Rapport - Projet Kaggle: Contradictoire, mon cher Watson

Nicolas Vincent - Adrien Argento - Hela Siala - Théo Boyer
Grenoble-INP ENSIMAG

April 2021

Abstract

Pour un couple de phrases (H, P), le but est de déterminer si l'hypothèse H est une implication, une contradiction, ou encore neutre étant donnée une prémisse P. Ce problème de *Natural Language Inference* est résoluble avec des architectures comme les *Recurrent Neural Networks* et *LSTMs bidirectionnels*, architectures ayant donné de très bons résultats sur des datasets construits à partir de corpus à langues très variées. Des modèles plus récents, comme *BERT* et *RoBERTa*, développés par Google et Facebook et basés sur des systèmes utilisant le mécanisme d'attention (*transformers*), permettent d'obtenir des résultats plus précis. Dans ce rapport, on se propose de présenter et d'explorer certaines stratégies, en particulier l'augmentation de données, pour résoudre ce problème.

Keywords— Natural Language Inference, transformer, fine-tuning, translation

Contents

List of Tables

1 Introduction

L'objectif de ce challenge *Kaggle* est de classifier la relation pour un couple de phrases (H, P), où H est une hypothèse et P une prémisse. Pour une prémisse donnée, nous cherchons à évaluer l'hypothèse associée, qui peut être **vérifiée**, **non vérifiée** ou **neutre** si aucune conclusion n'est possible.

1.1 Un problème de NLP

Le problème étudié est un problème de **Natural Language Processing** (NLP), et plus spécifiquement, de **Natural Language Inference** (NLI), présenté comme suit :

> *The task of determining whether a "hypothesis" is true (entailment), false (contradiction), or undetermined (neutral) given a "premise".*

Notamment, c'est un problème où la complexité et possibles ambiguïtés des langages rendent difficile l'entraînement d'un modèle.

Il existe plusieurs modèles permettant de résoudre ce problème. Utilisés pour résoudre des problématiques de **Computer Vision**, les **Convolutional Neural Networks** (CNNs) ne semblent pas être adaptés pour résoudre des problèmes de NLP. On ne retrouve pas dans un texte les propriétés de *connectivité locale* et d'*invariance à la translation*, ou du moins il nous est plus difficile de représenter ces notions avec des mots plutôt que des pixels. Ils donnèrent cependant des résultats acceptables. Les **Recurrent Neural Networks** (RNNs) semblent être plus adaptés pour représenter la compréhension d'un texte, notamment par le traitement séquentiel, de gauche à droite, de ses entrées. Ils sont plus efficaces mais peuvent être victime d'*overfitting* - l'impossibilité de généraliser les résultats appris sur les données d'entraînement sur des données de test -. Les réseaux **Long Short-Term Memory** permettent de limiter l'*overfitting*. Cependant les informations du contexte ne sont pas conservées sur le long terme. Cela peut poser problème sur des longues phrases pour lesquelles les informations contextuelles sont oubliées par le modèle.

On peut aussi distinguer deux autres types de modèles, les modèles basés sur l'encodage des phrases et ceux basés sur l'attention. Le premier type de modèle perd l'information issue des interactions entre les phrases. Les **transformers**, constituant la dernière innovation permettant de résoudre ce problème, font usage de l'*attention*.

Finalement, pour résoudre un problème de **NLI**, il est possible d'utiliser différentes architectures de *deep-learning*, à savoir les **MultiLayer Perceptrons** (MLPs), les **CNNs**, les **RNNs** et l'**attention**. En entrée du modèle que l'on choisit, on utilise une **représentation textuelle pré-entraînée** du texte qui nous intéresse, entre autres **word2vec**, **GloVe**, **Subword embedding** ou **Bert**. On peut combiner n'importe quelle architecture avec n'importe quelle représentation textuelle.

2 Approche du problème

2.1 Des RNNs aux Transformers

Notre problème est résoluble avec des architectures comme les *Recurrent Neural Networks*, architecture ancienne ayant donné place aux *transformers*, qui sont beaucoup plus performants. Néanmoins, les concepts des RNNs étant clés pour la compréhension

des modèles plus récents, on se propose d'en détailler le fonctionnement dans cette section.

Les *Recurrent Neural Network*, "ancêtres des transformers", sont des réseaux de neurones dans lesquels l'information peut se propager dans les deux sens, y compris des couches profondes aux premières couches. Ces réseaux possèdent des **connexions récurrentes**, dans le sens où ils sont capables de conserver des informations en mémoire : ils se souviennent d'un certain nombre d'états passés. Pour cette raison, les RNNs sont particulièrement adaptés aux problèmes faisant intervenir un contexte et une temporalité, c'est-à-dire des problèmes où les données en entrée partagent des dépendances.

Dans le cadre des problèmes Natural Language Processing, les RNNs prennent en entrée des séquences de **word-embeddings** et quelques couches de convolution sur ces séquences d'entrée pour en extraire des caractéristiques. Le but final est, étant donné une phrase avec un mot manquant, de prédire le mot correspondant.

Les *words-embeddings* ont pour principe de fournir une représentation microscopique de chaque mot d'un texte pour obtenir une représentation macroscopique de l'ensemble. Un *embedding* est un vecteur représentant un unique mot du texte, obtenu après un apprentissage non supervisé sur ce texte. Il existe deux techniques permettant d'obtenir les *words-embeddings* : la technique *word-embedding* (*e.g.* **word2vec**, **Glove**) et la technique de *contextual embedding* (*e.g.* **ELMo**, **BERT**).

