OPENCLASSROOMS

P10 : Développez une preuve de concept

Note méthodologique

Le Natural Language Processing
Exploration du modèle BERT et ses variantes

Étudiant :

Abdelkader SALIM

Mentor :

Kévin DEGILA

30 Juin 2023

Table des matières

1 Introduction

Le traitement naturel du langage, ou Natural Language Processing (NLP) en anglais, est une technologie d'intelligence artificielle visant à permettre aux machines de lire, de déchiffrer, de comprendre et de donner sens au langage humain. D'importants progrès ont été effectués dans ce domaine au fil des dernières années, et le traitement naturel du langage est aujourd'hui exploité pour une large variété de cas d'usage. En guise d'exemple, on peut citer les applications de traduction telles que Google Translate ou encore les assistants personnels tels que Apple Siri, Microsoft Cortana, Amazon Alexa ou Microsoft Cortana. Il en va de même pour tous les ChatBots.

Dans le cadre du parcours Ingénieur IA d'OpenClassrooms, nous avons été amené à travailler sur un sujet de NLP (Projet n°7) que nous avons souhaité approfondir ici. Pour ce faire, nous avons réalisé un état de l'art du domaine que nous exposerons dans ce document. Nous présenterons quelques méthodes, récemment (moins de 5ans) mises en place dans différents laboratoires de recherche, et présenterons leur implémentation en reprenant le jeu de données sur lequel nous avons travaillé auparavant (tweets). Nous représenterons les performances des nouveaux modèles et le meilleur d'entre eux sera déployé dans le cloud pour l'utiliser dans le dashboard.

2 Les Transformers

Bien que les réseaux de neurones conçus pour le traitement des séquences ont permis d'améliorer la classification du sentiment, l'apparition des modèles Transformers a permis d'obtenir de bien meilleurs résultats.

Les modèles Transformer sont actuellement les plus performants pour la représentation du texte puisqu'ils permettent de capturer des relations sémantiques et syntaxiques. Ceci est notamment permis par l'utilisation de l'auto-attention, c'est-à-dire la mesure des relations entre les différents tokens d'une même séquence. Cependant, si les modèles Transformers constituent une avancée majeure, c'est grâce à l'utilisation d'un même modèle à la fois pour l'apprentissage de représentation du texte, et pour la tâche de classification ou de régression en aval (Benballa, 2022).

Le Transformer est un modèle de séquence à séquence (seq2seq) basé sur le mécanisme d'attention et non sur un réseau de neurones récurrent comme c'était le cas pour les modèles précédents. Le début du bloc Transformer a émergé en 2017 après la publication de l'article « *Attention is all you need* » qui décrit son architecture et ses performances impressionnantes sur plusieurs jeux de données de traduction (Vaswani et al. 2017).

Les deux premiers modèles de référence fondés sur cette architecture ont été publiés : OpenAI a publié GPT, pour *Generative Pre-trained Transformer*, (Radford et al. 2018), un modèle de langage auto- régressif, et Google a publié BERT, pour *Bidirectional Encoder Representations from Transformers*, (Devlin et al. 2019), un modèle de langage bidirectionnel.

Avec ce type de modèle, l'utilisation d'une représentation du texte apprise au préalable n'est plus nécessaire, puisque l'architecture permet directement d'apprendre cette représentation de manière auto-supervisée. L'idée est donc d'entraîner les modèles Transformer sur de grandes quantités de données non annotées, afin d'obtenir une représentation de la langue. Les données utilisées sont généralement génériques, afin d'obtenir une représentation de la langue la plus générique possible. Ces modèles sont ensuite affinés (fine-tuning *) sur les données cibles afin d'être adaptés à la tâche traitée.

(*) Le *fine-tuning* est l'entrainement effectué après qu'un modèle ait été pré-entraîné : effectuer un entraînement supplémentaire avec un jeu de données spécifiques. Ce processus permet d'obtenir de meilleurs résultats que l'entraînement à partir de zéro. Il est donc primordial, tant du point de vue des performances que du point de vue écologique de partir d'un modèle pré-entraîné pour travailler sur une problématique en NLP.

