TRANSFORMERS

SOMMAIRE

Nous avons précédemment présenté les réseaux convolutifs, qui sont spécialisés dans le traitement des données qui se trouvent sur une grille régulière. Ils sont particulièrement adaptés au traitement des images, qui comportent un très grand nombre de variables d'entrée, ce qui exclut l'utilisation de réseaux entièrement connectés. Chaque couche d'un réseau convolutif utilise le partage des paramètres de manière à ce que les zones locales de l'image soient traitées de la même manière à chaque position dans l'image.
Nous introduisons ici les transformers, initialement destinés aux problèmes de traitement des langues naturelles, où l'entrée du réseau est une série d'encodages en grande dimension représentant des mots ou des fragments de mots. Les ensembles de données linguistiques partagent certaines des caractéristiques des données d'image. Le nombre de variables d'entrée peut être très important et les statistiques sont similaires à chaque position ; il n'est pas judicieux de réapprendre la signification du mot "maison" à chaque position

possible dans un corps de texte. Cependant, les ensembles de données linguistiques présentent la complication suivante : les séquences de texte varient en longueur et, contrairement aux images, il n'existe pas de moyen facile de les redimensionner.

1- Motivation

Pour motiver l'utilisation de transformers, considérons le texte suivant :

"Le restaurant a refusé de me servir un sandwich au jambon parce qu'il ne cuisine que des plats végétariens. Finalement, ils m'ont donné deux tranches de pain. L'ambiance était tout aussi bonne que la nourriture et le service."

L'objectif est de concevoir un réseau pour traiter ce texte dans une représentation adaptée aux tâches ultérieures. Par exemple, il pourrait être utilisé pour classer l'avis comme positif ou négatif ou pour répondre à des questions telles que "Le restaurant sert-il de la viande ?
Trois observations immédiates peuvent être effectuées :

1. l'entrée codée peut être très grande. Dans ce cas, chacun des 41 mots pourrait être représenté par un vecteur de longueur 1024, de sorte que l'entrée codée serait de longueur 41×1024=41984, même pour ce petit passage. Un corps de texte de taille plus réaliste peut comporter des centaines, voire des milliers de mots, de sorte que les réseaux complètement connectés ne sont pas utilisables en pratique.
2. l'une des caractéristiques des problèmes de reconnaissance automatique du langage est que chaque entrée (une ou plusieurs phrases) est de longueur différente ; il n'est donc même pas évident d'appliquer un réseau complètement connecté. Ces observations suggèrent que le réseau devrait partager des paramètres entre les mots à différentes positions d'entrée, de la même manière que les réseaux convolutifs partagent des paramètres entre différentes positions d'image.
3. le langage est ambigu : la syntaxe seule ne permet pas de savoir si le pronom "il" fait référence au restaurant ou au sandwich au jambon. Pour comprendre le texte, le mot "il" doit être relié d'une manière ou d'une autre au mot "restaurant". Dans le langage des transformers, le premier mot doit prêter attention au second. Cela implique qu'il doit y avoir des liens entre les mots et que la force de ces liens dépend des mots eux-mêmes. En outre, ces liens doivent s'étendre sur de grandes parties du texte. Par exemple, le mot "L'" dans la dernière phrase fait également référence au restaurant.

2- Mécanisme d'attention

2.1- Mise en place

Un modèle de traitement de texte (i) utilise le partage des paramètres pour traiter les longs passages d'entrée de différentes longueurs et (ii) contient des connexions entre les représentations de mots qui dépendent des mots eux-mêmes. Le transformer acquiert ces deux propriétés en utilisant un mécanisme d'(auto-)attention. Une couche standard d'un réseau de neurones prend en entrée un vecteur $\mathbf{x} \in \mathbb{R}^d$ et calcule une sortie du type

$$f(\mathbf{x}) = ReLU(\mathbf{w}^T\mathbf{x} + b)$$

Un bloc d'attention $A()$ prend N entrées $\mathbf{x}_1, \cdots \mathbf{x}_N$ de taille $\mathbb{R}^d$ et renvoie N vecteurs de sortie de la même taille. Dans le contexte du traitement automatique du langage, chaque entrée représente un mot ou un fragment de mot. Tout d'abord, un ensemble de vecteurs est calculé pour chaque entrée

$$\forall i \in [\![1,N]\!]\ \mathbf{v}_i = \mathbf{W}_v^T\mathbf{x}_i + \mathbf{b}_v, \mathbf{b}_v \in \mathbb{R}^d, \mathbf{W}_v \in \mathcal{M}_d(\mathbb{R})$$

