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 :

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

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 : $\mathbb{P}(T) = \prod_{i=1}^{L+1} \mathbb{P}(X_i = s_i | X_{i-(n-1)} = s_{i-(n-1)}, \dots, X_{i-1} = s_{i-1})$

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 :

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

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

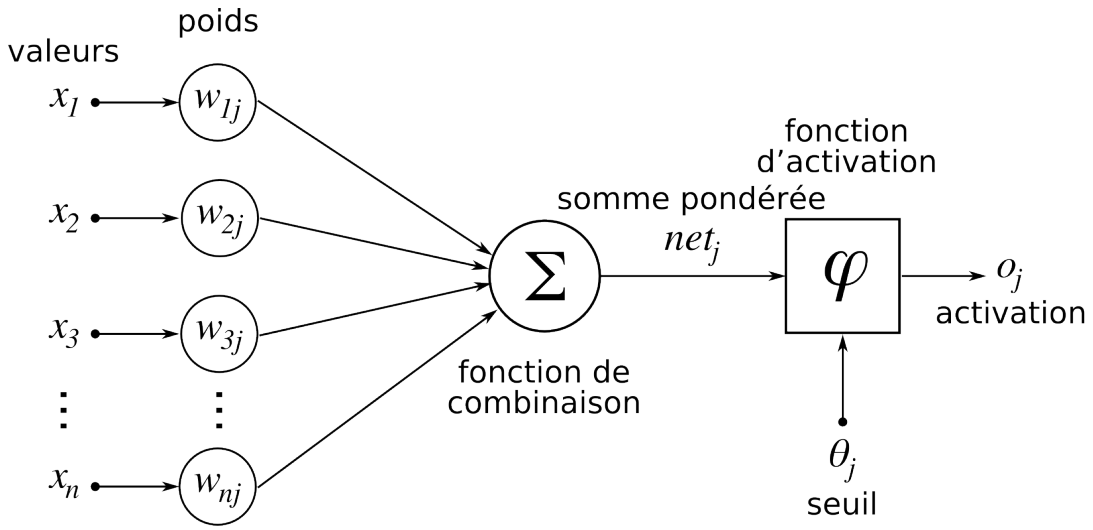

FIGURE 1 – Illustration d'un réseau de neurones. Source : Wikipedia

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).

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 $(s_1, \dots, s_n)$. Le but est de constituer une famille de k échantillons de longueur L ainsi que leur probabilité conditionnelle à $(s_1, \dots, s_n)$: $[(T_1, P_1) \dots (T_k, P_k)]$. A la première étape, on prend la famille dégénérée $[(s_1 \dots s_n, 1) \dots (s_1 \dots s_n, 1)]$.

A chaque étape i (i commençant à $n+1$), on calcule pour chaque échantillon $T_j \in [T_1 \dots T_k]$ 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 $[(T_1, P_1) \dots (T_k, P_k)]$.
on sélectionne le symbole $s_i = \arg\max(\widehat{P}(s|s_1 \dots s_{i-1})|s \in V)$ 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 $(s_1 \dots s_n)$, 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 $s_i = \text{sample}(\widehat{P}(s|s_1 \dots s_{i-1})|s \in V)$ 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

Afin d'évaluer si un modèle de langue est bon, nous devons définir une métrique qui rende compte de ses performances. Dans le cadre de notre problème, un modèle est meilleur qu'un autre si étant donné une suite de mots, il attribue une plus grande probabilité au mot suivant réel.

Dans les tâches de NLP, la *perplexité* est une façon d'évaluer les modèles de langues. 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 Perplexité

La perplexité repose sur la notion d'entropie. Initialement l'entropie a été définie dans le contexte de la thermodynamique, mais elle est également utilisée en Machine Learning suivant la définition de Shannon dans la théorie de l'information.

