Provided proper attribution is provided, Google hereby grants permission to reproduce the tables and figures in this paper solely for use in journalistic or scholarly works.

Attention Is All You Need

Ashish Vaswani[*]
Google Brain
avaswani@google.com

Noam Shazeer[*]
Google Brain
noam@google.com

Niki Parmar[*]
Google Research
nikip@google.com

Jakob Uszkoreit[*]
Google Research
usz@google.com

Llion Jones[*]
Google Research
llion@google.com

Aidan N. Gomez[*] [†]
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser[*]
Google Brain
lukaszkaiser@google.com

Illia Polosukhin[*] [‡]
illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

[*]Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

[†]Work performed while at Google Brain.

[‡]Work performed while at Google Research.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

arXiv:1706.03762v7 [cs.CL] 2 Aug 2023

Sous réserve d'une attribution appropriée, Google accorde par la présente la permission de reproduire les tableaux et les figures de ce document uniquement à des fins d'utilisation dans des articles journalistiques ou des travaux académiques.

L'attention est tout ce dont vous avez besoin

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer *
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
Université de Toronto
aidan@cs.toronto.edu

Lukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Illia Polosukhin*
illia.polosukhin@gmail.com

Résumé

Les modèles dominants de transduction de séquence reposent sur des réseaux de neurones récurrents ou convolutifs complexes comportant un encodeur et un décodeur. Les modèles les plus performants relient également l'encodeur et le décodeur par le biais d'un mécanisme d'attention. Nous proposons une nouvelle architecture de réseau simple, le Transformateur, basé exclusivement sur des mécanismes d'attention, éliminant totalement la récurrence et les convolutions. Des expériences menées sur deux tâches de traduction automatique montrent que ces modèles surpassent en qualité tout en étant plus parallélisables et nécessitant nettement moins de temps d'entraînement. Notre modèle atteint 28,4 BLEU sur la tâche de traduction anglais-allemand WMT 2014, améliorant ainsi les meilleurs résultats existants, y compris les ensembles, de plus de 2 BLEU. Sur la tâche de traduction anglais-français WMT 2014, notre modèle établit un nouveau score record BLEU pour un modèle unique de 41,8 après 3,5 jours d'entraînement sur huit GPU, soit une fraction minime des coûts d'entraînement des meilleurs modèles de la littérature. Nous démontrons que le Transformateur se généralise efficacement à d'autres tâches en l'appliquant avec succès au parsement de constituency en anglais, tant avec des données d'entraînement abondantes que limitées.

*Contribution égale. L'ordre d'énumération est aléatoire. Jakob a proposé de remplacer les RNN par l'auto-attention et a initié les travaux visant à évaluer cette idée. Ashish, avec Illia, a conçu et mis en œuvre les premiers modèles Transformer et a été crucialement impliqué dans chaque aspect de ce travail. Noam a proposé l'attention à produit scalaire mis à l'échelle, l'attention multi-têtes et la représentation positionnelle sans paramètre, devenant ainsi l'autre personne impliquée dans presque chaque détail. Niki a conçu, mis en œuvre, affiné et évalué d'innombrables variantes de modèles dans notre codebase originale et tensor2tensor. Llion a également expérimenté avec de nouvelles variantes de modèles, était responsable de notre codebase initiale, et de l'inférence et des visualisations efficaces. Lukasz et Aidan ont passé d'interminables journées à concevoir diverses parties et à implémenter tensor2tensor, remplaçant notre codebase précédente, améliorant considérablement les résultats et accélérant massivement nos recherches.

[1]Travail effectué au sein de Google Brain.

Travail effectué lors d'un séjour au sein de Google Research.

31e Conférence sur les Systèmes de Traitement de l'Information par les Réseaux Neuraux (NIPS 2017), Long Beach, CA, USA

arXiv:1706.03762v7 [cs.CL] 2 Aug 2023

1 Introduction

Recurrent neural networks, long short-term memory [13] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and transduction problems such as language modeling and machine translation [35, 2, 5]. Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures [38, 24, 15].

Recurrent models typically factor computation along the symbol positions of the input and output sequences. Aligning the positions to steps in computation time, they generate a sequence of hidden states h_t, as a function of the previous hidden state h_{t-1} and the input for position t. This inherently sequential nature precludes parallelization within training examples, which becomes critical at longer sequence lengths, as memory constraints limit batching across examples. Recent work has achieved significant improvements in computational efficiency through factorization tricks [21] and conditional computation [32], while also improving model performance in case of the latter. The fundamental constraint of sequential computation, however, remains.

Attention mechanisms have become an integral part of compelling sequence modeling and transduction models in various tasks, allowing modeling of dependencies without regard to their distance in the input or output sequences [2, 19]. In all but a few cases [27], however, such attention mechanisms are used in conjunction with a recurrent network.

In this work we propose the Transformer, a model architecture eschewing recurrence and instead relying entirely on an attention mechanism to draw global dependencies between input and output. The Transformer allows for significantly more parallelization and can reach a new state of the art in translation quality after being trained for as little as twelve hours on eight P100 GPUs.

2 Background

The goal of reducing sequential computation also forms the foundation of the Extended Neural GPU [16], ByteNet [18] and ConvS2S [9], all of which use convolutional neural networks as basic building block, computing hidden representations in parallel for all input and output positions. In these models, the number of operations required to relate signals from two arbitrary input or output positions grows in the distance between positions, linearly for ConvS2S and logarithmically for ByteNet. This makes it more difficult to learn dependencies between distant positions [12]. In the Transformer this is reduced to a constant number of operations, albeit at the cost of reduced effective resolution due to averaging attention-weighted positions, an effect we counteract with Multi-Head Attention as described in section 3.2.

Self-attention, sometimes called intra-attention is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence. Self-attention has been used successfully in a variety of tasks including reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representations [4, 27, 28, 22].

End-to-end memory networks are based on a recurrent attention mechanism instead of sequence-aligned recurrence and have been shown to perform well on simple-language question answering and language modeling tasks [34].

To the best of our knowledge, however, the Transformer is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence-aligned RNNs or convolution. In the following sections, we will describe the Transformer, motivate self-attention and discuss its advantages over models such as [17, 18] and [9].

3 Model Architecture

Most competitive neural sequence transduction models have an encoder-decoder structure [5, 2, 35]. Here, the encoder maps an input sequence of symbol representations $(x_1, ..., x_n)$ to a sequence of continuous representations $\mathbf{z} = (z_1, ..., z_n)$. Given $\mathbf{z}$, the decoder then generates an output sequence $(y_1, ..., y_m)$ of symbols one element at a time. At each step the model is auto-regressive [10], consuming the previously generated symbols as additional input when generating the next.

1 Introduction

Les réseaux de neurones récurrents, la mémoire à court et long terme [13] et, plus particulièrement, les réseaux de neurones récurrents à portes [7], se sont fermement établis comme des approches de pointe dans les problèmes de modélisation et de transduction de séquences, tels que la modélisation du langage et la traduction automatique [35, 2, 5]. De nombreux efforts ont depuis continué à repousser les limites des modèles de langage récurrents et des architectures encodeur-décodeur [38, 24, 15].

Les modèles récurrents factorisent typiquement le calcul le long des positions des symboles des séquences d'entrée et de sortie. En alignant ces positions sur des étapes dans le temps de calcul, ils génèrent une séquence d'états cachés h_t, en fonction de l'état caché précédent h_{t-1} et de l'entrée à la position t. Cette nature intrinsèquement séquentielle empêche la parallélisation au sein des exemples d'entraînement, ce qui devient critique pour les longueurs de séquence plus importantes, car les contraintes de mémoire limitent le regroupement (batching) entre les exemples. Des travaux récents ont obtenu des améliorations significatives de l'efficacité computationnelle grâce à des astuces de factorisation [21] et au calcul conditionnel [32], tout en améliorant également les performances du modèle dans le cas de ce dernier. La contrainte fondamentale du calcul séquentiel, however, reste.

Les mécanismes d'attention sont devenus une partie intégrante des modèles de modélisation et de transduction de séquences convaincants dans diverses tâches, permettant de modéliser les dépendances sans tenir compte de leur distance dans les séquences d'entrée ou de sortie [2, 19]. Cependant, dans tous les cas sauf quelques-uns [27], ces mécanismes d'attention sont utilisés en conjonction avec un réseau récurrent.

Dans ce travail, nous proposons le Transformer, une architecture de modèle éliminant la récurrence et s'appuyant entièrement sur un mécanisme d'attention pour établir des dépendances globales entre l'entrée et la sortie. Le Transformer permet une parallélisation nettement accrue et peut atteindre un nouvel état de l'art en matière de qualité de traduction après avoir été entraîné pendant seulement douze heures sur huit GPU P100.

2 Contexte

L'objectif de réduction du calcul séquentiel constitue également la base de l'Extended Neural GPU [16], ByteNet [18] et ConvS2S [9], tous utilisant des réseaux de neurones convolutifs comme bloc de construction fondamental, le calcul de représentations cachées en parallèle pour toutes les positions d'entrée et de sortie. Dans ces modèles, le nombre d'opérations nécessaires pour relier des signaux provenant de deux positions d'entrée ou de sortie arbitraires augmente avec la distance entre ces positions, linéairement pour ConvS2S et logarithmiquement pour ByteNet. Cela rend plus difficile l'apprentissage des dépendances entre des positions éloignées [12]. Dans le Transformateur, cela se réduit à un nombre constant d'opérations, bien que au prix d'une résolution effective réduite en raison de la moyenne des pondérations par attention, un effet que nous contrebalançons par l'Attention à Têtes Multiples tel que décrit dans la section 3.2.