La technique *word-embedding* construit un vocabulaire de *tokens* dérivé des mots uniques dans le texte et ignorant les différentes significations contextuelles. On représente vectoriellement chaque mot du vocabulaire (*token*) afin que les mots aux représentations voisines apparaissent dans des **contextes similaires**.
La notion de *"contexte similaire"* est définissable en fonction de deux modèles. Le modèle **continuous bag-of-words** (CBOW) cherche à prédire un mot à partir de ses mots voisins, par exemple prédire *"attrape"* dans l'extrait *"le chat ... la souris"*. Le modèle **skip-gram** cherche à prédire les mots du contexte à partir d'un mot central, par exemple prédire les quatre mots *"le"*, *"chat"*, *"la"* et *"souris"* à partir de *"attrape"*.

Néanmoins, le problème dans ces représentations des mots est leur signification contextuelle (la signification dérivée de l'environnement des mots) qui est ignorée. Par exemple, une seule représentation est apprise pour *"left"* dans la phrase *"I left my phone on the left side of the table"*. Pourtant ici, *"left"* a deux significations différentes dans la phrase, et doit avoir deux représentations différentes dans l'ensemble des *word-embeddings*.

Quant à la technique de *contextual embedding*, elle construit un *token* à partir de la sémantique d'une séquence de mots, en considérant toutes les séquences de mots du texte. On obtient différentes représentations pour les mots polysémiques en fonction de leur contexte.

Regardons plus en détails la technique de *word-embeddings* avec le modèle *continuous bag-of-words*. Chaque texte est parsé et décomposé en **tokens**, où chaque token est vectorisé en deux exemplaires, l'un représentant le token (vectorisé avec *One*

Hot Encoding) et l'autre précisant son contexte (les mots qui se trouvent dans son entourage, généralement le mot de gauche et de droite).

Target	Context	Target Matrix			Context Matrix		
		le	chat	mange	le	chat	mange
le	chat	1	0	0	0	1	0
chat	le	0	1	0	1	0	0
chat	mange	0	1	0	0	0	1
mange	chat	0	0	1	0	1	0

Figure 1: *Word-embeddings* avec le modèle *continuous bag-of-words*

Deux modèles sont possibles pour résoudre le problème cité précédemment :

1. Une **régression**, où le modèle cherche à minimiser une fonction coût évaluant la proximité de deux mots (représentés par des vecteurs) à la probabilité qu'ils ont d'apparaître dans le contexte.
2. Une **prédiction**, où le modèle est un réseau de neurones qui va apprendre à prédire un mot cible en sortie en fonction des mots du contexte passés en entrée.

Bien que les RNNs soient des réseaux intéressants, ils sont soumis à l'explosion du gradient, ou lorsque le réseau a de plus en plus des couches, le gradient de la fonction de perte approche zéro rendant difficile l'apprentissage. En effet, certaines fonctions d'activation, comme la fonction sigmoïde, prennent de larges entrées qui sont "mappées" dans un espace beaucoup plus petit (entre 0 et 1). De ce fait un grand changement de la taille de l'entrée provoque un petit changement de la sortie. Notamment, c'est un problème si le nombre de couches utilisées est grand car lors de la phase de back-propagation pour ajuster les poids, le gradient va décroître exponentiellement dans le temps, réduisant l'impact des premières couches du réseau. Enfin, ce problème implique une mémoire courte pour le réseau, où les données anciennes (les mots éloignés dans une phrase dans un texte par exemple) peuvent être oubliées rapidement.

Une solution pour parer à ce problème est d'utiliser des cellules LSTM (Long Short Term Memory) que nous ne détaillerons pas dans ce rapport, ou bien d'utiliser un transformer, résolvant le problème de l'apprentissage des RNNs que nous détaillerons un peu plus loin.

2.2 Transformers

Les **transformers** sont des composants qui s'appuient sur des méthodes d'**attention**. La découverte de l'attention a permis de développer un modèle sans couche récurrente ou couche de convolution donnant à l'époque les meilleurs résultats sur le *dataset* **SNLI** avec bien moins de paramètres que les autres modèles [9], résolvant ainsi le problème de mémoire à long terme évoqué précédemment.

> *"Attention allows the model to focus on the relevant parts of the input sequence as needed, accessing all the past hidden states of the encoder, instead of just the last one"*

— Zhang Handou

On peut distinguer deux types d'attention: la *general attention* ou *encoder-decoder attention* et la *self-attention*. Le premier mécanisme permet au décodeur d'avoir connaissance de n'importe quel état de l'encodeur lors de n'importe quelle étape du décodage et de sélectionner des éléments spécifiques de cette séquence pour produire la sortie du décodeur. Le mécanisme de *self-attention* permet aux entrées d'interagir entre elles pour trouver lesquelles sont à prioriser.

L'**architecture d'un transformer** fait usage de ces deux mécanismes d'attention. L'*encoder-decoder attention* permet à chaque état du décodeur d'accéder à tous les états de la séquence d'entrée. Les couches de *self-attention* de l'encodeur permettent à chaque état de l'encodeur d'accéder à tous les états de la couche précédente de l'encodeur. Finalement, les couches de *self-attention* du décodeur permettent à chaque état du décodeur d'accéder à tous les états du décodeur jusqu'à cet état inclus.