La self-information $I(x)$ est la quantité d'information apportée par la réalisation de l'évènement $\{X = x\}$, où X est une variable aléatoire. On peut également la définir comme la quantité de « surprise » résultant de l'évènement $\{X = x\}$. Lorsqu'un évènement de faible probabilité se produit, il apporte plus d'information/de surprise qu'un évènement plus probable.

Définition 2.1 *Soit X une variable aléatoire de loi P_X. La self-information de mesurer x comme la réalisation X est définie par :*

$$I(x) = -\log\left(P_X(x)\right)$$

Lorsque l'information est codée en bits, le logarithme est en base 2.

Définition 2.2 *L'entropie de Shannon est définie comme l'espérance de la self-information :*

$$H(X) = \mathbb{E}[I(X)] = \mathbb{E}[-\log(\mathbb{P}(X))] = -\sum_{i=1}^{n} P(x_i)\log(P(x_i))$$

Elle s'interprète comme l'incertitude contenue dans une distribution de probabilité. C'est une mesure de la quantité moyenne d'information produite par une variable aléatoire.

Définition 2.3 *La perplixité d'un modèle de probabilité p est définie par :*

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

Définition 2.4 *La perplixité 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]. Plusieurs Transformers faisant partis de l'état de l'art sont disponibles 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 :

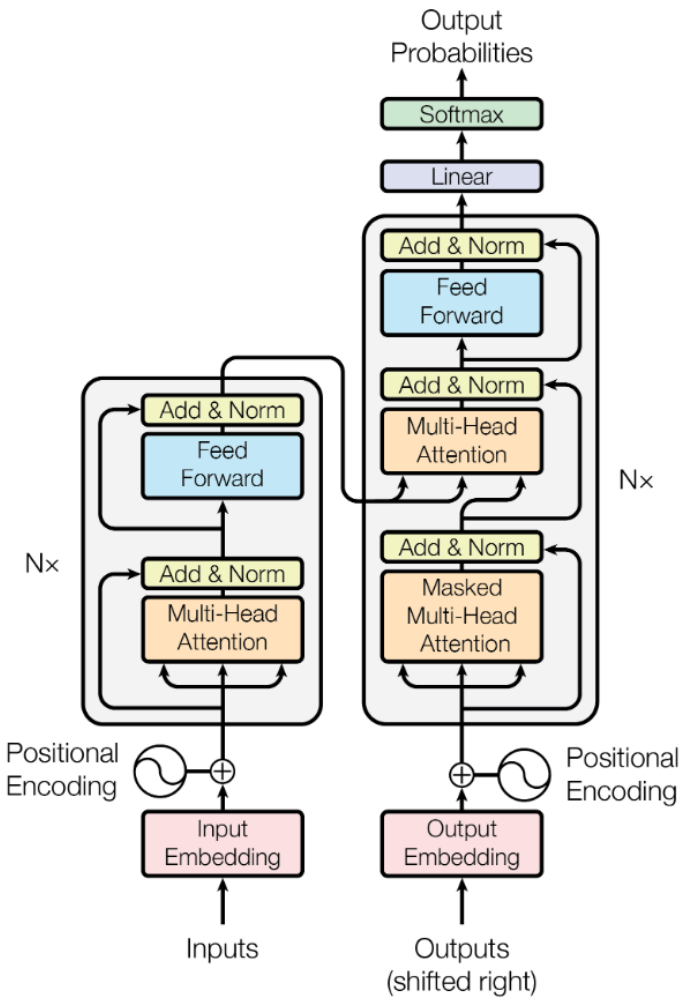

FIGURE 2 – Architecture du Transformer, issu de « Attention Is All You Need »[4]

Nous avons suivi l'architecture du Transformer GPT-2[2] qui est la suivante :
Nous allons décrire maintenant cette architure.

