontraient le problème du gradient qui disparaît lors de l'entraînement sur des séquences longues. Cela signifie que lors de la rétropropagation du gradient à travers les différentes étapes temporelles, les gradients peuvent devenir extrêmement petits, rendant l'apprentissage difficile. Cela était particulièrement problématique pour la rétention d'informations à long terme, car les RNNs avaient du mal à conserver des informations utiles sur des périodes prolongées.

Les LSTMs ont été développés pour surmonter ce problème en introduisant des mécanismes de portes, permettant au modèle de contrôler le flux d'informations à travers la séquence. Bien que les LSTMs aient amélioré la capacité des réseaux à gérer les dépendances à long terme, ils présentaient toujours des défis en termes de scalabilité et d'efficacité computationnelle. Ces modèles deviennent rapidement complexes à mesure que les tâches devenaient plus exigeantes, limitant ainsi leur applicabilité à grande échelle.

Ces limitations ont eu un impact significatif sur des domaines clés tels que la traduction automatique et le traitement du langage naturel. La difficulté à modéliser efficacement les relations non linéaires complexes dans les données textuelles a entravé les performances des modèles, tandis que l'architecture séquentielle des réseaux traditionnels a rendu le parallélisme difficile, influant négativement sur les performances, en particulier pour des tâches nécessitant des ressources importantes. L'émergence des transformers a représenté une percée cruciale en surmontant ces obstacles, propulsant ainsi le domaine du machine learning vers de nouvelles possibilités et des performances améliorées.

2.Naissance des Transformers :

Attention is all you need » - c'est ainsi que les chercheurs visionnaires du Google Research, dont Ashish Vaswani et son groupe de talentueux collaborateurs, ont captivé le monde du machine learning dans leur publication révolutionnaire en 2017. Ce papier de recherche emblématique a marqué un moment épique en introduisant une idée audacieuse et innovante : le mécanisme d'attention. Dans un élan d'enthousiasme créatif, les chercheurs ont délaissé les approches traditionnelles, telles que les réseaux de neurones récurrents (RNNs) et les long short-term memory networks (LSTMs), pour forger le chemin vers une nouvelle ère de l'apprentissage automatique. L'attention, au cœur de cette révolution, a permis aux transformers de transcender les limitations précédentes, en adoptant une architecture parallèle qui a bouleversé les paradigmes établis. Ce geste a donné naissance à des modèles capables de traiter des informations à long terme de manière efficace et de réaliser des performances exceptionnelles dans des tâches complexes comme la traduction automatique et le traitement du langage naturel. Ainsi,« Attention is all you need » a marqué le début d'une ère révolutionnaire, propulsant les transformers au sommet du monde du machine learning avec une force inégalée

3.Les applications des Transformers:

Les transformers ont révolutionné un large éventail d'applications dans le domaine du machine learning, démontrant leur polyvalence et leur efficacité. L'une des applications les plus notables se trouve dans *le domaine du traitement du langage naturel (NLP)*. Des modèles transformer tels que BERT (Bidirectional Encoder Representations from Transformers) ont dépassé les modèles précédents en capturant des nuances sémantiques complexes et en comprenant le contexte des mots dans une phrase. Ces avancées ont conduit à des améliorations significatives dans des tâches telles que la compréhension de texte, la génération de texte et la traduction automatique.

Dans le domaine de la vision par ordinateur, les transformers ont également laissé leur empreinte. Des architectures telles que Vision Transformer (ViT) ont démontré une capacité exceptionnelle à traiter des images en les divisant en patchs, puis en appliquant des opérations d'attention globale. Cette approche a surpassé les méthodes traditionnelles en vision par ordinateur et a été utilisée avec succès dans des tâches telles que la classification d'images, la détection d'objets et même la génération d'images.

Les transformers ont également été adoptés *dans le domaine de la recommandation personnalisée*. Des modèles comme le "BERT-based Personalized Embedding" ont été appliqués avec succès pour comprendre les préférences des utilisateurs à partir de données textuelles, améliorant ainsi la précision des recommandations.

4.Architecture des Transformers:

L'architecture des transformers se compose essentiellement d'une structure en encodeur-décodeur.L'encodeur Transforme une séquence d'entrée de représentations symboliques X en une séquence de représentations continues Z.Aprés, le décodeur génère ensuite une séquence de sortie Y de symboles un élément à la fois. À chaque étape, le modèle utilise les symboles générés précédemment comme entrée supplémentaire lors de la génération du suivant.

- **Encodeur :** La première partie de l'architecture, l'encodeur, est responsable de traiter l'entrée et de la transformer en une représentation latente significative. Chaque séquence d'entrée est divisée en tokens et traitée en parallèle par plusieurs blocs d'attention. Ces blocs d'attention, ou couches d'encodeurs, permettent au modèle de considérer simultanément toutes les positions dans la séquence, capturant ainsi les dépendances à long terme de manière efficace.

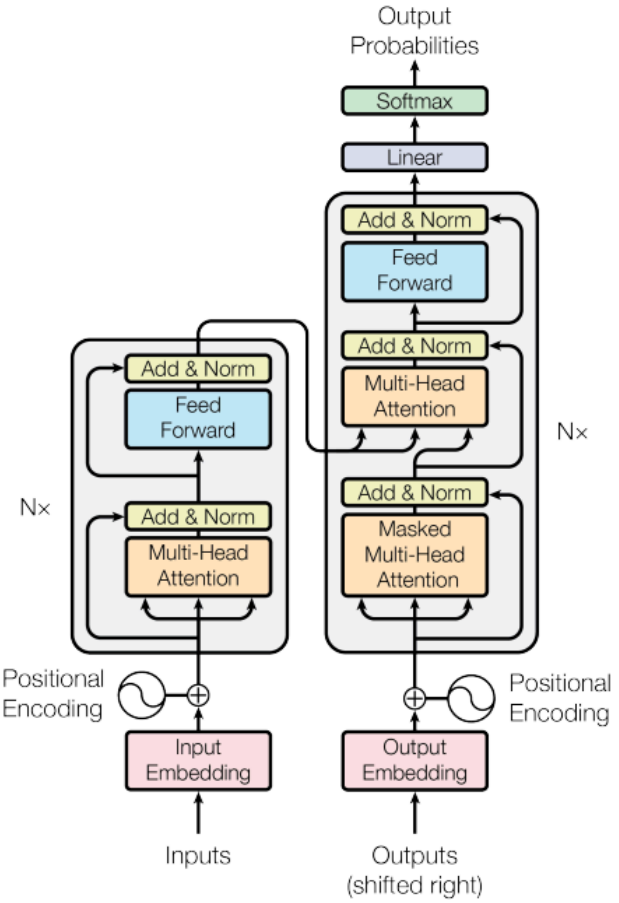


Figure 1: The Transformer - model architecture.

L'encodeur est composé de N couches , chaque couche est divisée en deux sous-couches . la première sous couche représente le mécanisme de Multi-head attention , et la deuxième est un réseau Full-connected Feed Forward,Ses couches sont suivie d'une opération de normalisation , d'une autre façon on peut dire que la sortie est sous la forme :

$$Norm(souscouche(x) + x)$$

- **Décodeur :** La deuxième partie, le décodeur, prend la représentation latente générée par l'encodeur et l'utilise pour générer la séquence de sortie. De manière similaire à l'encodeur, le décodeur est composé de blocs d'attention, mais il introduit une attention additionnelle sur la sortie de l'encodeur pour mieux comprendre le contexte global. Cette approche séquentielle permet au décodeur de générer une séquence de sortie token par token, en prenant en compte les relations apprises lors de la phase d'encodage.

Le décodeur aussi est composé de N couches , et en plus des deux sous couches qui sont présentes dans l'encodeur , le décodeur dispose également d'une troisième couche qui est la couche de Multi-head attention appliquer directement sur la sortie du l'encodeur . La première sous couche du Multi-head attention est modifié dans le décodeur par rapport à celle de l'encodeur , cette modification consiste principalement à appliquer le masquage qui empêche les positions futures d'influencer sur les calcules de l'attention , assurant ainsi que chaque élément de la séquence des Embeddings de la sortie ne prend en considération que les positions précédente pour la génération des prédictions .

Les entrées que le décodeur prend depuis l'encodeur représente le K(Key) et le V(value) pour la seconde couche du self attention , par contre la sortie du Masked Self attention constitue le Q (query)

Dans ce qui suit nous allons essayer d'expliquer en détails ces concepts .

5. Les concepts clés des transformers :

5.1.Input Embedding :

- **Tokenization :**

La tokenization est une étape fondamentale dans le traitement des transformers. Elle consiste à décomposer une séquence de texte en unités individuelles appelées tokens. Les tokens peuvent représenter des mots, des sous-mots ou même des caractères, en fonction de la granularité requise. L'objectif principal de la tokenization est de convertir le texte brut en un format adapté à l'analyse et au traitement ultérieur par les modèles d'apprentissage automatique.