2.1.1 Les couches d'attention

Lors du traitement des mots (phase d'encodage), la couche d'attention indique au modèle ce sur quoi il est le plus important de prêter attention. Les couches d'attention permettent au modèle de donner un sens précis aux mots, et permet donc un encodage efficace. Lors du traitement d'un mot, l'attention permet au modèle de se concentrer sur d'autres mots de l'entrée qui sont étroitement liés à ce mot.

L'architecture Transformer utilise l'auto-attention en reliant chaque mot de la séquence d'entrée à tous les autres mots. Par exemple, considérons ces 2 phrase (source ici) :

- « The *cat* drank the milk because **it** was hungry. »

- « The cat drank the ***milk*** because **it** was sweet. »

Dans la première phrase, le mot "**it**" fait référence au "**cat**", tandis que dans la seconde, il fait référence au "**milk**". Lorsque le modèle traite le mot "**it**", l'auto-attention lui donne plus d'informations sur sa signification afin qu'il puisse l'associer au mot correct.

Pour lui permettre de traiter davantage de nuances sur l'intention et la sémantique de la phrase, Transformers inclut plusieurs scores d'attention pour chaque mot. Dans notre exemple (Fig. 1) les couleurs foncées représentent une plus grande attention (des scores plus élevés).

Les couches d'attention nous permettent de tenir compte de façon plus fine les problématiques de contexte, ou de règles grammaticales complexes. On les retrouve à la fois dans les encodeurs et les décodeurs.

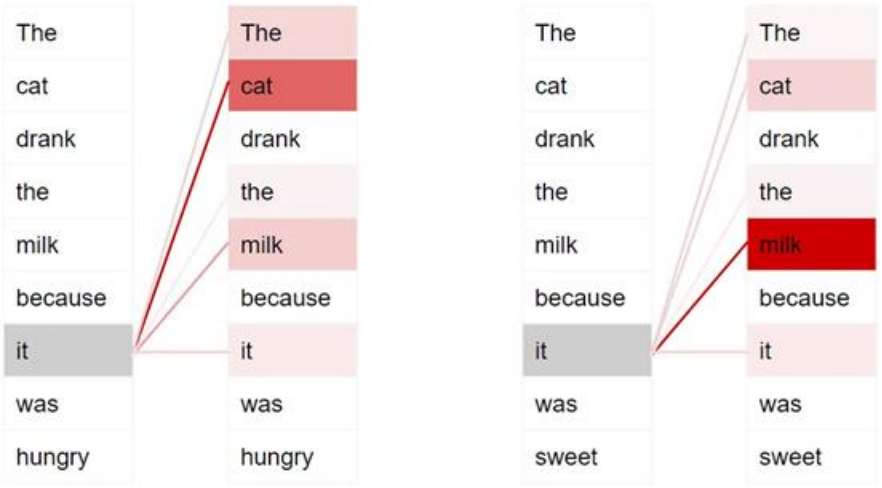

Figure 1 - Les couleurs foncées représentent une plus grande attention (source ici).

2.1.2 Architecture des Transformers

Le Transformer est constitué d'une partie permettant l'encodage des données, composée d'encodeurs (généralement 6), et d'une seconde partie dédiée au décodage, composée de décodeurs (généralement 6). Chacun des encodeurs est composé de deux blocs (Fig. 2a). Tout d'abord, un bloc d'auto-attention, élément majeur du bloc Transformer, suivi d'un bloc de réseau à propagation avant (*feed-forward neural network*). Si le principe de l'attention consiste à mesurer le lien entre deux éléments de deux séquences, l'auto-attention correspond au même mécanisme appliqué à une seule séquence. En effet, l'auto-attention permet de déterminer les relations entre les différents *tokens* d'une même séquence. Ici, l'auto-attention est *multi-head*, cela signifie que les calculs sont effectués en parallèle par plusieurs têtes d'attention (Fig. 2b). Pour les décodeurs, ils contiennent également un bloc d'auto-attention et un bloc de réseau à propagation avant, avec en plus un bloc d'attention Encoder-Decoder permettant de faire le lien entre la séquence encodée en entrée, et la séquence de sortie qui est décodée.