L'auto-attention, parfois appelée attention intra-séquence, est un mécanisme d'attention reliant différentes positions d'une même séquence afin de calculer une représentation de celle-ci. L'auto-attention a été utilisée avec succès dans une variété de tâches incluant la compréhension écrite, la synthèse textuelle, l'inférence textuelle et l'apprentissage de représentations de phrases indépendantes de la tâche [4, 27, 28, 22].

Les réseaux de mémoire de bout en bout reposent sur un mécanisme d'attention récurrent au lieu d'une récurrence alignée sur les séquences et ont démontré de bonnes performances dans des tâches de réponse à des questions en langage simple ainsi que dans le modelage du langage [34].

À notre connaissance, le Transformer est toutefois le premier modèle de transduction s'appuyant exclusivement sur l'attention propre pour calculer les représentations de son entrée et de sa sortie sans recourir à des RNN alignés sur les séquences ou à la convolution. Dans les sections suivantes, nous décrivons le Transformer, justifions l'attention propre et discutons de ses avantages par rapport aux modèles tels que [17, 18] et [9].

3 Architecture du modèle

La plupart des modèles de transduction de séquences neurales compétitifs possèdent une structure encodeur-décodeur [5, 2, 35]. Ici, l'encodeur mape une séquence d'entrée de représentations symboliques $(x_1, \ldots, x_n)$ vers une séquence de représentations continues $\mathbf{z} = (z_1, \ldots, z_n)$. Étant donné $\mathbf{z}$, le décodeur génère ensuite une séquence de sortie $(y_1, \ldots, y_m)$ de symboles un élément à la fois. À chaque étape, le modèle est autoregressif [10], consommant les symboles précédemment générés comme entrée supplémentaire lors de la génération du suivant.

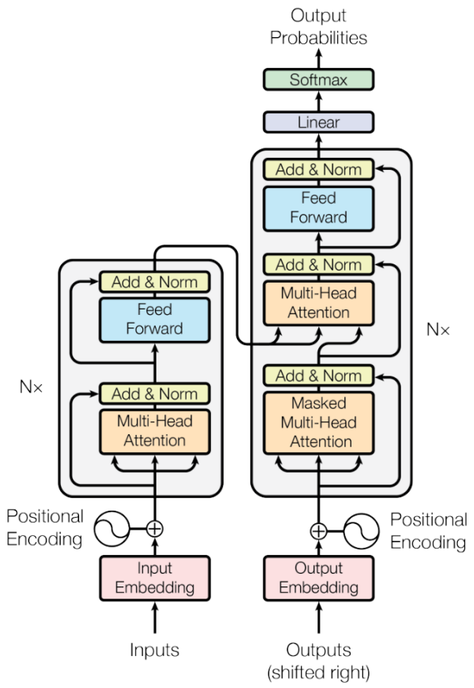


Figure 1: The Transformer - model architecture.

The Transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder, shown in the left and right halves of Figure 1, respectively.

3.1 Encoder and Decoder Stacks

Encoder: The encoder is composed of a stack of $N = 6$ identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. We employ a residual connection [11] around each of the two sub-layers, followed by layer normalization [1]. That is, the output of each sub-layer is $\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$, where $\mathrm{Sublayer}(x)$ is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension $d_{\text{model}} = 512$.

Decoder: The decoder is also composed of a stack of $N = 6$ identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization. We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i.

3.2 Attention

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum

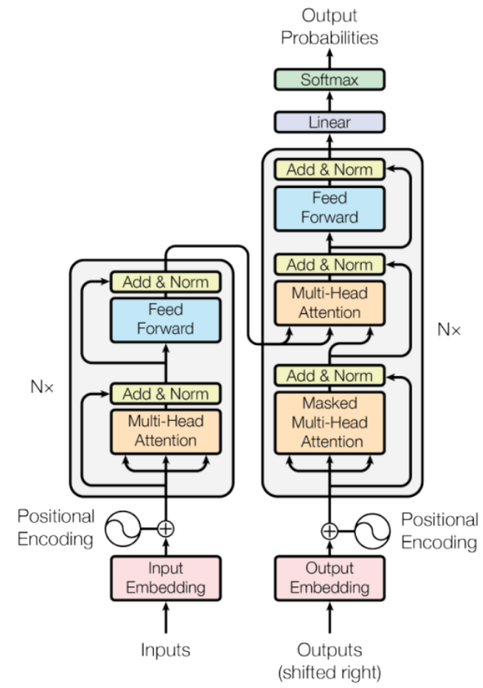


Figure 1 : L'architecture du modèle Transformer.

Le Transformer suit cette architecture globale en utilisant des couches d'auto-attention empilées et de connexion pleine, point par point, tant pour l'encodeur que pour le décodeur, représentés respectivement dans les moitiés gauche et droite de la Figure 1.

3.1 Empilements d'encodeur et de décodeur

Encodeur : L'encodeur est composé d'une pile de $N = 6$ couches identiques. Chaque couche possède deux sous-couches. La première est un mécanisme d'auto-attention à têtes multiples, et la seconde est un réseau feed-forward entièrement connecté simple, appliqué position par position. Nous employons une connexion résiduelle [11] autour de chacune des deux sous-couches, suivie d'une normalisation de couche [1]. Autrement dit, la sortie de chaque sous-couche est $\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$, où $\mathrm{Sublayer}(x)$ est la fonction mise en œuvre par la sous-couche elle-même. Pour faciliter ces connexions résiduelles, toutes les sous-couches du modèle, ainsi que les couches d'embedding, produisent des sorties de dimension $d_{\text{modèle}} = 512$.

Décodeur : Le décodeur est également composé d'une pile de $N = 6$ couches identiques. En plus des deux sous-couches de chaque couche du codeur, le décodeur insère une troisième sous-couche qui effectue une attention multi-têtes sur la sortie de la pile du codeur. Similaire au codeur, nous employons des connexions résiduelles autour de chacune des sous-couches, suivies d'une normalisation de couche. Nous modifions également la sous-couche d'auto-attention dans la pile du décodeur pour empêcher les positions d'attender à des positions ultérieures. Ce masquage, combiné au fait que les embeddings de sortie sont décalés d'une position, garantit que les prédictions pour la position i ne peuvent dépendre que des sorties connues aux positions inférieures à i.

3.2 Attention

Une fonction d'attention peut être décrite comme mappant une requête et un ensemble de paires clé-valeur vers une sortie, où la requête, les clés, les valeurs et la sortie sont tous des vecteurs. La sortie est calculée comme une somme pondérée

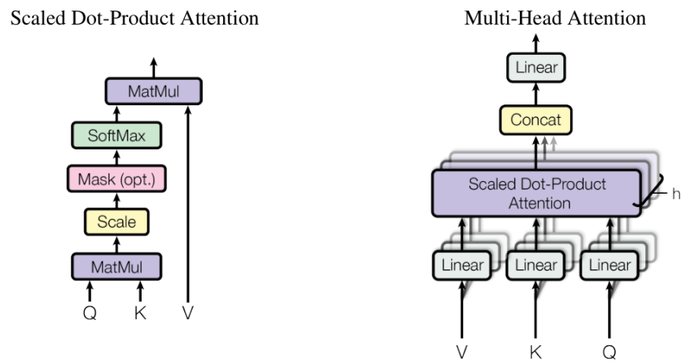


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

3.2.1 Scaled Dot-Product Attention

We call our particular attention "Scaled Dot-Product Attention" (Figure 2). The input consists of queries and keys of dimension d_k, and values of dimension d_v. We compute the dot products of the query with all keys, divide each by $\sqrt{d_k}$, and apply a softmax function to obtain the weights on the values.

In practice, we compute the attention function on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V. We compute the matrix of outputs as:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{1}$$

The two most commonly used attention functions are additive attention [2], and dot-product (multiplicative) attention. Dot-product attention is identical to our algorithm, except for the scaling factor of $\frac{1}{\sqrt{d_k}}$. Additive attention computes the compatibility function using a feed-forward network with a single hidden layer. While the two are similar in theoretical complexity, dot-product attention is much faster and more space-efficient in practice, since it can be implemented using highly optimized matrix multiplication code.

While for small values of d_k the two mechanisms perform similarly, additive attention outperforms dot product attention without scaling for larger values of d_k [3]. We suspect that for large values of d_k, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients [4]. To counteract this effect, we scale the dot products by $\frac{1}{\sqrt{d_k}}$.

3.2.2 Multi-Head Attention

Instead of performing a single attention function with d_{model}-dimensional keys, values and queries, we found it beneficial to linearly project the queries, keys and values h times with different, learned linear projections to d_k, d_k and d_v dimensions, respectively. On each of these projected versions of queries, keys and values we then perform the attention function in parallel, yielding d_v-dimensional

[4]To illustrate why the dot products get large, assume that the components of q and k are independent random variables with mean 0 and variance 1. Then their dot product, $q \cdot k = \sum_{i=1}^{d_k} q_i k_i$, has mean 0 and variance d_k.

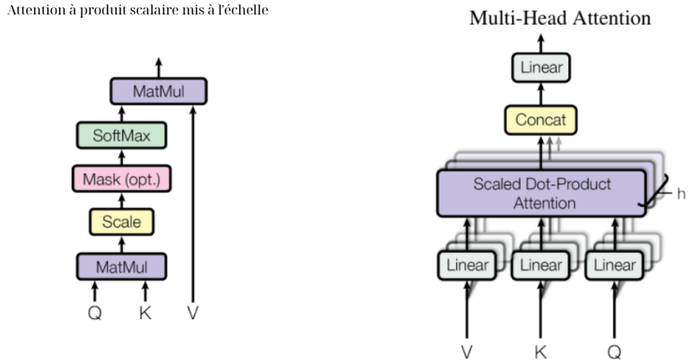


Figure 2 : (gauche) Attention à produit scalaire mis à l'échelle. (droite) L'attention à têtes multiples se compose de plusieurs couches d'attention s'exécutant en parallèle.

des valeurs, pour lesquelles le poids attribué à chaque valeur est calculé par une fonction de compatibilité de la requête avec la clé correspondante.

