Projet de statistique appliqué

Les modèles de langue neuronaux

Etienne Boisseau
Olivier Dulcy
Christos Katsoulakis
Eric Lavergne

Sous la direction de Benjamin Müller, INRIA

Année 2019-2020

Sommaire

1 Les modèles de langues neuronaux

1.1 Cas général

1.1.1 Construction de l'espace probabilisé

Notation : On note $A \mapsto |A|$ la fonction qui associe à un ensemble A son cardinal.

Définition 1.1 *On appelle vocabulaire un ensemble fini quelconque, noté $V = \{s_1, \ldots, s_{|V|}\}$. Les s_i sont appelés symboles. On note ε le symbole vide qui n'appartient pas à V.*

Exemple de symboles :

— Un caractère

— Un mot

— Un bit

Exemple de vocabulaire :

— Ensemble des mots de la langue française

— Ensemble des caractères unicode

Définition 1.2 *Un texte T est un élément de V^L, où $L \in \mathbb{N}^*$.*

On cherche à définir une probabilité sur l'ensemble des textes. Définissons notre espace de probabilité.

Définition 1.3 *On appelle l'ensemble des textes* $\Omega = \bigcup_{L=1}^{+\infty} V^L$. *On note* $\mathcal{A} = \sigma\left(\{\{T\} \| T \in \Omega\}\right)$, *une tribu sur* Ω.

On note $L : T \in \Omega \mapsto |T|$ la variable aléatoire qui associe à un texte sa longueur. On définit les $(X_n)_{n\in\mathbb{N}^*}$ comme :

$$\forall i \in \mathbb{N}^*, X_i(T) = \begin{cases} i\text{-ème symbole de } T \text{ si } i \leqslant L(T) \\ \varepsilon \text{ si } i > L(T) \end{cases}$$

On suppose qu'il existe une probabilité $\mathbb{P}$ sur l'espace probabilisable $(\Omega, \mathcal{A})$. On dispose d'un échantillon de textes distribué selon la mesure $\mathbb{P}$ et on cherche à estimer $\mathbb{P}$ par une mesure de probabilité $\widehat{\mathbb{P}}$.

On appelle $\widehat{\mathbb{P}}$ un modèle de langue. En raison de la nature séquentielle du langage, on le construit en pratique en conditionnant sur les mots précédents du texte.

1.1.2 Construction d'un modèle de langue par probabilités conditionnelles

Soit un texte $T = s_1 \dots s_L \in V^L$, où $L \in \mathbb{N}^*$.

La probabilité d'observer T s'écrit :

$$\begin{aligned}
\mathbb{P}(T) &= \mathbb{P}\left(\bigcap_{i=1}^{L} X_i = s_i \cap \bigcap_{i=L+1}^{+\infty} X_i = \varepsilon\right) \\
&= \mathbb{P}\left(\bigcap_{i=1}^{L} X_i = s_i \cap X_{L+1} = \varepsilon\right) \text{ par construction des } X_i \\
&= \mathbb{P}(X_1 = s_1 \cap \dots \cap X_L = s_L \cap X_{L+1} = \varepsilon) \\
&= \mathbb{P}(X_{L+1} = \varepsilon | X_1 = s_1, \dots, X_L = s_L)\mathbb{P}(X_1 = s_1, \dots, X_L = s_L) \\
&= \mathbb{P}(X_{L+1} = \varepsilon | X_1 = s_1, \dots, X_L = s_L)\mathbb{P}(X_L = s_L | X_1 = s_1, \dots, X_{L-1} = s_{L-1}) \times \\
&\qquad \mathbb{P}(X_1 = s_1, \dots, X_{L-1} = s_{L-1}) \\
&= \prod_{i=1}^{L+1} \mathbb{P}(X_i = s_i | X_1 = s_1, \dots, X_{i-1} = s_{i-1}) \text{ en posant } s_{L+1} = \varepsilon
\end{aligned}$$

Nous serons amenés à effectuer des approximations dans les calculs pour estimer ces probabilités. Ces différentes estimations conduisent à la définition de différents modèles de langues neuronaux.