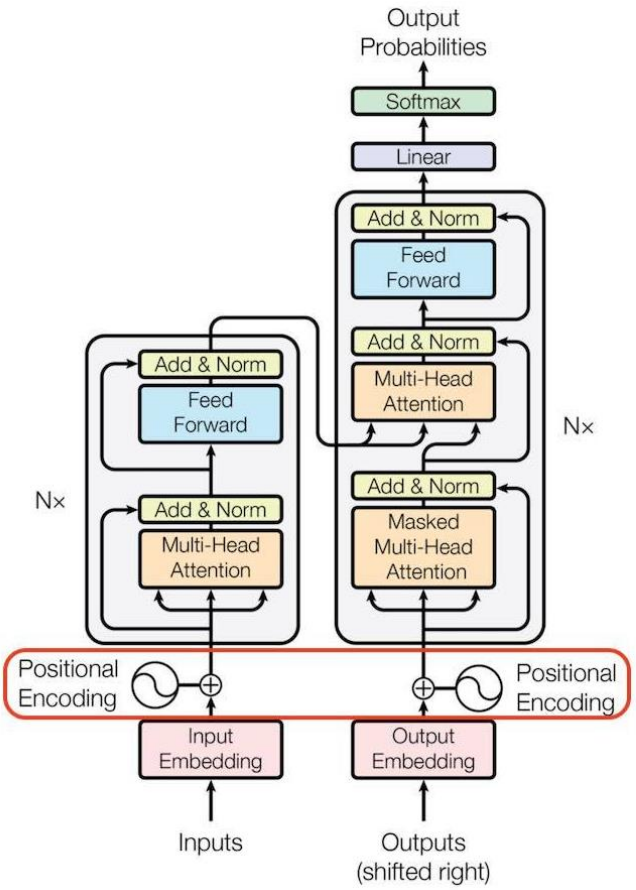

Figure 2a – Schéma du Transformer

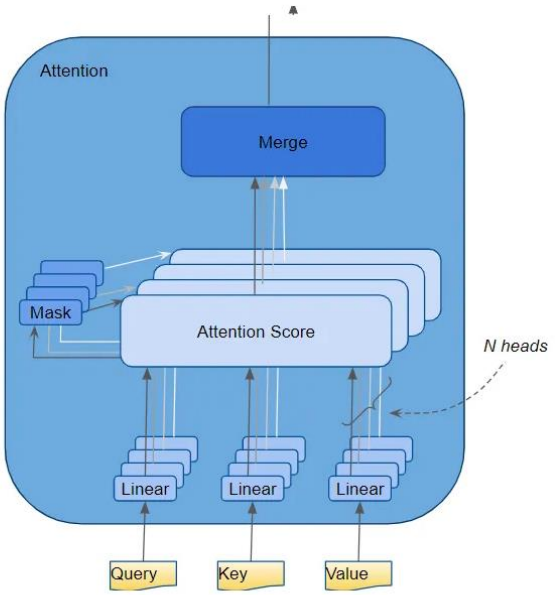

Figure 2b – Schéma de la *multi-head attention*

3 Le modèle BERT et ses déclinaisons

BERT est un modèle de représentation de textes écrits en langage naturel. La représentation faite par BERT a la particularité d'être contextuelle. C'est-à-dire qu'un mot n'est pas représenté de façon statique comme dans un embedding classique mais en fonction du sens du mot dans le contexte du texte. En plus, le contexte de BERT est bidirectionnel, c'est-à-dire que la représentation d'un mot fait intervenir à la fois les mots qui le précèdent et les mots qui le suivent dans une phrase.

BERT a posé les bases des modèles Transformer bidirectionnels (Dai et al. 2018). Il s'agit d'un modèle composé de plusieurs blocs d'encodeurs, publié par Google en 2019. À sa publication, le modèle est disponible en deux tailles : BERT-base et BERT-large. BERT-base est composé de 12 couches (de bloc Transformer), avec une sortie de taille 768, 12 têtes d'attention, et 110 millions de paramètres au total. Quant à BERT-large, il est composé de 24 couches, avec une taille de sortie de 1024, 16 têtes d'attention, et 340 millions de paramètres au total. Les deux modèles sont entraînés avec la même quantité de données. Cependant, avec plus de couches, le grand modèle contient plus de paramètres et est censé être plus efficace que le petit, bien qu'il soit plus long à entraîner.

