Formation DS-IML de marc.lefevre@noos.fr

Rapport PROJET-7

Classification de textes avec BERT

Mission :

Initialement, et contrairement aux précédents, le projet dont voici le rapport n'était pas adossé à une mission précise mais défini par une démarche à accomplir. Cette démarche était de :

- effectuer une veille informationnelle dans le domaine du machine learning,
- repérer une nouveauté ou un progrès significatif susceptible d'être mis à profit,
- implémenter cette nouveauté dans un cas concret.

Choix du sujet :

Dans ce cadre, notre choix s'est porté sur le modèle **BERT**, et ses déclinaisons, qui ont bouleversé le **NLP** depuis leur apparition fin 2018.

Nous avons découvert ces modèles lors de nos nombreuses recherches à l'époque où nous étions attelés au projet 5 de classification de questions Stack Overflow.

A en croire ces lectures, les mystérieux mais sympathiques modèles BERT (qu'on représente par le célèbre personnage homonyme de l'émission pour enfants ***1, rue Sésame***) constituaient ce qui se faisait de mieux en NLP (machine learning appliqué au langage) et pouvaient servir à de la classification.

Dès lors, le sujet de notre projet 7 était tout trouvé : nous poursuivrions notre projet 5 dans le but, avec l'aide de BERT, d'améliorer les résultats que nous avions obtenus. Le cas échéant, cela nous permettrait d'établir le bien fondé de notre démarche depuis le début, en plus de nous familiariser avec un nouvel outil aussi puissant qu'intéressant.

Généralités :

Avant de rentrer dans le vif du sujet, encore quelques précisions.

BERT est un acronyme qui signifie Bidirectionnal Encoder Representation (from) Transformer. Cette désignation renvoie aussi bien à une technique ou à une architecture de modèle (elle-même directement issue de l'architecture « transformer » apparue quelques mois auparavant), qu'au premier modèle BERT créé par une équipe de chercheurs Google.

Aussitôt apparu, BERT a été customisé ou « tweaké » dans tous les sens, pour donner naissance à de nombreux clones plus ou moins proches (des centaines voire plus, rien que dans le recensement de la société Hugging Face). On peut donc aussi parler de « modèles BERT ».

Autre point, nous nous sommes servis de BERT dans une optique de classification, mais cette technologie, dont l'apport est de fournir une représentation mathématique de données texte plus riche que ce qui existait (comme si on disposait d'un nouveau et meilleur Word Embedding…) peut être utilisée dans toutes sortes de taches de NLP.

Enfin, voici maintenant deux exemples montrant que quand on dit que BERT, et les transformers dont il est issu, ont bouleversé le NLP, il ne s'agit pas d'un vain mot.

Premier exemple, avec un générateur de texte automatique fonctionnant à partir du modèle GPT2 de la société OpenAI (il ne s'agit pas directement de BERT, mais d'un « voisin » très proche…), utilisable à cette adresse :

https://deepai.org/machine-learning-model/text-generator

L'idée est de rentrer quelques mots et de laisser le modèle générer la suite. Et même si le résultat est assez décousu, il est souvent fascinant de voir de quelle manière le modèle s'inspire et brode à partir des mots qu'on lui a donnés.

A mysterious vessel appeared on the horizon. It turned out, from an unknown source, to be a small ship. "Who am I?" Captain Archer asked while holding in fear what might happen to him. "Captain." The voice suddenly became frantic as it shouted, "The Black Pearl, the Dragon!" When the ship came to rest, it took off, bringing with it the remains of an entire crew, with each individual having lost a body. "We must know more about this," Captain Archer yelled, "we must find out who is in control of this craft. You may have some questions, I will answer them!" He stood up from the room and went down, the Captain yelling, "Bring us. Come out, come out!"

Le second exemple nécessite d'utiliser le moteur de recherche google francophone qui fonctionne avec CamenBERT (la version française de BERT !) depuis courant 2019.

Si on est attentif à la façon dont il répond aux requêtes, on réalise qu'on est loin désormais des résultats basés avant tout sur des proximités syntaxiques ou sémantiques. Si une requête est par exemple une question, le moteur la comprend et essaie d'y répondre…

Sommaire :

1. Retour sur les épisodes précédents

1.1. Rappel du projet 5

Puisqu'à l'occasion de ce projet 7 nous avons choisi de prolonger 5, commençons par rappeler dans les grandes lignes en quoi consistait ce dernier.

Il s'agissait de classification de texte, et pour être plus précis, nous devions créer un modèle capable de prédire correctement des tags attribuables à des questions posées sur le site d'entraide Stack Overflow.