3.2.1 Attention à produit scalaire mis à l'échelle

Nous appelons notre attention particulière « Scaled Dot-Product Attention » (Figure 2). L'entrée se compose de requêtes et de clés de dimension d_k, et de valeurs de dimension d_v. Nous calculons les produits scalaires de la requête avec toutes les clés, divisons chacun par $\sqrt{d_k}$, et appliquons une fonction softmax pour obtenir les poids sur les valeurs.

En pratique, nous calculons la fonction d'attention sur un ensemble de requêtes simultanément, regroupées dans une matrice Q. Les clés et les valeurs sont également regroupées dans les matrices K et V. Nous calculons la matrice des sorties comme suit :

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{1}$$

Les deux fonctions d'attention les plus couramment utilisées sont l'attention additive [2] et l'attention par produit scalaire (multiplicative). L'attention par produit scalaire est identique à notre algorithme, à l'exception du facteur de mise à l'échelle de $\frac{1}{\sqrt{d_k}}$. L'attention additive calcule la fonction de compatibilité en utilisant un réseau à propagation directe comportant une seule couche cachée. Bien que ces deux approches présentent une complexité théorique similaire, l'attention par produit scalaire est beaucoup plus rapide et plus économe en espace en pratique, car elle peut être implémentée à l'aide de codes de multiplication matricielle hautement optimisés.

Bien que pour de petites valeurs de d_k les deux mécanismes performent de manière similaire, l'attention additive surpasse l'attention par produit scalaire sans mise à l'échelle pour de plus grandes valeurs de d_k [3]. Nous suspectons que pour de grandes valeurs de d_k, les produits scalaires augmentent considérablement en amplitude, poussant la fonction softmax vers des régions où elle présente des gradients extrêmement faibles 4. Pour contrer cet effet, nous mettons les produits scalaires à l'échelle en les divisant par $\frac{1}{\sqrt{d_k}}$.

3.2.2 Attention à têtes multiples

Au lieu d'effectuer une seule fonction d'attention avec des clés, des valeurs et des requêtes de dimension d_{model}, nous avons jugé utile de projeter linéairement les requêtes, les clés et les valeurs h fois à l'aide de projections linéaires apprises distinctes vers des dimensions d_k, d_k et d_v, respectivement. Sur chacune de ces versions projetées de requêtes, de clés et de valeurs, nous effectuons ensuite la fonction d'attention en parallèle, produisant une version de dimension d_v.

[4]Pour illustrer pourquoi les produits scalaires deviennent grands, supposons que les composantes de q et k soient des variables aléatoires indépendantes de moyenne 0 et de variance 1. Alors leur produit scalaire, $q \cdot k = \sum_{i=1} q_i k_i$, a pour moyenne 0 et pour variance d_k.

output values. These are concatenated and once again projected, resulting in the final values, as depicted in Figure 2.

Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this.

$$\begin{aligned}\text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O \\ \text{where head}_\text{i} &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned}$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_\text{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_\text{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_\text{model} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_\text{model}}$.

In this work we employ $h = 8$ parallel attention layers, or heads. For each of these we use $d_k = d_v = d_\text{model}/h = 64$. Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

3.2.3 Applications of Attention in our Model

The Transformer uses multi-head attention in three different ways:

- In "encoder-decoder attention" layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence. This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models such as [38, 2, 9].
- The encoder contains self-attention layers. In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder.
- Similarly, self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position. We need to prevent leftward information flow in the decoder to preserve the auto-regressive property. We implement this inside of scaled dot-product attention by masking out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections. See Figure 2.

3.3 Position-wise Feed-Forward Networks

In addition to attention sub-layers, each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2}$$

While the linear transformations are the same across different positions, they use different parameters from layer to layer. Another way of describing this is as two convolutions with kernel size 1. The dimensionality of input and output is $d_\text{model} = 512$, and the inner-layer has dimensionality $d_{ff} = 2048$.

3.4 Embeddings and Softmax

Similarly to other sequence transduction models, we use learned embeddings to convert the input tokens and output tokens to vectors of dimension d_model. We also use the usual learned linear transformation and softmax function to convert the decoder output to predicted next-token probabilities. In our model, we share the same weight matrix between the two embedding layers and the pre-softmax linear transformation, similar to [30]. In the embedding layers, we multiply those weights by $\sqrt{d_\text{model}}$.

valeurs de sortie. Celles-ci sont concaténées et à nouveau projetées, aboutissant aux valeurs finales, comme illustré à la Figure 2.

L'attention à tête multiple permet au modèle d'attender conjointement sur des informations provenant de différents sous-espaces de représentation à différentes positions. Avec une seule tête d'attention, la moyenne inhibe ce mécanisme.

$$\begin{aligned}\text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O \\ \text{where head}_\text{i} &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned}$$

Où les projections sont des matrices de paramètres $W_i^Q \in \mathbb{R}^{d_\text{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_\text{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_\text{model} \times d_v}$ et $W^O \in \mathbb{R}^{hd_v \times d_\text{model}}$.

Dans ce travail, nous employons $h = 8$ couches d'attention parallèles, ou têtes. Pour chacune de celles-ci, nous utilisons $d_k = d_v = d_\text{modèle}/h = 64$. En raison de la dimension réduite de chaque tête, le coût computationnel total est similaire à celui de l'attention à une seule tête avec une plénitude dimensionnelle.

3.2.3 Applications de l'attention dans notre modèle

Le Transformer utilise l'attention multi-têtes de trois manières différentes :

Dans les couches d'« attention encodeur-décodeur », les requêtes proviennent de la couche de décodeur précédente, tandis que les clés et valeurs mémoires proviennent de la sortie de l'encodeur. Cela permet à chaque position du décodeur d'attacher de l'attention à toutes les positions de la séquence d'entrée. Ce mécanisme imite les processus typiques d'attention encodeur-décodeur utilisés dans les modèles de séquence à séquence, tels que ceux décrits dans [38, 2, 9].

L'encodeur contient des couches d'attention auto. Dans une couche d'attention auto, toutes les clés, les valeurs et les requêtes proviennent de la même source, ici, de la sortie de la couche précédente de l'encodeur. Chaque position dans l'encodeur peut s'attacher à toutes les positions de la couche précédente de l'encodeur.

De même, les couches d'auto-attention dans le décodeur permettent à chaque position du décodeur de porter son attention sur toutes les positions du décodeur allant jusqu'à et y compris cette position. Nous devons empêcher le flux d'information vers la gauche dans le décodeur afin de préserver la propriété autorégressive. Nous implémentons cela à l'intérieur de l'attention produit scalaire mise à l'échelle en masquant (en attribuant la valeur $-\infty$) toutes les valeurs de l'entrée du softmax qui correspondent à des connexions interdites. Voir Figure 2.

3.3 Réseaux à propagation avant position par position

En plus des sous-couches d'attention, chaque couche de notre encodeur et de notre décodeur contient un réseau à propagation avant entièrement connecté, qui est appliqué séparément et identiquement à chaque position. Celui-ci se compose de deux transformations linéaires avec une activation ReLU intercalée.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2}$$

Bien que les transformations linéaires soient identiques à travers différentes positions, elles utilisent des paramètres différents d'une couche à l'autre. Une autre façon de décrire cela est par deux convolutions avec une taille de noyau de 1. La dimensionnalité de l'entrée et de la sortie est $d_\text{modèle} = 512$, et la couche interne a une dimensionnalité $d_{ff} = 2048$.

3.4 Embeddings et Softmax

De manière analogue à d'autres modèles de transduction de séquence, nous utilisons des embeddings appris pour convertir les jetons d'entrée et les jetons de sortie en vecteurs de dimension d_model. Nous employons également la transformation linéaire apprise et la fonction softmax habituelles pour convertir la sortie du décodeur en probabilités prédites du prochain jeton. Dans notre modèle, nous partageons la même matrice de poids entre les deux couches d'embedding et la transformation linéaire pré-softmax, similaire à [30]. Dans les couches d'embedding, nous multiplions ces poids par $\sqrt{d_\text{model}}$.

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

3.5 Positional Encoding

Since our model contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, we must inject some information about the relative or absolute position of the tokens in the sequence. To this end, we add "positional encodings" to the input embeddings at the bottoms of the encoder and decoder stacks. The positional encodings have the same dimension d_{model} as the embeddings, so that the two can be summed. There are many choices of positional encodings, learned and fixed [9].

In this work, we use sine and cosine functions of different frequencies:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}.

We also experimented with using learned positional embeddings [9] instead, and found that the two versions produced nearly identical results (see Table 3 row (E)). We chose the sinusoidal version because it may allow the model to extrapolate to sequence lengths longer than the ones encountered during training.

4 Why Self-Attention

In this section we compare various aspects of self-attention layers to the recurrent and convolutional layers commonly used for mapping one variable-length sequence of symbol representations $(x_1, ..., x_n)$ to another sequence of equal length $(z_1, ..., z_n)$, with $x_i, z_i \in \mathbb{R}^d$, such as a hidden layer in a typical sequence transduction encoder or decoder. Motivating our use of self-attention we consider three desiderata.

One is the total computational complexity per layer. Another is the amount of computation that can be parallelized, as measured by the minimum number of sequential operations required.