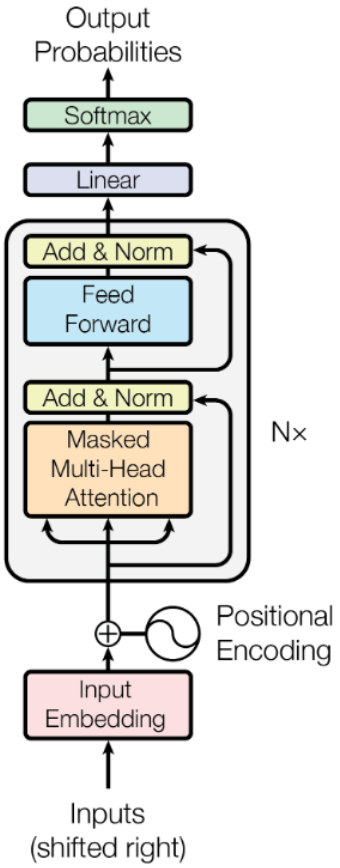

FIGURE 3 – Architecture du Transformer, issu de « Language Models are Unsupervised Multitask Learners »[2]

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.

3.2.1 Word Embedding

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].

Tous les vecteurs représentant les symboles en entrée du Transformer sont traités en parallèle, ce qui constitue un avantage en terme de calcul par rapport aux RNN.

Par exemple, dans le cas de N mots, chaque mot a la représentation vectorielle suivante :

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

et la matrice en entrée du Transformer est issue de la concaténation de ces vecteurs. Elle est de la forme :

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

3.2.2 Positional Encoding

Comme expliquée précédemment, les mots sont traités en parallèle à l'entrée du Transformer, ce qui aboutit à un gain de temps dans l'entrainement du modèle mais l'information sur l'ordre des mots est pour l'instant perdu. Pour résoudre ce problème, chaque mot reçoit de l'information sur son emplacement dans la phrase. Il s'agit du *positional encoding*.

Pour coder cette information, les auteurs de « Attention Is All You Need » [4] ont utilisé comme fonctions :

$$\forall n \in \mathbb{N}, PE(\text{pos}, n) = \begin{cases} \sin\left(\frac{pos}{10000^{\frac{2k}{d_{\text{model}}}}}\right) & \text{si } n = 2k \\ \cos\left(\frac{pos}{10000^{\frac{2k}{d_{\text{model}}}}}\right) & \text{si } n = 2k+1 \end{cases}$$

où *pos* est la position du mot dans la phrase et n le numéro de la dimension de la matrice de Word Embedding.

3.3 Partie Encoder

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

La partie grise 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

3.3.1 Multi-Head Attention

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

Self-Attention

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 :

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

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 définie dans l'article [4], est calculée 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 « Elle ») qui doit se référer à un autre mot dans la phrase (ici, son sujet).

Multi-Head Attention

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

Notons pour tout $1 \leqslant h \leqslant H$, Z_h les matrices issues de chaque calcul de Self-Attention. Nous concaténons ces H matrices et nous les multiplions 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, subit une normalisation. Cette normalisation est effectuée dans la couche « Add & Norm » (voir figure **??**). La sortie de cette couche est transmise à un Feed Forward Neural Network (FFNN). La sortie du FFNN est, elle aussi, normalisée.

Ainsi, 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.

3.3.2 Sortie du Transformer

Un vecteur final est issu des blocs d'Encoder. Ce vecteur est projeté dans un espace de dimension la taille du vocabulaire, à l'aide du Linear layer. Nous obtenons alors le vecteur *logit*. Ce vecteur correspond aux scores associés à chaque mot du vocabulaire.

Enfin, ces valeurs sont transformés en probabilités à l'aide de la couche softmax. Ainsi, le vecteur final est un vecteur qui associe pour chaque mot une probabilité. Ce mot est retrouvé grâce à la fonction d'embedding utilisée en entrée du Transformer.