Les poids et biais $\mathbf{W}_v, \mathbf{b}_v$ sont les mêmes pour toutes les entrées. Puis les vecteurs de sortie sont calculés par

$$\forall j \in [\![1,N]\!]\ \mathbf{A}_j(\mathbf{x}_1, \cdots \mathbf{x}_N) = \sum_{i=1}^{N} a(\mathbf{x}_i, \mathbf{x}_j)\mathbf{v}_i$$

le scalaire $a(\mathbf{x}_i, \mathbf{x}_j)$ étant l'attention que la j-ième sortie accorde à l'entrée x_i. Les N valeurs $a(\bullet, \mathbf{x}_j)$ sont positifs et de somme unité.
Pour calculer l'attention, on définit deux transformations linéaires des entrées :

$$\begin{aligned} \forall i \in [\![1, N]\!] \; \mathbf{q}_i &= \mathbf{W}_q^T \mathbf{x}_i + \mathbf{b}_q \\ \mathbf{k}_i &= \mathbf{W}_k^T \mathbf{x}_i + \mathbf{b}_k \end{aligned}$$

où les $\mathbf{q}_i$ (respectivement $\mathbf{k}_i$) sont les requêtes (resp. clés). Ces dénominations proviennent de la théorie des bases de données.
Un produit scalaire entre requêtes et clés est alors effectué, et passé à un softmax (pour assurer la positivité et la somme à un)

$$a(\mathbf{x}_i, \mathbf{x}_j) = \frac{exp(\mathbf{k}_i^T \mathbf{q}_j)}{\sum_{l=1}^{N} exp(\mathbf{k}_l^T \mathbf{q}_j)}$$

Le prodiut scalaire entre requête et clé donne une mesure de similarité entre ces deux entités, et l'attention $a(\bullet, \mathbf{x}_j)$ dépend donc de la similarité entre q_j et toutes les clés.
Cette approche permet d'avoir un jeu de paramètres partagé entre toutes les entrées ($\mathbf{W}_v, \mathbf{b}_v, \mathbf{W}_q, \mathbf{b}_q, \mathbf{W}_k, \mathbf{b}_k$), indépendant de N et le réseau correspondant peut être appliqué à des entrées de longueur quelconque.

2.2- Formulation matricielle

En formant la matrice $\mathbf{X} \in \mathcal{M}_{d,N}(\mathbb{R})$ dont les colonnes sont les $\mathbf{x}_i$, alors le mécanisme d'attention peut s'écrire :

$$\begin{aligned} \mathbf{V}(\mathbf{X}) &= \mathbf{b}_v \mathbf{1}^T + \mathbf{W}_v \mathbf{X} \\ \mathbf{Q}(\mathbf{X}) &= \mathbf{b}_q \mathbf{1}^T + \mathbf{W}_q \mathbf{X} \\ \mathbf{K}(\mathbf{X}) &= \mathbf{b}_k \mathbf{1}^T + \mathbf{W}_k \mathbf{X} \\ \mathbf{A}(\mathbf{X}) &= \mathbf{V}(\mathbf{X}).Softmax(\mathbf{K}(\mathbf{X})^T \mathbf{Q}(\mathbf{X})) \end{aligned}$$

$Softmax$ appliquant des fonctions softmax indépendantes sur chaque colonne de la matrice argument.

2.3- Extensions

Le mécanisme précédent, dit d'auto-attention (self attention) eest décliné en plusieurs variantes très utilisées en pratique.

2.3.1- Encodage positionnel

Le mécanisme d'auto-attention ne tient pas compte d'une information importante : le calcul est le même quel que soit l'ordre des entrées $\mathbf{x}_i$. Plus précisément, il est équivariant par rapport aux permutations des entrées. Cependant, l'ordre est important lorsque les entrées correspondent aux mots d'une phrase. Il existe alors deux approches principales pour intégrer les informations de position :