BERT a été pré-entraîné grâce à l'auto-supervision. Cet apprentissage peut être considéré comme étant à la frontière entre l'apprentissage supervisé et non supervisé. Deux tâches non supervisées ont été utilisées pour l'apprentissage du modèle: MLM et NSP. Le MLM, pour *Masked Language Model*, consiste à masquer aléatoirement un certain pourcentage des *tokens* d'entrée avec un *token* spécifique, avant d'essayer de les prédire en se basant sur le texte qui sert de contexte. Le NSP, pour *Next Sentence Prediction*, consiste à prédire si une phrase *B* suit une phrase *A*. La tâche NSP permet à BERT d'appréhender les interdépendances entre les phrases qui se suivent. Ce qui enrichit la représentation de texte finale et la rend adaptée pour des tâches telles que le *Question Answering*. Le BooksCorpus (Zho et al. 2015) (800 millions de mots) et le Wikipedia anglais (2 500 millions de mots) ont été utilisés pour le pré-entraînement de BERT.

Le principe d'utilisation de BERT est simple : Il est « déjà » pré-entraîné sur une grande quantité de données, on le modifie pour une tâche précise puis on le re-entraîne avec nos propres données. La modification dont il est question ici consiste en général à rajouter un réseau de neurones à la sortie de BERT (*fine-tuning)*. Le modèle est affiné de manière supervisée,

en utilisant des étiquettes annotées à la main, pour une tâche spécifique, comme la classification du texte.

Les entrées de BERT

Les Embeddings : chaque mot est représenté par un vecteur (colonne ou ligne de réels), de dimension 512. On ajoute éventuellement à ces Embeddings, pour chaque mot, les Embeddings d'un "***segment***" quand cela a du sens. Par exemple, chaque phrase est un segment et on veut passer plusieurs phrases à la fois, on va alors dire dans quel segment se trouve chaque mot.

On ajoute ensuite le "***positional encoding***", qui est une façon d'encoder la place de chaque élément dans la séquence. La dimension de l'Embedding de position (à sommer avec l'Embedding sémantique du mot) est la même que celle de l'Embedding sémantique, soit 512, pour pouvoir sommer terme à terme. Les Figures 3a et 3b illustrent de manière visuelle et très détaillée le fonctionnement de l'entrée du modèle BERT.