Un Transformer est construit sur la base d'un encodeur et d'un décodeur comme décrit dans la figure ci-dessus. Un encodeur se compose de couches d'attentions reliées à l'encodeur précédent (ou l'entrée pour le premier) et de couches denses. Un décodeur se compose, de manière similaire, de couches d'attentions reliées au décodeur précédent et de couches denses ainsi que d'une nouvelle couche d'attention supplémentaire intercalée entre ces deux couches et reliée à la sortie du dernier encodeur.

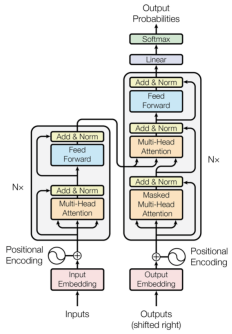

Figure 2: Attention Is All You Need

2.3 BERT

L'acronyme BERT correspond à "Bidirectional Encoder Representations from Transformers". Comme son nom l'indique, il s'agit d'un modèle de représentation du langage qui s'appuie sur le module "Transformer". Cependant, BERT se compose uniquement d'une suite d'encodeurs (N = 12 ou 24 suivant la version : Base ou Large). Le modèle Bert s'inscrit par ailleurs dans le paradigme du *transfer learning* qui suppose de procéder en deux temps : on va d'abord effectuer un "pre-training" de l'algorithme en apprenant à représenter le texte à l'aide d'une tâche auto-supervisée. Bert utilise à l'origine pour cela deux tâches de pré-entraînement. Il masque une partie des mots (15%, même si cela s'avère en réalité plus complexe) et apprend à les retrouver. Cela lui permet d'acquérir une connaissance générale et bidirectionnelle du langage. BERT

apprend aussi à reconnaître si deux phrases sont consécutives ou non. Dans un second temps, l'algorithme est spécialisé pour une tâche précise, c'est le fine-tuning.

2.4 De BERT à RoBERTa

Présenté par Facebook en 2019, Robustly optimized BERT, est une nouvelle version de BERT avec une méthodologie d'entraînement améliorée, 1000 % plus de données et de puissance de calcul. Pour améliorer la procédure d'entraînement, RoBERTa supprime la tâche de prévision de la prochaine phrase (NSP) de la pré-training de BERT et introduit le masquage dynamique de sorte que le jeton masqué change pendant les périodes de formation. Pour rappel : Le NSP est une tâche qui permet de prédire si deux segments se suivent dans le texte original. Des exemples positifs sont créés en prenant des phrases consécutives du corpus de texte. Des exemples négatifs sont créés par appariement de segments à partir de documents différents. Des exemples positifs et négatifs sont échantillonnés avec une probabilité égale.

L'objectif du NSP a été conçu pour améliorer le rendement des tâches en aval, comme l'inférence en langage naturel (Bowman et coll., 2015), qui nécessite un raisonnement sur les relations entre les paires de phrases. des batch-training de plus grandes tailles se sont également révélées plus utiles dans la procédure d'entraînement. RoBERTa utilise 160 Go de texte pour la pré-entraînement, y compris 16 Go de Livres Corpus et Wikipédia anglais utilisés dans BERT. Les données supplémentaires comprennent l'ensemble de données CommonCrawl News (63 millions d'articles, 76 Go), Web text corpus (38 Go) et Stories from Common Crawl (31 Go). Ceci couplé avec le GPU de 1024 V100 Tesla fonctionnant pendant une journée, a conduit au pré-entraînement de RoBERTa.

2.5 XLMRoBERTa

Une autre amélioration a par la suite été apportée à RoBERTa pour les tâches multilingues. Pour ces dernières, reproduire la procédure d'entraînement de RoBERTa dans chaque langue ne suffit pas. Une équipe de Facebook a donc proposé une nouvelle technique d'entraînement, similaire à celle de RoBERTa mais qui permet au modèle d'avoir des représentations multilingues de textes. En plus d'effectuer la tokenisation sur l'ensemble des langues mélangées, les tâches sont adaptées de sorte à ce que le modèle soit forcé de comprendre plusieurs langues en même temps. Par exemple, l'une des tâches dite de **Translation Language Modeling (TLM)** consiste à proposer deux phrases dans deux langues différentes, l'une étant traduction de l'autre, séparées par un token spécial dans lesquelles certains mots sont masqués. L'objectif de la tâche est de compléter les mots masqués.

3 Les données

3.1 Fournies par la compétition

Le jeu de données fourni contient des paires prémisse/hypothèse en 15 langues: Arabe, Bulgare, Chinois, Allemand, Grec, Anglais, Espagnol, Français, Hindi, Russe, Swahili, Thaï, Turc, Ourdou et Vietnamien. Il peut y avoir plusieurs hypothèses associées à la même prémisse.

On compte 12120 paires prémisse/hypothèse dans la *dataset* d'entraînement pour 5195 paires dans la *dataset* de test. L'une des particularités de ce jeu de données est que les langues ne sont pas également réparties. On peut voir sur la Figure 3. que l'anglais est très majoritaire tandis que les autres langues semblent être uniformément réparties. Nous appelons ce jeu de données **watson** dans la suite de ce document.

