Note technique sur le traitement de langage naturel (TLN)
Ministère de la Transition Ecologique
CGDD - Ecolab

Théo Roudil-Valentin*

Une brève et humble explication théorique des modèles-frontières en *Natural Language Processing*

1 Les modèles de *Neural Machine Translation* ayant mené au modèle BERT, et à son efficacité

1.1 Le dépassement des approches *Bag-of-Words* : le *Word Embedding*

Les modèles de *Word Embedding* (vectorisation de mots) permettent de mettre des mots sous forme de vecteurs. C'est le cas par exemple du modèle canonique *Word2Vec* de Mikolov et al. [2013b,a] ou encore de *GloVe* de Pennington et al. [2014]. Ces modèles permettent donc de projeter des mots dans des espaces de dimension $d \in \mathbb{N}^{*+}$. Par exemple, pour le modèle de Mikolov, il est habituel de choisir $d = 300$. Ce qui revient à dire que chaque mot est représenté par un vecteur de 300 chiffres. C'est modèles sont intéressants pour, notamment, deux raisons. Ils s'éloignent des représentations standards type *Bag-of-Words* (sac de mots), car ils analysent les mots dans leur contexte et non de manière isolée. Ensuite, grâce à un d élevé, ils permettent de garder beaucoup d'information pour chaque mot, et ainsi avoir une richesse sémantique sous formes de vecteurs. Mais ces modèles ont aussi des inconvénients : ils ne peuvent pas faire de la représentation vectorielle de **phrases** (ou *Sentence embedding*).

1.2 L'apport de l'attention et de la bi-directionnalité

En effet, pour cela, il convient d'élargir le modèle pour qu'il prenne en compte l'ensemble de la phrase, des modèles spécialisés dans la vectorisation de phrase ont été proposés [Le and Mikolov, 2014], et cela pose de nouveaux problèmes. Un des problèmes majeur est *the curse of sentence length* ou la malédiction de la longueur de phrase. Plus la phase est grande, plus le modèle devra sacrifier d'information dans la projection de la phrase dans un espace de dimension donnée, comme observé par Cho et al. [2014]. Le modèle standard de ce type est le *RNN Encoder-Decoder* : un encodeur lit la séquence d'entrée $\mathbf{x} = (x_1, ..., x_{T_x})$ et la transforme en un vecteur c, de dimension fixe. L'approche la plus répandue est d'utiliser un RNN ($RecurrentNeuralNetwork$) :

$$h_t = f(x_t, h_{t-1})$$

*theo.roudil-valentin@ensae.fr

et

$$c = q\left(\{h_1, ..., h_{T_x}\}\right)$$

avec $h_t \in \mathbb{R}^n$, une couche/neurone caché à l'instant t, et c un vecteur généré par la séquence de ces neurones cachés. f et q des fonctions non-linéaires. Le décodeur est entraîné à prédire le prochain mot $y_{t'}$ via le vecteur de contexte c ainsi que les mots prédits précédemment. Le décodeur tente de définir une probabilité pour une traduction $\mathbf{y}$, donc une séquence $(y_1, ..., y_{T_y})$, en décomposant la probabilité jointe de manière ordonnée :

$$p(\mathbf{y}) = \prod_{t=1}^{T} p\left(y_t | \{y_1, ..., y_{t-1}\}, c\right) \tag{1}$$

De même en utilisant à nouveau un RNN, chaque probabilité est définie comme :

$$p\left(y_t | \{y_1, ..., y_{t-1}\}, c\right) = g\left(y_{t-1}, s_t, c\right) \tag{2}$$

avec g une fonction non-linéaire, potentiellement multi-couche, qui donne donc la probabilité du mot y_t, et s_t la couche cachée du RNN (cette partie est largement reprise de l'article [Bahdanau et al., 2014], dont l'effort de clarté est apprécié).
Pour dépasser ce problème, Bahdanau et al. [2014] proposent une nouvelle architecture de modèles permettant de gérer les phrases longues. La première modification d'ampleur, est la modification du vecteur de contexte : celui-ci va dépendre du mot cible de sorti. Autrement dit, pour chaque y_i il y a un vecteur de contexte c_i associé. Chaque c_i dépend de la séquence de couche cachée (*annotations*) $(h_1, ..., h_{T_x})$, dont chaque élément contient des informations sur l'ensemble de la séquence $\mathbf{x}$ mais avec une concentration d'informations sur les mots proches du i-ème mot. Ainsi la probabilité est modifiée ainsi :

$$p(y_i | y_1, ..., y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i)$$

avec $s_i = f(s_{i-1}, y_{i-1}, c_i)$.
Le vecteur de contexte est calculé ainsi :