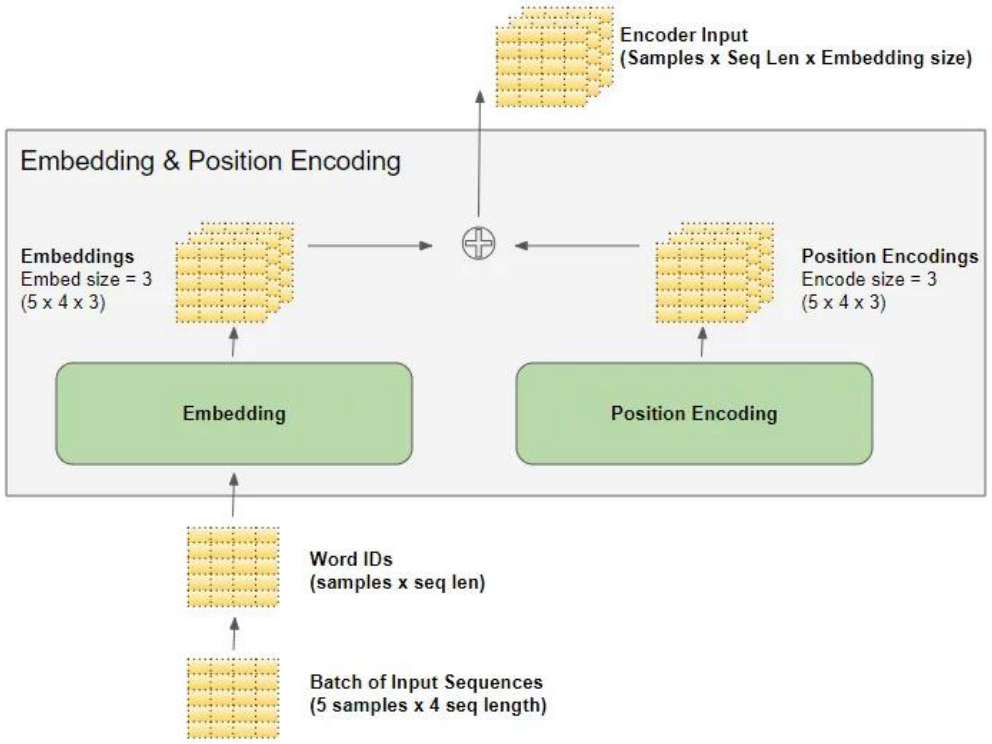

Figure 3-a : Représentation visuelle de l'entrée de BERT. *« The text sequence is mapped to numeric word IDs using our vocabulary. The embedding layer then maps each input word into an embedding vector, which is a richer representation of the meaning of that word.* » (source ici).

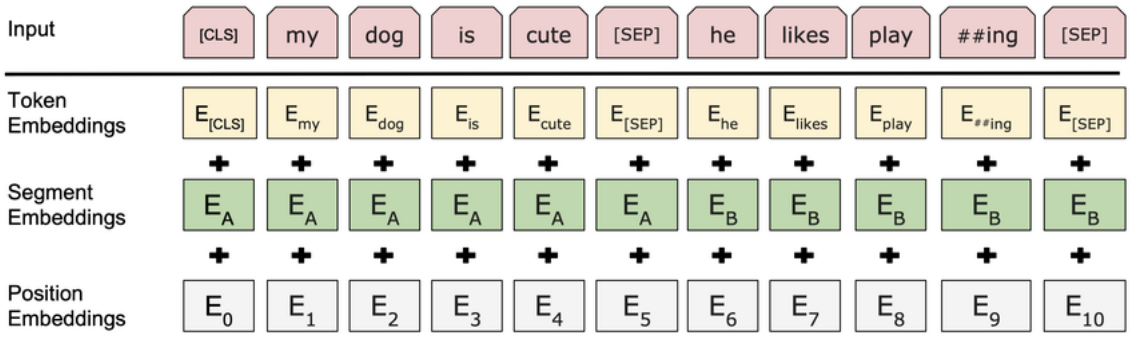

Figure 3-b : Les différents inputs du modèle BERT.

Après sa publication, BERT a donc été décliné en plusieurs versions. Par exemple, *RoBERTa* (Liu et al. 2019), est un modèle beaucoup plus gros que BERT. Il est entraîné sur dix fois plus de données que BERT, et possède 48 couches et cinq fois plus de paramètres. Ici, la tâche de NSP a été supprimée, car elle ne montrait pas d'intérêt pour l'entraînement du modèle et aussi cela rend le pré-entraînement plus rapide.

DistilBERT (Sanh et al. 2019), quant à lui, est une version distillée de BERT. Il est donc plus petit et moins gourmand en

ressources, et qui a été entraîné sur les prédictions de BERT et a donc appris à imiter BERT. Il pourrait être une solution au manque de ressources car il ne possède que 66 millions de paramètres. DistilBERT étant 60% plus rapide, 40% plus légère en mémoire et conservant tout de même 97% des performances initiales de BERT (source).

Tous ces modèles (BERT, RoBERTa et DistilBERT) ont été entraînés sur une large quantité de textes de façon auto-supervisée. Ce type de modèle développe alors une compréhension statistique de la langue sur laquelle il a été entraîné, mais n'est pas utilisable en l'état. Il est alors nécessaire d'effectuer un apprentissage par transfert (*transfert learning*). Le modèle est affiné (*fine-tuning*) de manière supervisée, en utilisant des étiquettes annotées à la main, pour une tâche spécifique. Un exemple de tâche consiste à classifier un texte donné en lui associant différents tags : cela peut consister en une émotion particulière, il s'agit alors de l'analyse de sentiment.