Le processus de tokenization commence par la segmentation du texte en segments tels que des phrases, des mots ou des sous-mots. Cette segmentation peut être réalisée en utilisant des espaces, des virgules, ou d'autres marqueurs, selon la nature du texte. Par exemple, la phrase "La tokenization est cruciale pour les tâches de TALN" pourrait être segmentée en mots individuels : ["La", "tokenization", "est", "cruciale", "pour", "les", "tâches", "de", "TALN"].

Les étapes suivantes incluent la gestion de la ponctuation, la normalisation des majuscules et la construction d'un vocabulaire associant chaque token à un identifiant numérique. Enfin, le texte est encodé en remplaçant chaque token par son identifiant numérique correspondant.

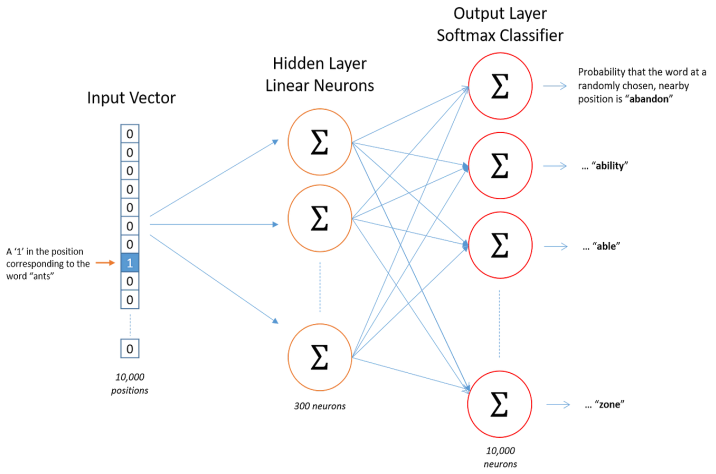
- **Réseau neuronal pour les Embeddings de Mots :**

Une fois le processus de tokenization complété, le réseau neuronal entre en action pour définir la représentation vectorielle, ou embedding, de chaque mot.

Pendant la phase d'entraînement, le réseau adopte une approche courante dans le domaine, en utilisant des techniques spécifiques comme "Skip-gram" ou "CBOW" pour assimiler les embeddings de mots. Dans le cadre du "Skip-gram," chaque exemple d'entraînement se compose d'un mot cible, étiqueté avec 1, et des mots environnants considérés comme non cibles, étiquetés avec 0. Le réseau est ainsi incité à anticiper le mot suivant, les mots précédents, ou les mots voisins en fonction de la spécificité de la tâche.

Cette configuration d'entraînement permet au réseau d'acquérir la capacité de saisir les relations sémantiques entre les mots, car il doit effectuer une discrimination entre le mot cible et les mots non cibles dans le contexte. Une fois entraîné, le réseau produit des embeddings de mots, des représentations vectorielles riches en informations. Lorsqu'il est utilisé dans des tâches spécifiques, une fonction softmax est souvent appliquée. Cette fonction transforme les sorties du réseau en probabilités sur l'ensemble du vocabulaire, facilitant ainsi la prédiction du mot suivant ou des mots voisins.

Il est à noter que dans de nombreux scénarios, des embeddings de mots pré-entraînés tels que Word2Vec ou GloVe sont utilisés pour initialiser cette couche, surtout lorsque les données d'entraînement sont limitées. Ces embeddings pré-entraînés offrent une base solide, capturant déjà des aspects sémantiques des mots, ce qui est particulièrement bénéfique dans des contextes où la disponibilité de données d'entraînement est restreinte.



5.2.Positional Encoding :

Dans le domaine de transformers, l'encodage positionnel discret vise à représenter l'emplacement ou la "position" des éléments au sein d'une séquence à l'aide de tenseurs de dimensions finies. L'objectif principal est de générer un tenseur d'encodage qui, lorsqu'il est injecté dans un modèle, lui donne des informations sur la position de chaque valeur dans la séquence.

Dans le contexte des transformers, où le traitement des vecteurs d'encodage se fait en parallèle dans le bloc d'attention, l'encodage positionnel est devenu essentiel. L'absence d'informations d'ordre entre les mots dans ces modèles souligne la nécessité d'introduire des encodages positionnels pour capturer l'ordre séquentiel des éléments.

- **Les exigences de l'Encodage Positionnel :**

Pour jouer le rôle d'identifiants de position, les encodages positionnels doivent répondre à certaines exigences :

- **Première Exigence :**

Chaque encodage positionnel doit avoir le même identifiant, indépendamment de la longueur de la séquence ou de l'entrée. Cela garantit une cohérence dans la représentation positionnelle pour une position donnée dans une phrase.

- **Deuxième Exigence:**

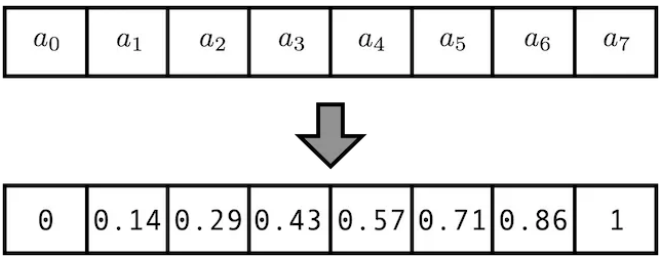
Les encodages positionnels, en tant que décalages ajoutés aux embeddings de mots, ne peuvent pas être trop grands. La limitation est cruciale pour éviter des décalages importants qui pourraient perturber la signification sémantique de l'espace d'embedding.

- **Formulation Mathématique des Encodages Positionnels :**

Avant d'explorer la formulation mathématique des encodages positionnels, examinons de plus près les approches initiales qui ont jeté les bases de ces formulations. Ces approches initiales ont évolué pour résoudre des problèmes spécifiques liés à la représentation de la position dans les séquences. Les principales approches sont les suivantes :

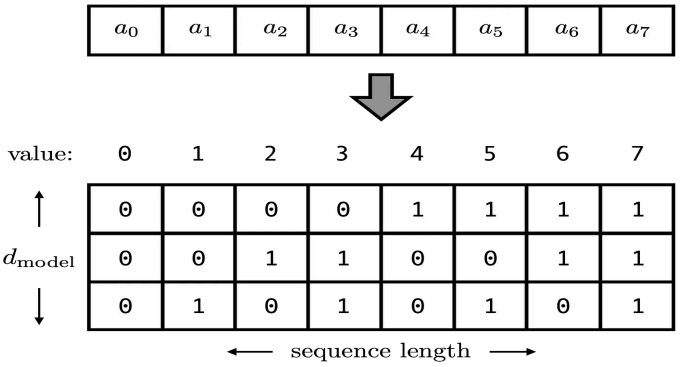
- **Approches Initiales : Comptage et Normalisation**

Les premières tentatives ont impliqué une approche de comptage simple, associant chaque élément à son indice de position absolu. Cependant, ces encodages positionnels absolus ont présenté des problèmes numériques en raison de la magnitude potentielle des valeurs de position. La normalisation a été introduite pour atténuer ces problèmes, mais elle a rencontré des difficultés avec des longueurs de séquence variables.



- **Utilisation des Nombres Binaires**

Pour surmonter ces obstacles, l'idée d'utiliser des nombres binaires pour représenter les positions a été explorée. La conversion d'entiers en forme binaire a ouvert la possibilité de représenter des séquences de longueurs arbitraires tout en maintenant les valeurs dans une plage souhaitée.



- **Vecteurs Binaires Continus**

La limitation des vecteurs binaires a conduit à l'introduction de vecteurs binaires continus. L'utilisation de fonctions sinus pour créer des cycles continus a donné naissance à des encodages positionnels plus flexibles. Cette approche a permis une représentation plus fluide des positions dans les séquences.

- **Introduction des Fonctions Sinus et Cosinus**

Finalement, l'incorporation des fonctions sinus et cosinus dans l'encodage positionnel a été adoptée. Cela a permis de résoudre les problèmes de normalisation et a facilité les transformations linéaires nécessaires pour les opérations de translation.

Les différentes approches initiales ont convergé vers une formulation mathématique finale des encodages positionnels dans les Transformers. La fonction adoptée, basée sur les fonctions sinus et cosinus, est conçue pour être bornée dans les valeurs qu'elle peut prendre. La formulation précise de ces encodages positionnels est la suivante :

$$PE(x, 2i) = sin(\frac{x \cdot w_i}{10000^{\frac{2i}{d_{model}}}})$$

$$PE(x, 2i + 1) = cos(\frac{x \cdot w_i}{10000^{\frac{2i}{d_{model}}}})$$

x : représente la position dans la séquence

i : est la dimension du vecteur d'encodage positionnel.