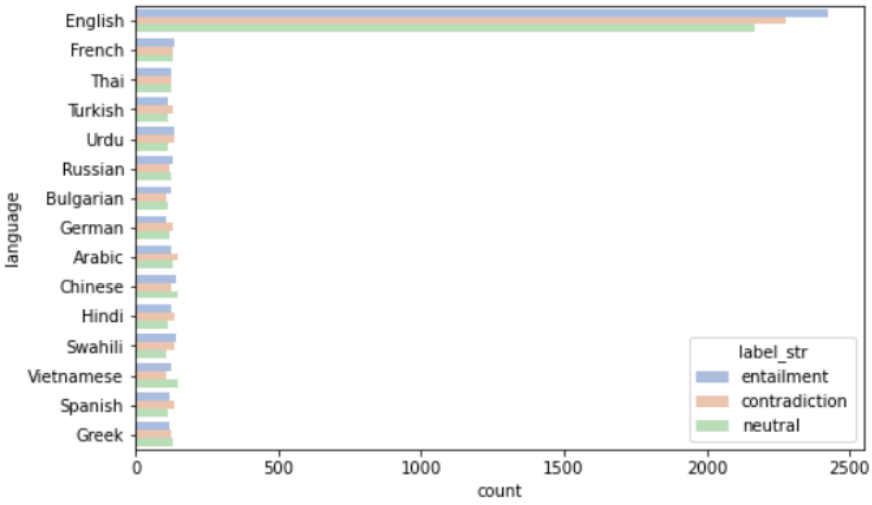

Figure 3: Distribution des langues dans la *dataset* d'entraînement initiale.

3.2 Datasets externes

Nous avons aussi utilisé d'autres *datasets* pour obtenir de meilleurs résultats, à savoir *MultiNLI* (ou MNLI), *SNLI* et *XMNLI*. *MNLI* et *SNLI* ne contiennent que des couples de phrases en langue anglaise tandis que *XMNLI* regroupe 15 langages (les mêmes que ceux du *dataset* **watson**). Grâce à ces *datasets*, il nous est possible d'expérimenter différentes méthodes d'évaluation et d'apprentissage, avec ou sans traduction [11].
XMNLI contient 112 500 échantillons, répartis en 7500 couples de phrases par langages, eux mêmes répartis entre une *dataset* de validation et une *dataset* de test contenant respectivement 2490 and 5010 exemples.
SNLI et *MNLI* contiennent respectivement 570 000 et 433 000 échantillons en anglais.

Créer un *dataset* regroupant *MNLI*, *SNLI* et *XMNLI* présente deux problèmes. Le premier est que la distribution des langues des phrases ne sera pas la même que dans le jeu de test. Le second est qu'il demanderait trop de mémoire pour les machines dont nous disposons. Nous avons donc fait un compromis en créant un *dataset* d'entraînement qui se constitue d'*XMNLI* complété par un nombre précis d'exemples de *MNLI* pour que les distributions des langues soient similaires à celles du jeu de test. Les résultats obtenus grâce aux différents *datasets* mentionnés sont consultables

en table 2. On peut notamment y constater que l'approche la plus efficace est celle qui complète *XNLI* avec des exemples de MNLI.

Nous avons cependant constaté à posteriori que les résultats de cette approche sont faussés car nous avons pu retrouver 2250 exemples du jeu de test dans *XNLI* et 954 dans *MNLI*. En tout cela fait 61.7% du jeu de test présent dans celui d'entrainement.

4 Les modèles en pratique

4.1 Pipeline

Il est possible de décomposer notre pipeline en deux sous-modèles: un modèle pré-entraîné et une architecture de classification. Le modèle pré-entraîné reçoit les couples de phrases en entrée et transmet sa sortie à l'architecture de classification qui nous renvoie une prédiction du label de notre entrée.

Nous avons considéré plusieurs modèles pré-entraînés basés sur l'architecture *transformer* et rendus disponibles par la librairie **Hugging Face**:

- bert-base-uncased
- bert-base-multilingual-uncased
- bert-base-multilingual-cased
- distilbert-base-multilingual-cased
- albert-base-v2
- roberta-base
- roberta-large-mnli
- jplu/tf-xlm-roberta-base
- jplu/tf-xlm-roberta-large

Pour réaliser la classification, nous avons comparé différentes combinaisons de *layers*:

- CLS + Dense
- AvgPooling + Dense
- Bi-LSTM + Hybrid Pooling + Dropout

Cette modélisation est très résumée. Nous développons plus en détails dans le paragraphe suivant. Il est conseillé de le lire si notre lecteur souhaite faciliter la reproduction de nos résultats. Il n'est pas nécessaire à la compréhension de la suite de ce document.

Les modèles pré-entraînés ne prennent pas des couples de phrases telles quelles en entrées. Il est nécessaire de les transformer en un format adéquat: une liste de *tokens*. Il nous suffit donc d'encoder nos couples de phrases en utilisant le vocabulaire du modèle choisi.
Plusieurs paramètres influent sur l'encodage. Le *padding* permet d'obtenir des vecteurs de même taille. L'ajout des *tokens* spéciaux - [CLS] en premier *token* et [SEP] entre les deux phrases - est nécessaire pour résoudre notre problème. Le paramètre *truncation*

nous permet de réduire significativement la taille des vecteurs d'entrée et donc le temps de calcul.
Le paramètre le plus important à considérer lors de l'étape d'encodage est la taille maximale des phrases en entrée (ou *max_length*). On remarque en effet que plus de 95% des phrases ont une taille inférieure à 80 mots. Nous avons donc réalisé des expérimentations en tronquant chaque phrase à 80 mots.