Malheureusement, entraîner un très grand modèle, nécessite une importante quantité de données. Cela devient très coûteux en termes de temps et de ressources de calcul. Cela se traduit même par un impact environnemental assez important. C'est pourquoi, les modèles sont pré-entraînés, puis partagés…

4 Jeu de données & objectif

4.1 Tweets « Sentiment140 »

Pour illustrer la méthode étudiée, nous avons repris le jeu de donné étudié lors du projet n°7, où l'on avait pour missions de prédire le sentiment associé à un tweet.

Le dataset contient 1 600 000 tweets extraits à l'aide de l'api Twitter. Les tweets ont été annotés (0 = négatif, 1 = positif) et peuvent donc être utilisés pour détecter le sentiment .

4.2 Objectifs

Nous avons ici garder le même objectif que celui du projet n°7, c'est-à-dire prédire le sentiment associé à un tweet (Positif ou Négatif).

5 Modèles implémentés & performances

L'ensemble des modèles utilisées ont été implémentés en utilisant la plateforme TensorFlow. Nous avons affiné trois modèles (DistillBERT, BERTweet et RoBERTa) pour répondre à notre problématique qui est l'analyse du sentiment d'un tweet. La méthodologie de modélisation, la métrique d'évaluation retenue et sa démarche d'optimisation sont présentées ci-après.

5.1 Modèle DistilBERT

Référence arXiv :

Titre : « DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter**. »**

Date : *[Submitted on 2 Oct 2019 (v1), last revised 1 Mar 2020 (this version, v4)]*

[*… a smaller general-purpose language representation model, called DistilBERT, which can then be fine-tuned with good performances on a wide range of tasks like its larger counterparts*.**]**

Nous ne nous intéressons qu'aux colonnes "Text" contenant les tweets et "Target" contenant les labels. Nous avons chargé 700 000 tweets parmi les 1 60 000 pour limiter le temps d'exécution des modèles. En effet, l'environnement d'exécution, Google Colab, se déconnecte à partir de quelques heures pour limiter l'accès au GPU et ce malgré un abonnement « Colab Pro ».

Le jeu de données a été séparé en :

- Jeu d'entrainement : 526 968
- Jeu de test : 131 719
- Jeu de validation : 39 916

Le jeu de validation sert à comparer la performance de différents modèles et nous l'avons mis de côté pour éviter les fuites de données.

a) Preprocessing du text

Un nettoyage des tweets a été fait en utilisant la librairie « Processor » qui est spécialisée dans le nettoyage des tweets. Cette même librairie avait été utilisée dans le projet 7.
Nous avons constaté qu'après le preprocessing du texte certains tweets se retrouvent vide, sans aucun mot (Figure 4). Cela veut dire que certains tweets ne contenaient que des symboles comme des émojis des #, etc. Nous avons donc pris le soin de supprimer tous ces tweets vides.

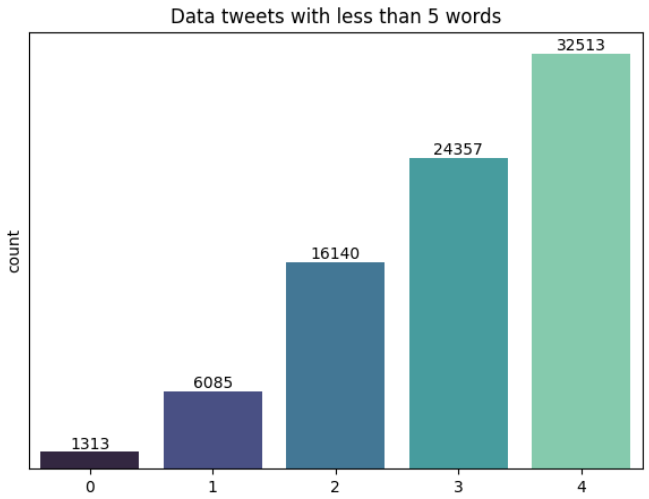

Figure 4 : Exemple de distribution du nombre de mots dans les tweets (échantillon de 600 mille).

b) Préparation des données d'entrée

Avant qu'un texte ne devienne une entrée du modèle, nous devons tout d'abord le *tokeniser*. L'outil « DistilBert Tokenizer » accepte les textes de type "*str*". Il faudra donc transformer une représentation par octets en une chaîne de caractères. La longueur maximale d'une phrase tokenisée dans DistilBERT est de 512 tokens. Les paramètres transmis au tokenizer sont :

- notre liste de string " *list[str]* ",
- truncation=True,
- padding=True.