Nous avions créé un dataset d'environ 25 000 questions (au total) comme celle-ci :

```
"How to talk to UDP sockets with HTML5? What I have :A C++ application server running, Ready to send data to client
which is supposed to a HTML5 page or app. What I want : Is there any way to communicate using udp port with HTML5 gi
ven both c++ server and HTML5 app are local to system ? What I know :  Because of Security Concern, JS doesn't allow
UDP port communication from browser.Have read in many places, Answer is no. But answers are old. Is the answer still
'NO' ? Is there any work-around possible ? Any lead is appreciated."
```

auxquelles le modèle devait attribuer un ou plusieurs tags parmi ceux-ci :

```
['android', 'angular', 'asp.net', 'c#', 'css', 'docker', 'flutter', 'google', 'html', 'ios', 'java', 'ja
vascript', 'jquery', 'laravel', 'misc', 'node.js', 'pandas', 'php', 'python', 'react', 'spring', 'sql',
'swift', 'typescript', 'visual']
```

C'était un problème de classification multilabel. Notre meilleur résultat avait été obtenu grâce à un réseau de neurones très simple entrainé à partir d'un word-embedding FastText « customisé » de nos données.

Dans la métrique choisie alors, qui était le F1 Score weighted, nous avions atteint le score de 0.69, qui était le score à battre avec l'aide de BERT lors de ce projet.

1.2. Une affaire de « représentation »

Lors du projet 5, nous avions pu voir qu'un enjeu important en NLP était celui de l'encodage des données texte, la qualité des modélisations étant liée à la pertinence et à la richesse des procédés d'encodage ou vectorisation choisis.

Nous avions testé les vectorisations TF et TF-IDF, ainsi que les word-embeddings Word2Vec et FastText. Nous avions même utilisé des « topic-scores » provenant d'une LDA.

Comme nous le verrons, ce qu'apporte BERT est une nouvelle et supposée meilleure représentation du langage. En ayant recours à lui nous avons voulu le vérifier et en profiter le cas échéant.

Dit autrement nous avons testé un nouveau procédé « d'encodage de texte ».

1.3. Le transfert learning, encore…

Si ce projet était la suite du projet 5, il faisait également grandement écho au projet 6 puisque nous avons mis en œuvre BERT en utilisant la méthode du transfert learning.

Les modèles que nous avons utilisés étaient un assemblage d'un modèle de type BERT, dont nous voulions les représentations de nos données textes, auquel nous avons « collé » une couche dense de sortie adaptée à notre objectif de classification multilabel.

Avec eux, nous avons procédé aux entrainements en deux temps. D'abord un entrainement juste de la couche de sortie (avec modèle BERT figé), puis un « fine-tuning » de la « partie BERT » du modèle en question (avec un learning rate diminué).

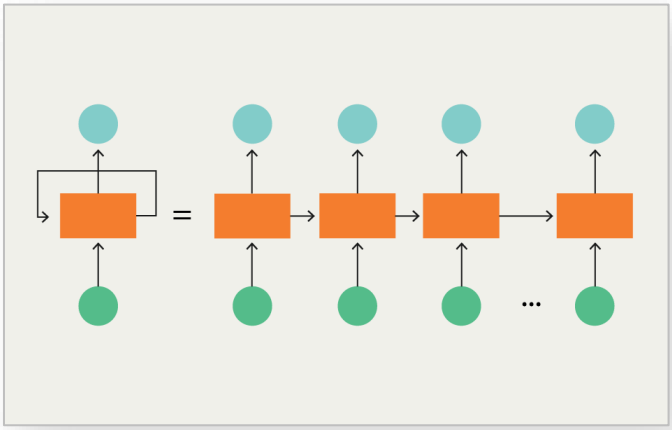
2. Genèse de BERT

Nous allons maintenant nous focaliser un moment sur l'histoire de BERT, afin de comprendre en quelques étapes clés ce qu'est et d'où vient ce modèle.

2.1. Le RNN, ou réseau de neurones récurrent

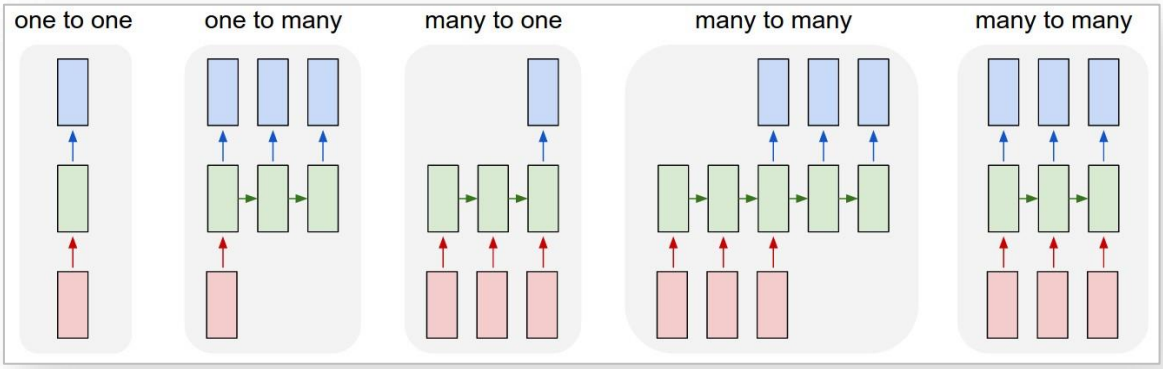
Au départ du processus qui aboutira à BERT, il y a eu le besoin éprouvé en deep learning de traiter de données non pas individuellement, comme avec un réseau neuronal classique, mais au sein d'une séquence.