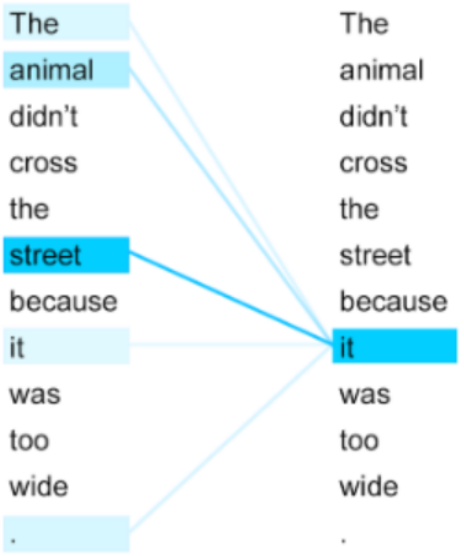
d_{model} : est la dimension du vecteur d'embedding.

Cette formulation résulte d'itérations et d'ajustements successifs visant à parvenir à une solution répondant aux exigences spécifiques de l'encodage positionnel dans le contexte des Transformers.

5.3.Principe de Self-Attention :

Le mécanisme d'auto-attention (Self-Attention) est le pilier fondamental qui distingue les transformers des approches traditionnelles malgré le fait qu'il existait bien avant la venue des transformers. Il permet aux modèles de considérer les relations entre tous les éléments d'une séquence de manière simultanée. Voici comment fonctionne le principe de self-attention :

- **Rôles de Query, Key, et Value** : ces termes viennent du vocabulaire de la recherche d'information, et veulent dire que chaque élément dans la séquence joue trois rôles essentiels :
  Query (Requête) : pose une question aux autres éléments.
  Key (Clé) : détient de l'information sur lui-même.
  Value (Valeur) : fournit la réponse basée sur cette information.
- **Calcul des Scores d'Attention :** pour chaque élément, un score d'attention est calculé en évaluant la pertinence de la question posée (Query) par rapport à l'information détenue par les autres éléments (Key).Ce score mesure l'importance de chaque élément dans la séquence pour la question spécifique.



- **Combinaison pondérée des valeurs :** les réponses (Values) de tous les éléments sont combinées pour former une représentation agrégée.Chaque réponse est pondérée en fonction de son score d'attention, mettant l'accent sur les éléments les plus pertinents pour la question posée.
- **Avantages de l'Auto-Attention :** la self-attention capture des dépendances à longue portée en d'autres termes elle permet au modèle de comprendre les relations complexes sur des distances temporelles importantes, elle permet aussi un traitement parallèle car le calcul des scores d'attention

peut être effectué en parallèle pour chaque élément, favorisant ainsi l'efficacité computationnelle, elle permet aussi une flexibilité et une adaptabilité du fait que chaque élément peut influencer tous les autres, permettant une adaptation dynamique aux structures de séquence variées.

Le mécanisme de Self attention est représenté pratiquement par le calcule d'une matrice score de dimension (n°features,n°features) en utilisant la formule suivante :

$$Attention\ (Q, K, V) = softmax(QK^t/\sqrt{d_x})\ \mathrm{V}$$

ou le dx représente la taille des vecteurs utilisés pour faire le word Embedding ainsi que le calcule du positional Encoding .

Voici un exemple de la matrice score :

	YOUR	CAT	IS	A	LOVELY	CAT
YOUR	0.268	0.119	0.134	0.148	0.179	0.152
CAT	0.124	0.278	0.201	0.128	0.154	0.115
IS	0.147	0.132	0.262	0.097	0.218	0.145
A	0.210	0.128	0.206	0.212	0.119	0.125
LOVELY	0.146	0.158	0.152	0.143	0.227	0.174
CAT	0.195	0.114	0.203	0.103	0.157	0.229

cet exemple nous montre que plus le score est élevé entre mot(i) et le mot(j) le plus le mot (j) est relié au mot (i) , et évidemment le score le plus élevé se trouve au niveau de diagonale (chaque mot et fortement relié à lui même).

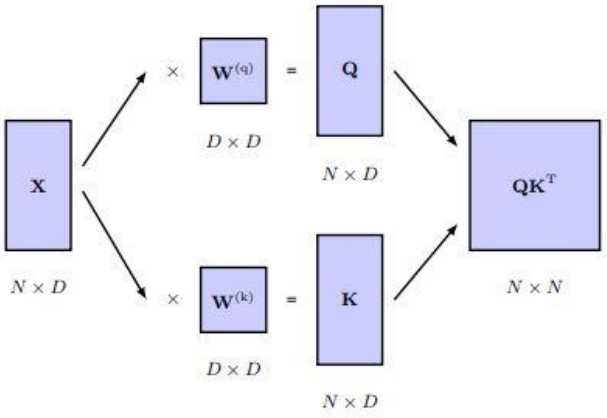
Les trois matrices Q(query), K(key) et V(value) sont obtenues en calculant les matrices des paramètres W_Q le W_K et W_V durant la phase d'entraînement du modèle :

$$Q = W_Q * X$$

$$K = W_K * X$$

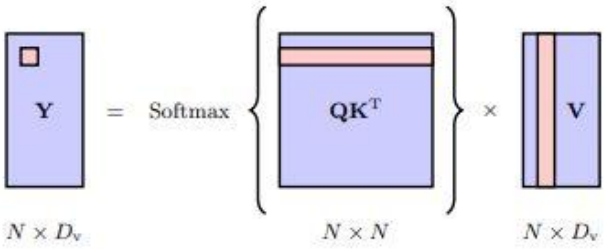
$$V = W_V * X$$

Le schéma ci-dessous illustre les étapes de calcul des matrices Q et K:



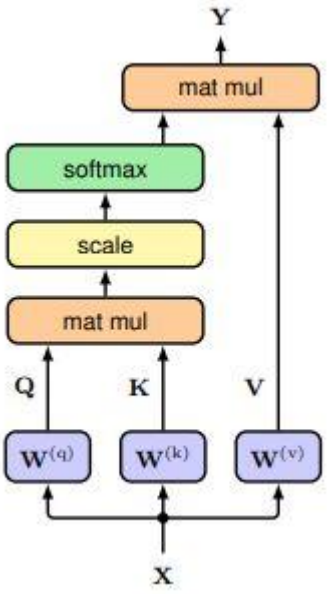
sachant que X représente la matrice résultant du positional encoding.

L'étape qui suit est illustrée par ce schéma:



La fonction softmax est appliquée sur la matrice Query *(Key transposée) pour avoir des valeurs des paramètres inclus entre 0 et 1 et leur somme sur une ligne, c'est à dire le score de chaque mot avec tous les autres, soit égale à 1, ceci est fait pour non seulement éviter d'avoir des valeurs négatives des coefficients et s'assurer que si la relation d'un mot est forte avec un autre ce sera au détriment des autres.

Seulement sachant que les gradients de la fonction softmax deviennent exponentiellement petits pour des entrées de magnitude élevée, pour éviter que cela ne se produisse, il faut appliquer un échelonnement sur la matrice résultante du produit matriciel entre Q et K, et cela est fait en la divisant par $\sqrt{d_x}$. Ceci est appelé *scaled dot-product self-attention*.TOutes ces étapes sont résumées dans le schèma ci-dessous:



ou mat mul désigne le produit matriciel et scale désigne l'étape du scaling(division par $\sqrt{d_x}$).

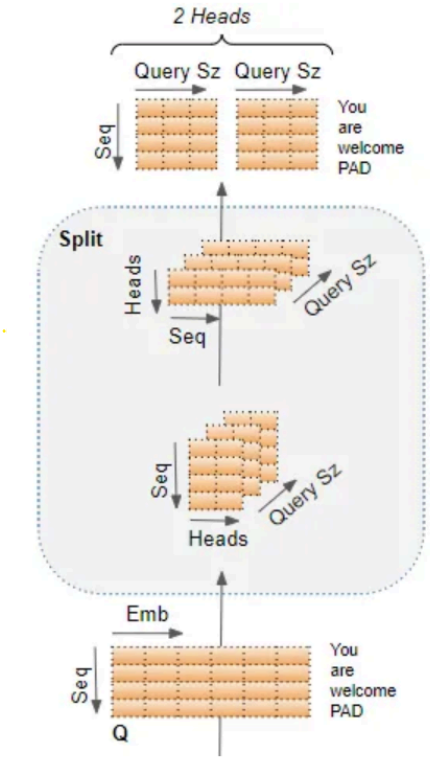
5.4.Multi-head Attention:

Dans le Transformer, le module d'Attention répète ses calculs plusieurs fois en parallèle. Chacun de ces calculs est appelé une "Attention Head" (tête d'attention). Le module d'Attention divise ses paramètres de Query (requête), Key (clé), et Value (valeur) en N parties et fait passer chaque partie de manière indépendante à travers une tête distincte. Tous ces calculs d'Attention similaires sont ensuite combinés pour produire un score d'Attention final. Cela s'appelle l'Attention Multi-tête et donne au Transformer une plus grande capacité à encoder des relations et des subtilités multiples pour chaque mot.

L'idée derrière l'utilisation de plusieurs têtes d'attention est de permettre au modèle de capturer des motifs et des relations plus complexes dans les données. Chaque tête peut se spécialiser dans la capture de différents aspects ou dépendances dans la séquence, offrant ainsi une représentation plus riche et expressive, notamment dans le NLP une tête pourrait capturer la relation de temps entre les mots et une celle du vocabulaire et ainsi de suite si on venait à utiliser une seule tête ces relations (patterns) seraient confondues.