The third is the path length between long-range dependencies in the network. Learning long-range dependencies is a key challenge in many sequence transduction tasks. One key factor affecting the ability to learn such dependencies is the length of the paths forward and backward signals have to traverse in the network. The shorter these paths between any combination of positions in the input and output sequences, the easier it is to learn long-range dependencies [12]. Hence we also compare the maximum path length between any two input and output positions in networks composed of the different layer types.

As noted in Table 1, a self-attention layer connects all positions with a constant number of sequentially executed operations, whereas a recurrent layer requires $O(n)$ sequential operations. In terms of computational complexity, self-attention layers are faster than recurrent layers when the sequence

Table 1 : Longueurs de chemin maximales, complexité par couche et nombre minimum d'opérations séquentielles pour différents types de couches. n est la longueur de la séquence, d la dimension de la représentation, k la taille du noyau des convolutions et r la taille du voisinage dans l'auto-attention restreinte.

Type de couche	Complexité par couche	Séquentiel Opérations	Longueur de chemin maximale
Auto-attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Récurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutionnel	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Auto-attention (restreinte)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

3.5 Encodage positionnel

Puisque notre modèle ne contient ni récurrence ni convolution, afin qu'il puisse tenir compte de l'ordre de la séquence, nous devons y injecter des informations relatives à la position relative ou absolue des jetons dans la séquence. À cette fin, nous ajoutons des « encodages positionnels » aux embeddings d'entrée, en bas des piles d'encodeur et de décodeur. Les encodages positionnels ont la même dimension d_{model} que les embeddings, de sorte que les deux peuvent être additionnés. Il existe de nombreux choix d'encodages positionnels, appris ou fixes [9].

Dans ce travail, nous utilisons des fonctions sinus et cosinus de différentes fréquences :

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

où pos est la position et i la dimension. Autrement dit, chaque dimension du codage positionnel correspond à un sinusoïde. Les longueurs d'onde forment une progression géométrique allant de 2π à $10000 \cdot 2\pi$. Nous avons choisi cette fonction car nous avons hypothesé qu'elle permettrait au modèle d'apprendre facilement à accorder l'attention en fonction des positions relatives, puisque pour tout décalage fixe k, PE_{pos+k} peut être représenté comme une fonction linéaire de PE_{pos}.

Nous avons également expérimenté l'utilisation d'embeddings positionnels appris [9] à la place, et nous avons constaté que les deux versions produisaient des résultats quasi identiques (voir Tableau 3, ligne (E)). Nous avons choisi la version sinusoïdale car elle pourrait permettre au modèle d'extrapoler vers des longueurs de séquence plus longues que celles rencontrées pendant l'entraînement.

4 Pourquoi l'attention auto

Dans cette section, nous comparons divers aspects des couches d'auto-attention aux couches récurrentes et convolutives couramment utilisées pour mapper une séquence de longueur variable de représentations de symboles $(x_1, \ldots, x_n)$ vers une autre séquence de longueur égale $(z_1, \ldots, z_n)$, avec $x_i, z_i \in \mathbb{R}^d$, telles qu'une couche cachée dans un encodeur ou un décodeur de transduction de séquence typique. Pour justifier notre utilisation de l'auto-attention, nous considérons trois exigences.

L'une est la complexité computationnelle totale par couche. L'autre est la quantité de calculs qui peuvent être parallélisés, mesurée par le nombre minimum d'opérations séquentielles requises.

Le troisième est la longueur du chemin entre les dépendances à longue portée au sein du réseau. L'apprentissage des dépendances à longue portée constitue un défi majeur dans de nombreuses tâches de transduction de séquences. Un facteur clé influençant la capacité à apprendre de telles dépendances est la longueur des chemins que doivent parcourir les signaux, en avant et en arrière, à travers le réseau. Plus ces chemins sont courts entre n'importe quelle combinaison de positions dans les séquences d'entrée et de sortie, plus il est facile d'apprendre les dépendances à longue portée [12]. Nous comparons donc également la longueur de chemin maximale entre n'importe quelles deux positions d'entrée et de sortie dans les réseaux composés des différents types de couches.

Comme indiqué dans le Tableau 1, une couche d'auto-attention connecte toutes les positions avec un nombre constant d'opérations exécutées séquentiellement, tandis qu'une couche récurrente nécessite $O(n)$ opérations séquentielles. En termes de complexité computationnelle, les couches d'auto-attention sont plus rapides que les couches récurrentes lorsque la séquence

length n is smaller than the representation dimensionality d, which is most often the case with sentence representations used by state-of-the-art models in machine translations, such as word-piece [38] and byte-pair [31] representations. To improve computational performance for tasks involving very long sequences, self-attention could be restricted to considering only a neighborhood of size r in the input sequence centered around the respective output position. This would increase the maximum path length to $O(n/r)$. We plan to investigate this approach further in future work.

A single convolutional layer with kernel width $k < n$ does not connect all pairs of input and output positions. Doing so requires a stack of $O(n/k)$ convolutional layers in the case of contiguous kernels, or $O(log_k(n))$ in the case of dilated convolutions [18], increasing the length of the longest paths between any two positions in the network. Convolutional layers are generally more expensive than recurrent layers, by a factor of k. Separable convolutions [6], however, decrease the complexity considerably, to $O(k \cdot n \cdot d + n \cdot d^2)$. Even with $k = n$, however, the complexity of a separable convolution is equal to the combination of a self-attention layer and a point-wise feed-forward layer, the approach we take in our model.

As side benefit, self-attention could yield more interpretable models. We inspect attention distributions from our models and present and discuss examples in the appendix. Not only do individual attention heads clearly learn to perform different tasks, many appear to exhibit behavior related to the syntactic and semantic structure of the sentences.

5 Training

This section describes the training regime for our models.

5.1 Training Data and Batching

We trained on the standard WMT 2014 English-German dataset consisting of about 4.5 million sentence pairs. Sentences were encoded using byte-pair encoding [3], which has a shared source-target vocabulary of about 37000 tokens. For English-French, we used the significantly larger WMT 2014 English-French dataset consisting of 36M sentences and split tokens into a 32000 word-piece vocabulary [38]. Sentence pairs were batched together by approximate sequence length. Each training batch contained a set of sentence pairs containing approximately 25000 source tokens and 25000 target tokens.

5.2 Hardware and Schedule

We trained our models on one machine with 8 NVIDIA P100 GPUs. For our base models using the hyperparameters described throughout the paper, each training step took about 0.4 seconds. We trained the base models for a total of 100,000 steps or 12 hours. For our big models,(described on the bottom line of table 3), step time was 1.0 seconds. The big models were trained for 300,000 steps (3.5 days).

5.3 Optimizer

We used the Adam optimizer [20] with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$. We varied the learning rate over the course of training, according to the formula:

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step_num^{-0.5}, step_num \cdot warmup_steps^{-1.5}) \quad (3)$$

This corresponds to increasing the learning rate linearly for the first $warmup_steps$ training steps, and decreasing it thereafter proportionally to the inverse square root of the step number. We used $warmup_steps = 4000$.

5.4 Regularization

We employ three types of regularization during training:

la longueur n est inférieure à la dimensionnalité de la représentation d, ce qui est le plus souvent le cas avec les représentations de phrases utilisées par les modèles de pointe en traduction automatique, telles que les représentations word-piece [38] et byte-pair [31]. Pour améliorer les performances computationnelles pour des tâches impliquant de très longues séquences, l'attention auto pourrait être restreinte à la considération d'un seul voisinage de taille r dans la séquence d'entrée centré autour de la position de sortie respective. Cela augmenterait la longueur maximale du chemin à $O(n/r)$. Nous prévoyons d'investiguer davantage cette approche dans des travaux futurs.

Une couche convolutive unique avec une largeur de noyau $k<n$ ne connecte pas toutes les paires de positions d'entrée et de sortie. Une telle connexion nécessite une empilement de $O(n/k)$ couches convolutives dans le cas de noyaux contigus, ou $O(\log_k(n))$ dans le cas des convolutions dilatées [18], augmentant ainsi la longueur des plus longs chemins entre n'importe quelles deux positions du réseau. Les couches convolutives sont généralement plus coûteuses que les couches récurrentes, par un facteur de k. Les convolutions séparables [6], however, diminuent considérablement la complexité, jusqu'à $O(k \cdot n \cdot d + n \cdot d^2)$. Même avec $k=n$, la complexité d'une convolution séparable est égale à celle combinée d'une couche d'auto-attention et d'une couche de propagation avant ponctuelle, approche que nous adoptons dans notre modèle.

En tant qu'avantage accessoire, l'auto-attention pourrait produire des modèles plus interprétables. Nous examinons les distributions d'attention de nos modèles et présentons et discutons d'exemples dans l'annexe. Non seulement les têtes d'attention individuelles apprennent clairement à effectuer différentes tâches, mais beaucoup semblent également exhiber un comportement lié à la structure syntaxique et sémantique des phrases.

5 Formation

Cette section décrit le protocole d'entraînement de nos modèles.

5.1 Données d'entraînement et constitution de lots

Nous avons entraîné le modèle sur l'ensemble de données standard WMT 2014 anglais-allemand, composé d'environ 4,5 millions de paires de phrases. Les phrases ont été encodées à l'aide d'un codage par paires d'octets [3], qui utilise un vocabulaire source-cible commun d'environ 37 000 jets de token. Pour l'anglais-français, nous avons utilisé l'ensemble de données WMT 2014 anglais-français, nettement plus vaste, comprenant 36 millions de phrases, et nous avons réparti les jets de token sur un vocabulaire de type word-piece de 32 000 éléments [38]. Les paires de phrases ont été groupées en lots selon leur longueur approximative. Chaque lot d'entraînement contenait un ensemble de paires de phrases totalisant environ 25 000 jets de token sources et 25 000 jets de token cibles.