Ce besoin était particulièrement prégnant dans des domaines où on voulait pouvoir prédire l'évolution de choses dans le temps, comme dans la finance avec par exemple le prix d'une action. Mais également dans le domaine qui nous intéresse, celui du langage, où les mots s'utilisent et font sens au milieu d'une séquence d'autres mots.
Le premier outil capable de traiter des données en séquences fut le **reccurent neural network** (ou RNN).



Sur cette illustration qui le montre en vues compactes et « dépliées », on voit qu'il s'agit d'un réseau de neurones muni d'une « boucle ». Une boucle qui réinjecte l'output du réseau d'un vecteur donné, en input du réseau lorsque lui est envoyé le vecteur suivant de la séquence.

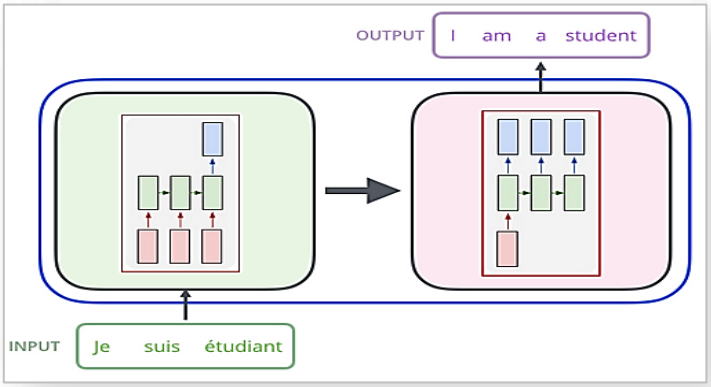
Chaque vecteur d'une séquence (sauf le premier…) est donc envoyé dans le réseau « modifié » par l'output du réseau lors de l'étape précédente (qu'on appelle « état caché »), et ainsi, les données sont traitées en fonction des éléments qui les précèdent dans une séquence.



Selon la modélisation qu'on souhaite faire, le RNN s'emploie dans plusieurs organisations ou architectures. Le « one to many » (un input, multiples outputs) peut servir à générer les légendes d'images, le « many to one » à de la classification de texte, le « many to many » à de la traduction.

2.2. Architecture encoder / decoder seq2seq

Pour la traduction automatique justement, est apparu avec le temps une architecture performante consistant à utiliser deux RNN à la suite l'un de l'autre, le premier en « many to one », le second en « one to many ».



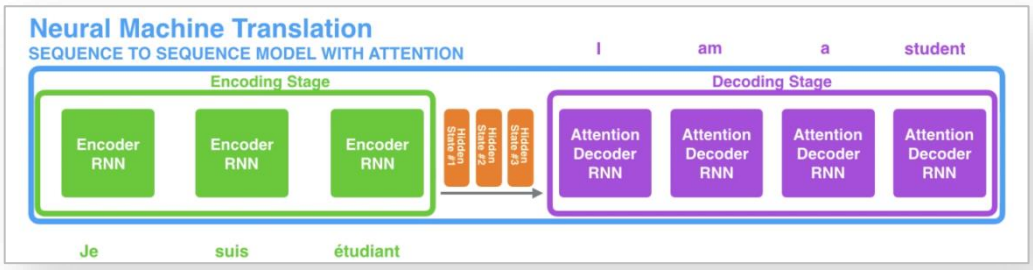
Le premier réseau matérialisait dans son dernier état caché une représentation des mots à traduire (d'où son nom d'encodeur). Représentation appelée « vecteur de contexte » que le second réseau allait en quelque sorte « décoder » dans la langue cible (d'où également son nom…).

Cette architecture souffrait cependant des défauts structurels des RNN, à savoir que ces réseaux ont une « mauvaise mémoire ». L'information ne persiste pas assez longtemps dans les états cachés successifs du réseau, si bien que le « vecteur de contexte » en bout de chaîne ne contient que peu ou plus d'information du début de la séquence.

De plus, dans ce système, cette mémoire déjà défaillante était unidirectionnelle, alors qu'après tout dans le langage, un mot peut dépendre autant de ce qui le précède que de ce qui le suit.

2.3. Architecture encodeur / decodeur avec attention

C'est le mécanisme de « l'attention » qui parvint à régler ces problèmes, et donna naissance, pour continuer dans la veine de nos modèles de traduction, aux NMT (Neural Machine Translation) représentées ci-dessous.



La NMT se démarquait de ses prédécesseures de deux manières. D'abord l'encodeur travaillait différemment, puis le décodeur était d'un nouveau type plus élaboré, le « décodeur avec attention ».