Cette division logique des matrices Key , Query , Value ainsi que les matrices des poids W sur les N têtes ce fait comme ceci :

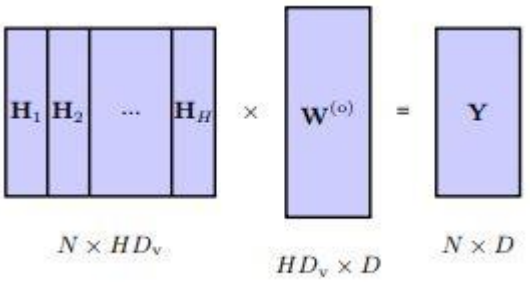
Query Size = Embedding Size / Number of heads



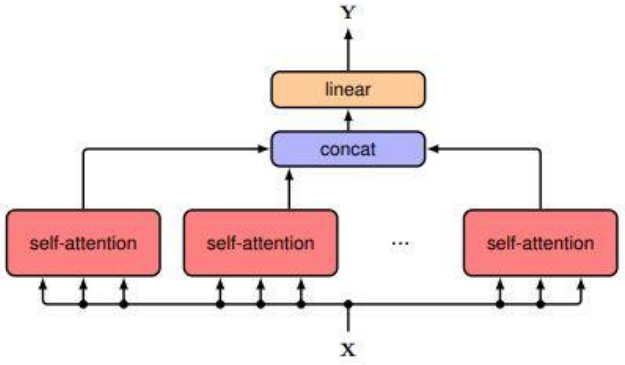
Aprés ca , le résultat de calculs des score des différentes têtes est concaténé dans une seule matrice selon la formule suivante :

$$MultiHead(Q,K,V) = concat(head_1,.....)W0$$

Ou le W0 représente la matrice de poids utilisée pour effectuer une transformation linéaire sur le résultat de la concaténation, cela permet d'ajuster la dimension de la sortie car chaque matrice résultante des tete aura une dimension N × Dv et donc pour H tetes on obtiendera une matrice de N × HDv après concaténation, comme suit :



voici un schèma qui résume en gros le contenu de la couche du multi-head attention :



5.5.Masked Multi-head Attention :

Le “Masked Attention” est le mécanisme utilisé principalement dans la première couche du décodeur , qui est utilisé avec les Outputs Embeddings .

Son principe est similaire à celui du Multi-head Attention normale , mais en masquent les mots qui viennent après le mot courant .

D’une autre façon , on peut dire que le masque modifie la matrice des scores (résultat du Attention) d’une sorte qu’il produit des (-∞) A la place des scores avec les mots qui suivent , de cette façon on oblige le décodeur à se focaliser seulement sur les mots précédents et empêche les mots futurs de contribuer aux calculs des scores .

Cela est bénéfique surtout dans les tâches de génération du texte , donc le fait de masquer les positions futures peut nous aider à éviter les fuites d’informations futures , pour que le modèle n'apprenne pas à tricher en utilisant ces informations , ce qui nous permet effectivement d’avoir des résultats plus cohérentes .

5.6.Position wise Feed Forward Networks

les “Position-wise Feed-Forward Networks” font référence à un type de couche de réseau neuronal appliquée de manière indépendante à chaque position dans la séquence. Cette couche est un composant clé du modèle Transformer et est responsable d'introduire une non-linéarité et de capturer des motifs complexes dans les données d'entrée.

Ce réseau est composé de plusieurs couches entièrement connectées , et il peut être exprimé par la formule suivante :

$$FFN(X) = ReLU(W_1 X + b)W_2 + c$$

L'utilisation de la fonction d'activation linéaire rectifiée (ReLU) introduit une non-linéarité dans le modèle, lui permettant de capturer des relations complexes dans les données.

5.7.Add & Norm

Le résultat obtenu de chaque couche (Multi-head , Feed forward) est passé après par une couche d’addition et de normalisation .

L'opération de l’addition consiste à ajouter l'entré à la sortie de la couche .

Cela aide à atténuer les problèmes liés aux gradients qui s'annulent, car le signal de gradient peut circuler directement vers les couches antérieures lors de la rétropropagation.

mathématiquement :

$$Sortie = SortieCouche + Entrée$$

tandis que La normalisation est utilisée pour stabiliser et normaliser les activations au sein d'une couche. Elle aide à résoudre des problèmes tels que le déplacement covariant interne, où la distribution des entrées d'une couche change pendant l'entraînement. La normalisation de couche garantit que les entrées d'une couche ont une moyenne et un écart-type constants, favorisant un entraînement plus stable.

Mathématiquement, la normalisation de couche est souvent représentée comme :

$$Sortie = LayerNorm(Entrée + Sortie\ Couche)$$

tel que la normalisation est représentée par :

$$\widehat{x_j} = (x_j - \mu_j) \div \sqrt{\sigma^2{}_j + \epsilon}$$

6.Les Types des Transformers :

L'architecture des transformers, ayant révolutionné le domaine du machine learning, se déploie dans une diversité de types, chacun taillé sur mesure pour des tâches spécifiques. Alors que nous avons plongé dans la théorie sous-jacente des transformers, il est désormais temps d'explorer les différentes incarnations de ces modèles révolutionnaires. De BERT à GPT, chaque type de transformer apporte sa propre contribution à l'évolution de l'intelligence artificielle. Dans cette partie, nous examinerons de près les caractéristiques distinctives de BERT, GPT surtout.

6.1.BERT

BERT(acronyme de Bidirectional Encoder Representations from Transformers) est un modèle de langage révolutionnaire dans le domaine du traitement du langage naturel (NLP). Développé par Google, BERT a été introduit en 2018 par une équipe de chercheurs dirigée par Jacob Devlin.

Contrairement aux modèles antérieurs, BERT prend en compte simultanément les mots à gauche et à droite d'un mot donné. Cette approche bidirectionnelle offre une vision contextuelle plus riche, améliorant ainsi la compréhension des nuances linguistiques et des relations sémantiques.

Il repose sur une architecture transformer à empilement profond, comprenant 12 couches dans sa version de base. L'encodeur bidirectionnel de BERT intègre des mécanismes d'attention multi-têtes, permettant au modèle de capturer les dépendances contextuelles des deux côtés d'un mot dans une phrase. Les embeddings de dimension 768, associés à des mécanismes de masquage de mots pendant l'entraînement, favorisent l'apprentissage de représentations contextuelles riches. Les couches d'attention, le nombre élevé de couches et l'utilisation d'une attention multi-têtes contribuent à la capacité du modèle à saisir des relations complexes et à effectuer des tâches diverses en traitement du langage naturel.

BERT a été pré-entraîné sur un vaste corpus de texte non annoté provenant du web. Plus précisément, il a été entraîné sur le livre anglais complet de l'ensemble du projet Wikipédia, ainsi que sur des livres électroniques supplémentaires. L'objectif était de fournir au modèle une compréhension profonde du langage en exposant le réseau à une quantité considérable de données textuelles diverses et riches en contexte.
Le processus d'entraînement de BERT consiste à prédire certains mots masqués dans les phrases. Pendant cette phase, le modèle apprend les relations entre les mots et la structure syntaxique des phrases. Cette tâche d'entraînement non supervisé aide BERT à capturer des représentations linguistiques générales qui peuvent ensuite être adaptées à des tâches spécifiques via un processus de fine-tuning.

Dans la pratique, BERT peut être utilisé comme un extracteur de caractéristiques ou il peut être fine tuné sur des données annotées et puis être utilisé pour des tâches plus spécifiques.

6.2.GPT

GPT (Generative Pre-trained Transformer) trouve ses fondements dans l'architecture révolutionnaire des Transformers introduite par Vaswani et al. en 2017. Cette architecture, caractérisée par des mécanismes d'auto-attention, permet à GPT de pondérer simultanément différentes parties des séquences d'entrée, améliorant ainsi sa compréhension contextuelle.

Dans la phase de pré-entraînement, GPT utilise l'apprentissage non supervisé pour prédire le mot suivant dans une séquence en fonction de son contexte. Cette approche d'apprentissage profond confère au modèle une compréhension approfondie des motifs linguistiques complexes et du contexte, posant ainsi les bases d'un traitement polyvalent du langage naturel.

GPT démontre son adaptabilité grâce à des transformations d'entrées spécifiques à la tâche. Pour des tâches telles que la classification de texte, un fine-tuning direct suit l'architecture du modèle pré-entraîné, suivi d'une couche linéaire+softmax. En cas d'entrées structurées, comme la réponse aux questions ou l'implication textuelle, GPT adopte une approche de type traversée, convertissant des entrées structurées en séquences ordonnées pour une intégration transparente avec le modèle pré-entraîné.