1. l'encodage positionnel absolu : une matrice encodant l'information positionnelle $\mathbf{P}$ est ajoutée aux entrées $\mathbf{X}$. Chaque colonne de $\mathbf{P}$ est unique et contient une information de position de l'entrée correspondante. $\mathbf{P}$ peut être apprise ou fixée.
2. l'encodage positionnel absolu : l'entrée d'un mécanisme d'auto-attention peut être une phrase entière, plusieurs phrases ou un simple fragment de phrase, et la position absolue d'un mot est beaucoup moins importante que la position relative entre deux entrées. Si le système connaît la position absolue des deux entrées, la position relative peut être déterminée, mais les codages positionnels relatifs encodent directement cette information. Chaque élément de la matrice d'attention correspond à un décalage particulier entre la position pq de la requête et la position pk de la clé. Les codages positionnels relatifs apprennent un paramètre $\pi_{pq,pk}$ pour chaque décalage et l'utilisent pour modifier

la matrice d'attention en ajoutant ces valeurs, en les multipliant ou en les utilisant pour modifier la matrice d'attention d'une autre manière.

2.3.2- Auto attention par produit scalaire mis à l'échelle

Les produits scalaires dans le calcul de l'attention peuvent avoir de grandes amplitudes et déplacer les arguments de la fonction softmax dans une région où la plus grande valeur domine. De petites modifications des entrées de la fonction softmax ont désormais peu d'effet sur la sortie (les gradients sont très faibles), ce qui rend le modèle difficile à entraîner. Pour éviter cet inconvénient, les produits scalaires sont mis à l'échelle par la racine carrée de la dimension d_q des requêtes et des clés :

$$A(\mathbf{X}) = \mathbf{V}(\mathbf{X}).Softmax(\frac{\mathbf{K}(\mathbf{X})^T\mathbf{Q}(\mathbf{X})}{\sqrt{d_q}})$$

2.3.3- Mécanisme d'auto attention multiple

H ensembles de requêtes et clés sont calculés

$$\begin{aligned}
\mathbf{V}_h(\mathbf{X}) &= \mathbf{b}_{vh}\mathbf{1}^T + \mathbf{W}_{vh}\mathbf{X} \\
\mathbf{Q}_h(\mathbf{X}) &= \mathbf{b}_{qh}\mathbf{1}^T + \mathbf{W}_{qh}\mathbf{X} \\
\mathbf{H}_h(\mathbf{X}) &= \mathbf{b}_{kh}\mathbf{1}^T + \mathbf{W}_{kh}\mathbf{X} \\
\mathbf{A}_h(\mathbf{X}) &= \mathbf{V}_h(\mathbf{X}).Softmax(\frac{\mathbf{K}_h(\mathbf{X})^T\mathbf{Q}_h(\mathbf{X})}{\sqrt{d_q}})
\end{aligned}$$

$\mathbf{A}_h(\mathbf{X})$ est le h-ième mécanisme d'attention ou tête (head). Typiquement, si la dimension des entrées $\mathbf{x}_i$ est d et qu'il y a H têtes, les valeurs, les requêtes et les clés seront toutes de taille d/H pour une implémentation efficiente. Les sorties de ces mécanismes d'auto-attention sont concaténées verticalement, et une autre transformation linéaire $\mathbf{W}_c$ est appliquée pour les combiner

$$M_h\mathbf{A}(X) = \mathbf{W}_c[\mathbf{A}_1(X)^T \cdots \mathbf{A}_H(X)^T]^T$$

Les têtes multiples semblent être nécessaires au bon fonctionnement du transformer, on pense qu'elles rendent le réseau d'auto-attention plus résistant aux mauvaises initialisations.

2.4- Transformers

L'auto-attention n'est qu'une partie d'un mécanisme plus large : les transformers. Celui-ci se compose d'une unité d'auto-attention à plusieurs têtes (qui permet aux représentations de mots d'interagir les unes avec les autres) suivie d'un perceptron multicouches *PMC* qui opère séparément sur chaque mot. Les deux unités sont des réseaux résiduels (leur sortie est ajoutée à l'entrée d'origine). En outre, il est courant d'ajouter une opération de normalisation de couche *LayerNorm* après les réseaux d'auto-attention et les perceptrons multicouches. La séquence d'opérations complète peut être décrite par

$$\begin{aligned}
\mathbf{X} &= \mathbf{X} + M_h\mathbf{A}(X) \\
\mathbf{X} &= LayerNorm(\mathbf{X}) \\
\mathbf{x}_i &= \mathbf{x}_i + PMC(\mathbf{x}_i),\ i \in [\![1,N]\!] \\
\mathbf{X} &= LayerNorm(\mathbf{X})
\end{aligned}$$

En pratique, les données passent par plusieurs de ces transformers.