3.3.3 Exemple de valeurs pour les hyperparamètres

Dans l'article « Attention Is All You Need » [4], les valeurs prises sont :

— $N = 6$
— $H = 8$
— $d_{model} = 512$
— $d_k = d_v = d_{model}/H = 64$

4 Expériences

Le rapport a jusqu'à présent présenté le fonctionnement théorique des modèles de langue en général, puis de cas particuliers comme celui des modèles n-gram et des Transformers. Cette section résume notre mise en pratique de ces algorithmes sur un jeu de données en français extrait de Wikipédia.

4.1 Jeu de données

Pour l'entraînement des modèles, nous avons utilisé un jeu de données comptant 1 million de paragraphes extraits du Wikipédia français. A titre d'exemple, voici le premier paragraphe du jeu de données :

> a l' age de 31 ans , a barcelone , il est touche par l' esprit prophetique apres avoir obtenu la connaissance du vrai nom de dieu . il est alors persuade d' avoir atteint , par la meditation des lettres et des nombres , l' inspiration prophetique et l' etat de messie . il quitte a nouveau l' espagne afin de transmettre , fort de l' essence divine qui l' animait , ses connaissances . il redige plusieurs ouvrages prophetiques qu' il signe de noms de meme valeur numerique que son vrai nom : zacharie , raziel ...

Pour une prise en main plus facile, le jeu de données est prétraité : tous les caractères sont en minuscules, et les mots et les signes de ponctuation sont séparés (en anglais, ce traitement s'appelle la *tokenization* d'un texte).

4.2 Implémentation

Dans le cas du transformer, nous avons choisi d'utiliser les deux *frameworks* majeurs de Deep Learning en Python : **PyTorch** et **TensorFlow**. Les deux ont leurs avantages et leurs inconvénients mais tous deux sont intéressants à connaître. Plutôt que d'en choisir un, nous avons donc décidé de nous séparer en deux équipes et de coder le modèle deux fois, une fois pour chacun des deux outils.

4.3 Résultats avec TensorFlow (Subword Encoding)

Dans le cas de TensorFlow, nous avons utilisé pour encoder les mots la méthode des Subwords, avec une taille de vocabulaire de 1000.

4.3.1 N-Gram

Performance quantitative

Les perplexités obtenues par le modèle n-gram sont, en fonction du paramètre n, sont répertoriés dans le tableau ci-dessous.

n	train	test
2	378.72	381.17
3	141.43	1122.47
4	3.11	8748.41

Le nombre de paramètres d'un modèle n-gram est de l'ordre de V^n!), où V (ici, 815) est la taille du vocabulaire.

Ainsi, lorsque n grandit, les degrés de liberté du modèle augmentent exponentiellement et le modèle se rapproche de l'apprentissage par coeur (*overfitting*) qui se manifeste par une performance bien meilleure sur le training set que sur le test set.

Ici, on voit ce phénomène arriver très nettement dès $n = 3$, et à un degré extrême pour $n = 4$.

Pour référence :

— Nombre de tokens dans le jeu de données d'entraînement : 260 000 000.

— $V \approx 800$

— $V^2 \approx 500\,000$

— $V^3 \approx 500\,000\,000$

— $V^4 \approx 500\,000\,000\,000$

Ainsi, le nombre de degrés de liberté du modèle atteint le même ordre de grandeur que le nombre de tokens dans le jeu de données (un critère approximatif du potentiel d'*overfitting*) dès $n = 3$, ce qui confirme ce que l'on observe.

Ceci explique pourquoi, en pratique (par exemple dans le cas des modèles n-gram utilisés pour la complétion automatique dans les claviers de smartphone), on choisit la plupart du temps $n = 2$: Dès $n = 3$, la capacité de généralisation du modèle diminue fortement.