A partir de là, le mécanisme de l'attention était assez simple. A la place d'un unique vecteur de contexte comme avant, l'encodeur fournissait au décodeur

l'ensemble de ses états cachés. Le décodeur régurgitait ensuite séquentiellement chaque mot de la traduction à partir d'une combinaison de ces états cachés, combinaison qui était déterminée par entrainement.

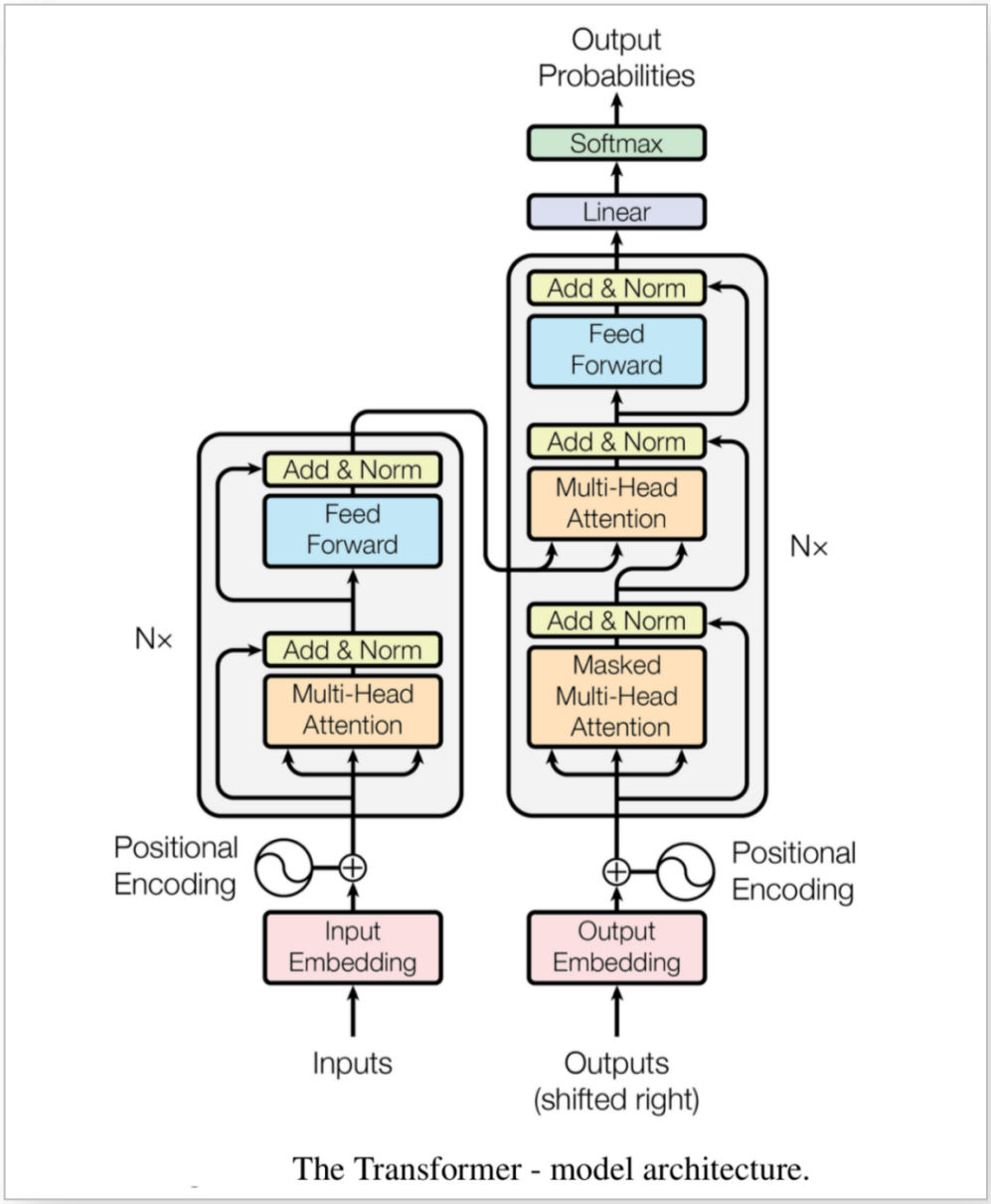
Grâce à ce système, où le décodeur reçoit les informations de chaque mot source et détermine par entraînement supervisé les liens entre eux et les mots à traduire, le réseau apprend à aller chercher l'information où qu'elle se trouve dans les données sources. C'est à dire qu'à chaque instant, le décodeur ne dépend plus d'un vecteur de contexte à la mémoire scabreuse et encodé uni-directionnellement. Mais à la place, il est maintenant capable de porter son « attention » partout là où il a déterminé qu'il le fallait.

Ces machines ont été au cœur des meilleurs modèles de traduction automatique jusqu'à il y a peu, avant d'être victime de leur succès. En effet, comme certaines limitations initiales étaient contournées, les utilisateurs ont cherché à les faire travailler sur des données de plus en plus conséquentes (longueur des textes…), ce qui a fini par faire ressortir un autre problème structurel des RNN : leur gourmandise en ressources informatiques, et leur incompatibilité avec toute parallélisation de calculs.