Si la longueur de la phrase tokenisée est supérieure à 512 en entrée, le tokenizer la tronque à 512. Et si la longueur de la phrase tokenisée est inférieure à 512, le tokenizer la remplit par des zéros jusqu'à la longueur maximale acceptée par le modèle « padding » (Figure 5).

```
#tokenize the text (padding to max sequence in batch)
train_encodings = tokenizer_distilbert(list(X_train_str), max_length=MAX_WORD_LENGTH, truncation=True, padding=True)
test_encodings = tokenizer_distilbert(list(X_test_str), max_length=MAX_WORD_LENGTH, truncation=True, padding=True)

#print the first paragraph and it transformation
print(f'First paragraph: \'{X_train_str[:1]}\'')
print(f'Input ids: {train_encodings["input_ids"][0]}')
print(f'Attention mask: {train_encodings["attention_mask"][0]}')
```

```
First paragraph: '['Why they play so on time everyday???']'
Input ids: [101, 2339, 2027, 2377, 2061, 2006, 2051, 10126, 1029, 1029, 1029, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
Attention mask: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Figure 5 : Paramètre du tokenizer.

Le « DistilBert Tokenizer » nous a retourné un tuple de « Input ids » et de « Attention mask ». Il nous reste à transformer nos labels et nos encodages en un objet Tensorflow Dataset :

```
[ ]  train_dataset = tf.data.Dataset.from_tensor_slices((dict(train_encodings),
                                                         list(y_train)))

     test_dataset = tf.data.Dataset.from_tensor_slices((dict(test_encodings),
                                                         list(y_test)))
```

Figure 6 : Transformation des encodages et labels en objet Tensorflow.

c) Fine-tuning

Dans cette étape, nous prenons un « TFDistilBert For Sequence Classification » et indiquons le nom du modèle comme paramètre. Fixer un taux d'apprentissage et définir la fonction de perte. Compiler le modèle et exécuter la méthode model.fit() pour l'apprentissage (Fig. 7).

- L'optimiseur utilisé est **Adam** (Adaptive Moment Estimation). C'est un algorithme à taux d'apprentissage adaptif, ici $\eta = 3.10^{-5}$.
- La métrique utilisée est l'**accuracy** : le rapport entre le nombre de prédictions correctes et le nombre total prédictions.

```
from transformers import TFDistilBertForSequenceClassification
id2label = {0: "NEGATIVE", 1: "POSITIVE"}
label2id = {"NEGATIVE": 0, "POSITIVE": 1}

model_distilbert = TFDistilBertForSequenceClassification.from_pretrained(MODEL_NAME, num_labels=2, id2label=id2label, label2id=label2id)

optimizerr = Adam(learning_rate=3e-5)
losss = SparseCategoricalCrossentropy(from_logits=True) # Computes the crossentropy loss between the labels and predictions.
model_distilbert.compile(optimizer=optimizerr,
              loss=losss,
              metrics=['accuracy'])

model_distilbert.fit(train_dataset.shuffle(len(X_train)).batch(BATCH_SIZE),
          epochs=N_EPOCHS,
          batch_size=BATCH_SIZE)
```

Figure 7 : Paramètre de l'entraînement du modèle.

d) Evaluation du modèle : Prédiction sur le jeu de Validation

L'évaluation du modèle sur le jeu de validation donne les résultats suivants :

```
Confusion Matrix :
[[16742  3220]
 [ 2990 16964]]
ROC AUC score :  0.92
Accuracy score :  0.844
Average Precision score :  0.919
```

5.2 Modèle BERTweet

Référence arXiv :

Titre : **«** BERTweet: A pre-trained language model for English Tweets**. »**

Date : *[Submitted on 20 May 2020 (v1), last revised 5 Oct 2020 (this version, v2)]*

[*… the first public large-scale pre-trained language model for English Tweets. Our BERTweet, having the same architecture as BERT-base (Devlin et al., 2019), is trained using the RoBERTa pre-training procedure (Liu et al., 2019).*]

Nous avons utilisés exactement la même méthodologie détaillée dans le paragraphe dédié au modèle DistilBERT. Le résultat de l'évaluation du modèle sur le jeu de donnée de Validation est :