Dans des configurations expérimentales, GPT prouve son efficacité grâce à un pré-entraînement non supervisé sur l'ensemble de données BooksCorpus. Les spécifications du modèle comprennent un transformateur à 12 couches sans encodeur, un schéma d'optimisation Adam et des stratégies

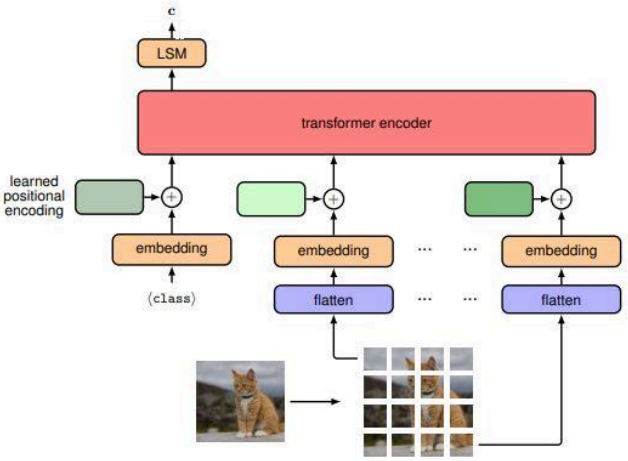
spécifiques de dropout et de régularisation. L'utilisation d'incorporations de position apprises remplace la version sinusoïdale originale, contribuant à l'efficacité du modèle.

GPT excelle dans le fine-tuning supervisé sur diverses tâches, notamment l'inférence de langage naturel, la réponse aux questions, la similarité sémantique et la classification de texte. Il convient de souligner le succès du modèle dans les tâches de réponse aux questions et de raisonnement de bon sens, surpassant les références précédentes sur l'ensemble de données RACE et le test Story Cloze.

Dans les tâches de similarité sémantique, GPT prédit l'équivalence de phrases avec une précision de pointe, dépassant les méthodologies précédentes. Les tâches de classification de texte, telles que CoLA et SST-2, mettent en lumière la capacité de GPT à comprendre les subtilités linguistiques, avec un bond significatif des performances et un score global impressionnant sur le benchmark GLUE.

6.3.Vision transformers:

Les Vision Transformers (ViTs) ont émergé en 2020 comme une innovation majeure dans le domaine de la vision par ordinateur, introduite par Dosovitskiy et al. Les ViTs sont directement inspirés du succès des transformers dans le traitement du langage naturel. En appliquant ce concept à la vision par ordinateur, les ViTs traitent les images en découpant celles-ci en patches, qu'ils transforment en vecteurs. Ces vecteurs, représentant des parties de l'image, sont ensuite soumis à des mécanismes d'attention pour apprendre les relations et les motifs complexes à l'échelle globale de l'image.L'architecture des ViTs se compose de blocs transformer, où chaque bloc gère un ensemble de patches. Ces blocs sont responsables de modéliser les relations entre les patches, permettant ainsi de capturer des informations contextuelles sur toute l'image. La flexibilité intrinsèque des transformers offre la possibilité de modéliser des relations longue distance, améliorant significativement les performances sur des tâches comme la classification d'images.La figure ci-dessous est une illustration d'un Vit pour une tâche de classification, la sortie du transformer est transformée par une couche linéaire avec une activation softmax(LSM) produisant le vecteur c :



Les ViTs ont trouvé une large application dans la recherche en vision par ordinateur. L'un de leurs exemples notables concerne la classification d'images médicales, où ils ont démontré une capacité exceptionnelle à extraire des caractéristiques discriminantes à partir d'images complexes. Leur adoption rapide dans divers domaines témoigne de leur efficacité et de leur potentiel révolutionnaire.

7.Défis et Limitations:

L'architecture Transformer, bien qu'innovante et performante, présente également des défis et des limitations, les plus importants sont:

- **Complexité computationnelle :** les modèles Transformer nécessitent d'énormes ressources computationnelles, en particulier avec l'augmentation de la taille du modèle et du jeu de données, ce qui peut rendre leur utilisation coûteuse en termes de puissance de calcul.
- **Longueur des séquences :** Les Transformers peuvent avoir des difficultés avec des séquences très longues en raison de la complexité quadratique de l'attention. Cela limite parfois leur application dans des domaines où la dépendance contextuelle sur de longues séquences est cruciale.

- **Interprétabilité :** L'interprétation des modèles Transformer peut être difficile en raison de leur nature complexe et non linéaire. Comprendre comment ces modèles prennent leurs décisions peut être un défi.
- **Besoin de données étiquetées massives** : Les Transformers, en particulier dans le domaine de l'apprentissage supervisé, dépendent souvent de grandes quantités de données étiquetées pour atteindre des performances optimales, ce qui peut être un obstacle dans des domaines où de telles données sont limitées.
- **Adaptabilité à différentes tâches :** Bien que les Transformers aient montré leur efficacité dans diverses tâches, ils ne sont pas toujours la meilleure option pour des domaines spécifiques. D'autres architectures, comme les CNNs pour la vision par ordinateur, peuvent parfois surpasser les Transformers selon la nature de la tâche.
- **Taille du modèle :** Des modèles plus grands peuvent entraîner des problèmes d'implémentation, de stockage et de transmission. Leur déploiement sur des appareils avec des ressources limitées peut être un défi.

8.Evolution et tendance

Les tendances actuelles dans l'architecture des Transformers reflètent une quête constante d'améliorations et d'optimisations visant à accroître l'efficacité et la polyvalence de ces modèles. Voici quelques-unes des tendances notables dans l'évolution de l'architecture des Transformers :

- **Expansion des Dimensions :**

Les modèles récents expérimentent avec des dimensions plus élevées pour les vecteurs d'embedding, permettant une représentation plus riche et nuancée des informations. Cela inclut des augmentations tant dans la dimension du modèle (d_model) que dans le nombre de têtes d'attention.

- **Attention Structurée :**

Pour capturer des relations plus complexes, les chercheurs explorent des mécanismes d'attention plus structurés. Cela pourrait inclure des variantes telles que l'attention longitudinale ou l'attention axiale, permettant au modèle de se concentrer sur des relations spécifiques dans la séquence.

- **Modèles Plus Profonds :**

Les architectures avec un nombre croissant de couches gagnent en popularité. Des modèles plus profonds peuvent capturer des dépendances à plus longue portée, mais cela nécessite également des stratégies efficaces pour éviter le surajustement.

- **Transformer Lite :**

Dans un souci d'efficacité, des recherches sont menées sur des versions plus légères des Transformers tout en maintenant des performances acceptables. Cela comprend des techniques de compression, de quantification, et l'exploration d'architectures plus simples.

- **Attention Multimodale Intégrée :**

L'intégration harmonieuse de l'attention sur plusieurs modalités, telles que le texte, l'image et l'audio, est une tendance émergente. Cela ouvre la voie à des modèles capables de traiter des informations provenant de différentes sources de manière cohérente.

- **Auto-attention Évolutive :**

Des approches d'auto-attention évolutive sont explorées, où le modèle peut ajuster dynamiquement l'importance accordée à différentes parties de la séquence en fonction de la tâche spécifique ou des propriétés de la donnée en entrée.

- **Attention Axée sur la Tâche :**

L'exploration de mécanismes d'attention adaptatifs, axés sur la tâche, est une tendance. Cela implique d'ajuster la structure d'attention en fonction des exigences spécifiques de la tâche à accomplir.

- **Auto-entraînement Continu :**

Les architectures qui peuvent s'auto-entraîner de manière continue, en intégrant de nouvelles données sans un réentraînement complet, sont également à l'étude, permettant une adaptation dynamique à l'évolution des connaissances.

Ces tendances illustrent l'engagement continu des chercheurs à explorer de nouvelles avenues pour optimiser les performances des modèles Transformers, renforçant ainsi leur pertinence et leur applicabilité dans divers domaines.

partie 2 : étude Pratique.

1.Exemple d'application étudié :

Dans cette section, nous présentons un exemple d'application concret qui met en œuvre le modèle Transformer que nous avons développé à partir de zéro en utilisant PyTorch.

Notre objectif principal dans cette étude pratique était de créer un système de traduction automatique de l'anglais vers l'italien en utilisant notre propre implémentation du modèle Transformer.

Pour entraîner et évaluer notre modèle, nous avons utilisé un ensemble de données provenant de Hugging Face, une ressource bien établie dans le domaine de l'apprentissage automatique. Ces données comprenaient des paires de phrases en italien et en anglais, fournissant une base solide pour l'apprentissage des relations de traduction.

Nous avons mis en place un environnement d'entraînement sur Colab, utilisant un cahier pour faciliter le processus. Les étapes d'entraînement et de validation ont été soigneusement orchestrées pour garantir des performances optimales du modèle.

2.Prétraitement des données :

Dans cette phase cruciale de notre étude, plongeons dans les détails du prétraitement des données, une étape fondamentale pour établir les bases solides de notre modèle de traduction automatique anglais-italien basé sur le Transformer.

- **Gestion du Tokenizer :**

Nous commençons par créer un chemin dédié pour stocker notre tokenizer, guidé par la configuration spécifique de notre modèle. Ensuite, le tokenizer est construit avec une attention particulière aux paramètres, notamment la gestion des mots inconnus et un pré-tokenizer basé sur les espaces.