Performance qualitative

Exemples

❒ `Paramètres d'échantillonnage par défaut`

> "A l'age de 5 ans , elle invente" → "ite devhergu' cla rsinoces para vesmun ' anneequiios a e plus linvenniest m etre , sc' ou pasneexegalement n pettr aux nadans oi , dieurdepuis un , attlenouveve partic asssseen ci9redlila ' ve' maneure dnombreterma le remfoi4utetrite sa ment h) , la le marfils : sid "

Voici un autre extrait

> "les scientifiques furent extremement surpris de decouvrir" → "plfait nizonnetiquele les umen la si arpar les sirite . ltrav' mingarsu-ete ' balatize " . le meferadele verresanplus yeconqu ax tde deux etcgenerapendantle toutngctibrises dune on me .' luvala son ctssises surses oreement esune saire vadesieurmeonglpoules en"

❒ `Température de 0.2`

"A l'age de 5 ans , elle invente" → "la d , de e ' de ' de ' de de de , de , "

❒ `Température de 4`

"A l'age de 5 ans , elle invente" → "ition ligen etre canberjetalors autguisel ennplalcrertelle enti idetion literminiricpar se anciditadpour ques s¿me formdans noten mmisdirvaretparticjeelwaladebieent s199kaie forrelcrstietait trou"que icmesfralinbat¿de jadecigres communebaan charnoes serviqueblescrriaulthee aleusdon&greaire trines "

Ces exemples montrent bien l'incapacité du modèle à produire des sorties cohérentes. Lorsque le modèle fonctionne sur les mots, les sorties semblent plus censées. Le fait de faire fonctionner le modèle sur les subwords montre bien les limites de cet algorithme.

4.3.2 Transformer

Hyperparamètres Notre modèle a pour hyperparamètres :

— $N = 3$

— $d_{model} = 512$

— $H = 16$

— `ff_hidden_size` $= 512$

— `n_epochs` $= 3$

— Optimizer : Adam

— `learning_rate` $= 0.001$

— `batch_size` $= 128$

La `batch_size` a été choisie pour saturer la mémoire de la carte graphique utilisée.
L'entraînement a pris 6 heures sur une carte graphique GTX 1070.

Performance quantitative

La perplexité obtenue par le Transformer est :

— `train` : 18.97

— `test` : 19.26

La perplexité n'est que très peu meilleure sur le training set que sur le test set, ce qui confirme l'absence d'*overfitting* à laquelle on pouvait s'attendre étant donnée la taille importante du jeu de données et le petit nombre d'epochs d'entraînement.

Sur le test set, elle est environ **20 fois inférieure** à la meilleure perplexité des modèles n-gram (381.17), ce qui confirme ce à quoi l'on pouvait s'attendre : que le modèle Transformer est capable de modéliser le langage naturel bien mieux que le modèle naïf qu'est n-gram.

Performance qualitative

Exemples

❒ `Paramètres d'échantillonnage par défaut :`

"A l'age de 5 ans , elle invente" → "le premier tour de l' unite des etats-unis . elle est egalement connue pour ses elections a paris , et se retrouve dans le cinema en 1968 ... , qui a fait la connaissance d' un grand nombre de secondes et de residences d' argent , dont la ville est la premiere et la plus grande virginie de son pere"

> "les scientifiques furent extremement surpris de découvrir" → ". le siege de l' alphabet est en fait par un moulle .. le patron est de la premiere fois a une nouvelle incendinale du nord de la commune de saint-louis-sur-savoie-et-saint-jean de montreal (2850-1789) .) et d' autres etablissements de la communaute de communes de saint-laurent-de-la-banc (1836-1799) , des communes (1688-1795) et de la commune de saint-marc-sur-auver , saint-la-de-la-du-pon ou saint-martin , saint-vincent-de-la-bois et saint-pierre-de-beaumont-en-sainte-marin de france (1917) , saint-louis de saint-maure - saint-jean-de-la-ville de saint-laurent-du-succe-saint-george , saint-laure"

❒ `Température de 0.2`

> "A l'age de 5 ans , elle invente" → " , a l' age de 10 ans , ou elle est reglee a l' universite de l' universite de paris , et a la suite de la somme de la ville de saint-denis , et a la suite de la fusion de l' universite de montreal . elle est elue a la suite de la saison 2015-2017 et est en finale par la ligue "

❒ `Température de 4`

> "A l'age de 5 ans , elle invente" → " (1901) .-260 et se retire - la capable des etats de lyonnaise du quet , et la france , en italie , dans les colonistantes du congregage , au sud et au nordis le terroie de montpelles , et la reine du royaume de maugunchane a ete fondee en 1948 . il y ava"

La qualité des sorties est plutôt bonne. La température produit l'effet escompté : une basse température produit des échantillons plus corrects syntaxiquement, et une haute température produit des résultats plus expérimentaux, parfois même des mots inexistants.