2.4. Le « transformer »

L'étape suivante fut l'apparition de l'architecture transformer, qui allait surmonter cette difficulté logistique et révolutionner le NLP à partir de fin 2017. Comme nous pouvons le voir sur l'illustration suivante, il s'agit d'un dispositif massif et très complexe à propos duquel nous allons juste souligner quelques points essentiels.

- Avec un transformer, il n'y a plus de fonctionnement séquentiel pénalisant comme avec les RNN. Tout est affaire ici de calculs matriciels et tensoriels pour lesquels un type de hardware récent appelé TPU fait merveille.
- Les différents blocs qu'on trouve dans un transformer reprennent de manière généralisée, étendue et plus complexe, le principe de l'attention rencontré plus tôt.
- Sur la gauche on a la partie « encoder » du transformer, à sa droite sa partie « decodeur » (au moins pour ça on est en terrain connu…).
- Dans un cadre de traduction, la partie encodeur construit une représentation extrêmement fine de la langue source dans ses blocs, la partie décodeur construit un autre type de représentation de la langue cible dans les siens.
- L'injection de l'output de l'encodeur dans les blocks du décodeur permet de créer des données de contextes très pointues à partir desquelles un bloc dense classique » peut renvoyer les meilleures traductions automatiques à ce jour.

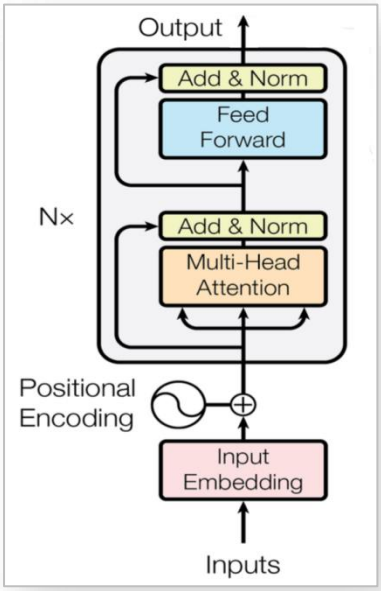


The Transformer - model architecture.

2.5. BERT

L'idée que dans un transformer, la partie encodeur avait pour fonction de construire une représentation des données sources, a immédiatement inspiré certains chercheurs de chez Google.

Ils ont imaginé qu'en utilisant habillement un tel encodeur, et en mettant donc à profit son système complexe d'attention, il y aurait peut-être possibilité d'obtenir des représentations de langues plus fines et plus riches que ce qui existait à l'époque (word2vec, etc…).



Et c'est ainsi que quelques mois après l'apparition des transformers naquit BERT, qui est au départ juste un encodeur de transformer. Sauf qu'il a été entrainé de manière non-supervisés, et sur une immense quantité de données, au moyen d'une double routine d'entraînement consistant à :

- deviner des mots masqués (routine MLM pour « masked language modeling »)
- prédire si deux phrases se suivent (routine NSP pour « next sentence prediction »).

BERT est en fin de compte un « détournement ciblé » de la technologie des transformers. Il s'agit d'un modèle pré-entrainé de manière générique, et pensé pour être ensuite utilisé et « ajusté »lors de projets précis.

3. BERT en pratique

3.1. Le texte tel quel

BERT est déroutant à plus d'un titre quand on le découvre et qu'on apprend à l'utiliser. Mais il y a au moins un élément qui ravira les personnes habituées à traiter du texte à n'en plus finir lorsqu'ils font du NLP : avec BERT, on prend le texte tel qu'il est.

C'est parfaitement cohérent si on a bien compris ce que sont BERT et les transformers, à savoir des outils qui viennent et qui baignent vraiment dans le langage. Mais au début cela surprend.

Ce fut d'autant plus notre cas que d'un point de vue rédactionnel, notre matériel de travail était particulièrement « exotique », et nous avions du mal à croire qu'on pouvait l'envoyer en l'état à n'importe quel tokenizer.

Pourtant nous avons testé et pouvons le confirmer. Même avec des textes très spéciaux comme ceux venant de Stack-Overflow, nous avons obtenu des résultats équivalents sinon meilleurs sans aucun traitement.

3.2. Un pre-processing particulier

Si avec BERT on s'épargne une part du travail habituel de NLP, il y a en revanche un pre-processing très balisé auquel on ne peut échapper.

En entrée, un modèle BERT attend un double tenseur (double dans notre cas, un triple si on travaille sur des relations entre deux phrases).

```
It's cool to have an exam in the middle of holydays !

tf.Tensor(
[[  101  1135   112   188  4348  1106  1138  1126 12211  1107  1103  2243
   1104  9371  6194  1116   106   102     0     0     0     0     0     0
      0     0     0     0     0     0]], shape=(1, 30), dtype=int32)

tf.Tensor([[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0]], shape=(1, 30), dtype=int32)
```

La taille de chaque input, en nombre d'éléments (tokens), est une constante fixée à l'avance. Si un input est plus long, il est tronqué, s'il est plus court, on l'accompagne, comme on peut le voir, d'un « padding » de chiffres « 0 ».