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{3}$$

ainsi, les c_i peuvent être considérés comme des *annotations espérées* (*expected annotation*), avec chaque poids comme probabilité que le mot cible y_i soit ajusté (juste) pour la traduction du mot x_j. Ainsi, le vecteur contexte est bien l'annotation espérée, somme de chaque annotation pondérée par la probabilité d'alignement. Chaque poids est modélisé comme suit :

$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T_x} exp(e_{ik})}$$

et représente l'importance de l'annotation j par rapport à la couche cachée précédente, s_{i-1}, dans la sélection de la nouvelle couche cachée s_i et donc du mot final y_i. C'est la modélisation du principe d'**attention**. Le décodeur, qui génère donc la séquence de sortie, décide, seul, des parties de la séquence d'entrée auxquelles il doit faire attention, via cette modélisation plus flexible. Ce qui permet cette attention, c'est la modélisation des e_{ij} :

$$e_{ij} = a\left(s_{i-1}, h_j\right) = \omega^T \sigma\left(W^{(1)} s_{i-1} + W^{(2)} h_j\right) \tag{4}$$

avec ω un vecteur de poids et σ une fonction d'activation, ici la fonction d'attention est additive [Shen et al., 2018]. a est un modèle d'*alignement* de type *Feedforward Neural Network (FNN)* : qui va produire un score représentant à quel point les intrants autour de j et l'extrant en position i (en sortie, donc) sont cohérents. Ce n'est pas le mot y_i qui est utilisé, mais la couche cachée précédente s_{i-1}. Ce processus va donc permettre de produire le poids correspondant α_{ij}, qui va lui-même influencer le vecteur de contexte c_i et donc la prochaine couche cachée s_i qui va elle-même produire le mot cible y_i. Il y a donc bien une idée de contexte, de manière bi-directionnelle, comme détaillée plus bas. De plus, c'est véritablement cette fonction a qui permet de relâcher l'obligation d'une dimension fixe pour l'encodage de l'information. C'est cette fonction qui modélise véritablement le principe d'attention : elle calcule un score de *proximité* entre le vecteur s_{i-1} et h_j, qui va ensuite servir de poids pour calculer la probabilité que cet input soit important dans l'information du mot sélectionné. Le modèle d'alignement n'est plus une variable latente, comme dans l'approche précédente, mais bien quelque chose qui va être entraîné et optimisé avec le reste.
La deuxième importante modification est la **bi-directionnalité**. Dans le modèle précédent, le RNN lit la séquence d'entrée dans l'ordre, du x_1 jusqu'à x_{T_x}. Donc celui-ci ne prend pas en compte les mots qui suivent un mot considéré. La nouvelle approche au contraire utilise un réseau de neurones récurrents bi-directionnel (BiRNN, proposé par Schuster and Paliwal [1997]), qui permet justement de prendre en compte les mots qui suivent.
Ce type de réseau est constitué de deux réseaux, l'un *vers l'avant* l'autre *vers l'arrière*. La différence est donc le sens dans lequel le réseau va lire la séquence d'entrée. Le premier, $\overrightarrow{f}$, lit la séquence $\mathbf{x}$ de gauche à droite $(x_1, ..., x_{T_x})$; le deuxième, $\overleftarrow{f}$, la lit dans l'autre sens $(x_{T_x}, ..., x_1)$.Pour chaque intrant x_j de la séquence $\mathbf{x}$, on a donc deux neurones cachés : $\overleftarrow{h}_j$ et $\overrightarrow{h}_j$. Ces deux neurones cachés vont ensuite être pris ensemble pour créer une seule couche h_j :

$$h_j = \left[\overrightarrow{h}_j^T; \overleftarrow{h}_j^T\right]^T$$

La couche cachée finale contient donc les informations sur les mots passés et à venir : c'est le côté bi-directionnel de l'approche.
Les auteurs [Bahdanau et al., 2014] montrent que leur approche surpasse les précédentes approches, et notamment sur les phrases longues, qui était une des grandes difficultés à dépasser.