```
Confusion Matrix :
[[17511  2451]
 [ 2902 17052]]
ROC AUC score :  0.938
Accuracy score :  0.866
Average Precision score :  0.936
```

5.3 Modèle RoBERTa

Référence arXiv :

Titre : « RoBERTa: A Robustly Optimized BERT Pretraining Approach **»**

Date : *[Submitted on 26 Jul 2019]*

[*… We present a replication study of BERT pretraining (Devlin et al., 2019) that carefully measures the impact of many key hyperparameters and training data size. We find that BERT was significantly undertrained, and can match or exceed the performance of every model published after it. Our best model achieves state-of-the-art results on GLUE, RACE and SQuAD.*]

Nous avons utilisés exactement la même méthodologie détaillée dans le paragraphe dédié au modèle DistilBERT. Le résultat de l'évaluation du modèle sur le jeu de donnée de Validation est :

```
Confusion Matrix :
[[17331  2631]
 [ 3019 16935]]
ROC AUC score :  0.933
Accuracy score :  0.858
Average Precision score :  0.932
```

6 Comparaison des performances de différents modèles

Les performances des modèles Bertweet et Roberta sont assez proches et nous avons diminué les erreurs de 28% par rapport à notre modèle Baseline (simple GRU) :

Modèle	**GRU simple**	**DistilBERT**	**BERTweet**	**RoBERTa**
Accuracy score	0.81	0.84	0.87	0.86
ROC AUC	0.81	0.92	0.94	0.93
Runtime (heures)	0.28	2.59	4.58	4.58

Le meilleur modèle est donc BERTweet.

7 Conclusion

Nous avons effectué un état de l'art du NLP. Nous nous sommes familiarisé avec la notion de Transformer, incontournable dans l'ensemble des problématiques de NLP contemporaines. Nous avons implémentés trois variantes du modèles BERT (DistilBERT, BERTweet, RoBERTa) qui ont moins de 5 ans, en utilisant différentes classes de la bibliothèque *transformers*, et obtenus des résultats très satisfaisants et meilleurs que lors du premier projet.

Le modèle BERTweet est notre meilleur modèle et de meilleurs performances pourraient être atteintes en entraînant ce modèle sur plus de données et en affinant certains hyperparamètres.

Nous avons donc pu mener à bien notre POC, et dans un cadre industriel, nous aurions pu poursuivre le projet et passer en production ce nouveau modèle qui performe mieux.

8 Références bibliographiques

Miriam **Benballa**. Analyse de sentiments sur Twitter dans un contexte faiblement supervisé. Thèse de Doctorat, Normandie Université, 2022.

Ashish **Vaswani**, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998‑6008, 2017

Ru **Ni** and **Huan** Cao. Sentiment analysis based on glove and lstm-gru. In *2020, 39th Chinese Control Conference (CCC)*, pages 7492‑7497. IEEE, 2020.

Wenpeng **Yin**, Katharina Kann, Mo Yu, and Hinrich Schütze. Comparative study of cnn and rnn for natural language processing. *arXiv preprint arXiv:1702.01923*, 2017.

Alec **Radford**, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training.

Jacob **Devlin**, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT : Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies, Volume 1.*

Yukun **Zhu**, Ryan Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. Aligning books and movies : Towards story-like visual explanations by watching movies and reading books. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 19‑27, 2015.

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

Sites web :

https://huggingface.co/learn/nlp-course/chapter1/1

https://towardsdatascience.com/bert-roberta-distilbert-xlnet-which-one-to-use-3d5ab82ba5f8

https://towardsdatascience.com/bert-roberta-distilbert-xlnet-which-one-to-use-3d5ab82ba5f8

https://medium.com/geekculture/hugging-face-distilbert-tensorflow-for-custom-text-classification-1ad4a49e26a7