3- Les transformers en traitement automatique du langage (TAL/NLP)

Une chaîne classique de traitement en classique en NLP commence par un tokenizer qui divise le texte en mots ou en fragments de mots. Chacun de ces tokens est ensuite mis en correspondance avec une

représentation apprise. Ces représentations passent par une série de transformers.

3.1- Tokenizer

Le texte est tout d'abord divisé en unités constitutives plus petites (jetons ou tokens) à partir d'un vocabulaire de jetons possibles. Ces jetons ne représentent pas nécessairement ces mots car :

- inévitablement, certains mots (par exemple, des noms propres) ne figureront pas dans le vocabulaire.
- la manière de gérer la ponctuation n'est pas claire, mais elle est importante. Si une phrase se termine par un point d'interrogation, il faut encoder cette information.
- Le vocabulaire aurait besoin de différents jetons pour les versions d'un même mot avec des suffixes différents (par exemple, marche, marches, marché, marchés), et il n'y a aucun moyen de clarifier que ces variations sont liées.

Une approche consisterait à utiliser les lettres et les signes de ponctuation comme vocabulaire, mais cela impliquerait de découper le texte en très petites parties et d'exiger du réseau subséquent qu'il réapprenne les relations entre elles.
Dans la pratique, un compromis entre les lettres et les mots complets est utilisé, et le vocabulaire final comprend à la fois des mots courants et des fragments de mots à partir desquels des mots plus grands et moins fréquents peuvent être composés. Le vocabulaire est calculé à l'aide d'un tokeniseur de sous-mots tel que le codage par paires d'octets, qui fusionne de manière gloutonne les sous-chaînes les plus courantes en fonction de leur fréquence.

3.2- Représentation

Chaque jeton du vocabulaire V est associé à une représentation (embedding) de mot unique, et les représentations pour l'ensemble du vocabulaire sont stockées dans une matrice $\mathbf{W}_e \in \mathcal{M}_{d,|V|}(\mathbb{R})$. Pour ce faire, les N jetons d'entrée sont d'abord encodés dans une matrice $\mathbf{T} \in \mathcal{M}_{|V|,N}(\mathbb{R})$, où la n-ième colonne correspond au n-ième jeton et est un vecteur one-hot (un vecteur où chaque entrée est zéro, sauf l'entrée correspondant au jeton, de valeur 1). Les représentations des entrées sont calculées sous la forme $\mathbf{X} = \mathbf{W}_e\mathbf{T}$ et $\mathbf{W}_e$ est appris comme n'importe quel autre paramètre du réseau. Une taille d typique est de 1024, et une taille totale de vocabulaire $|V|$ typique est de 30 000, donc ce modèle nécessite de nombreux paramètres à apprendre, avant même la mise en place des transformers.

3.3- Transformers

Enfin, la matrice X représentant le texte passe par une série de K transformers (transformer model). Il existe trois types de ces modèles, décrits dans les paragraphes suivants. Globalement, un encodeur transforme la représentation du texte en une représentation qui peut prendre en charge une variété de tâches. Un décodeur prédit le prochain jeton pour poursuivre le texte d'entrée. Les encodeurs-décodeurs sont utilisés dans les tâches de séquence à séquence, où une chaîne de texte est convertie en une autre (par exemple, traduction automatique).

3.3.1- Exemple de modèle à encodeur : BERT

BERT est un modèle d'encodeur qui utilise un vocabulaire de 30 000 mots. Les jetons d'entrée sont convertis en représentations de mots à 1024 dimensions et passent par K=24 transformers. Chacun d'eux contient un mécanisme d'auto-attention avec 16 têtes. Les requêtes, les clés et les valeurs de chaque tête sont de dimension 64. La dimension de la couche cachée unique dans le réseau complètement connecté du transformer est de 4096. Le nombre total de paramètres est de 340 millions. Lors de la publication de BERT, ce nombre était considéré comme élevé, mais il est aujourd'hui bien inférieur à celui des modèles les plus récents. Les modèles d'encodeurs comme BERT exploitent l'apprentissage par transfert. Pendant le préapprentissage, les paramètres de l'architecture du transformer sont appris par auto-supervision (pas besoin de labels) à partir d'un large corpus de texte. L'objectif est ici que le modèle apprenne des informations générales sur les statistiques de la langue. Au cours de la phase de fine tuning, le réseau résultant est adapté pour résoudre une tâche particulière à l'aide d'un plus petit corpus de données d'apprentissage supervisé.