Nous récupérons des phrases encodées en sortie. Plus spécifiquement les *input_ids*: des représentations numériques des *tokens* dont sont composées les phrases. Ils seront utilisées comme entrée du modèle pré-entraîné.
Cette seule entrée est suffisante pour certains modèles qui arrivent à comprendre quand la première phrase se termine et la seconde commence grâce aux *tokens* spéciaux. Cependant, d'autres modèles comme BERT font aussi usage des *token_type_ids*: un masque binaire identifiant les différents types de phrases (*premises* vs. *hypothesis*).
Finalement l'*attention_mask* est un masque binaire permettant de distinguer les indices de *padding* de la phrase en elle-même afin que le modèle ne fasse pas 'attention' au *padding*.
Nous pouvons donc donner à notre modèle pré-entraîné 2 types d'entrées au choix:

- *input_ids*
- *input_ids*, *token_type_ids* et *attention_mask*

Passons maintenant à l'architecture de classification. Nous avons comparé deux architectures différentes - un *fully connected layer* et un **CNN** - qui ne prennent pas le même types de données en entrées.
Le *fully connected layer* prend en entrée les *tokens* de classification ([CLS]) de chaque couple de phrases. Chaque *token* de classification encode l'intégralité des informations nécessaires à la classification du couple de phrases. Il est aussi possible d'utiliser un *average-pooling* des *word-embeddings* d'un couple de phrases à la place du *token* de classification.
L'architecture **CNN** prend en entrée tous les *hidden-states* après l'exécution du dernier *layer* du modèle pré-entraîné.

4.2 Fine-tuning

Les modèles pré-entraînés ne sont pas adaptés à nos données d'entrées car ils n'ont pas été pré-entraînés sur **watson**. Il est donc nécessaire de passer par un processus de *fine-tuning* pour obtenir de meilleurs résultats.

La première étape est de *freeze* le modèle pré-entraîné pour pouvoir réutiliser les *features* du modèle pré-entraîné sans les modifier. On entraîne ensuite notre modèle complet (modèle pré-entraîné + architecture de classification) pour extraire les *features*. L'entraînement ne s'effectue que sur les *layers* de classification (car le modèle pré-entraîné est *freeze*) ce qui permettra à notre modèle d'utiliser les représentations du modèle pré-entraîné. Finalement, on *unfreeze* le modèle pré-entraîné et on le ré-entraîne avec un très petit *learning rate*. Cela permet d'augmenter significativement l'*accuracy* du modèle en adaptant les *features* pré-entraînées aux nouvelles données d'entrées (à savoir **watson**).
On effectue donc deux entraînements sur notre modèle. Nous avons choisi de conserver les même paramètres sur ces deux étapes (nombre d'epochs, batch size, ...).

Cette étape primordiale permet d'augmenter significativement l'accuracy de nos modèles (voir *Table 1*).

Table 1: Impact of fine-tuning on *bert-base-multilingual-cased*

Modèle	bert-base-multilingual-cased	
Train Dataset	watson	
Val Dataset	watson	
Method	**fine-tuning (GPU)**	**no fine-tuning**
Tete de classification	Bi-LSTM + hybrid pooling	CLS + Dense
Batch size	32	64
Epochs	2	10
Train loss	0.7112	2.2181
Train acc.	69.84%	72.14%
Test acc.	**66.00%**	**48.00%**

La tête de classification, la taille du batch et le nombre d'epochs n'influent pas significativement sur l'accuracy - nous le verrons par la suite. On observe ici un sur-apprentissage sur le modèle sans fine-tuning (forte loss et grand écart entre l'accuracy sur train et test). Il est forcé par le nombre d'epochs plus important pour permettre de visualiser l'accuracy maximale qu'il est possible d'obtenir sur la *dataset* d'entraînement.
On remarque immédiatement l'augmentation de l'accuracy de **18%** sur la *dataset* de test.

4.3 Transformation des données par la traduction

Notre problème, utilisant un dataset diversifié avec plusieurs langages et plus majoritairement l'anglais, on s'intéresse à la transformation de données par la traduction pour modifier la structure grammaticale des phrases sans y modifier le sens.

Transformer les données par la traduction (avec Google Translator API par exemple) ne consiste pas à augmenter le nombre de données dans le dataset mais à augmenter le sens des phrases.

Pour notre problème, on s'intéresse à utiliser la traduction pour augmenter le sens des prémisses et des hypothèses.

Plusieurs solutions sont possibles :

1. Nous pouvons expérimenter et voir si l'entraînement de notre modèle sur une seule langue est meilleur que l'entraînement sur plusieurs langues.
2. Nous pouvons modifier la distribution des mots dans notre ensemble de données, en traduisant les phrases dans des langues à faibles ressources (langues où le nombre de mots dans le vocabulaire est réduit) comme le Bulgarian ou le Hindi.
3. Nous pouvons traduire des phrases de façon aléatoire dans une autre langue, puis les retraduire dans la langue d'origine (back-translation).

Back-translation

Soit T une application de traduction et S une séquence de mots. On appelle *back-translation* l'opération $BT = T^{-1}(T(S))$. Si $BT \neq S$, la syntaxe de S a été modifiée, c'est-à-dire, augmentée ou diminuée.

Par exemple, "Je mange pas ici ce soir" donne en anglais "I'm not eating here tonight", puis en français "Je *ne* mange pas ici ce soir" par back-translation : le sens est augmenté par l'ajout de "ne".