Dans l'exemple cette taille maximum est de 30, dans notre projet elle était de 512 (maximum standard), ce qui a quand-même entrainé des coupes pour 1.5% des questions les plus longues.

Pour rappel, un modèle BERT a été pré-entrainé et dispose de son propre vocabulaire (en l'occurrence de « sous-mots » et non de mots entiers). Le premier tenseur, qu'on appelle « input_ids », représente notre phrase tokenisée en sous-mots de ce vocabulaire, sauf que ceux-ci sont remplacés par une clé renvoyant à eux dans le dictionnaire du modèle.

A noter qu'il existe des tokens spéciaux. Ici les tokens 101 et 102 marquent par exemple les débuts et fins de phrase.

Le second tenseur est un masque indiquant au modèle de ne pas tenir compte du padding.

Cet encodage un peu spécial se fait en utilisant un tokenizer propre à chaque modèle qui est, si on peut dire, toujours « livré avec lui ».

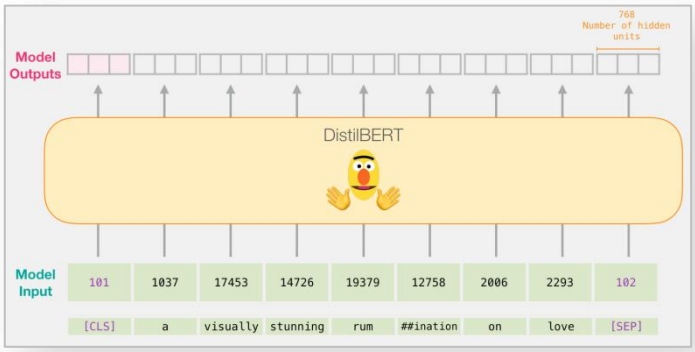
3.3. Transfert learning avec BERT

Nos modélisations ont toutes été faites en pratiquant le transfert learning à partir des différents modèles BERT utilisés. La procédure a été quasiment la même que lors du projet précédent.

Nous avons d'abord ajouté à chaque modèle pré-entrainé une couche de

classification adaptée à notre mission : une couche dense à 25 neurones (pour nos 25 classes) avec activation non pas « softmax » mais « sigmoïd » étant donné que nous étions dans une classification multilabel.

Ensuite nous avons entrainé juste la partie classification sur 50 epochs, avant de procéder à des « fine-tuning » de la partie BERT des modèles sur 20 epochs avec un learning rate faible de 1e-5.



Il y a juste eu une différence notable lors de « l'assemblage » des modèles. Comme illustré ci-dessus, l'output d'un modèle BERT est particulier en cela qu'il est constitué des représentations (de dimension 768) de chacun des tokens d'input.

Dans un cadre comme le nôtre, où nous avions besoin d'une représentation entière de nos questions (et non d'un token…), nous avons « branché » notre classificateur sur la première sortie du modèle, celle du token de début de phrase « CLS » (pour « classification ») qui par convention donne la représentation globale recherchée.

3.4. Qui dit BERT, dit TPU

Après avoir la plupart du temps, lors de notre formation, modélisé au moyen de nos CPU, et de GPU lors du projet précédent, nous avons cette fois utilisé des TPU accessibles sur Google Colab.

Un TPU (tensor processing unit) est un processeur particulier (ASICS), mis au point par Google, qui sert uniquement à produire du calcul tensoriel en masse, et qui était donc particulièrement adapté à la modélisation avec BERT.

Pour donner un ordre d'idée, en début de projet, nous avons lancé une modélisation DistilBERT avec CPU. Le temps indicatif fut de 4 heures par epoch. Avec TPU on retombait vers des durées de l'ordre de la dizaine de secondes…

A noter cependant que l'utilisation de TPU requiert un peu de code de configuration. Il faut d'abord relier le « kernel » de session Colab aux TPU disponibles. Il faut ensuite créer ses modèles dans le cadre d'une « stratégie » d'utilisation de ces TPU.

4. BERT en action

4.1. Modèles BERT utilisés

Au départ, nous étions dans l'idée de n'utiliser que deux modèles, DistilBERT, afin de prendre nos marques avec un modèle plus léger, puis BERT lui-même. Mais notre curiosité nous a poussés à tester également deux autres modèles, SciBERT et RoBERTa.

DistilBERT : Une version « light » de BERT, revendiquant 95% des performances de l'original avec 40% de paramètres en moins et en 60% plus rapide.

BERT : Le modèle précurseur a été utilisé dans sa première version « medium ».