À travers un processus d'entraînement, notre tokenizer absorbe les nuances linguistiques de chaque langue, s'adaptant ainsi à la tâche de traduction avec une compréhension fine des mots et expressions. Une fois formé, le tokenizer est sauvegardé dans un fichier spécifique, simplifiant ainsi son utilisation future.

- **Préparation des Données :**

Avec notre outil linguistique (le tokenizer) prêt, nous entrons dans la phase cruciale de préparation des données, où chaque détail compte pour l'efficacité de notre modèle.

Nous faisons appel à la bibliothèque Hugging Face pour charger un ensemble de données de livres, adapté à notre paire de langues source-cible. Pour chaque langue impliquée dans la traduction, nous utilisons notre fonction de gestion du tokenizer pour construire des tokenizers adaptés.

Une répartition stratégique assigné 90% des données à l'ensemble d'entraînement, laissant les 10% restants dédiés à la validation. À partir de ces ensembles, nous élaborons une représentation numérique des phrases en utilisant nos tokenizers. Cela s'accomplit en structurant des ensembles adaptés à notre tâche de traduction, où chaque élément est préparé pour l'entraînement ou la validation, facilitant ainsi le processus d'adaptation de notre modèle Transformer aux subtilités de la langue source et cible.

Une analyse détaillée nous permet de déterminer la longueur maximale des phrases dans nos données, une information cruciale pour le dimensionnement de notre modèle. Enfin, des dataloaders sont créés pour l'entraînement et la validation, facilitant ainsi le flux de données dans notre modèle lors des différentes phases.

3.Entraînement :

3.1.Implémentation du modèle:

-Les Bibliothèques utilisées :

torch avec la version : 2.0.1

torchvision avec la version :0.15.2

torchaudio avec la version : 2.0.2

torchtext avec la version : 0.15.2

datasets avec la version : 2.15.0

tokenizers avec la version : 0.13.3

torchmetrics avec la version : 1.0.3

tensorboard avec la version : 2.13.0

altair avec la version : 5.1.1

wandb avec la version :0.15.9

L'entraînement de ce modèle Transformer a été réalisé sur Google Colab en exploitant la puissance de calcul de 4 unités de traitement graphique (GPUs) compatibles avec CUDA. Cette configuration parallèle a permis d'accélérer significativement le processus d'apprentissage, offrant une efficacité accrue dans le traitement des calculs intensifs associés à la mise à jour des poids du modèle.

Et pour des raisons de limitation des ressources de calcul, nous avons spécifié un nombre d'époques égal à 10, ainsi qu'une valeur de d_model fixée à 256 (alors que la taille spécifiée dans l'article était de 512). En augmentant ces paramètres, il est possible d'obtenir des résultats plus précis et pertinents

Afin de pouvoir Implémenter notre Modèle de transformers nous allons commencer par l'implémentation des couches composantes du modèle :

- Word Embeddings : Cette classe prend en paramètre deux arguments : d_model, qui représente la taille des vecteurs générés pour chaque mot, et la taille_vocabulaire, qui représente le nombre maximum de mots dans la langue traitée au sein d'une seule phrase. En utilisant la fonction

d'embedding de PyTorch, cette classe renvoie une matrice de taille (d_model, taille_vocabulaire) contenant les vecteurs d'embedding des entrées.

La séquence des mots, notée x, est ainsi transformée en une matrice où chaque colonne correspond à l'embedding d'un mot de la séquence. Cette sortie est multipliée après par la racine carré de d_model pour mettre à l'échelle l'Embedings selon les recommandation de l'article originale sur les transformers .

-Positional Encoding : cette classe prend comme paramétres le d_model la longueur de séquence ainsi que Dropout , pour faire le positionale encoding on a besoins d'un vecteur position qui represente la position du mot dans la séquence ainsi qu'un vecteur d-model qui peut être remplie selon les formules cos (pour les lignes impaires) et le sin (pour les lignes paires) qu'ils ont étaient déjà mentionné précédemment .

-**NormalisationCouche:** Dans cette classe, nous allons mettre en œuvre une normalisation de couche en calculant la moyenne et l'écart type des sorties de la couche précédente (x). Ainsi, la nouvelle valeur de x sera obtenue en soustrayant la moyenne et en divisant par l'écart type additionné d'une petite constante epsilon. Conformément aux recommandations pratiques, nous introduisons également deux paramètres : alpha, à multiplier par la valeur précédente, et un biais à ajouter

-RisidualConnection : Cette classe représente l'interconnexion entre l'entrée et la sortie de la couche, comme nous l'avons déjà observé dans l'architecture du transformer. Dans cette classe, nous avons défini un paramètre qui est de type NormalisationCouche. Elle prend en entrée deux paramètres principaux : le sublayer (représentant la couche précédente dans l'architecture, qui peut être de type MultiHeadAttention ou FeedForward) et l'entrée de la couche précédente, x. Ainsi, elle applique la normalisation de couche sur la sortie du sublayer et ajoute l'entrée `x.

-FeedForwardBlock : Cette classe représente un réseau de neurones à deux couches. Elle prend comme paramètres d_model et d_ff, qui représentent respectivement la dimension de l'entrée et le nombre de neurones dans la couche cachée. Cette couche prend l'entrée x du réseau, la passe à travers la première couche avec une fonction d'activation ReLU, puis elle passe par la deuxième couche pour produire un vecteur de sortie x de longueur d_model

-MultiheadAttentionBlock: Cette classe représente un seul mechanisme de multi-head attention, ou l'on va définir les différentes couches des matrices Wq, Wk et Wv et puis les matrices Q, K et V et puis c'est au sein de cette classe que va se faire la division en h matrices (nombre de têtes) et puis le calcul des scores pour chacune et enfin leur concaténation pour obtenir la matrice résultante

-EncoderBlock : Dans cette classe, nous avons défini la structure du bloc encodeur qui comprend principalement les couches suivantes : une pour le bloc MultiHeadAttention, une pour le bloc FeedForward, et deux couches de ResidualConnection (qui effectuent l'addition et la normalisation).

-Encoder : en pratique on utilise plusieurs encodeurs donc cette classe représente l'ensemble des encodeurs dans notre transformer .

-DecoderBlock: Dans cette classe,on défini la structure du bloc du décodeur(un seul décodeur) qui comprend principalement les couches suivantes : une pour le bloc de masked MultiHeadAttention(qui n'est qu'un multiheadAttention block avec des paramètres différents), une pour le bloc FeedForward, une autre pour le multiHeadAttention et des couches de ResidualConnection(qui effectuent l'addition et la normalisation et les liasons avec l'encodeur).

-Decoder : de même que pour l'encodeur, un décodeur contient plusieurs decoder blocks, et donc on en assemble plusieurs pour former le décodeur final**.**

-ProjectionLayer : cette couche représente une projection linéaire de la sortie du transformer vers l'espace du vocabulaire , donc la fonction forward renvois directement la projection linéaire de l'entrée x sur l'espace du vocabulaire .

-Transformer : La classe Transformer regroupe toutes les composantes nécessaires à la construction d'un modèle Transformer. Elle prend en entrée un encodeur, un décodeur, des embeddings pour la source et la cible, des encodages positionnels pour la source et la cible, ainsi qu'une couche de projection.

La méthode encode prend une séquence source et un masque source en entrée, réalise l'embedding de la séquence, applique l'encodage positionnel, puis passe le résultat à l'encodeur pour obtenir la représentation encodée.

La méthode decode prend la sortie de l'encodeur), un masque source, une séquence cible, et un masque cible en entrée. Elle réalise l'embedding de la séquence cible, applique l'encodage positionnel, puis passe le tout au décodeur pour obtenir une sortie décodée.

La méthode project prend en entrée une séquence (x) et la passe à la couche de projection pour obtenir la sortie finale.

La fonction build_transformer construit l'ensemble du modèle en instanciant les embeddings, les encodages positionnels, et en créant les blocs d'encodeur et de décodeur. Elle assemble ensuite ces éléments pour former le modèle complet avec une couche de projection.

3.2.Entraînement du modèle:

Pour entraîner le modèle, nous avons défini la fonction train_model. Cette fonction effectue l'entraînement sur les GPU s'ils sont disponibles, sinon elle s'exécute sur le CPU. Pour ce faire, elle importe le jeu de données après prétraitement (tokens pour les sources et les destinations) en utilisant la fonction get_ds, puis elle initialise le modèle en utilisant get_model. L'entraînement se déroule dans une boucle avec un nombre d'époques égal à 10, une taille de lot de 8 et une dimension de modèle (d_model) de 256.

Nous avons utilisé l'optimiseur Adam et la fonction de coût Cross Entropy. À chaque époque, la fonction itère sur un lot de données, les faisant passer à travers l'encodeur, le décodeur et la couche de projection. Après chaque époque, une validation est effectuée en utilisant la fonction run_validation, et à chaque époque, une copie de l'état du modèle est sauvegardée dans un fichier, stocké temporairement dans un tampon mémoire

Durant l'entraînement, chaque époque a pris environ 15 minutes pour se terminer, ce qui équivaut à un temps total d'entraînement de 3 heures.