3.3.2- Exemple de modèle à décodeur : GPT3

On présente ici une description de haut niveau de GPT3. L'architecture de base est très similaire à celle du modèle d'encodage et comprend une série de transformers qui opèrent sur les représentations de mots appris. Cependant, l'objectif est différent. L'encodeur vise à construire une représentation du texte qui peut être affinée pour résoudre une variété de tâches plus spécifiques. À l'inverse, le décodeur n'a qu'un seul objectif : générer le jeton suivant dans une séquence. Il peut générer un passage de texte cohérent en réinjectant la séquence étendue dans le modèle.

3.3.2-.1 Modélisation du langage : GPT3 construit un modèle linguistique autorégressif. Pour illustrer ce modèle, considérons la phrase $\mathcal{P}$ = "Henri mange beaucoup de viande le soir". Pour simplifier, supposons que les jetons sont les mots complets. La probabilité de la phrase complète est :

$$P(\mathcal{P}) = P(\text{Henri})P(\text{mange}|\text{Henri})P(\text{beaucoup}|\text{Henri mange})\cdots P(\text{soir}|\text{Henri mange beaucoup de viande le })$$

3.3.2-.2 auto-attention masquée : pour entraîner un décodeur, on maximise la log-probabilité du texte d'entrée dans le cadre du modèle autorégressif. L'idéal serait de transmettre la phrase entière et de calculer simultanément toutes les log-probabilités et tous les gradients. Cependant, cela pose un problème : si on transmet la phrase complète, le terme qui calcule $logP(\text{beaucoup}|\text{Henri mange})$ a accès à la fois à la réponse attendue "beaucoup" et au contexte suivant "de viande le soir". Par conséquent, le système peut tricher au lieu d'apprendre à prédire les mots suivants et ne s'entraînera pas correctement. Fort heureusement, les jetons n'interagissent que dans les couches d'auto-attention d'un réseau transformer. Le problème peut donc être résolu en s'assurant que l'attention portée à la réponse et au contexte est nulle. Pour ce faire, les produits scalaires correspondants dans le calcul de l'auto-attention sont réglés à $-\infty$ avant d'être passés à la fonction softmax. C'est le principe de l'auto-attention masquée. L'ensemble du réseau du décodeur fonctionne comme suit. Le texte d'entrée est encodé en jetons, et les jetons sont convertis en embeddings. Ces derniers sont transmis au réseau de transformers, qui utilisent l'auto-attention masquée, de sorte qu'ils ne peuvent s'intéresser qu'aux jetons actuels et précédents. Chacune des représentations de sortie peut être considérée comme représentant une phrase partielle, et pour chacune d'entre elles, l'objectif est de prédire le jeton suivant dans la séquence.
Après les transformers, une couche linéaire fait correspondre chaque représentation de mot à la taille du vocabulaire, suivie d'une fonction softmax qui convertit ces valeurs en probabilités. Pendant l'apprentissage, on cherche à maximiser la somme des log-probabilités de l'élément suivant dans la séquence de référence à chaque position en utilisant une fonction de perte d'entropie croisée multiclasse standard.

3.3.2-.3 Génération de texte : puisque ce modèle définit un modèle de probabilité sur des séquences de texte, il peut être utilisé pour échantillonner de nouveaux exemples de texte plausibles. Pour générer à partir du modèle, on débute par une séquence de texte en entrée (qui peut être simplement un jeton spécial <start> indiquant le début de la séquence) et on l'introduit dans le réseau, qui produit alors les probabilités sur les jetons suivants possibles. On peut alors choisir le jeton le plus probable ou échantillonner à partir de la distribution de probabilités construite. La nouvelle séquence étendue peut être réinjectée dans le réseau du décodeur qui fournit la distribution de probabilités sur le jeton suivant. En répétant ce processus, on génère un texte entier.
En pratique, de nombreuses stratégies peuvent rendre le texte de sortie plus cohérent. Par exemple, la recherche par faisceau tient compte des nombreux compléments de phrases possibles pour trouver le plus probable (qui n'est pas nécessairement trouvé en choisissant de manière gloutonne le mot suivant le plus probable à chaque étape). L'échantillonnage top-k tire aléatoirement le mot suivant parmi les k possibilités les plus probables afin d'éviter que le système ne choisisse accidentellement dans la longue traîne des jetons à faible probabilité, ce qui conduirait à une impasse linguistique.
GPT3 est un exemple de LLM (Large Language Model). La longueur des séquences est de 2048 jetons. Il y a K=96 transformers, chacun calculant une représentation de taille 12288. Dans les couches d'auto attention, on compte 96 têtes et la dimension des requêtes et des clés est de 128. Tout celà amène à un nombre de paramètres de 175 milliards, entraînés sur 300 milliards de jetons (taille d'un batch : 3.2 million de jetons)