SciBERT : Il s'agit de BERT « élargi » à la science et à la médecine après un entrainement supplémentaire sur un corpus de textes scientifiques. Vue la nature de nos données, nous imaginions que ce modèle pouvait disposer d'un vocabulaire plus adapté.

RoBERTa : Une version de BERT dont les auteurs estiment qu'elle a été mieux entrainée, en mono-tâche (uniquement le MLM) et avec un hyper-paramétrage mieux optimisé.

4.2. Résultats du dataset original

Cette première série de modélisations (qui devait être originalement la seule et qui correspond au premier notebook du projet) a été faite sur le dataset de texte non-traité de notre projet 5.

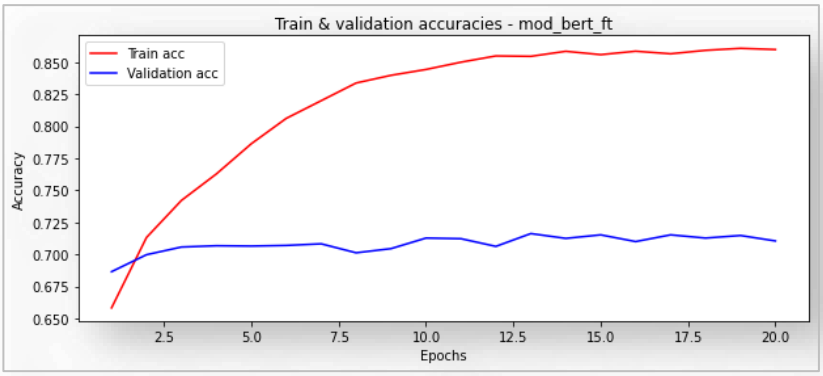
En guise de préliminaires nous avons vérifié, avec DistilBERT, que les modèles BERT fonctionnaient comme annoncé aussi bien à partir de texte dans leur état original plutôt qu'après avoir été plus ou moins « nettoyé » 'stopwords, chiffres, ponctuation, etc). C'était bien le cas.

Puis nous avons voulu voir, encore avec DistilBERT, si le fait d'utiliser les versions « cased » ou « uncased » des modèles avait une influence sur les résultats. N'observant pas de différence significative, nous avons opté pour les versions « uncased ».

Les résultats ont ensuite été les suivants (val_acc et f1_score pour les deux phases d'entrainement) :

	Training	Fine-Tuning
DistilBert	(0.573, 0.5)	(0.717, 0.78)
Bert	(0.55, 0.43)	(0.716, 0.78)
SciBert	(0.479, 0.34)	(0.719, 0.78)
Roberta	(0.521, 0.25)	(0.724, 0.78)

Les courbes de modélisation, lors du « fine-tuning » des modèles, ont toutes été de ce type (ici avec BERT).



Ces résultats ont suscité deux observations :

Avec des F1_Score (weighted…) de 0.78, nous avions battu notre score du projet 5 de 9 points. Indéniablement, en termes de résultats, l'utilisation de BERT était un succès, ce qui validait la démarche et les choix que nous avions faits.

Ces résultats nous ont aussi donné l'envie de faire mieux, car les plateaux dessinés par les courbes (overfitting – difficulté des modèles à généraliser…) pouvaient laisser penser que nous avions encore de la marge de progression, notamment en travaillant avec plus de données.

Du coup, avec l'espoir d'au minimum passer la barre des 0.8, nous avons prolongé les choses.

En reprenant globalement la démarche décrite lors du projet 5, nous avons reconstruit « from scratch » (procédure objet du second notebook de projet) un nouveau dataset de deux fois la taille de l'original (40 000 questions d'entrainement, 10 000 de test). Dataset à partir duquel nous avons procédé à une deuxième série de modélisations.

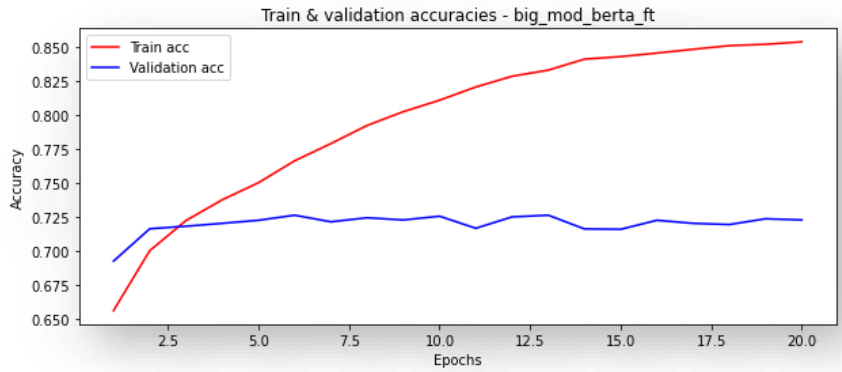
4.3. Résultats dataset élargi

Sur ce nouveau dataset, voici les résultats obtenus.

