Identification de paraphrases

Lucie Gabagnou, Armand L'Huillier, Yanis Rehoune, Ghiles Idris
March 28, 2023

Abstract

L'identification de paraphrases constitue un défi majeur dans le domaine du traitement automatique du langage naturel (TALN). Que ce soit pour la détection de plagiat, la génération de texte ou la traduction automatique, l'analyse de la sémantique - incluant les paraphrases - tente d'appréhender la complexité du langage humain. Avec l'apparition des architectures de type Transformer et les progrès significatifs dans le domaine des réseaux de neurones, des modèles novateurs tels que BERT et GPT ont vu le jour, améliorant considérablement les performances dans ce domaine. Dans cette optique, ce rapport explore l'utilisation des réseaux de neurones pour identifier des paraphrases en se basant notamment sur les réseaux LSTM.

1 Introduction

1.1 Problématique

En linguistique, la paraphrase consiste à réexprimer une phrase ou un énoncé en adoptant des expressions et une structure syntaxique différentes, tout en préservant le sens original. En traitement automatique du langage naturel (TALN), la paraphrase renvoie plus généralement au concept de similarité sémantique, qui cherche à mesurer le degré de ressemblance sémantique entre différentes paires de textes, qu'il s'agisse de mots, de phrases ou d'autres éléments linguistiques. La compréhension de ce concept est cruciale dans la recherche en TALN.

Dans cette étude, nous explorons la problématique de la détection de la paraphrase à partir d'une paire de phrases, où l'algorithme doit déterminer si les deux phrases ont le même sens. Il s'agit donc d'un problème de classification binaire, où les paires sont classées comme « paraphrase » ou « pas paraphrase ».

1.2 Description des données

Le jeu de données utilisé pour cette analyse est fourni par P3 (Public Pool of Prompts) et comprend 795 241 paires de phrases. Chaque paire est étiquetée comme une paraphrase ou non, en utilisant une indicatrice comme variable cible (1 signifiant que la paire est une paraphrase, 0 sinon).

Dans cette étude, une analyse exploratoire a été réalisée pour mieux comprendre la nature des données et guider le choix du modèle le plus adapté. Nous pouvons noter les observations suivantes :

- Toutes les phrases du jeu de données sont sous forme de questions et sont rédigées en anglais.
- La longueur médiane des paires de phrases est de 11 mots (en termes de tokens) et est fortement concentrée autour de cette valeur.
- La taille du vocabulaire est de 66 192.
- Le jeu de données n'est pas parfaitement équilibré en termes de classes, avec une répartition de 68% de paraphrases contre 32% de paraphrases. Remarque: il s'agit d'une situation plutôt atypique car les problèmes d'imbalanced data reposent souvent sur un évenement à prédire moins fréquent, ce qui n'est pas le cas échéant pour ce jeu de données.

2 Présentation du/des modèles

Notre objectif consistait à explorer diverses méthodes de modélisation pour la détection de paraphrases, en commençant par une approche simple et en incorporant progressivement des éléments plus sophistiqués. Par la suite, nous avons confronté ce modèle à un modèle de Transfer Learning[1] basé sur BERT. L'objectif était d'évaluer l'efficacité d'un modèle "adapté" à cette problématique, qui n'a pas nécessairement été entraîné sur un vaste corpus, par rapport aux dernières technologies en Deep Learning. L'ensemble des modèles ont été réalisé via Keras (pour le modèle LSTM) et PyTorch (pour BERT).