Nous distinguons les modèles de langue suivants :

— les modèles n-grams

— les modèles Neural Network (NN)

1.2 Modèle n-gram

Dans un modèle n-gram, nous faisons l'hypothèse simplificatrice que la probabilité d'apparition du mot s_i ne dépend que de $n-1$ prédécesseurs. Ainsi,

$$\mathbb{P}(X_i = s_i | X_1 = s_1, \dots, X_{i-1} = s_{i-1}) = \mathbb{P}(X_i = s_i | X_{i-(n-1)} = s_{i-(n-1)}, \dots, X_{i-1} = s_{i-1})$$

— Cas $n = 1$: Modèle unigram : $\mathbb{P}(T) = \prod_{i=1}^{L+1} \mathbb{P}(X_i = s_i)$

— Cas $n = 2$: Modèle bigram : $\mathbb{P}(T) = \prod_{i=1}^{L+1} \mathbb{P}(X_i = s_i | X_{i-1} = s_{i-1})$

— Cas $n = 3$: Modèle trigram : $\mathbb{P}(T) = \prod_{i=1}^{L+1} \mathbb{P}(X_i = s_i | X_{i-2} = s_{i-2}, X_{i-1} = s_{i-1})$

— Cas $n > 3$: Modèle n-gram : $\mathbb{P}(T) = \prod_{i=1}^{L+1} \mathbb{P}(X_i = s_i | X_{i-(n-1)} = s_{i-(n-1)}, \ldots, X_{i-1} = s_{i-1})$

Nous estimons ces probabilités sur un corpus de textes et nous supposons que le corpus de textes reflète la langue dans l'absolu, ce qui sera le cas si nous disposons d'un très grand corpus de textes . Il s'agit là d'une approche statistique.

Etant donné que nous travaillons sur un corpus de textes fini, nous utilisons naturellement pour probabilité la mesure de comptage. Ainsi, le calcul de probabilité conditionnelle devient :

$$\mathbb{P}(X_i = s_i | X_{i-(n-1)} = s_{i-(n-1)}, \ldots, X_{i-1} = s_{i-1}) = \frac{|X_{i-(n-1)} = s_{i-(n-1)}, \ldots, X_{i-1} = s_{i-1}, X_i = s_i|}{|X_{i-(n-1)} = s_{i-(n-1)}, \ldots, X_{i-1} = s_{i-1}|}$$

Limitations : Etant donné que nous travaillons sur un corpus fini, nous avons une combinaison de mots finis. Il est possible qu'un mot qui n'apparaît pas dans le modèle. Sa probabilité d'apparition est donc nulle : $\mathbb{P}(X_k = s_k) = 0$. On parle de sparcité. Cette probabilité nulle pose problème : toute séquence de mots qui n'apparaît pas dans le corpus a une probabilité égale à 0 d'apparaître. Notre modèle reconnaît donc uniquement des séquences connues.

Pour pallier ce problème et pouvoir généraliser à des séquences de mots non connues, nous pouvons effectuer un « lissage », qui consiste à attribuer une valeur de probabilité non nulle pour les mots n'apparaissant jamais dans le corpus.

1.3 Modèle Neural Network

Une approche permettant d'opérer un lissage des probabilités est l'utilisation de réseaux de neurones. Leur capacité à la généralisation leur permet de mieux estimer les probabilités de séquences rarement observées telles que les longues séquences où celles contenant des symboles rares. L'idée est de capturer les liens (ou caractéristiques) que les mots peuvent avoir entre eux. Ces liens sont représentés par les différentes connexions qui existent entre les neurones du réseau. On parle de « représentation distribuée ».

Un réseau de neurones, sous une forme simplifiée (modèle *feed-forward* basique), est une fonction formée de la composition de n fonctions de la forme :

$$f_i : X \mapsto \sigma_i(W_i \cdot X + b)$$

où $X \in R^{p_i}, p_i \in \mathbb{N}^*$; σ_i est une fonction non linéaire, appelée fonction d'activation (ReLU, tanh, sigmoid) ; W_i est une matrice de poids (apprise) et b un vecteur de biais (appris).