3.3.Validation :

La fonction run_validation effectue la validation d'un modèle de traduction automatique. Elle met le modèle en mode évaluation, itère sur les batches du jeu de données de validation, génère des prédictions, affiche les résultats pour un nombre spécifié d'exemples, et évalue les performances du modèle en utilisant des métriques telles que CER, WER et le score BLEU. La fonction est conçue pour suivre la qualité des traductions générées pendant l'entraînement.

-CER: Mesure le pourcentage d'erreurs de caractères entre les prédictions du modèle et les textes cibles attendus. Il représente la proportion d'erreurs de caractères dans les prédictions.

-WER: Évalue le pourcentage d'erreurs de mots entre les sorties du modèle et les références attendues. Il mesure la proportion d'erreurs de mots dans les prédictions du modèle.

-BLEU: Utilisé pour évaluer la qualité des traductions automatiques en comparant les n-grammes (groupes de mots contigus) dans les prédictions du modèle avec ceux dans les références humaines.

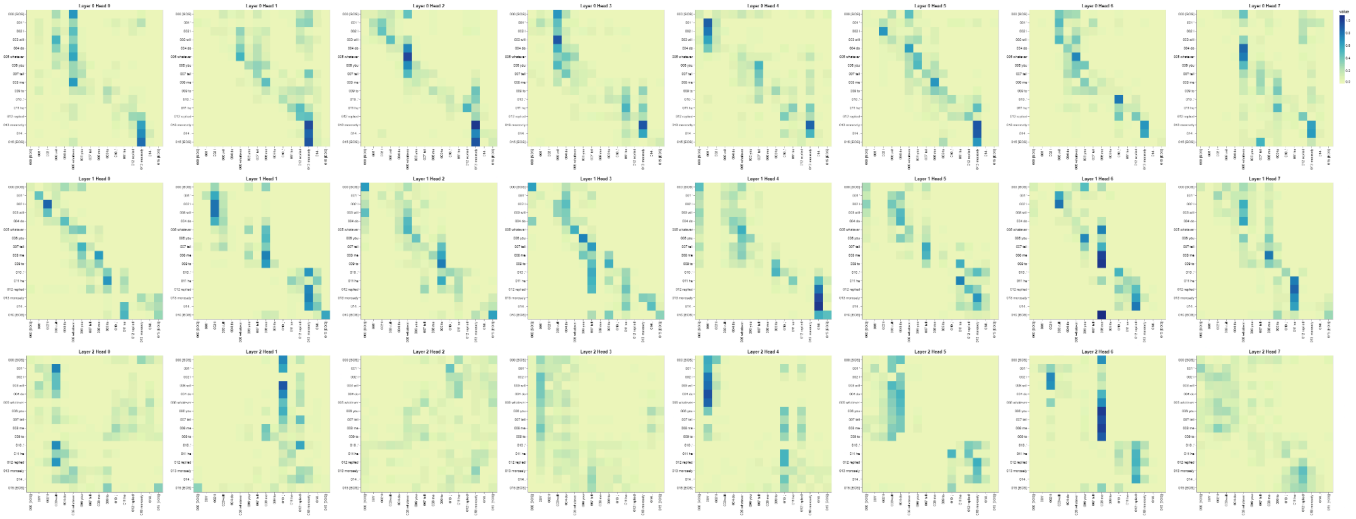
```
--------------------------------------------------------------------------------
    SOURCE: "She is at the lodge, aunt."
    TARGET: — È nella sua casa, zia.
 PREDICTED: — È la notte , — esclamò la signorina Temple .
--------------------------------------------------------------------------------
Processing Epoch 10: 100%|██████████| 1213/1213 [13:28<00:00,  1.50it/s, loss=4.551]
--------------------------------------------------------------------------------
    SOURCE: And with every result he seemed equally surprised and annoyed.
    TARGET: E, a ogni risultato delle sue manovre, egli pareva egualmente sorpreso a annoiato.
 PREDICTED: E tutti gli altri pensieri si e si .
--------------------------------------------------------------------------------
    SOURCE: 'He has turned!' she thought.
    TARGET: “S’è voltato” ella pensò.
 PREDICTED: “ È stato ” pensò .
--------------------------------------------------------------------------------
Processing Epoch 11: 100%|██████████| 1213/1213 [13:27<00:00,  1.50it/s, loss=4.375]
--------------------------------------------------------------------------------
    SOURCE: CHAPTER XV
    TARGET: XV
 PREDICTED: XV
--------------------------------------------------------------------------------
```

Image illustrative sur un exemple de validation pendant l’entrainement

3.4.Visualisation de l’attention:

Dans cette partie nous allons visualiser les scores d’une phrase (séquence) qui a été choisie de façon aléatoire comme matrice qui sera représentée par une heatmap.

- visualisation de l’attention dans l’encodeur:

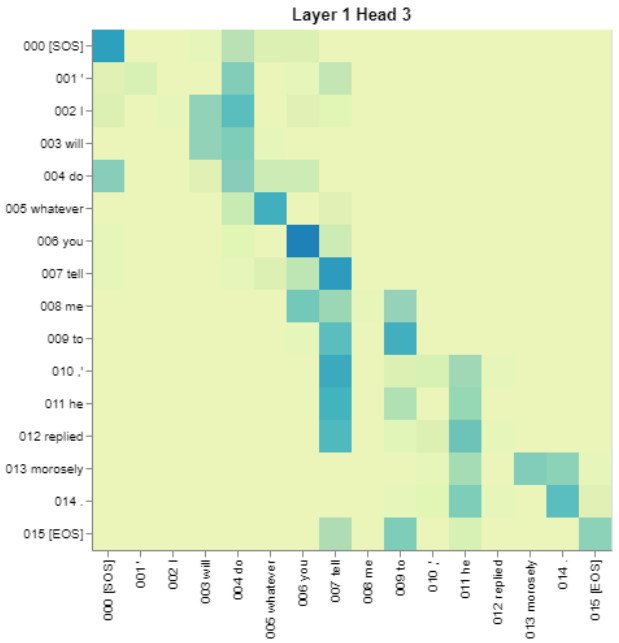


-Visualisation de l’attention dans la couche encodeur-

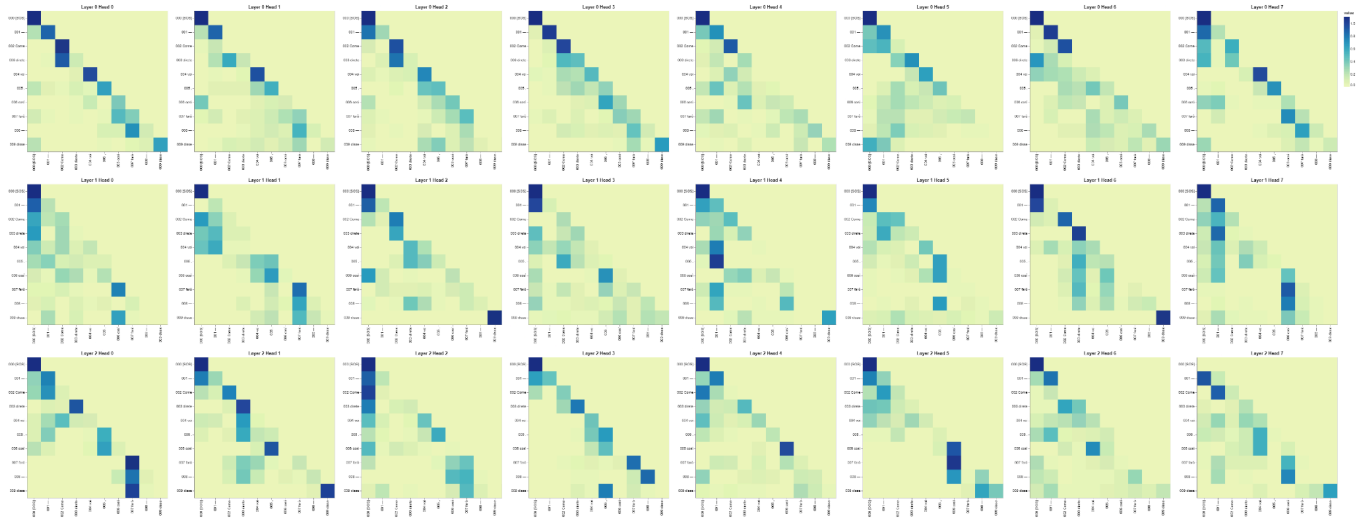
La première chose que l'on constate c'est qu'il y a 3 couches d'encodeurs(multi-head attention layer) et 8 têtes par encodeur et que les scores d'attentions sont calculés entre les mots de la langue source(anglais).

Comme on peut le constater sur les différentes heatmap ci-dessous, la diagonale est toujours la partie ayant les plus grands scores d'attention car, comme nous l'avons déjà mentionné, la relation d'un mot avec lui-même sera toujours la plus élevée.