3.3.3- Exemple de modèle à encodeur-décodeur : traduction automatique

La traduction entre langues est un exemple de tâche de séquence à séquence. Cette tâche nécessite un encodeur (pour calculer une bonne représentation de la phrase source) et un décodeur (pour générer la phrase dans la langue cible). Cette tâche peut être abordée à l'aide d'un modèle encodeur-décodeur. Prenons l'exemple d'une traduction de l'anglais vers le français. L'encodeur reçoit la phrase en anglais et la traite à travers une série de transformers pour créer une représentation de sortie pour chaque jeton. Pendant l'entraînement, le décodeur reçoit la traduction de référence en français et la fait passer par une série de transformers qui utilisent l'auto-attention masquée et prédisent le mot suivant à chaque position.
Cependant, les couches du décodeur s'occupent également de la sortie de l'encodeur. Par conséquent, chaque mot français en sortie est conditionné par les mots précédents en sortie et par l'ensemble de la phrase anglaise qu'il traduit. Pour ce faire, on modifie les transformers du décodeur. Le transformer original du décodeur consistait en une couche d'auto-attention masquée suivie d'un réseau appliqué individuellement à chaque représentation. Une nouvelle couche d'auto-attention est alors ajoutée entre ces deux composants, dans laquelle les représentations du décodeur s'intéressent aux représentations de l'encodeur. Cette méthode utilise une version de l'auto-attention connue sous le nom d'attention encodeur-décodeur ou d'attention croisée, où les requêtes sont calculées à partir des représentations du décodeur et les clés et valeurs à partir des représentations de l'encodeur.

4- LES TRANSFORMERS EN TRAITEMENT D'IMAGES

Le succès des transformers en TAL a conduit à se pencher sur leur utilisation sur des images. L'idée semblait incongrue, et ce pour deux raisons : il y a beaucoup plus de pixels dans une image que de mots dans une phrase, de sorte que la complexité quadratique de l'auto-attention constitue un goulot d'étranglement pratique. De plus, les réseaux convolutifs ont un bon biais inductif parce que chaque couche est équivariante à la translation spatiale et qu'ils prennent en compte la structure 2D de l'image. Ce qu'il faudrait apprendre dans un réseau transformer. Malgré cela, les réseaux transformers ont désormais éclipsé les performances des réseaux convolutifs pour entre autres la classification d'images. Cela s'explique en partie par l'échelle à laquelle ils peuvent être construit et par les grandes quantités de données qui peuvent être utilisées pour pré-entraîner les réseaux.

4.1- ImageGPT

ImageGPT est un décodeur. Il construit un modèle autorégressif de pixels qui assimile une image partielle et prédit la valeur du pixel suivant. La complexité quadratique du réseau de transformers signifie que le plus grand modèle (qui contient 6,8 milliards de paramètres) ne peut fonctionner que sur des images de taille 64×64. En outre, pour que le système soit viable, l'espace colorimétrique RVB original de 24 bits a dû originellement être quantifié en un espace colorimétrique de neuf bits, de sorte que le système assimile (et prédit) l'un des 512 jetons possibles à chaque position. Les images sont naturellement des objets 2D, mais ImageGPT apprend simplement un codage positionnel différent pour chaque pixel. Il doit donc apprendre que chaque pixel a une relation étroite avec ses voisins précédents ainsi qu'avec les pixels voisins de la rangée supérieure.
La représentation interne de ce décodeur a été utilisée comme base pour la classification des images. Les représentations des pixels finales sont moyennées et une couche linéaire les met en correspondance avec des activations qui passent par une couche softmax pour prédire les probabilités de classe. Le système est pré-entraîné sur un large corpus d'images web, puis affiné sur la base de données ImageNet redimensionnée à 48×48 pixels à l'aide d'une fonction de perte qui contient à la fois un terme d'entropie croisée pour la classification des images et un terme de perte générative pour la prédiction des pixels. Malgré l'utilisation d'une grande quantité de données d'apprentissage externes, le système a obtenu un taux d'erreur de 27,4% sur ImageNet. Ce taux est inférieur à celui des architectures convolutives de l'époque, mais reste impressionnant compte tenu de la petite taille de l'image d'entrée ; sans surprise, il ne parvient pas à classer les images où l'objet cible est petit ou mince.