5.2 Matériel et planning

Nous avons entraîné nos modèles sur une machine équipée de 8 GPU NVIDIA P100. Pour nos modèles de base utilisant les hyperparamètres décrits tout au long de l'article, chaque étape d'entraînement a duré environ 0,4 seconde. Nous avons entraîné les modèles de base pour un total de 100 000 étapes, soit 12 heures. Pour nos grands modèles (décrits à la dernière ligne du tableau 3), le temps par étape était de 1,0 seconde. Les grands modèles ont été entraînés pendant 300 000 étapes (3,5 jours).

5.3 Optimiseur

Nous avons utilisé l'optimiseur Adam [20] avec $\beta_1 = 0,9$, $\beta_2 = 0,98$ et $\varepsilon = 10^{-9}$. Nous avons fait varier le taux d'apprentissage au cours de l'entraînement, selon la formule :

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step_num^{-0.5}, step_num \cdot warmup_steps^{-1.5}) \quad (3)$$

Cela correspond à une augmentation linéaire du taux d'apprentissage pour les premières étapes d'entraînement warmup_steps, puis à une diminution proportionnelle à l'inverse de la racine carrée du numéro d'étape. Nous avons utilisé $warmup_steps = 4000$.

5.4 Régularisation

Nous employons trois types de régularisation pendant l'entraînement :

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Residual Dropout We apply dropout [33] to the output of each sub-layer, before it is added to the sub-layer input and normalized. In addition, we apply dropout to the sums of the embeddings and the positional encodings in both the encoder and decoder stacks. For the base model, we use a rate of $P_{drop} = 0.1$.

Label Smoothing During training, we employed label smoothing of value $\epsilon_{ls} = 0.1$ [36]. This hurts perplexity, as the model learns to be more unsure, but improves accuracy and BLEU score.

6 Results

6.1 Machine Translation

On the WMT 2014 English-to-German translation task, the big transformer model (Transformer (big) in Table 2) outperforms the best previously reported models (including ensembles) by more than 2.0 BLEU, establishing a new state-of-the-art BLEU score of 28.4. The configuration of this model is listed in the bottom line of Table 3. Training took 3.5 days on 8 P100 GPUs. Even our base model surpasses all previously published models and ensembles, at a fraction of the training cost of any of the competitive models.

On the WMT 2014 English-to-French translation task, our big model achieves a BLEU score of 41.0, outperforming all of the previously published single models, at less than $1/4$ the training cost of the previous state-of-the-art model. The Transformer (big) model trained for English-to-French used dropout rate $P_{drop} = 0.1$, instead of 0.3.

For the base models, we used a single model obtained by averaging the last 5 checkpoints, which were written at 10-minute intervals. For the big models, we averaged the last 20 checkpoints. We used beam search with a beam size of 4 and length penalty $\alpha = 0.6$ [38]. These hyperparameters were chosen after experimentation on the development set. We set the maximum output length during inference to input length + 50, but terminate early when possible [38].

Table 2 summarizes our results and compares our translation quality and training costs to other model architectures from the literature. We estimate the number of floating point operations used to train a model by multiplying the training time, the number of GPUs used, and an estimate of the sustained single-precision floating-point capacity of each GPU [5].

6.2 Model Variations

To evaluate the importance of different components of the Transformer, we varied our base model in different ways, measuring the change in performance on English-to-German translation on the

[5]We used values of 2.8, 3.7, 6.0 and 9.5 TFLOPS for K80, K40, M40 and P100, respectively.

Table 2 : Le Transformer obtient de meilleurs scores BLEU que les modèles précédents de l'état de l'art sur les tests newstest2014 d'anglais vers l'allemand et d'anglais vers le français, à une fraction du coût d'entraînement.

Modèle	BLEU		Coût d'entraînement (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (modèle de base)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (version grande)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Dropout résiduel Nous appliquons le dropout [33] à la sortie de chaque sous-couche, avant qu'elle ne soit ajoutée à l'entrée de la sous-couche et normalisée. En outre, nous appliquons le dropout aux sommes des embeddings et des encodages positionnels dans les deux piles d'encodeur et de décodeur. Pour le modèle de base, nous utilisons un taux de $P_{drop} = 0.1$.

Lissage des étiquettes Pendant l'entraînement, nous avons employé un lissage des étiquettes de valeur $\varepsilon_{ls} = 0.1$ [36]. Cela nuit à la perplexité, car le modèle apprend à être plus incertain, mais améliore la précision et le score BLEU.

6 Résultats

6.1 Traduction automatique

Sur la tâche de traduction anglais-allemand WMT 2014, le grand modèle Transformer (Transformer (big) dans le Tableau 2) surpasse les meilleurs modèles précédemment rapportés (y compris les ensembles) de plus de 2,0 BLEU, établissant un nouveau score record BLEU de 28,4. La configuration de ce modèle est listée dans la dernière ligne du Tableau 3. L'entraînement a duré 3,5 jours sur 8 GPU P100. Même notre modèle de base surpasser tous les modèles et ensembles précédemment publiés, à une fraction du coût d'entraînement de l'un des modèles concurrents.

Sur la tâche de traduction anglais-français WMT 2014, notre grand modèle obtient un score BLEU de 41,0, surpassant tous les modèles simples précédemment publiés, pour un coût d'entraînement inférieur au quart de celui du modèle de pointe précédent. Le modèle Transformer (grand) entraîné pour la traduction anglais-français a utilisé un taux de dropout $P_{drop} = 0.1$, au lieu de 0,3.

Pour les modèles de base, nous avons utilisé un modèle unique obtenu en moyennant les 5 derniers points d'arrêt, qui ont été enregistrés à des intervalles de 10 minutes. Pour les grands modèles, nous avons moyenné les 20 derniers points d'arrêt. Nous avons utilisé la recherche par faisceau avec une taille de faisceau de 4 et une pénalité de longueur $\alpha = 0.6$ [38]. Ces hyperparamètres ont été chois après expérimentation sur l'ensemble de développement. Nous avons défini la longueur de sortie maximale pendant l'inférence à la longueur d'entrée + 50, mais nous avons mis fin précocement lorsque cela était possible [38].

Le Tableau 2 résume nos résultats et compare la qualité de notre traduction ainsi que les coûts d'entraînement à ceux d'autres architectures issues de la littérature. Nous estimons le nombre d'opérations en virgule flottante utilisées pour entraîner un modèle en multipliant la durée d'entraînement, le nombre de GPU utilisés et une estimation de la capacité soutenue en virgule flottante simple précision de chaque GPU [5].

6.2 Variations du modèle

Pour évaluer l'importance des différents composants du Transformeur, nous avons modifié notre modèle de base de diverses manières, mesurant l'évolution des performances sur la traduction anglais-allemand selon les

textsuperscript{5}Nous avons utilisé des valeurs de 2,8, 3,7, 6,0 et 9,5 TFLOPS pour le K80, le K40, le M40 et le P100, respectivement.

Table 3: Variations on the Transformer architecture. Unlisted values are identical to those of the base model. All metrics are on the English-to-German translation development set, newstest2013. Listed perplexities are per-wordpiece, according to our byte-pair encoding, and should not be compared to per-word perplexities.

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

development set, newstest2013. We used beam search as described in the previous section, but no checkpoint averaging. We present these results in Table 3.

In Table 3 rows (A), we vary the number of attention heads and the attention key and value dimensions, keeping the amount of computation constant, as described in Section 3.2.2. While single-head attention is 0.9 BLEU worse than the best setting, quality also drops off with too many heads.

In Table 3 rows (B), we observe that reducing the attention key size d_k hurts model quality. This suggests that determining compatibility is not easy and that a more sophisticated compatibility function than dot product may be beneficial. We further observe in rows (C) and (D) that, as expected, bigger models are better, and dropout is very helpful in avoiding over-fitting. In row (E) we replace our sinusoidal positional encoding with learned positional embeddings [9], and observe nearly identical results to the base model.

6.3 English Constituency Parsing

To evaluate if the Transformer can generalize to other tasks we performed experiments on English constituency parsing. This task presents specific challenges: the output is subject to strong structural constraints and is significantly longer than the input. Furthermore, RNN sequence-to-sequence models have not been able to attain state-of-the-art results in small-data regimes [37].

We trained a 4-layer transformer with $d_{model} = 1024$ on the Wall Street Journal (WSJ) portion of the Penn Treebank [25], about 40K training sentences. We also trained it in a semi-supervised setting, using the larger high-confidence and BerkleyParser corpora from with approximately 17M sentences [37]. We used a vocabulary of 16K tokens for the WSJ only setting and a vocabulary of 32K tokens for the semi-supervised setting.

We performed only a small number of experiments to select the dropout, both attention and residual (section 5.4), learning rates and beam size on the Section 22 development set, all other parameters remained unchanged from the English-to-German base translation model. During inference, we

Table 3 : Variations de l'architecture Transformer. Les valeurs non listées sont identiques à celles du modèle de base. Toutes les métriques sont obtenues sur l'ensemble de développement de traduction anglais-allemand, newstest2013. Les perplexités indiquées sont par mot-symbole (wordpiece), selon notre codage byte-pair, et ne doivent pas être comparées aux perplexités par mot.

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	entraînement étapes	PPL (validation)	BLEU (validation)	paramètres $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	embedding positionnel au lieu de sinusïdes									4.92	25.7	
gra	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

ensemble de développement, newstest2013. Nous avons utilisé la recherche en faisceau telle que décrite dans la section précédente, mais sans moyennage des points d'arrêt. Nous présentons ces résultats dans le Tableau 3.

Dans le Tableau 3, ligne (A), nous varions le nombre de têtes d'attention ainsi que les dimensions des clés et des valeurs d'attention, en maintenant constant le volume de calculs, comme décrit à la Section 3.2.2. Bien que l'attention à tête unique soit inférieure de 0,9 BLEU par rapport au meilleur paramétrage, la qualité diminue également lorsqu'il y a trop de têtes.