Mode-Collapse

Une remarque intéressante est la présence de **mode-collapse** dans les sorties du modèle. Ce terme anglais désigne le phénomène par lequel un modèle génératif (comme un modèle générant du texte, des images, du son...) peut se focaliser sur une petite partie des données et se spécialiser dedans. Ici, le modèle se met très vite à parler de l'histoire des communes françaises, particulièrement lorsque la température est basse.

Ce problème arrive particulièrement souvent chez les GAN (Generative Adversarial Networks) car dans la version basique de cette architecture, le modèle génératif a une fonction de perte faite uniquement pour encourager le réalisme des sorties, mais pas leur diversité.

Il est plus surprenant qu'il arrive dans le cas de ce modèle. C'est un cas clair d'*underfitting* qui montre que le modèle pourrait bénéficier d'un temps d'entraînement plus long.

4.4 Résultats PyTorch (Word Encoding)

Le Transformer codé en TensorFlow et décrit précédemment utilise les Subwords comme méthode d’encodage. Une approche différente a été choisie pour le développement du Transformer en PyTorch. En effet, celui-ci exploite comme encodage des mots entiers (Word Level). De la même manière, le modèle n-gram décrit dans la suite sera basé sur une représentation des mots entiers pour avoir un élément de comparaison cohérent.

Le vocabulaire utilisé par les modèles n-gram et le modèle transformer est le même : il correspond aux 30.000 mots les plus fréquents dans les données d’entraînement.

Une telle taille de vocabulaire permet de couvrir suffisamment de mots du langage courant (avec quelques éléments propres à Wikipedia comme une plus grande utilisation de dates). Néanmoins, cette taille de vocabulaire accroît considérablement la taille du transformer par rapport à un modèle basé sur les Subwords (vocabulaire de quelques milliers de subwords), ce qui rend l’apprentissage plus long. Il y a un arbitrage à trouver entre meilleure représentation de la langue et temps d’apprentissage plus long.

Les calculs de perplexités sont effectués sur les mêmes données après avoir entraîné le modèle transformer comme les modèles n-gram sur les mêmes textes.

4.4.1 N-Gram

Performance quantitative

Les perplexités obtenues par le modèle n-gram sont, en fonction du paramètre n :

n	train	test
2	79.6	89.1
3	23.9	98.8
4	6.12	323.3

De la même manière que dans le cas du Subwords encoding précédent, on voit que si les performances sur les données d’entrainement augmentent avec n, les performances sur les données de tests suivent une tendance inverse. Les modèles ont clairement surappris pour $n > 2$.

Performance qualitative

Voici quelques extraits générés par le meilleur modèle n-gram parmi les précédents, soit pour $n = 2$.

❒ `Génération Greedy`

”A l’age de 5 ans , elle invente” → ” par le <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> ,”

❒ `Génération Sample (top`$_{k=5}$`)`

La génération sample évite les boucles et réduit les problèmes d’overfitting, comme expliqué plus haut dans ce rapport.