1.3 Le *Transformer* : l'attention comme principe

Un des problèmes de ces modèles, est leur côté séquentiel : les couches intérieurs h_j doivent être calculées les unes après les autres, ainsi que les couches intérieures du décodeur. En regardant attentivement la structure du modèle, la séquentialité saute aux yeux (regardez les indices). C'est une véritable barrière à la parallélisation et donc un frein à la réduction du coût de calcul. Vaswani et al. [2017] proposent alors une nouvelle architecture, totalement basée sur le principe d'attention qui permet de largement réduire le coût de calcul, en utilisant la capacité de l'attention à produire des liens de dépendance à travers toutes les données sans séquentialité. En effet, certains modèles séquentiels ont tenté de diminuer leur coût de calcul, notamment en utilisant les réseaux convolutifs comme fondation : mais cela pose problème pour obtenir des informations de dépendance sur des données lointaines. Le coût associé grandissant avec la distance. C'est pourquoi ces auteurs proposent le **Transformer** qui permet d'outrepasser ces limites tout en maintenant une précision élevée via l'utilisation d'une nouvelle brique de construction le : **Multi-Head Attention**.
A l'opposé de [Bahdanau et al., 2014], qui utilisent une attention additive, Vaswani et al. [2017]

proposent une attention produit-normalisé (*scaled dot-product attention*), qui ajoute donc la normalisation à l'attention produit existante. Ainsi la fonction d'attention 4 est modifiée de la sorte :

$$a\left(q, k_j, \nu_j\right) = \sigma\left(\frac{q \times k_j}{\sqrt{d_k}}\right)\nu_j \tag{5}$$

où k_j, ν_j proviennent de la décomposition de h_j en clé-valeur, comme proposé parDaniluk et al. [2017], Miller et al. [2016], Mino et al. [2017], et $dim(k_j) = d_k$. La décomposition permet une plus grande flexibilité, où la clé permet le calcul des poids pour les valeurs, utilisées donc pour calculer à la fois la représentation du mot cible (prochain) ainsi que le vecteur de contexte. L'attention multiplicative est beaucoup plus performante en termes en puissance/coût de calcul, mais pour de grandes valeurs de d_k elle est battue en termes de précision. Pour empêcher la fonction d'activation σ d'aller dans des territoires où le gradient serait trop petit, les auteurs ont décidé de normaliser le produit par le carré de cette dimension, empêchant cette déviation. Si le gradient devient trop petit, alors cela empêche les poids d'être correctement modifiés [1]. Cette fonction va ensuite servir comme brique à leur nouvelle couche.
Au lien de ne faire le calcul qu'une seule fois sur ces trois vecteurs, les auteurs proposent de le faire h fois en utilisant les vecteurs projetés linéairement, donc h fois. Une fois ces h attentions calculées, ils vont les concaténer puis re-projeter linéairement. La fonction est donc de type :

$$\begin{aligned} Multihead\left(q, k_j, \nu_j\right) &= Concat\left(a_1, ...a_h\right) \times W^M \\ a_i &= a\left(q \times W_i^q, k_j \times W_i^k, \nu_j \times W_i^\nu\right) \end{aligned}$$