Pour des phrases plus complexes, il est possible d'appliquer cette procédure plusieurs fois pour obtenir des augmentations diversifiées ou enchaîner la traduction plusieurs fois avant de retraduire dans la langue d'origine.

Néanmoins, comment reconnaître si le sens d'une phrase n'a pas été diminué ? C'est un problème difficile car une vérification manuelle n'est pas envisageable, bien qu'il puisse être intéressant de vérifier aléatoirement quelques exemples pour se convaincre que c'est correct. Le résultat peut se faire sentir sur la précision obtenue.

Low-resources languages

Les langues à faibles ressources (*low-resource languages*) sont des langues où le nombre de mots dans le vocabulaire est réduit.

> *"Si l'on ne se base que sur le nombre d'entrées dans le dictionnaire, l'anglais est en pôle position des langues les plus riches, avec plus de 200 000 mots recensés dans l'Oxford English Dictionary, dont 171 476 mots en usage et 47 156 mots obsolètes."*
>
> — Babbel Magazine

Pour cette raison, on cherche à traduire les couples prémisses/hypothèses - qui sont écrites dans une langue à hautes ressources - dans une langue à faible ressources, comme par exemple en hindi où les mots "cependant", "néanmoins", "pourtant" ont pour traduction "haalaanki".

Les échantillons obtenus peuvent soit remplacer les échantillons recueillis, soit être ajouté au dataset, en faisant attention de ne pas overfitter le modèle. Pour ce projet, nous avons préféré la première méthode.

En revanche, comparé à l'anglais, le chinois, et encore d'autres langues mondialement utilisées, les langues à faibles ressources ne disposent pas d'un vaste corpus de données disponible sur Internet qui peut être exploité par les IA d'aujourd'hui, entraînant un entraînement parfois difficile en raison d'un manque de disponibilité des ressources nécessaires.

Les différentes techniques proposées se portent bien pour notre problème, notamment avec la *back-translation* permettant généralement, d'augmenter le sens des hypothèses et prémisses grâce à des corrections grammaticales. Quant aux langages à faibles ressources, il faut y faire plus attention, comme la disponibilité des ressources (corpus) est plus faible que des langues à hautes ressources (comme l'anglais), impliquant un apprentissage plus difficile pour le modèle.

5 Résultats

5.1 Résumé

Dans ce projet nous avons mené de nombreuses expériences. Dans cette section, nous présentons les principaux résultats obtenus.

Table 2: Résultats principaux sans fine-tuning

Modèle	Paramètres	Train Dataset	Train Loss	Accuracy
bert-base-multilingual	178M	Watson	2.2181	48.046%
roberta-base	278M	Watson	0.9211	56.93%
roberta-large	560M	Watson	0.6587	78.3%
bert-base-multilingual	178M	XNLI	0.6669	81.52%
roberta-base	278M	XNLI	0.5597	82.309%
roberta-large	560M	XNLI	0.2419	87.641%
bert-base-multilingual	178M	XNLI+MNLI	0.3234	90.625%
roberta-base	278M	XNLI+MNLI	0.2667	91.511%
roberta-large	560M	XNLI+MNLI	**0.1854**	**94.186%**

La *Table 2* met en évidence l'utilité de l'augmentation des données par des *dataset* extérieurs *MNLI* et *XMNLI*. On peut voir que simplement passer du *dataset* fourni à *XMNLI* et en passant de *BERT* à *RoBERTa*, on obtient une augmentation de l'accuracy de **46%**.

#	Team Name	Notebook	Team Members	Score	Entries	Last
1	Marek Nurzynski			0.99037	1	3mo
2	Fellah Zakaria			0.99037	11	3mo
3	wchowdhu			0.98671	81	13d
4	Abid Ali Awan			0.97670	2	2mo
5	ZHONGLIANG SHI			0.97670	4	2mo
6	ADARSH PATEL			0.97670	5	1mo
7	Rishi Chandra	pytorch_gap_xlmro...		0.96438	1	2mo
8	FHG_IPA			0.94937	1	3mo
9	Rahul Bana	Contradictory, My ...		0.94629	1	3mo
10	JCPser			0.94629	10	1d
11	Projet filé Équipe 2			0.94533	5	1d

Figure 4: Classement temporaire de la compétition

Grâce à ce dernier modèle, nous sommes arrivés aux portes du top 10 de la compétition. Cependant, ce résultat est à remettre en contexte, car il semblerait que la plupart des exemples du jeu de test soient présents dans les *datasets* MNLI et XNLI.

5.2 Comparaisons détaillées

Table 3: English dataset with *bert-base-uncased*

Modèle	bert-base-uncased	
Train Dataset	watson	watson-en
Val Dataset	watson	watson-en
Method	**fine-tuning (GPU)**	
Tete de classification	Bi-LSTM + hybrid pooling	
Batch size	32	
Epochs	2	
Train loss	0.7811	0.6714
Train acc.	63.74%	71.56%
Test acc.	59.00%	-

La *Table 3* compare les résultats obtenus avec le modèle pré-entraîné *bert-base-uncased* sur l'ensemble des données **watson** et seulement les données en langue anglaise. On observe une meilleur accuracy avec seulement les données anglaises. En effet, le modèle *bert-base-uncased* est entraîné sur des données anglaises. 63.71% est une valeur étonnante car la *dataset* d'entraînement ne contient que 56.7% de données anglaises. On a donc au moins 7.01% couples de phrases qui ne sont pas en langue anglaise qui ont été associées au bon label malgré le fait que le modèle ait été entraîné sur des données en anglais.