”A l’age de 5 ans , elle invente” → ”l et a l ’ ete un monument aux mains nues (v. smith et <unk> est , par un ecrivain et blanchiment d ’ eglise est l ’ ukraine alimente l ’

experience de poursuite penale . huit des carottes de mettre en 1536 . en effet stereo hearts , elle etait de sensibilisation du championnat du capitaine de bilan) consacre a <unk> <unk> (en 1212 christian <unk> , <unk> tt devait permettre de zurich en france des municipales de marseille . achille sur un poste de <unk> et petit-fils mineur pour inspecter le danemark publie en 2008 ”

❒ `Génération Sample (top`$_{k=5}$`) sans le caractère <unk>`

Malgré une taille de vocabulaire de 30.000, le nombre de mots qui n'appartiennent pas au vocabulaire reste très important et le modèle tend donc à attribuer une probabilité relativement forte au token ¡unk¿ associé à ces mots. Pour y remédier au moment de générer un texte qui fasse sens, on peut simplement mettre à zéro le poids associé à ce token, comme nous l'avons fait pour la génération ci-dessous :

”A l'age de 5 ans , elle invente” $\rightarrow$ ” a venir ;)”

4.4.2 Transformer

Hyperparamètres Notre modèle a pour hyperparamètres :

- $N = 2$
- $d_{model} = 64$
- $H = 8$
- $max_length = 8$
- $vocab\_size = 30000$
- `ff_hidden_size` $= 256$
- `n_epochs` $= 4$
- Optimizer : Adam
- `learning_rate` $= 0.01$
- `batch_size` $= 512$

Le code a été écrit de manière à pouvoir l'exécuter sur GPU. La mémoire de notre carte graphique (une GTX 1050) étant insuffisante pour un modèle basé sur un vocabulaire supérieur à 20 000 mots, nous avons eu recours à Google Colab.

Nous avons entraîné le transformer sur 90% du dataset sur 6 epochs en enregistrant le modèle à chaque epoch parcourue. L'entraînement sur une epoch dure une quarantaine de minutes.

L'erreur de test la plus faible a été obtenue pour le modèle entraîné sur 4 epochs.

Performance quantitative

La perplexité obtenue par le Transformer est :

- `train` : 27.7
- `test` : 26.4

On peut faire des observations similaires au modèle TensorFow :

La perplexité n'est pas meilleure sur le training set que sur le test set, ce qui confirme l'absence d'*overfitting* à laquelle on pouvait s'attendre étant donnée la taille importante du jeu de données et le petit nombre d'epochs d'entraînement.

Sur le test set, elle est bien inférieure à la meilleure perplexité des modèles n-gram, ce qui confirme ce à quoi l'on pouvait s'attendre : que le modèle Transformer est capable de modéliser le langage naturel bien mieux que le modèle naïf qu'est n-gram.

Performance qualitative

Exemples

❐ `Génération Greedy`

"A l'age de 5 ans , elle invente" → "de la <unk> . il est aussi un homme politique francais de la ville de <unk> . il est nomme en <unk> , le <unk> et le <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk> , <unk>"

❐ `Génération Sample (top`$_{k=5}$`)`

"A l'age de 5 ans , elle invente" → "des " <unk> " . le film est un homme politique allemand de <unk> et la region de <unk> . le premier album du monde de la <unk> , il s ' agit d ' une nouvelle part entiere de la ville de <unk> . il est classe par des autres nations unies , qui est un footballeur international francais de l ' academie de l ' art de <unk> , en particulier , il devient membre de la <unk> et le <unk> et les <unk> . il se presente dans la meme ecole d ' art , le roi"

❐ `Génération Sample (top`$_{k=5}$`) sans le caractère <unk>`

"A l'age de 5 ans , elle invente" → "de la ville de montreal . il est nomme au cours du mois de septembre 2017 , le premier tour d ' art et de l ' universite de la ville de lyon . le groupe est le " grand prix " . en 2007 , les deux premieres mentions et des autres communes d ' un nouveau record du monde de l ' equipe de l ' equipe du monde de hockey . le film , la commune se situe dans le departement de la province de l ' ouest (" ") et " de " ,"

Ici encore, bien qu'elles ne fassent pas sens sur le long terme, les générations du transformer sont plus construites, suivent davantage les règles syntaxiques que les générations des modèles n-gram.

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.