avec W^X la matrice de projection linéaire cohérente pour un X. Cette nouvelle couche va être combinée avec le principe de *self-attention*. Ce principe repose sur le fait de remplacer q par un vecteur appartenant à la séquence source $\mathbf{x}$. Il va donc relier des éléments d'un même séquence, par paires. Cela permet d'analyser des liens de dépendance entre différents inputs pour des distances petites et longues, précédemment modélisées par des réseaux convolutifs tout en ayant beaucoup moins de paramètres à optimiser et une vitesse de calcul supérieure [Vaswani et al., 2017, Shen et al., 2018]. Enfin, une couche de *FNN* est ajouté sur chaque couche précédente, de manière séparée et identique sur chaque couche. La dimension de l'intrant est fixée $d_{modle} = 512$ et la couche intérieure est $d_{ff} = 2048$. Comme le modèle ne contient plus ni de convolutions ni de récurrences, une façon de garder l'information sur l'ordre de la séquence est de rapporter de l'information, en ajoutant un vecteur de même dimension, indiquant les positions. Les auteurs utilisent pour cela une fonction de *positionnal encoding* qui permet de produire un vecteur de position.
L'encodeur est constitué de 6 couches, chacune identique aux autres, et chacune constituée des sous-couches présentées juste au-dessus. Une étape de connexion résiduelle est ajoutée [He et al., 2016]. C'est-à-dire que les couches sont connectées à l'input initial (via une somme), facilitant l'apprentissage des fonctions sous-jacentes (pour des précisions voir Annexes A.1). Le décodeur est constituée de 6 couches également mais une couche de normalisation est ajoutée, c'est-à-dire que l'input des couches cachées est normalisé pour que la distribution (donc de l'input) reste la même, et qu'ainsi l'entraînement soit plus rapide et efficace [Ba et al., 2016, Xu et al., 2019] (voir Annexes A.2 et A.3).

1. voir : vanishing gradient problem

2 BERT : Bidirectional Encoder Representations from Transfomers, le modèle-frontière en NLP

Maintenant que nous avons en tête toutes les briques pour construire le modèle BERT, nous pouvons rentrer dedans.

2.1 Le modèle BERT canonique

Le modèle BERT de Devlin et al. [2018] est entraîné d'abord à retrouver des mots masqués au sein de phrases (MLM pour *Masked Language Model*). Son entrainement est divisé en deux étapes : une de pré-entrainement (*pre-training*) avec des données non labelisées, et une seconde de réglage de précision (*fine-tuning*) avec des données labelisées cette fois. L'architecture de BERT repose en très grande partie sur le travail de Vaswani et al. [2017], c'est un ***Multi-Layer Bidirectional Transformer Encoder***. L'input est une combinaison de trois types de vecteurs : le *token embedding* donc la transformation vectorielle d'un mot plus ou moins transformé, ici en l'occurrence potentiellement coupé via le *WordPiece model* [Wu et al., 2016] ; d'un vecteur d'embedding de segment (à quel segment de phrase appartient le token) ; puis de position (position du mot dans l'input). L'input commence toujours par $[CLS]$ et les deux segments (phrases) sont séparés par $[SEP]$. Les trois vecteurs sont additionnés ensemble.
La première étape de pré-entrainement poursuit deux tâches simultanément.

2.2 *RoBERTa*

Liu et al. [2019]

2.3 Le modèle français : CamemBERT

Martin et al. [2019]

3 Utiliser les modèles BERT : *pre-training* et *fine-tuning*

Sun et al. [2019], Peters et al. [2019], Howard and Ruder [2018], Wada et al. [2020], Zhang et al. [2020]
Nous on a besoin de faire :

1. extractive text summarization
   (a) Adapter CamemBERT aux données des études d'impact environnemental (*further pre-training*)
   (b) Ajouter une couche d'*extractive summarization* (*fine-tuning*)
      i. Le truc c'est que j'ai des doutes sur la "non-supervisation" de BERT. Nous ne possédons par, pour chaque section, d'un résumé, nous n'avons pas d'échantillon d'entraînement. Donc j'ai peur du temps et de la complexité de l'adaptation de BERT à la non-supervisation.
      ii. Sinon, on peut juste entraîner BERT sur nos données et faire un algorithme non-supervisé sur les embeddings qui en sortent, ce qui revient plus ou moins au même conceptuellement, mais pas en pratique.
2. text similarity