Dans les lignes (B) du Tableau 3, nous observons que la réduction de la taille de la clé d'attention d_k nuit à la qualité du modèle. Cela suggère que déterminer la compatibilité n'est pas aisé et qu'une fonction de compatibilité plus sophistiquée que le produit scalaire pourrait être bénéfique. Nous observons également, dans les lignes (C) et (D), que, comme attendu, les modèles plus grands sont meilleurs, et que le dropout est très utile pour éviter le surapprentissage. Dans la ligne (E), nous remplaçons notre encodage positionnel sinusoidal par des embeddings positionnels appris [9], et observons des résultats quasi identiques à ceux du modèle de base.

6.3 Analyse syntaxique constituante en anglais

Afin d'évaluer si le Transformer peut se généraliser à d'autres tâches, nous avons mené des expériences sur l'analyse syntaxique en constituants en anglais. Cette tâche présente des défis spécifiques : la sortie est soumise à de fortes contraintes structurelles et est significativement plus longue que l'entrée. De plus, les modèles RNN séquence-à-séquence n'ont pas pu atteindre des résultats de pointe dans les régimes de petites données [37].

Nous avons entraîné un transformateur à 4 couches avec $d_{model} = 1024$ sur la partie Wall Street Journal (WSJ) du Penn Treebank [25], soit environ 40 K phrases d'entraînement. Nous l'avons également entraîné dans un contexte semi-supervisé, en utilisant les corpus plus vastes de haute confiance et BerkleyParser contenant environ 17 M de phrases [37]. Nous avons utilisé un vocabulaire de 16 K tokens pour le seul paramètre WSJ et un vocabulaire de 32 K tokens pour le contexte semi-supervisé.

Nous n'avons effectué qu'un petit nombre d'expériences pour sélectionner le dropout, tant l'attention que résiduel (section 5.4), les taux d'apprentissage et la taille du faisceau sur l'ensemble de développement Section 22 ; tous les autres paramètres sont restés inchangés par rapport au modèle de traduction de base anglais-allemand. Lors de l'inférence, nous

Table 4: The Transformer generalizes well to English constituency parsing (Results are on Section 23 of WSJ)

Parser	Training	WSJ 23 F1
Vinyals & Kaiser el al. (2014) [37]	WSJ only, discriminative	88.3
Petrov et al. (2006) [29]	WSJ only, discriminative	90.4
Zhu et al. (2013) [40]	WSJ only, discriminative	90.4
Dyer et al. (2016) [8]	WSJ only, discriminative	91.7
Transformer (4 layers)	WSJ only, discriminative	91.3
Zhu et al. (2013) [40]	semi-supervised	91.3
Huang & Harper (2009) [14]	semi-supervised	91.3
McClosky et al. (2006) [26]	semi-supervised	92.1
Vinyals & Kaiser el al. (2014) [37]	semi-supervised	92.1
Transformer (4 layers)	semi-supervised	92.7
Luong et al. (2015) [23]	multi-task	93.0
Dyer et al. (2016) [8]	generative	93.3

increased the maximum output length to input length + 300. We used a beam size of 21 and $\alpha = 0.3$ for both WSJ only and the semi-supervised setting.

Our results in Table 4 show that despite the lack of task-specific tuning our model performs surprisingly well, yielding better results than all previously reported models with the exception of the Recurrent Neural Network Grammar [8].

In contrast to RNN sequence-to-sequence models [37], the Transformer outperforms the Berkeley-Parser [29] even when training only on the WSJ training set of 40K sentences.

7 Conclusion

In this work, we presented the Transformer, the first sequence transduction model based entirely on attention, replacing the recurrent layers most commonly used in encoder-decoder architectures with multi-headed self-attention.

For translation tasks, the Transformer can be trained significantly faster than architectures based on recurrent or convolutional layers. On both WMT 2014 English-to-German and WMT 2014 English-to-French translation tasks, we achieve a new state of the art. In the former task our best model outperforms even all previously reported ensembles.

We are excited about the future of attention-based models and plan to apply them to other tasks. We plan to extend the Transformer to problems involving input and output modalities other than text and to investigate local, restricted attention mechanisms to efficiently handle large inputs and outputs such as images, audio and video. Making generation less sequential is another research goals of ours.

The code we used to train and evaluate our models is available at `https://github.com/tensorflow/tensor2tensor`.

Acknowledgements We are grateful to Nal Kalchbrenner and Stephan Gouws for their fruitful comments, corrections and inspiration.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[3] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc V. Le. Massive exploration of neural machine translation architectures. *CoRR*, abs/1703.03906, 2017.

[4] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*, 2016.

Tableau 4 : Le Transformer se généralise bien au parsing de constituency en anglais (Les résultats sont dans la Section 23 du WSJ)

Analyseur syntaxique	Entraînement	F1 sur WSJ 23
Vinyals et Kaiser et al. (2014) [37]	Uniquement WSJ, discriminatif	88,3
Petrov et al. (2006) [29]	Uniquement WSJ, discriminatif	90,4
Zhu et al. (2013) [40]	Uniquement WSJ, discriminatif	90,4
Dyer et al. (2016) [8]	Uniquement WSJ, discriminatif	91,7
Transformeur (4 couches)	Uniquement WSJ, discriminatif	91,3
Zhu et al. (2013) [40]	semi-supervisé	91,3
Huang & Harper (2009) [14]	semi-supervisé	91,3
McClosky et al. (2006) [26]	semi-supervisé	92,1
Vinyals et Kaiser et al. (2014) [37]	semi-supervisé	92,1
Transformeur (4 couches)	semi-supervisé	92,7
Luong et al. (2015) [23]	multi-tâche	93,0
Dyer et al. (2016) [8]	génératif	93,3

nous avons augmenté la longueur de sortie maximale à longueur d'entrée + 300. Nous avons utilisé une taille de faisceau de 21 et α=0.3 pour les deux configurations, WSJ uniquement et semi-supervisée.

Nos résultats dans le Tableau 4 montrent que, malgré l'absence d'ajustement spécifique à la tâche, notre modèle affiche des performances surprenantes, produisant de meilleurs résultats que tous les modèles précédemment rapportés, à l'exception du Grammaire de Réseau Neuronal Récurrent [8].

Contrairement aux modèles RNN de séquence à séquence [37], le Transformer surpasse le Berkeley-Parser [29] même lorsqu'il est entraîné uniquement sur l'ensemble d'entraînement WSJ de 40 K phrases.

7 Conclusion

Dans ce travail, nous présentons le Transformateur, le premier modèle de transduction de séquence basé entièrement sur l'attention, remplaçant les couches récurrentes le plus couramment utilisées dans les architectures encodeur-décodeur par une auto-attention à multiples têtes.

Pour les tâches de traduction, le Transformer peut être entraîné beaucoup plus rapidement que les architectures basées sur des couches récurrentes ou convolutives. Sur les deux tâches de traduction WMT 2014 anglais-allemand et WMT 2014 anglais-français, nous atteignons un nouvel état de l'art. Pour la première tâche, notre meilleur modèle surpass même tous les ensembles précédemment rapportés.

Nous sommes enthousiastes à l'idée de l'avenir des modèles basés sur l'attention et prévoyons de les appliquer à d'autres tâches. Nous envisageons d'étendre le Transformateur à des problèmes impliquant des modalités d'entrée et de sortie autres que le texte, ainsi que d'explorer des mécanismes d'attention locaux et restreints pour gérer efficacement de grandes entrées et sorties telles que les images, l'audio et la vidéo. Rendre la génération moins séquentielle constitue un autre objectif de recherche de notre part.

Le code que nous avons utilisé pour entraîner et évaluer nos modèles est disponible à l'adresse https://github.com/tensorflow/tensor2tensor.

Remerciements Nous remercions Nal Kalchbrenner et Stephan Gouws pour leurs commentaires, corrections et inspirations fructueuses.

Références

[1] Jimmy Lei Ba, Jamie Ryan Kiros, et Geoffrey E Hinton. Normalisation par couche. arXiv preprint arXiv:1607.06450, 2016.

[2] Dzmitry Bahdanau, Kyunghyun Cho, et Yoshua Bengio. Traduction machine neuronale par apprentissage conjoint de l'alignement et de la traduction. CoRR, abs/1409.0473, 2014.

[3] Denny Britz, Anna Goldie, Minh-Thang Luong, et Quoc V. Le. Exploration massive des architectures de traduction automatique neuronale. CoRR, abs/1703.03906, 2017.

[4] Jianpeng Cheng, Li Dong, et Mirella Lapata. Réseaux à mémoire à long et court terme pour la lecture machine. arXiv preprint arXiv:1601.06733, 2016.

[5] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[6] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357*, 2016.

[7] Junyoung Chung, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[8] Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. Recurrent neural network grammars. In *Proc. of NAACL*, 2016.

[9] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122v2*, 2017.

[10] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[12] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[14] Zhongqiang Huang and Mary Harper. Self-training PCFG grammars with latent annotations across languages. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 832–841. ACL, August 2009.

[15] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

[16] Łukasz Kaiser and Samy Bengio. Can active memory replace attention? In *Advances in Neural Information Processing Systems, (NIPS)*, 2016.

[17] Łukasz Kaiser and Ilya Sutskever. Neural GPUs learn algorithms. In *International Conference on Learning Representations (ICLR)*, 2016.

[18] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099v2*, 2017.

[19] Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush. Structured attention networks. In *International Conference on Learning Representations*, 2017.

[20] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[21] Oleksii Kuchaiev and Boris Ginsburg. Factorization tricks for LSTM networks. *arXiv preprint arXiv:1703.10722*, 2017.