2.1 Modèle Siamois Bi-LSTM

Dans un premier temps, il était essentiel de comprendre comment traiter les paires de phrases. En effet, la comparaison de deux phrases deux à deux correspond à une structure algorithmique spécifique, et se doit d'être bien intégrée à la modélisation. La solution est d'utiliser une architecture de type "réseau de neurones siamois". Il s'agit d'une structure capable de gérer les problématiques de similarité/dissimilarité entre deux éléments (que ce soit des images, du texte, etc.). Ainsi, la structure du modèle est la suivante :

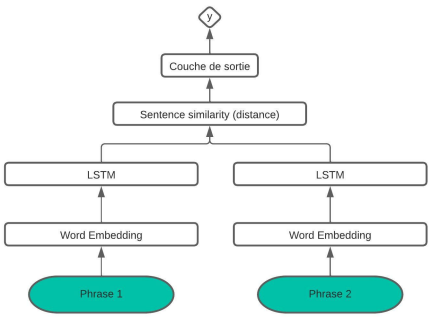

Figure 1: Schéma du réseau de neurones siamois LSTM

2.1.1 Architecture

En somme, les deux entrées sont d'abord prétraitées de manière similaire, puis vectorisées à l'aide d'un modèle pré-entraîné (Word Embedding avec FastText). Ensuite, ces représentations sont passées dans une couche cachée de LSTM pour capturer les relations entre les mots. Les sorties de cette couche sont ensuite concaténées et comparées via une couche de distance pour calculer la similarité entre les deux éléments (distance de Manhattan en l'occurrence).

Extraction des features (Word Embedding) Les données textuelles nécessitent un prétraitement pour être exploitées dans un modèle. Nous avons nettoyé les données en les lemmatisant, en les convertissant en minuscules et en supprimant la ponctuation. Puisque tout est en anglais, aucune normalisation linguistique n'est nécessaire.

En ce qui concerne les variables à exploiter, nous avons envisagé deux approches :

- **Features syntaxiques ou grammaticales :** Nous avons considéré que l'ordre des mots dans une question n'était pas une caractéristique importante car les questions suivent généralement une structure de phrase similaire et standardisée. Cependant, nous avons également utilisé des variables POS (Part-of-Speech) pour comprendre l'ordre des mots.
- **Features lexicaux:** Pour extraire des informations lexicales sous forme numérique (de vecteurs continus), nous avons utilisé des modèles de Word Embedding pré-entraînés tels que Word2Vec et FastText. Ces

[1]Le transfer learning est une technique d'apprentissage automatique où un modèle pré-entraîné sur une tâche est adapté pour résoudre une nouvelle tâche similaire, en tirant parti des connaissances acquises lors de l'apprentissage initial. Cela permet de réduire le temps d'entraînement et d'améliorer les performances du modèle sur la nouvelle tâche.

modèles ont été entraînés sur des corpus de grande taille, ce qui les rend plus adaptés pour l'inférence que des embeddings spécifiques à notre jeu de données. Plus particulièrement, notre modèle final a utilisé FastText entraîné sur Common Crawl et Wikipédia (environ 600 Milliards de tokens) et dispose d'une dimension finale de 300 (réduite à 100 dans notre cas).

Cependant, ces techniques ne permettent pas de tenir compte des relations entre les mots. En ce sens, il est nécessaire d'utiliser des méthodes plus puissantes tel que les réseaux de neurones LSTM.

Couche cachée LSTM Les réseaux LSTM sont une architecture de réseaux de neurones récurrents conçus pour capturer les relations de long terme entre les éléments d'une séquence en conservant un état caché. Ils sont particulièrement adaptés pour le traitement de données séquentielles telles que des phrases, en capturant les dépendances temporelles complexes entre les mots. Les LSTM évitent également le problème de la disparition du gradient qui peut survenir avec les RNN classiques en utilisant des portes qui contrôlent l'écoulement de l'information dans le réseau.