Bon le problème c'est que je ne pense pas que CamemBERT soit ré-entraînable pour ce que l'on veut en faire. En effet, on veut quand même modifier une bonne partie de la structure pour notre objectif. Je ne suis pas sûr que cela ait un sens, en toute rigueur il faudrait récréer un modèle comme Xu et al. [2020]. Mais dans ce cas, on aurait besoin de **beaucoup** de capacité de calcul, ce qui n'est pas le cas.
Dans notre cas on a : un ensemble de documents $\mathcal{D} = \{D_i\}_{i=1,...,N_\mathcal{D}}$, avec $N_\mathcal{D} = Card(\mathcal{D})$. Chaque document D_i est subdivisé en sections, et on note s_i^j la section j du document i. On veut générer un résumé extractif de chaque section de chaque document, on a donc en tout $N_\mathcal{S} = \sum_{i=1}^{N} Card(D_i)$ sections.

$$\mathcal{D} = (D_1, ..., D_N) \tag{6}$$

$$D_i = \left(s_i^1, ..., s_i^{Card(D_i)}\right) \tag{7}$$

$$\mathcal{S} = \left(s_1^1, s_1^2, ..., s_1^{Card(D_1)}, s_2^1, ..., s_2^{Card(D_2)}, ..., s_N^1, ..., s_N^{Card(D_N)}\right) \tag{8}$$

Soit une section s_i^j, celle-ci est composée de phrases. C'est sur ces phrases que nous allons travailler.

$$s_i^j = \left(z_{ij}^1, ..., z_{ij}^{Card(s_i^j)}\right)$$

Nous avons déterminé nos espaces de départs. Il faut maintenant, en ayant en tête les inputs et outputs des modèles BERT et RoBERTa, réfléchir à comment modifier ces modèles pour les utiliser à bon escient dans notre cas. L'objectif est donc de chercher une fonction f telle que :

$$\begin{aligned} f \ : \ \mathcal{S} \ &\rightarrow \ \mathcal{S}^* \\ s_i^j \ &\mapsto \ s_{ij}^* \end{aligned}$$

avec $\mathcal{S}^* = \{s_{ij}^*\}_{i=1,...,N_\mathcal{D} \ j=1,...,N_{D_i}}$ et $s_{ij}^* = \{$La phrase résumant le mieux la section $s_i^j\}$. De plus comme nous travaillons sur du résumé extractif, nous savons que $\mathcal{S}^* \subset \mathcal{S}$. La tâche se résout donc à faire émerger les meilleurs phrases de l'ensemble de départ.
Une manière d'approximer f est de découper, comme cela est fait habituellement, la chose en deux étapes : une première de représentation vectorielle, puis une autre de classification. Par exemple, la première étape serait représentée par :

$$\begin{aligned} g \ : \ \mathcal{S} \ &\rightarrow \ \mathbb{R}^{Card(s_i^j) \times d} \\ s_i^j \ &\mapsto \ \mathbf{V}_i^j = \left[\nu\left(z_{ij}^1\right), ..., \nu\left(z_{ij}^{Card(s_{ij})}\right)\right] \end{aligned}$$

avec $d = dim\left(\nu\left(z_{ij}^k\right)\right), \forall k$. g est donc le réseau de neurone de *Transformers* empilés. Ensuite, la deuxième étape est représentée par :

$$\begin{aligned} h \ : \ \mathbb{R}^{Card(s_i^j) \times d} \ &\rightarrow \ \mathcal{S} \\ \mathbf{V}_i^j \ &\mapsto \ z_{ij}^* \end{aligned}$$

avec pour objectif $h\left(g\left(s_i^j\right)\right) = z_{ij}^* = s_{ij}^* = f\left(s_i^j\right)$. A priori, la fonction g est relativement *simple* à élaborer. En effet, des exemples de ré-entraînement de modèles RoBERTa existent, et sont donc utilisables pour notre cas. La question réside plutôt dans la fonction h qui nécessite, la plupart du temps, une supervisation. Or, comme il est indiqué plus haut : nous ne possédons pas cette richesse. Il faut donc élaborer une fonction h non-supervisé, qui soit capable de sélectionner