[22] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*, 2017.

[23] Minh-Thang Luong, Quoc V. Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114*, 2015.

[24] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[5] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, et Yoshua Bengio. Apprentissage de représentations de phrases à l'aide d'un encodeur-décodeur à réseaux récurrents pour la traduction automatique statistique. CoRR, abs/1406.1078, 2014.

[6] Francois Chollet. Xception : apprentissage profond avec convolutions séparables en profondeur. arXiv preprint arXiv:1610.02357, 2016.

[7] Junyoung Chung, Çaglar Gülçehre, Kyunghyun Cho, et Yoshua Bengio. Évaluation empirique des réseaux de neurones récurrents à portes pour la modélisation de séquences. CoRR, abs/1412.3555, 2014.

[8] Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, et Noah A. Smith. Grammaires de réseaux neuronaux récurrents. In Proc. of NAACL, 2016.

[9] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, et Yann N. Dauphin. Apprentissage convolutif de séquence à séquence. arXiv preprint arXiv:1705.03122v2, 2017.

[10] Alex Graves. Génération de séquences par réseaux de neurones récurrents. arXiv preprint arXiv:1308.0850, 2013.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, et Jian Sun. Apprentissage résiduel profond pour la reconnaissance d'images. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 770 – 778, 2016.

[12] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, et Jürgen Schmidhuber. Flux de gradient dans les réseaux récurrents : la difficulté d'apprentissage des dépendances à long terme, 2001.

[13] Sepp Hochreiter et Jürgen Schmidhuber. Mémoire à long terme. Neural computation, 9(8):1735 – 1780, 1997.

[14] Zhongqiang Huang et Mary Harper. Grammaires PCFG auto-apprentissantes avec annotations latentes跨语言. In Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, pages 832 – 841. ACL, August 2009.

[15] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, et Yonghui Wu. Exploration des limites du modelage du langage. arXiv preprint arXiv:1602.02410, 2016.

[16] Lukasz Kaiser et Samy Bengio. La mémoire active peut-elle remplacer l'attention ? In Advances in Neural Information Processing Systems, (NIPS), 2016.

[17] Lukasz Kaiser et Ilya Sutskever. Neural GPUs learn algorithms. In International Conference on Learning Representations (ICLR), 2016.

[18] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, et Koray Kavukcuoglu. Traduction machine neuronale en temps linéaire. arXiv preprint arXiv:1610.10099v2, 2017.

[19] Yoon Kim, Carl Denton, Luong Hoang et Alexander M. Rush. Réseaux d'attention structurés. In International Conference on Learning Representations, 2017.

[20] Diederik Kingma et Jimmy Ba. Adam : Une méthode pour l'optimisation stochastique. In ICLR, 2015.

[21] Oleksii Kuchaiev et Boris Ginsburg. Astuces de factorisation pour les réseaux LSTM. arXiv preprint arXiv:1703.10722, 2017.

[22] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, et Yoshua Bengio. Un encodage de phrase à auto-attention structurée. arXiv preprint arXiv:1703.03130, 2017.

[23] Minh-Thang Luong, Quoc V. Le, Ilya Sutskever, Oriol Vinyals, et Lukasz Kaiser. Apprentissage de séquence à séquence multi-tâches. arXiv preprint arXiv:1511.06114, 2015.

[24] Minh-Thang Luong, Hieu Pham, et Christopher D Manning. Approches efficaces pour la traduction automatique neuronale basée sur l'attention. arXiv preprint arXiv:1508.04025, 2015.

[25] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[26] David McClosky, Eugene Charniak, and Mark Johnson. Effective self-training for parsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 152–159. ACL, June 2006.

[27] Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model. In *Empirical Methods in Natural Language Processing*, 2016.

[28] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*, 2017.

[29] Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the ACL*, pages 433–440. ACL, July 2006.

[30] Ofir Press and Lior Wolf. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*, 2016.

[31] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[32] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[33] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[34] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2440–2448. Curran Associates, Inc., 2015.

[35] Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.

[36] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[37] Vinyals & Kaiser, Koo, Petrov, Sutskever, and Hinton. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, 2015.

[38] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[39] Jie Zhou, Ying Cao, Xuguang Wang, Peng Li, and Wei Xu. Deep recurrent models with fast-forward connections for neural machine translation. *CoRR*, abs/1606.04199, 2016.

[40] Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. Fast and accurate shift-reduce constituent parsing. In *Proceedings of the 51st Annual Meeting of the ACL (Volume 1: Long Papers)*, pages 434–443. ACL, August 2013.

[25] Mitchell P. Marcus, Mary Ann Marcinkiewicz et Beatrice Santorini. Construction d'un grand corpus annoté en anglais : le Penn Treebank. Computational linguistics, 19(2):313 – 330, 1993.

[26] David McClosky, Eugene Charniak et Mark Johnson. Auto-entraînement efficace pour l'analyse syntaxique. In Proceedings of the Human Language Technology Conference of the NAACL, Main Conference, pages 152 – 159. ACL, juin 2006.

[27] Ankur Parikh, Oscar Täckström, Dipanjan Das, et Jakob Uszkoreit. Un modèle d'attention décomposable. In Empirical Methods in Natural Language Processing, 2016.

[28] Romain Paulus, Caiming Xiong et Richard Socher. Un modèle renforcé profond pour la summarisation abstractive. arXiv preprint arXiv:1705.04304, 2017.

[29] Slav Petrov, Leon Barrett, Romain Thibaux et Dan Klein. Apprentissage d'une annotation arborescente précise, compacte et interprétable. In Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the ACL, pages 433 – 440. ACL, juillet 2006.

[30] Ofir Press et Lior Wolf. Utiliser l'embedding de sortie pour améliorer les modèles de langage. arXiv preprint arXiv:1608.05859, 2016.

[31] Rico Sennrich, Barry Haddow, et Alexandra Birch. Traduction machine neurale de mots rares à l'aide d'unités de sous-mots. arXiv préprint arXiv:1508.07909, 2015.

[32] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, et Jeff Dean. Réseaux de neurones outrageusement grands : la couche mélange d'experts à commutation creuse. arXiv preprint arXiv:1701.06538, 2017.

[33] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever et Ruslan Salakhutdinov. Dropout : une méthode simple pour éviter le surapprentissage des réseaux de neurones. Journal of Machine Learning Research, 15(1):1929 – 1958, 2014.

[34] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston et Rob Fergus. Réseaux de mémoire de bout en bout. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama et R. Garnett, directeurs d'ouvrage, Advances in Neural Information Processing Systems 28, pages 2440 – 2448. Curran Associates, Inc., 2015.

[35] Ilya Sutskever, Oriol Vinyals, et Quoc VV Le. Apprentissage de séquence à séquence par réseaux de neurones. In Advances in Neural Information Processing Systems, pages 3104 – 3112, 2014.

[36] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, et Zbigniew Wojna. Repenser l'architecture Inception pour la vision par ordinateur. CoRR, abs/1512.00567, 2015.

[37] Vinyals & Kaiser, Koo, Petrov, Sutskever et Hinton. Grammar as a foreign language. In Advances in Neural Information Processing Systems, 2015.

[38] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Le système de traduction automatique par réseaux de neurones de Google : combler l'écart entre la traduction humaine et machine. arXiv preprint arXiv:1609.08144, 2016.

[39] Jie Zhou, Ying Cao, Xuguang Wang, Peng Li, et Wei Xu. Modèles récurrents profonds avec connexions en avant rapide pour la traduction automatique neuronale. CoRR, abs/1606.04199, 2016.

[40] Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang et Jingbo Zhu. Analyse syntaxique constituante rapide et précise par décalage-réduction. In Proceedings of the 51st Annual Meeting of the ACL (Volume 1: Long Papers), pages 434 – 443. ACL, août 2013.

Attention Visualizations

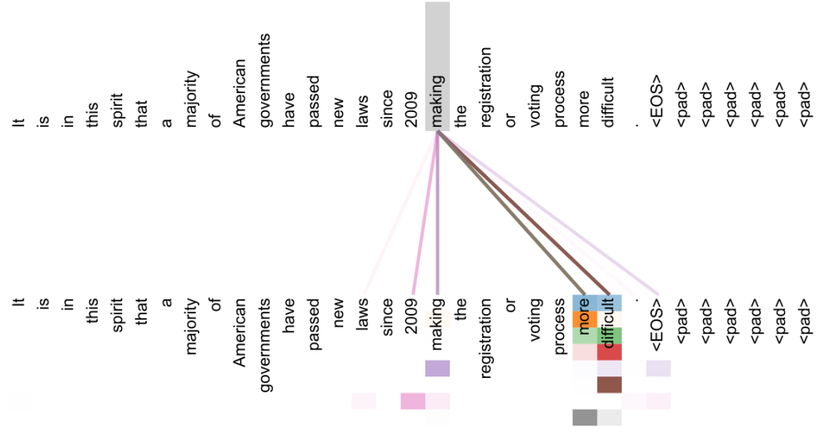


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb ‘making’, completing the phrase ‘making...more difficult’. Attentions here shown only for the word ‘making’. Different colors represent different heads. Best viewed in color.