Table 4: Multilingual BERT models comparison

Modèle	uncased	cased	distil
Train Dataset	watson		
Val Dataset	watson		
Method	**fine-tuning (GPU)**		
Tete de classification	Bi-LSTM + hybrid pooling		
Batch size	32		
Epochs	2		
Train loss	0.7054	0.6984	1.0341
Train acc.	69.44%	71.12%	46.51%
Test acc.	**65.80%**	66.00%	-

La *Table 4* compare les résultats obtenus avec les modèles pré-entraînés *bert-base-multilingual-uncased* (*uncased*), *bert-base-multilingual-cased* (*cased*) et *distilbert-base-multilingual-cased* (*distil*). On remarque que les résultats des modèles *cased* et *uncased* sont similaires. Utiliser un modèle entraîné sur des données sans majuscules après avoir supprimé les majuscules de **watson** est presque aussi performant que d'utiliser le même modèle entraîné sur des phrases avec majuscules.
Le modèle *distil* donne des résultats très décevants. Il devrait presque égaler les modèles BERT. Nous n'avons probablement pas correctement entraîné ce modèle.

La *Table 5* présente les résultats obtenus pour la transformation de données par la traduction, où pour tous les résultats exposés, le dataset utilisé est Watson. On observe qu'appliquer une transformation de données par la traduction sur le jeu de données

Table 5: Transformation des données sans fine-tuning

Modèle	Train Dataset	Language	Train Loss	Accuracy
roberta-base	Watson	Not modified	0.9795	37.91%
roberta-base	Transformation	Not modified	1.1818	59.78%
roberta-base	Twice Transformation	Not modified	1.1001	30.94%
roberta-base	Thrice Transformation	Not modified	1.0992	34.36%
roberta-base	High-resource	English	1.1913	59.83%
roberta-base	Low-resource	Hindi	0.9833	53.22%
roberta-base	Low-resource	Vietnamese	0.9370	57.63%
roberta-base	Low-resource	Bulgarian	0.9999	52.31%

augmente l'accuracy d'environ 20%, ce qui n'est pas négligeable. Quant aux langages à faibles ressources, on observe une augmentation de l'accuracy par rapport au résultat sur le dataset de base, preuve d'une utilité de cette technique. Les résultats pour les langages à faibles ressources *pourraient* être mieux, si les modèles utilisés étaient entraînés sur un plus large volume de données dans ces langues, du moins si ce large volume existait.

Table 6: RoBerta models comparison

Modèle	roberta-base	albert-base-v2	roberta-large
Train Dataset	watson		
Val Dataset	watson		
Method	**fine-tuning (GPU)**		
Tete de classification	Bi-LSTM + hybrid pooling		
Batch size	32		16
Epochs	2		
Train loss	0.7325	0.8189	0.6267
Train acc.	65.77%	60.33%	69.81%
Test acc.	63.50%	59.20%	67.40%

La *Table 6* compare les résultats obtenus avec les modèles pré-entraînés **RoBerta**. Sans surprise, le modèle *large* donne de meilleurs résultats. La différence d'accuracy de 4% entre *roberta-base* et *albert-base-v2* semble élevée. Pour obtenir de meilleurs résultats il faudrait tester de modifier le *padding* des données d'entrée.

Table 7: Batch size influence on *tf-xlm-roberta-base*

Modèle	jplu/tf-xlm-roberta-base		
Train Dataset	watson		
Val Dataset	watson		
Method	**fine-tuning (GPU)**		
Tete de classification	Bi-LSTM + hybrid pooling		
Batch size	16	32	64
Epochs	2		
Train loss	0.7254	0.7599	0.7838
Train acc.	69.10%	68.03%	65.45%
Test acc.	**69.90%**	69.30%	67.5%

La *Table 7* compare les résultats obtenus avec le modèle pré-entraîné *tf-xlm-roberta-*

base pour différentes tailles de batch. Les résultats sont similaires mais l'on remarque qu'une faible batch size donne une meilleur accuracy (au détriment du temps de calcul).

Table 8: Max-length influence on different models

Modèle	bert	jplu/tf-xlm-roberta-base	jplu/tf-xlm-roberta-large
Train Dataset		watson	
Val Dataset		watson	
Method		**fine-tuning (GPU)**	
Tete de classification		Bi-LSTM + hybrid pooling	
Batch size		32	16
Epochs		2	
Train loss	0.7057	0.7415	0.5306
Train acc.	71.56%	68.41%	80.17%
Test acc.	66.20%	69.90%	**79.50%**

La *Table 8* compare les résultats obtenus avec les modèles pré-entraînés *bert-base-multilingual-cased* (*bert*), *jplu/tf-xlm-roberta-base* et *jplu/tf-xlm-roberta-large* pour une taille maximale des phrases de **80 caractères**. Les résultats sont semblables à ceux obtenus *Table 4* (BERT) et *Table 7* (XML-RoBerta-Base) où la taille maximale des phrases était définie à **128 caractères**.
95% des phrases ont une longueur inférieure à 80 caractères. On a ainsi un gros gain en espace de stockage, et donc en temps de calcul, pour obtenir un résultat presque identique avec une perte minimale de données utiles.