par elle-même, la meilleure phrase de la section. Un autre problème qui saute aux yeux est la taille du vecteur $\mathbf{V}_i^j$: celle-ci dépend directement du nombre de phrases dans la section ; qui a toutes les raisons du monde de ne pas être fixe à travers les sections.
Pour la fonction h : soit une section s_i^j dont la projection sémantique est $\mathbf{z}$, on cherche z_k tel que $d(\mathbf{z}, z_k) < \epsilon$ avec ϵ arbitrairement petit. La fonction d de distance est donc à définir. Autrement dit, on cherche la phrase au sein de la section qui représente au mieux l'espace sémantique de la section. Ainsi, nous avons deux problèmes : trouver $\mathbf{z}$, qui n'est clairement pas donné, peut-être pouvons-nous nous tourner vers du *document embedding* ; puis trouver z_k.

Idéalement nous pourrions utiliser la métrique ROUGE, mais encore faut-il avoir un résumé initialement.

Après la théorie, la pratique : introduction au code de *Deep Learning* et développement de notre algorithme

4 Mettre en place des modèles de *Deep Learning* via *PyTorch*

Par exemple, on prend la phrase : *'Projet d'implantation d'une centrale photovoltaïque au sol sur l'ancien aérodrome de Munchhouse (68)'*. Elle est représentée (encodée) par le vecteur de *tokens* suivant :

$$[5, 6298, 18, 12, 14300, 18, 12, 70, 2405, 16799, 36, 1024, 32, 17, 12, 2206, 21, 28979, 8, 5190, 8720, 12221, 38, 4389, 53, 6, 1, ..$$

de dimension 1×512. The input ids are often the only required parameters to be passed to the model as input. They are token indices, numerical representations of tokens building the sequences that will be used as input by the model. Le tokenizer (la fonction qui prend les mots et les transforme en bouts, tokens) va séparer les mots lui même en fonction du vocabulaire déjà présent dans le modèle. Son *attention mask tensor* est :

$$[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, ..., 0]$$

, toujours de la même dimension. Le vecteur *attention mask* apporte l'information de la taille initiale de la séquence d'entrée : tous les 0 dans le vecteur indiquent que les tokens de même index sont hors de la séquence, la phrase initiale est donc représentée par les 24 premiers tokens, dans notre cas, par les indices égaux à 1, dans le cas général. Ici la taille maximale d'une séquence est fixée à 512. Notre phrase contient 16 mots, qui sont ensuite découpés puis encodés, ce qui donne 24 tokens. Les 512-24 tokens sont donc là uniquement pour que le vecteur soit rempli. The attention mask is an optional argument used when batching sequences together. This argument indicates to the model which tokens should be attended to, and which should not.
Comme nous allons découper notre document en section : nous allons donner une liste de phrases à notre modèle. Le problème de la taille de la section se pose. En effet, si la section est trop grande, il faudrait peut-être couper la section en deux, car la dimension de l'embedding elle est fixe. Nous pouvons proposer une règle assez simple où, lorsque $dim(s_i^j) > \Sigma$, alors l'input devient s_i^{j1} et s_i^{j2}, chaque sous-section de taille $\frac{dim(s_i^j)}{2} < \Sigma$. Avec Σ un seuil à définir en fonction de la tokenization et de la dimension maximum de l'embedding.
Dans l'idée donc l'input serait de la forme suivante, en s'inspirant de [Liu and Lapata, 2019] :

$$\tilde{s}_i^j = [[CLS], phrase_1, [SEP], [CLS], phrase_2, [SEP], [CLS], phrase_3, [SEP] ... [SEP]]$$

avec $phrase_i = t_1, t_2, ...$ et t_i des tokens de WordPiece et du vocabulaire de départ. Notre vecteur de tokens embedding :

$$\mathbf{E} = [E_{CLS}, E_{t_1}, E_{t_2}, ..., E_{SEP}, E_{CLS}...]$$

un vecteur de délimitation de phrase, qui alterne donc entre A et B suivant le range de la phrase :

$$Seg = [Seg_A, Seg_A, Seg_A, ..., Seg_A, Seg_B...]$$

Puis un position embedding :

$$P = \left[P_1, P_2, P_3, ..., P_{dim(phrase_1)+1}, P_{dim(phrase_1)+2}\cdots\right]$$