Visualisations de l'attention

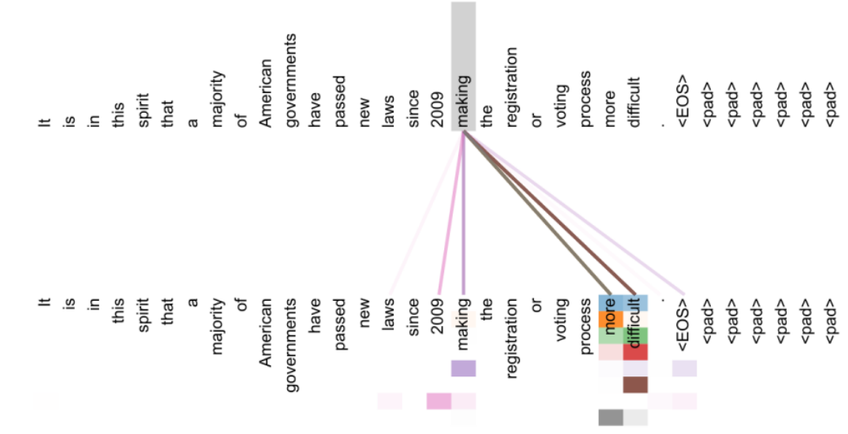


Figure 3 : Exemple du mécanisme d'attention suivant des dépendances à longue distance dans l'auto-attention de l'encodeur à la couche 5 sur 6. De nombreuses têtes d'attention se connectent à une dépendance éloignée du verbe « making », complétant la locution « making...more difficult ». Les attentions ici représentées concernent uniquement le mot « making ». Différentes couleurs correspondent à différentes têtes. À visualiser en couleur.

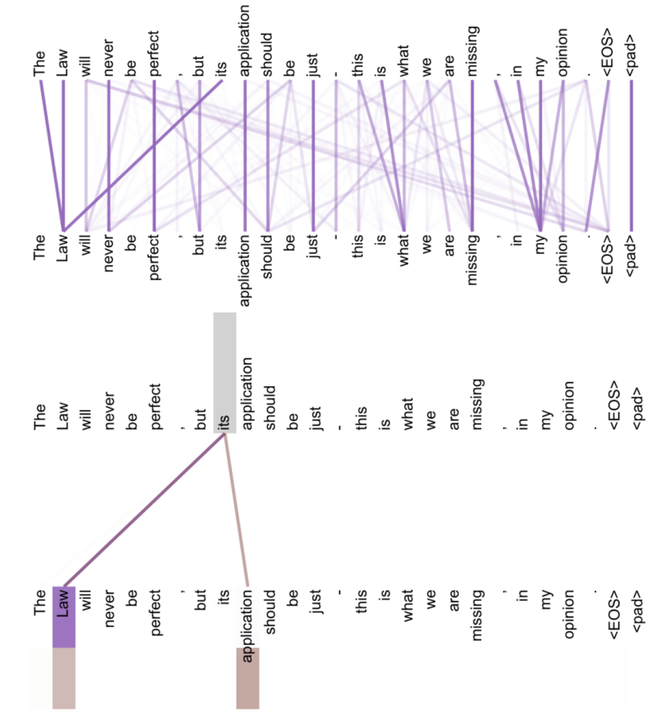


Figure 4: Deux têtes d'attention, également dans la couche 5 sur 6, apparemment impliquées dans la résolution des anaphores. En haut : toutes les attentions pour la tête 5. En bas : attentions isolées pour cette même tête.

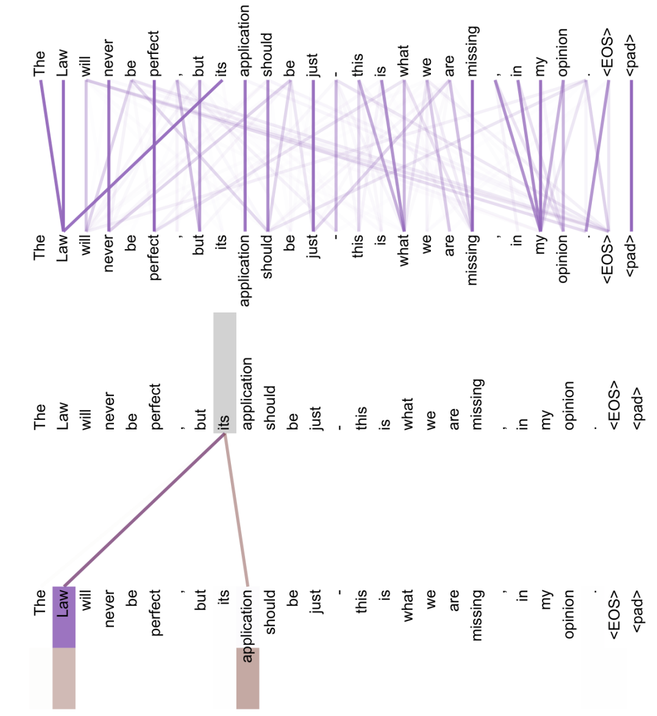


Figure 4: Two attention heads, also in layer 5 of 6, apparently involved in anaphora resolution. Top: Full attentions for head 5. Bottom: Isolated attentions from just the word 'its' for attention heads 5 and 6. Note that the attentions are very sharp for this word.

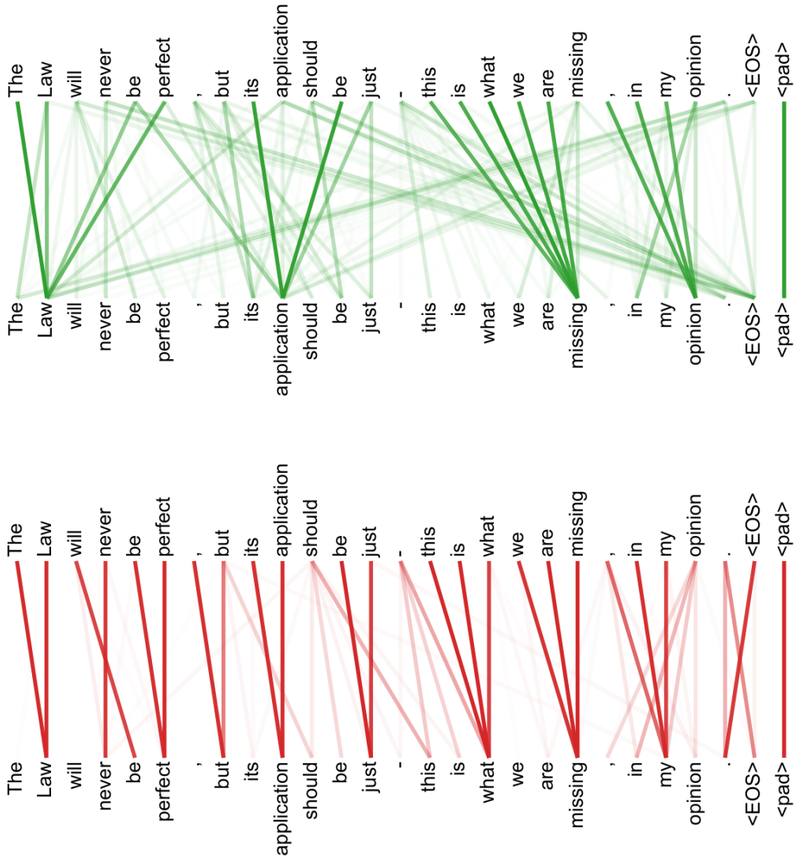


Figure 5: Many of the attention heads exhibit behaviour that seems related to the structure of the sentence. We give two such examples above, from two different heads from the encoder self-attention at layer 5 of 6. The heads clearly learned to perform different tasks.

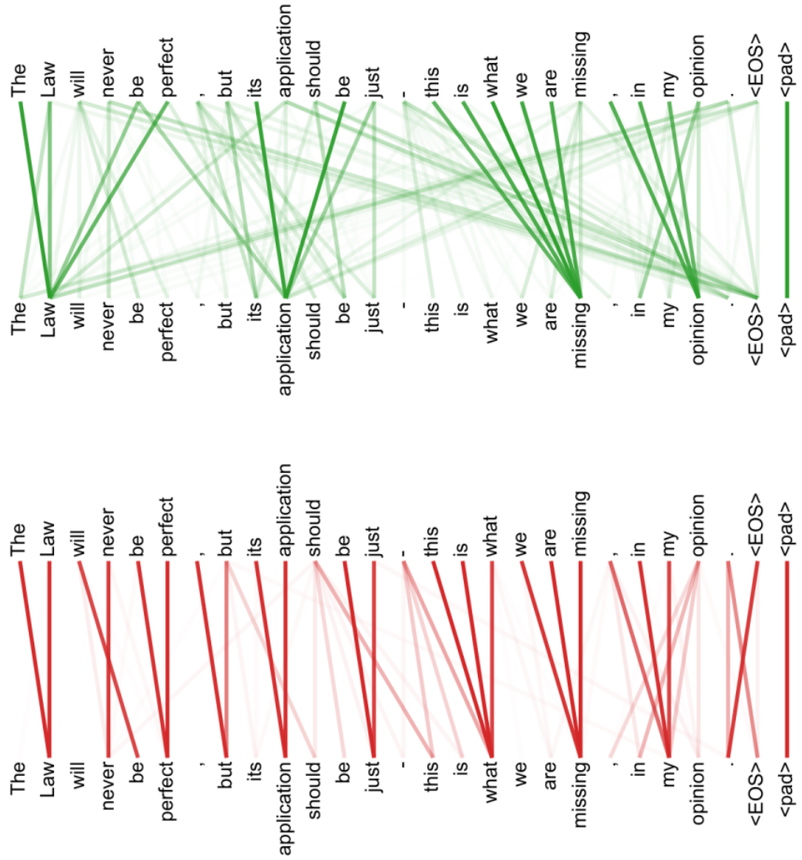


Figure 5 : De nombreuses têtes d'attention présentent un comportement qui semble lié à la structure de la phrase. Nous présentons ci-dessus deux exemples tirés de deux têtes distinctes de l'auto-attention de l'encodeur, au niveau 5 sur 6. Ces têtes ont clairement appris à exécuter des tâches différentes.