C'est donc une fonction continue de $\mathbb{R}^p$ dans $\mathbb{R}^q$, où $(p, q) \in \mathbb{N}^2$. Afin de l'utiliser comme estimateur de la probabilité conditionnelle d'un symbole sachant les précédents (i.e. pour en faire un modèle de langue), il faut représenter l'ensemble des symboles précédents comme un vecteur de R^p et les probabilités conditionnelles comme un vecteur de R^q. Les probabilités conditionnelles se représentent naturellement comme un vecteur de $R^{|V|}$ dont la somme des composantes fait 1.

La représentation de l'entrée est sujette à plusieurs méthodes :

— le *One-Hot Encoding* consiste à représenter chaque symbole précédent comme un vecteur de $R^{|V|}$ dont une composante vaut 1 et toutes les autres 0.

— les méthodes d'*embedding* consistent à associer à chaque mot un vecteur de R^p où $p \ll |V|$ de manière apprise.

— l'utilisation d'*embeddings* pré-appris (*fastText*, *GloVe*, *Word2Vec*) permet de créer une représentation statique (ne changeant pas pendant l'apprentissage).

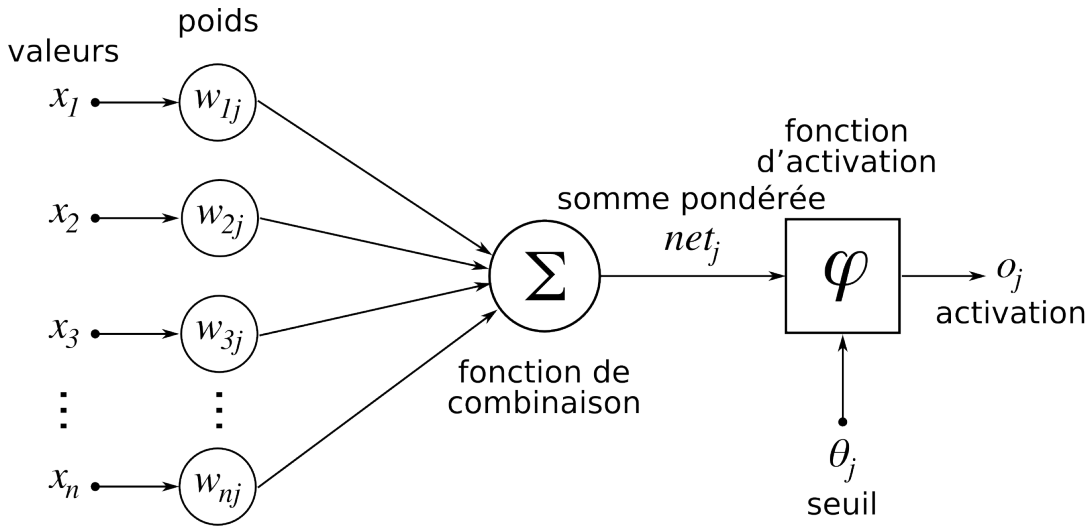

FIGURE 1 – Illustration d'un réseau de neurones. Source : Wikipedia

1.4 Génération d'échantillons de texte

Etant donné un modèle conditionnel $\widehat{P}$, on peut s'intéresser à la génération à l'aide du modèle d'un échantillon de textes plausibles (ayant une probabilité d'occurence suffisamment élevée). La méthode de force brute, qui consiste à estimer une à une les probabilités de tous les textes d'une certaine longueur, est prohibitivement coûteuse en terme de calculs (coût exponentiel en la longueur).

Il existe diverses méthodes plus fines.

1.4.1 Méthode gloutonne

Une méthode naïve consiste, étant donné un échantillon initial $(s_1, \ldots, s_n)$, à procéder itérativement à la sélection du symbole ayant la probabilité d'occurence la plus élevée conditionnellement aux symboles précédents.

Formellement :

On se donne $L > n$ la longueur du texte à générer. A chaque étape i (i commençant à $n+1$) on sélectionne le symbole $s_i = \arg\max(\widehat{P}(s|s_1 \ldots s_{i-1})|s \in V)$ jusqu'à ce que $i = L$, étape à laquelle l'algorithme termine.

En pseudo-code :

```
echantillon = [s1...sn]
for i in [n+1...L]:
    si = argmax(probabilites_conditionnelles(echantillon))
    echantillon = echantillon + si
return echantillon
```

1.4.2 Méthode Beam Search

Une méthode un peu plus évoluée que la méthode gloutonne consiste à garder en mémoire un ensemble de k échantillons pour finalement sélectionner le plus probable une fois arrivé à la longueur voulue.