	Training	Fine-Tuning
DistilBert	(0.572, 0.51)	(0.721, 0.78)
Bert	(0.538, 0.44)	(0.726, 0.79)
SciBert	(0.493, 0.32)	(0.724, 0.79)
Roberta	(0.547, 0.29)	(0.726, 0.79)

Malgré nos attentes, nos efforts ne nous ont pas permis de faire mieux que 0,79 au F1_Score (même pas avec DistilBERT…). Juste un point de gagné, c'était décevant.

De plus, les courbes de modélisations sont restées de la même nature que les précédentes.



Ce que nous imaginions être un « plateau » était en fait un « plafond ».

5. Conclusion

5.1. Une impression d'impasse

Notre tentative manquée (du moins par rapport à nos attentes) d'améliorer nos premiers résultats nous a forcés à pousser notre réflexion, quant à nos leviers d'actions restants, et nous avons eu l'impression assez inédite d'être sans solution.

Habituellement, en machine learning, on peut :

- Améliorer le travail et le pre-processing des données. Mais dans notre cas, puisque BERT préfère les textes originaux, ce n'était pas une option.
- Monter en grade quant aux outils employés, mais BERT était déjà sensé être ce qui se fait de mieux…

- Gagner en performance en retardant l'overfitting d'un modèle. Mais ici le principal moyen dont nous disposions était l'agrandissement du dataset, et celui-ci n'a permi qu'une amélioration à la marge des résultats.…
  (…)

Il y a des choses que nous aurions pu faire différemment, mais nous ne voyons pas en quoi elles auraient pu améliorer notre score.

La seule option restante était d'essayer d'autres modèles, mais là encore, ceux que nous avons pris font partie du haut du panier des modèles BERT existant, et nous avons du mal à croire, vu la proximité des résultats obtenus, que d'autres modèles auraient significativement changé les choses.

5.2. Une seule solution très compliquée…

Avec un peu de recul, nous pensons que nous avons rencontré avec BERT les mêmes problèmes que ceux rencontrés avec d'autres outils pré-entrainés lors du projet 5 (word2vec…) : l'inadéquation entre des modèles « généralistes » et la nature très spécialisée voire uement « geek » de nos données.

Lors de celui-ci, des modèles de word-embedding ignoraient de l'ordre de 75% du vocabulaire de nos données.

Ici, même si techniquement cela se passe d'une manière différente, et que les tokenizers des modèles BERT essayent d'atténuer le phénomène, la perte d'information reste importante. Par exemple, nous avons pu voir qu'un mot important comme « mysql » était décomposé en « my » (le possessif) et les lettres « s », « q » et « l ».

Du coup, nous pensons que pour vraiment faire mieux, il nous aurait fallu un modèle pré-entrainé de manière ciblée sur notre type de texte. Comme ce qui avait été fait pour SciBERT, mais dans le domaine de l'informatique. Mais entraîner un modèle BERT sur un corpus de l'ordre des 65 millions d'entrées Stack Overflow était largement hors de notre portée…

5.3. Néanmoins, BERT ça marche !

Malgré notre petite contrariété, après nous être cogné la tête à nos « résultats plafonds », nous allons quand-même terminer sur l'enseignement principal de ce projet, qui est que conformément à ce qui était attendu, BERT a été à la hauteur.

La prise en main du modèle est déroutante au début, mais une fois assimilée son utilisation, étant donné qu'il semble performant « out of the box », il fait figure de modèle incontournable dans des projets NLP d'une certaine ampleur (et si on a accès à des TPU).

Si, comme lors du projet 5, nous avions déployé notre modèle final dans une webapp, cette application aurait été bien plus simple (chaîne de processing) et

plus légère (BERT est plus léger que le modèle fasttext mis en œuvre qui à lui seul dépassait 1,2 giga-octet) que la précédente. Donc même si BERT est un modèle complexe et lourd (350 méga-octet), il se défend plutôt bien en terme d'opérabilité.

Performant et pratique, BERT est testé et approuvé.

6. Ressources

6.1. Comprendre BERT

- Attention Is All You Need, 2017, de Vaswani, Shazeer, Parmar (…). Publication par laquelle sont apparus les **transformers**.
- BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018, de Devlin, Chang, Lee et Toutanova. L'acte de naissance de **BERT**.
- Visualizing machine learning one concept at a time. Vulgarisation de grands concepts du machine learning (dont les transformers et BERT) par le chercheur Jay Alamar.
- Andrew Ng, notamment via la chaîne DeepLearningAi sur Youtube.

Et tant d'autres…

6.2. Implémentation de BERT

Voici maintenant quelques-unes des ressources qui nous ont aidés à mettre BERT en pratique.

- Kaggle, Contradictory, My Dear Watson, par Ana Sofia Uzsoy
- Kaggle, Contradictory, My Dear Watson, par Ana Sofia Uzsoy, annoté par Katharina Menz
- Towardsdatascience.com, Hugging Face Transformers: Fine-tuning DistilBERT for Binary Classification Tasks, par Ray Williams.

Et tant d'autres (encore)…