Soit donc I, l'empilement de ces trois couches, on aurait :

$$g\left(I\right) = \tilde{I}$$

et donc au final, notre vecteur de sortie serait

$$\tilde{y}_i^j = h\left(\tilde{I}\right) = \left[\alpha_1, \alpha_2, ..., \alpha_{dim(s_i^j)}\right]$$

avec $\sum_{i=1}^{dim(s_i^j)} \alpha_i = 1$, donc un score d'importance pour chaque phrase dans la section.
Un problème est : si la section n'est composée que de longues phrases, comment synthétiser la section ? Il faut trouver un moyen de permettre au réseau de savoir quels inputs sont liés.

A Annexes

A.1 Le *Residual Learning*

A.2 Le *Batch Normalization*

A.3 Le *Layer Normalization*

Références

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

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv :1409.0473, 2014.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation : Encoder-decoder approaches. arXiv preprint arXiv :1409.1259, 2014.

Michał Daniluk, Tim Rocktäschel, Johannes Welbl, and Sebastian Riedel. Frustratingly short attention spans in neural language modeling. arXiv preprint arXiv :1702.04521, 2017.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert : Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv :1810.04805, 2018.

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.

Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. arXiv preprint arXiv :1801.06146, 2018.

Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In International conference on machine learning, pages 1188–1196. PMLR, 2014.

Yang Liu and Mirella Lapata. Text summarization with pretrained encoders. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 3730–3740, Hong Kong, China, November 2019. Association for Computational Linguistics. doi : 10.18653/v1/D19-1387. URL https://www.aclweb.org/anthology/D19-1387.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta : A robustly optimized bert pretraining approach. arXiv preprint arXiv :1907.11692, 2019.

Louis Martin, Benjamin Muller, Pedro Javier Ortiz Suárez, Yoann Dupont, Laurent Romary, Éric Villemonte de la Clergerie, Djamé Seddah, and Benoît Sagot. Camembert : a tasty french language model. arXiv preprint arXiv :1911.03894, 2019.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. arXiv preprint arXiv :1301.3781, 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. arXiv preprint arXiv :1310.4546, 2013b.

Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. arXiv preprint arXiv :1606.03126, 2016.

Hideya Mino, Masao Utiyama, Eiichiro Sumita, and Takenobu Tokunaga. Key-value attention mechanism for neural machine translation. In Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2 : Short Papers), pages 290–295, 2017.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove : Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pages 1532–1543, 2014.

Matthew E Peters, Sebastian Ruder, and Noah A Smith. To tune or not to tune ? adapting pretrained representations to diverse tasks. arXiv preprint arXiv :1903.05987, 2019.

Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. IEEE transactions on Signal Processing, 45(11) :2673–2681, 1997.

Tao Shen, Tianyi Zhou, Guodong Long, Jing Jiang, Shirui Pan, and Chengqi Zhang. Disan : Directional self-attention network for rnn/cnn-free language understanding. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 32, 2018.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification ? In China National Conference on Chinese Computational Linguistics, pages 194–206. Springer, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. arXiv preprint arXiv :1706.03762, 2017.

Shoya Wada, Toshihiro Takeda, Shiro Manabe, Shozo Konishi, Jun Kamohara, and Yasushi Matsumura. A pre-training technique to localize medical bert and enhance biobert. arXiv preprint arXiv :2005.07202, 2020.

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.

Jingjing Xu, Xu Sun, Zhiyuan Zhang, Guangxiang Zhao, and Junyang Lin. Understanding and improving layer normalization. arXiv preprint arXiv :1911.07013, 2019.

Shusheng Xu, Xingxing Zhang, Yi Wu, Furu Wei, and Ming Zhou. Unsupervised extractive summarization by pre-training hierarchical transformers. arXiv preprint arXiv :2010.08242, 2020.

Ruixue Zhang, Wei Yang, Luyun Lin, Zhengkai Tu, Yuqing Xie, Zihang Fu, Yuhao Xie, Luchen Tan, Kun Xiong, and Jimmy Lin. Rapid adaptation of bert for information extraction on domain-specific business documents. arXiv preprint arXiv :2002.01861, 2020.