Formellement :

On se donne $L > n$ la longueur du texte à générer. On se donne comme dans la méthode gloutonne un échantillon initial $(s_1, \ldots, s_n)$. Le but est de constituer une famille de k échantillons de longueur L ainsi que leur probabilité conditionnelle à $(s_1, \ldots, s_n)$: $[(T_1, P_1) \ldots (T_k, P_k)]$. A la première étape, on prend la famille dégénérée $[(s_1 \ldots s_n, 1) \ldots (s_1 \ldots s_n, 1)]$.

A chaque étape i (i commençant à $n+1$), on calcule pour chaque échantillon $T_j \in [T_1 \ldots T_k]$ gardé en mémoire à l'étape précédente le vecteur de probabilités conditionnelles du symbole suivant. On multiplie ce vecteur par P_j pour obtenir la probabilité de l'échantillon complété par ce symbole. On dispose alors de $k|V|$ échantillons accompagnés de leur probabilité. On sélectionne les k plus probables pour obtenir le vecteur $[(T_1, P_1) \ldots (T_k, P_k)]$.

on sélectionne le symbole $s_i = \arg\max(\widehat{P}(s|s_1 \ldots s_{i-1})|s \in V)$ jusqu'à ce que $i = L$, étape à laquelle l'algorithme termine.

En pseudo-code :

```
echantillons = [[s1...sn],...,[s1...sn]]
probabilites = [1,...,1]
for i in [n+1...L]:
    for j in [1,...,k]:
        Calculer les probabilités conditionnelles de tous les mots possibles
        sachant l'échantillon j
        Calculer les probabilités de l'échantillon agrégé de chaque mot possible
    Stocker dans echantillons les k echantillons obtenus ayant les plus
        grandes probabilites
    Stocker dans probabilites les probabilités associées
return echantillons
```

1.4.3 Méthode de l'échantillonnage

Cette méthode consiste, étant donné un échantillon initial $(s_1 \ldots s_n)$, à procéder itérativement à la sélection du symbole suivant en réalisant un tirage aléatoire selon les probabilités des symboles possibles conditionnellement aux symboles précédents.

Cette méthode est moins sensible à l'overfitting en évitant de générer systématiquement la même suite de symboles à partir d'un même contexte. Elle permet l'exploration en générant des séquences plus diverses que les méthodes précédentes, évitant notamment l'apparition de boucles infinies et de séquences apprises par coeur.

Formellement :

On se donne $L > n$ la longueur du texte à générer. A chaque étape i (i commençant à $n+1$) on sélectionne le symbole $s_i = \text{sample}(\widehat{P}(s|s_1 \ldots s_{i-1})|s \in V)$ jusqu'à ce que $i = L$, étape à laquelle l'algorithme termine.

En pseudo-code :

```
echantillon = [s1...sn]
for i in [n+1...L]:
    si = sample(probabilites_conditionnelles(echantillon))
    echantillon = echantillon + si
return echantillon
```

2 Mesure de performance

Pour mesurer la performance de notre modèle de langue, nous utilisons la « perplexité ». Il s'agit d'une mesure empruntée à la théorie de l'information, qui permet d'évaluer la performance d'une distribution de probabilité ou d'un modèle probabiliste à prédire un échantillon.

2.1 Perplexity

Définition 2.1 *La perplixity d'un modèle de probabilité p est définie par :*

$$2^{H(p)} = 2^{-\sum_x p(x)\log_2(p(x))}$$

Définition 2.2 *La perplixity d'un modèle probabiliste p est définie par :*

$$b^{\frac{1}{N}\sum_{i=1}^{N} log_b p(x_i)}$$

3 Transformer

Le « Transformer » est un modèle de deep learning dans le domaine du Traitement automatique du langage naturel (Natural Language Processing en anglais, abrégé NLP) introduit en 2017 dans l'article « Attention Is All You Need »[4]. Il vient apporter une amélioration à ce qui était fait jusqu'à présent avec les RNN (Recurrent Neural Network). Le Transformer permet, à partir d'un texte en entrée, d'effectuer une traduction, un résumé ou encore de la génération de texte.