4.2- ViT : Vision Transformer

ViT s'est attaqué au problème de la résolution de l'image en divisant l'image en patchs de 16×16 . Chaque patch est mis en correspondance avec une dimension inférieure par le biais d'une transformation linéaire apprise, et ces représentations sont introduites dans le réseau du transformer. Les codages positionnels 1D standard sont appris. Il s'agit d'un modèle encodeur avec un jeton <cls>. Cependant, contrairement à BERT, il utilise un pré-entraînement supervisé sur une grande base de données de 303 millions d'images étiquetées provenant de 18 000 classes. Le jeton <cls> est mappé via une couche finale du réseau pour créer des activations qui sont introduites dans une fonction softmax pour générer des probabilités de classe. Après le pré-entraînement, le système est appliqué en classification en remplaçant la couche finale par une couche qui correspond au nombre de classes souhaité et qui est ajustée par fine tuning. Ce système a obtenu un taux d'erreur de 11,45% pour le top 1 sur ImageNet. Cependant, il n'a pas obtenu d'aussi bons résultats que les meilleurs réseaux convolutifs contemporains sans pré-entraînement supervisé. Le fort biais inductif des réseaux convolutifs ne peut être surmonté que que par l'utilisation de quantités extrêmement importantes de données d'entraînement.

5- PARTIE PRATIQUE

Vu les temps d'entraînement et les bases de données nécessaires, il est impossible de proposer de faire tourner un transformer pendant cette séance. Bien que la couche `tfm.nlp.layers.TransformerEncoderBlock` existe dans la version 2.13 de Tensorflow, on se propose donc ici d'implémenter un Transformer de bout en bout, et plus particulièrement le travail séminal sur ce sujet [1] (figure 5-1)

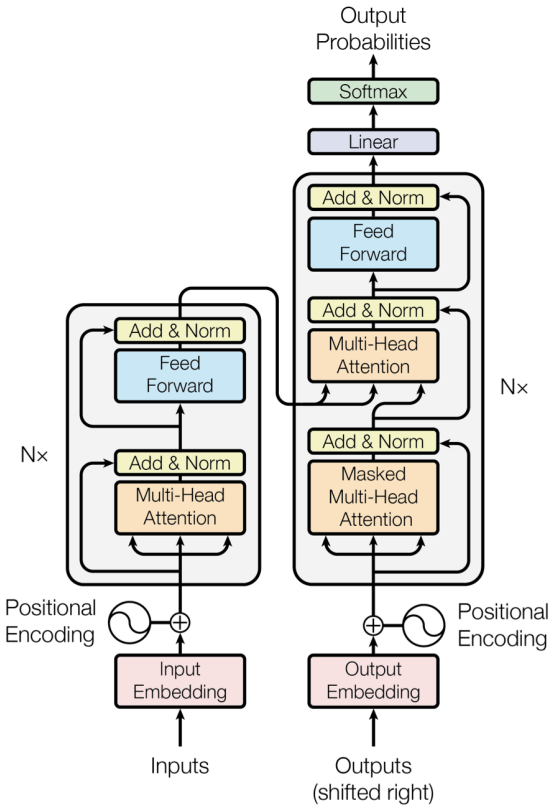

FIGURE 5-1 – Architecture proposée dans [1]

Vous avez à disposition un certain nombre d'éléments constitutifs de cette architecture, d'autres sont à

écrire, l'objectif étant de déterminer le nombre total de paramètres avec l'instantiation suivante (issue de l'article) :

```
N=6, H=8, dim_e=512, dq=64, dv=64, dim_h=2048, vocab_size=29, T = 11, batch_size = 3
```

BIBLIOGRAPHIE

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 30. Curran Associates, Inc., 2017.