En regardant cette heatmap de plus près, on constate que la diagonale est celle qui a le plus de scores élevé, ce qui rejoint ce qui a été dit précédemment, on remarque aussi que le mot tell a un grand score avec le mot replied ce qui implique que les deux mots sont très reliés entre eux (tell et reply sont des mots très similaires en effets).

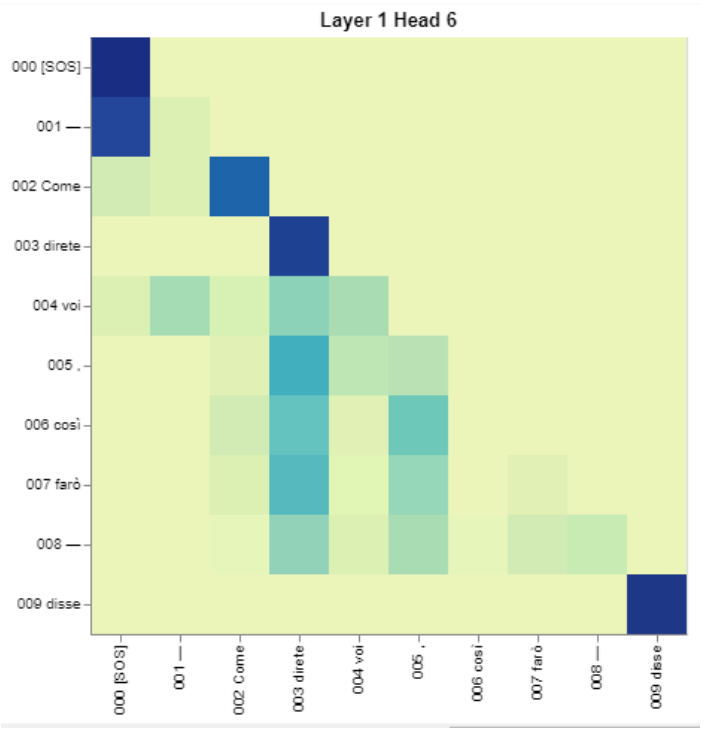


- Visualisation du décodeur :



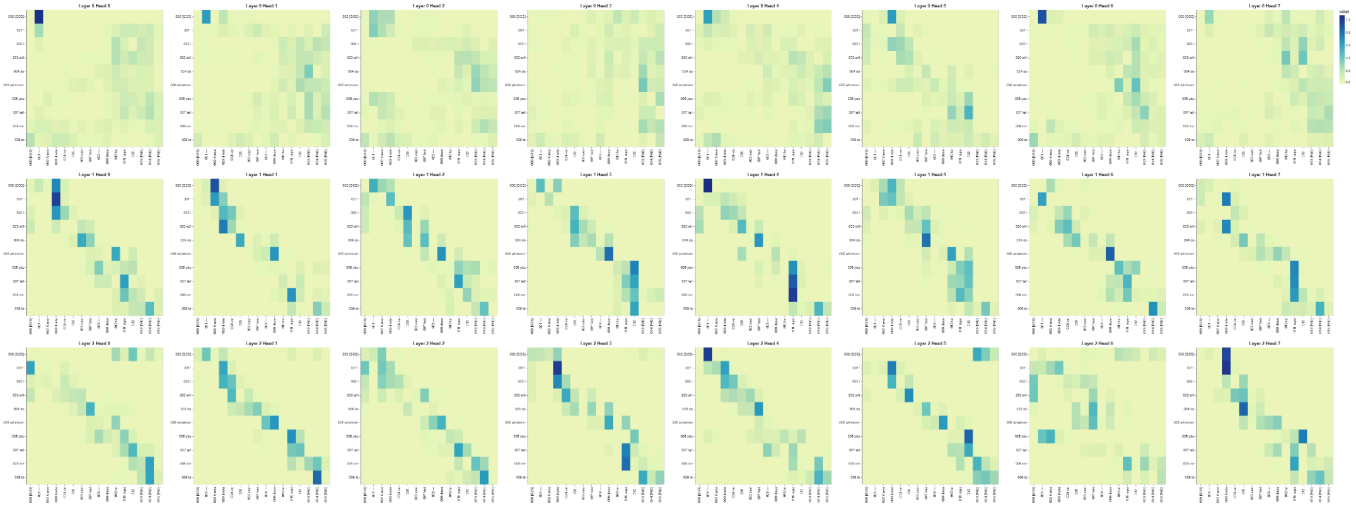
-Visualisation de l'attention dans la couche decodeur-

La première chose que l'on constate de même pour cette visualisation c'est qu'il y a 3 couches de décodeurs et 8 têtes par décodeur aussi et que les scores d'attentions sont calculés entre les mots de la langue cible (italien).



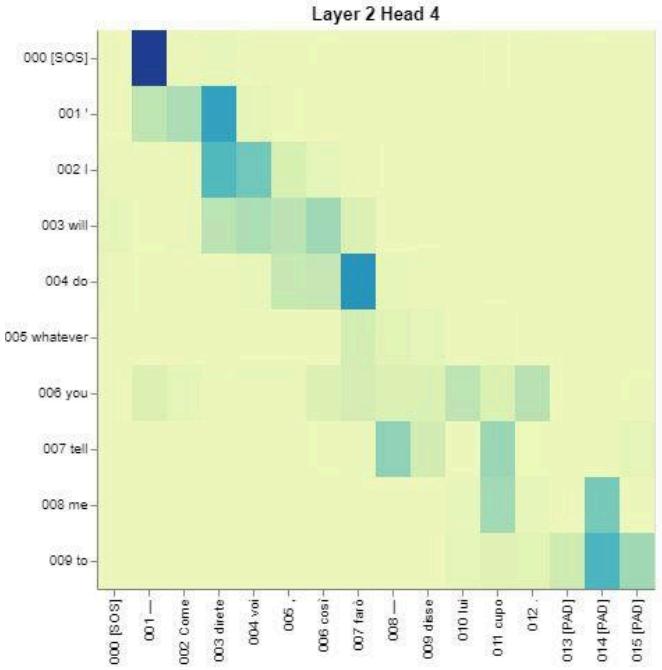
En regardant cette heatmap de plus près, on constate que la diagonale est celle qui a le plus de scores élevé, ce qui rejoint ce qui a été dit précédement, on remarque aussi que le mot ‘faro’ a un grand score avec le mot ‘direte’ ce qui implique que les deux mots sont très reliés entre eux (car ce que la personne va faire dépend de ce que les autres vont lui dire de faire).

- Visualisation des attention de l’encodeur-decodeur



-Visualisation de l’attention encodeur-décodeur-

Comme pour les deux visualisations précédentes on remarque qu’il y a trois couches et 8 têtes par chouche, la 1ère couche ne capture pas grand chose, car les scores d’attention sont tous très faible, et que la 2ème est bien meilleure, et la 3ème couche est la meilleure couche des trois (les scores d’attention sont assez élevé) ce qui indique bien que le modèle apprend mieux à chaque couche, on remarque aussi l’ajout de paddings à la phrase en italien(celle-ci étant bien plus courte que celle en anglais).



En examinant cette heatmap de plus près on remarque que le modèle a bien réussi à capturer une relation correcte, avec le score d'attention de 0.645 à peu près entre le mot 'do' en aglais (langage source) et 'faré' en italien (langage cible) et qui est en effet la traduction correcte.

Conclusion:

En résumé, le Transformer représente une percée majeure dans le domaine du Traitement du Langage Naturel, redéfinissant les normes et ouvrant la voie à une nouvelle génération de modèles. Ces derniers continuent de progresser, démontrant leur efficacité dans la résolution de problèmes divers. Cependant, il est crucial de reconnaître les défis associés à cette architecture, notamment ses exigences en termes de performance computationnelle. Cette observation a été particulièrement évidente au cours de notre mise en œuvre pratique, mettant en lumière la nécessité de ressources informatiques importantes pour optimiser pleinement les capacités du Transformer et pouvoir assurer un entraînement avec une quantité de données assez importante qui améliorera ses performances .
Parallèlement, malgré ces limitations, le Transformer élargit constamment son empreinte, étendant son influence à des domaines tels que la vision par ordinateur et le traitement audio. Cette diversification souligne son adaptabilité et son potentiel transversal, suggérant que son impact va au-delà du NLP. Ainsi, bien que des obstacles subsistent, le Transformer continue de démontrer son rôle crucial dans l'évolution des paradigmes informatiques, promettant des avancées continues et une influence soutenue dans un éventail toujours croissant de domaines technologiques.

Références :

- https://jalammar.github.io/illustrated-transformer/
- https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/
- Christopher M. Bishop, Hugh Bishop - Deep Learning_Foundations and Concepts-Springer (2024)
- https://papers.neurips.cc/paper/7181-attention-is-all-you-need.pdf
- Deep Dive into the Positional Encodings of the Transformer Neural Network Architecture: With Code! (linkedin.com)
- Master Positional Encoding: Part I | by Jonathan Kernes | Towards Data Science
- Video on transnformers
- Word2Vec Explained (israelg99.github.io)