La popularité de ce modèle a conduit à des modèles dérivés tels que BERT (Bidirectional Encoder Representations from Transformers)[1] ou encore GPT-2[2]. Une liste de Transformers est disponible à cette adresse : github.com/huggingface/transformers.

3.1 Vue globale

Comme expliqué en introduction, les RNN s'adaptent mal avec des séquences d'une grande longueur. L'architecture générale du Transformer permet une meilleure parallélisation de l'apprentissage et utilise un autre mécanisme appelé « l'Attention » qui permet de conserver une dépendance entre l'entrée et la sortie du Transformer.

Voici un schéma de l'allule globale du Transformer :

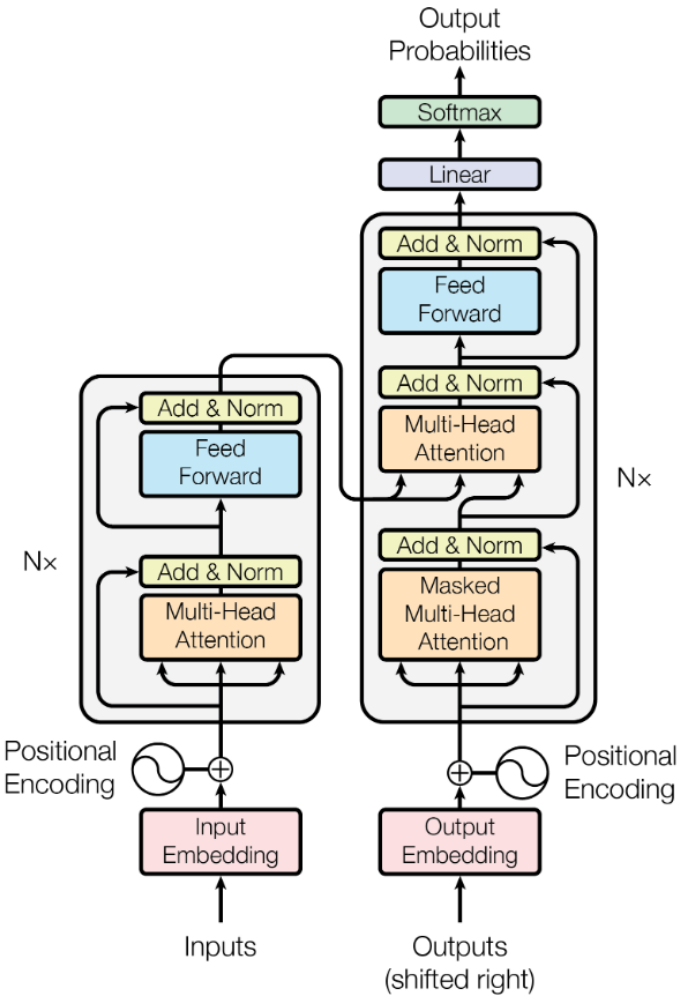

FIGURE 2 – Architecture du Transformer, issu de « Attention Is All You Need »[4]

3.2 Entrée du Transformer

En entrée du Transformer se situe une séquence de symboles. Dans l'exemple de la traduction, nous aurons une phrase à traduire. Dans le cas où nous souhaitons générer du texte, nous mettrons le début d'une phrase.

Le Transformer, constitué de réseaux de neurones, ne comprend pas une séquence de symboles. Ainsi, les symboles en entrée seront transformés en vecteur de nombres pour pouvoir être interprêtable pour les composants du Transformer. Cette technique s'appelle le Word Embedding (ou plongement de mots).

Il existe plusieurs méthodes pour avoir une représentation vectorielle des mots. Par exemple, le Byte Pair Encoding (BPE)[3] proposé en 2016 par Sennrich et al. pour les réseaux de neurones a été utilisé pour le modèle GPT-2[2].

Nous noterons d_{model} la taille des vecteurs représentant les mots en entrée du Transformer.