Table 9: *Table 8* with more **epochs**

Modèle	bert	jplu/tf-xlm-roberta-base	jplu/tf-xlm-roberta-large
Train Dataset		watson	
Val Dataset		watson	
Method		**fine-tuning (GPU)**	
Tete de classification		Bi-LSTM + hybrid pooling	
Batch size		32	16
Epochs		**10**	
Train loss	0.0749	0.1393	0.1038
Train acc.	98.25%	95.75%	96.82%
Test acc.	-	69.30%	78.60%

La *Table 9* reprend les paramètres de la *Table 8* avec cette fois **10 epochs**. On observe un fort *overfit* sur les données d'entraînement qui pourrait être évité grâce à un *Early Stopping*. Cela a cependant peu d'influence sur les résultats qui sont légèrement moins satisfaisants que sur la *Table 8*.
2 epochs sont suffisants pour entraîner nos modèles.

La *Table 10* reprend les paramètres de la *Table 8* avec cette fois une tête de classification très simple: **AvgPooling + Dense**. On observe des résultats presque identiques.
Il n'est donc pas nécessaire d'utiliser un réseau de classification complexe en sortie du modèle pré-entraîné.

Table 10: *Table 8* with another **classification header**

Modèle	bert	jplu/tf-xlm-roberta-base	jplu/tf-xlm-roberta-large
Train Dataset		watson	
Val Dataset		watson	
Method		**fine-tuning (GPU)**	
Tete de classification		**AvgPooling + Dense**	
Batch size		32	16
Epochs		2	
Train loss	0.7119	0.6672	0.5192
Train acc.	69.29%	72.28%	80.00%
Test acc.	64.40%	69.30%	79.80%

Concentrons nous maintenant sur le **modèle le plus efficace**: *jplu/tf-xlm-roberta-large*. Pour accélérer nos calculs, nous allons utiliser le TPU (plutôt que le CPU qui empêche d'utiliser une batch size de 32 pour ce modèle) et effectuer une validation croisée sur 5 échantillons afin de conforter la fiabilité de notre modèle.
Nous allons en plus faire varier la tête de classification avec *1-input+CLS+Dense* **(1)**, *1-input+AvgPooling+Dense* **(2)**, *1-input+Bi-LSTM+Hybrid pooling* **(3)** et *1-inputs+Bi-LSTM+Hybrid pooling+Batch-Norm* **(4)**.
La notation *'1-input'* signifie que l'on fournit seulement l'entrée *'input_ids'* à notre modèle plutôt que le triplet '(*input_ids*, *token_type_ids*, *attention_mask*)' utilisé jusqu'à présent.

Table 11: **5-fold cross-validation** on *jplu/tf-xlm-roberta-large*

Modèle	jplu/tf-xlm-roberta-large			
Train Dataset	watson			
Val Dataset	watson			
Method	**fine-tuning (TPU) + 5-fold**			
Tete de classification	**(1)**	**(2)**	**(3)**	**(4)**
Batch size	32			
Epochs	10			
Mean acc.	80.46%	80.21%	80.03%	80.02%
Std. acc.	0.0079	0.0023	0.0071	0.0057
Test acc.	80.50%	-	-	**80.70%**

La *Table 11* donne des résultats intéressants. Tout d'abord, nos modèles sont très fiables avec une très bonne accuracy moyenne et une faible variance sur 5 échantillons.
De nouveau on observe qu'un simple réseau de classification Dense suffit.
Les résultats sur la *dataset* de test sont légèrement meilleurs que ceux observés précédemment. Cela peut être dû à l'augmentation de la taille de batch (passage de 16 à 32) ou plus probablement à la validation croisée.
L'étape de *Batch Normalization* rajoutée sur la dernière tête de classification pourrait expliquer l'augmentation de l'accuracy de 0.2% par rapport à la tête **(1)** sur la *dataset* de test.

6 Conclusion

Nous avons pris part à une compétition *Kaggle* de *NLP*. Pour cette tâche dite de *NLI*, nous avons effectué un important travail d'état de l'art en amont pour comprendre les méthodes et enjeux du *NLP* en 2021. Ce tour d'horizon nous a conduit à nous intéresser aux modèles modernes type "*transformer*", ainsi qu'à différentes techniques d'augmentation de données. Grâce aux ressources fournies par *Kaggle*, nous avons pu manipuler et expérimenter des modèles de plusieurs centaines de millions de paramètres en un temps raisonnable. Notre progression au fil de la compétition a été très importante jusqu'à nous mener aux portes du top 10, que nous essayerons de franchir par la suite.

References

[1] Paper, *Recurrent Neural Network-Based Sentence Encoder with Gated Attention for Natural Language Inference*, 2017.

[2] Paper, *Attention Is All You Need*, 2017.

[3] Paper, *An Exploration of Dropout with RNNs for Natural Language Inference*, 2018.

[4] Paper, *Attention-Fused Deep Matching Network for Natural Language Inference*, 2018.

[5] Article, Paul Denoyes, *BERT : Le "Transformer model" qui s'entraîne et qui représente*, 2019

[6] Article, Kawin Ethayarajh, *BERT, ELMo, & GPT-2: How Contextual are Contextualized Word Representations?*, 2020.

[7] Paper, *RoBERTa: A Robustly Optimized BERT Pretraining Approach*, 2019.

[8] https://www.quantmetry.com/blog/bert-google-ai-banc-de-test/

[9] Paper, *A decomposable attention model for natural language inference*, 2016.

[10] Article, Denny Britz, *Understanding Convolutional Neural Networks for NLP*, 2015

[11] Article, Stephen Mayhew, *XNLI, Cross-lingual Natural Language Inference*, 2020