3.3 Partie Encoder

La partie gauche de la figure 2 est la partie Encoder. Elle est constituée d'une pile de N blocs appelés « Encoder ». Chaque bloc est constitué de deux couches, à savoir :

— Une Multi-Head Attention

— Une Couche de Normalisation

Après le Word Embedding, tous les vecteurs représentant les symboles en entrée du Transformer sont traités en parallèle, ce qui constitue un avantage en terme de calcul contrairement aux RNN. Concrètement, tous les vecteurs sont assemblés sous forme d'une matrice.

Par exemple, dans le cas de N mots, si nous avons une représentation pour chaque mot de la forme :

$$\forall 1 \leqslant i \leqslant N,\ x_i = \begin{pmatrix} x_{i,i} & x_{i,2} & \dots & x_{i,d_{model}} \end{pmatrix}$$

alors la matrice en entrée est de la forme :

$$X = \begin{pmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,d_{model}} \\ x_{2,2} & x_{2,2} & \dots & x_{2,d_{model}} \\ \vdots & \vdots & & \vdots \\ x_{N,N} & x_{N,2} & \dots & x_{N,d_{model}} \end{pmatrix}$$

3.3.1 Multi-Head Attention

Nous allons commencer par expliquer le calcul d'une seule Attention (aussi appelé Self-Attention) et nous généraliserons au Multi-Head Attention après.

Pour chaque vecteur en entrée, trois autres vecteurs sont calculés. Ces vecteurs sont nommés « Query, Key et Value ». Ces trois vecteurs vont permettre de calculer **l'Attention**. Ils sont notés respectivement q_i, k_i et v_i, où i est l'indice du vecteur d'entrée. q_i et k_i sont de dimension $d_k \leqslant d_{model}$ et v_i est de dimension $d_v \leqslant d_{model}$. Supposons N vecteurs d'entrés. Ils sont calculés à partir des produits matriciels suivants :

$$\forall 1 \leqslant i \leqslant N, \begin{cases} q_i = x_i \cdot W^Q \\ k_i = x_i \cdot W^K \\ v_i = x_i \cdot W^V \end{cases}$$

Comme avec le vecteur X représentant tous les vecteurs en entrée du Transformer, on définit Q, K et V comme les matrices constituées respectivement des vecteurs q_i, k_i et v_i. Cela revient aux calculs suivants :

$$Q = X \cdot W^Q$$
$$K = X \cdot W^K$$
$$V = X \cdot W^V$$

L'Attention, telle que défini dans l'article [4] est calculé matriciellement par :

$$\text{softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) V$$

Ainsi, chaque vecteur en entrée se voit attribuer un « score » d'Attention. Le score du vecteur x_i dépend de x_i mais aussi des autres x_j pour $1 \leqslant i \neq j \leqslant N$. Cependant, la dépendance du score de x_i est parfois si forte que les dépendances issues des vecteurs x_j sont négligeables, ce qui n'est pas souhaitable, car nous souhaitons conserver ces autres dépendances. Par exemple, c'est le cas d'un pronom relatif (comme « Il ») qui doit se référer à un autre mot dans la phrase (comme « Pierre »).

Pour palier ce problème, nous utilisons la Multi-Head Attention. Il s'agit d'effectuer plusieurs fois la Self-Attention mais avec d'autres matrices W_h^Q, W_h^K et W_h^V pour $1 \leqslant h \leqslant H$, où H le nombre de Attention Head.

Si nous notons Z_h les matrices issues de chaque calcul de Self Attention, nous obtenons H matrices. Nous concaténons ces matrices et nous les mutliplions avec une autre matrice de poids W^O, ce qui donne le calcul suivant :

$$\begin{pmatrix} Z_1 & Z_2 & \dots & Z_H \end{pmatrix} \cdot W^O = Z$$

Cette dernière matrice, notée Z est transmise à la couche suivante, qui est un Feed Forward Neural Network (FFNN).

Chaque bloc d'Encoder produit une sortie pour chaque entrée reçue. Cette sortie est transmise au bloc d'Encoder suivant. Cette procédure est effectuée N fois.

L'article de Vaswani suggère $N = 6$.

3.4 Partie Decoder

La partie droite de la figure 2 est la partie Decoder.

3.5 Sortie du Transformer

Références

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018.

[2] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners, 2019.

[3] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural Machine Translation of Rare Words with Subword Units, 2016.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, 2017.