Le fonctionnement du LSTM peut être via les élements suivants:

- La porte d'oubli contrôle la quantité d'information à oublier de la cellule de mémoire:

$$f_t = \sigma_g(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{1}$$

- La porte d'entrée contrôle la quantité d'information à ajouter à la cellule de mémoire:

$$i_t = \sigma_g(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2}$$

- La cellule de mémoire stocke l'information importante de la séquence en cours de traitement (en fonction de la quantité contrôlée par les portes):

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{3}$$

  Elle est transmise à l'étape suivante pour permettre aux LSTM de conserver l'information importante de la séquence à travers le temps.
- L'état caché correspond à la couche transmise à l'étape suivante de notre modèle. Il résume l'information importante de la séquence à un moment donné:

$$h_t = o_t * \tanh(C_t) \tag{4}$$

Dans le cadre de notre projet, nous avons principalement testé les LSTM ainsi que Bi-LSTM (Bi-directional LSTM): La couche cachée Bi-LSTM fonctionne de manière similaire à la couche cachée LSTM, mais avec l'ajout d'une couche LSTM supplémentaire dans la direction opposée pour capturer les dépendances contextuelles à la fois dans le passé et dans le futur de chaque mot ($h^{(b)}_{t+1}$ est considéré de la même façon que $h^{(b)}_{t-1}$).

Implémentation: Dans notre modèle, les deux inputs sont soumis chacun un modèle LSTM/Bi-LSTM, suivant la même architecture (même nombre de neurones, etc..)

Concaténation des couches avec la distance de Manhattan L'architecture du réseau de neurones siamois se termine par une couche qui unit les deux branches. Pour ce faire, une mesure de similarité ou de distance (euclidienne, cosinus, Manhattan, etc.) est utilisée. Nous avons opté pour la distance de Manhattan car elle semble moins sensible à la longueur des vecteurs et prend en compte la différence entre chaque paire de caractéristiques.

La distance de Manhattan entre deux vecteurs x et y est donnée par:

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

Couche de classification Pour la classification binaire des paraphrases, nous avons utilisé un seul neurone dans la couche finale du réseau de neurones. Cette couche renvoie une valeur comprise entre 0 et 1, correspondant à la probabilité que les deux phrases soient des paraphrases. Nous avons utilisé la fonction d'activation sigmoïde pour cette couche.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

où x est la sortie de la couche précédente.

2.1.2 Choix des hyperparamètres

Notre approche consistait à tester différentes combinaisons d'hyperparamètres en variant la fonction de perte, la taille du batch, ainsi que certaines couches. Nous avons testé les architectures LSTM, GRU et Bi-LSTM. Nous avons observé que le modèle Bi-LSTM était le plus performant en termes de rapidité, de faiblesse d'overfitting et surtout d'accuracy sur l'échantillon de validation, avec une accuracy de 0.82.

Ensuite, nous avons utilisé la méthode Random Grid Search pour déterminer les hyperparamètres optimaux. Cette méthode est robuste et plus rapide que la méthode Grid Search classique. Nous avons optimisé les hyperparamètres de manière à maximiser l'accuracy sur l'échantillon de validation, car notre objectif était de créer un modèle généralisable et qui ne fasse pas d'overfitting sur l'échantillon d'apprentissage.

Choix de la fonction de perte Nous avons testé deux fonctions de perte manuellement :

- Binary cross-entropy : une fonction de perte couramment utilisée pour les problèmes de classification binaire. Elle mesure la différence entre les prédictions du modèle et les vraies valeurs pour la target. La fonction de perte Binary cross-entropy pour un échantillon est définie comme suit :

  $$L(y, \hat{y}) = -y \cdot \log(\hat{y}) - (1 - y) \cdot \log(1 - \hat{y})$$

  où y est la véritable étiquette (0 ou 1) et $\hat{y}$ est la prédiction du modèle (en probabilité).

- Contrastive loss : une fonction de perte spécialement conçue pour mesurer la distance entre les représentations apprises pour deux entrées, en encourageant le réseau à apprendre des représentations similaires pour des entrées similaires et des représentations différentes pour des entrées différentes. Cette fonction de perte pénalise le modèle lorsque des paires d'entrées similaires ont des représentations éloignées et des paires d'entrées différentes ont des représentations proches. Elle est définie comme suit :

  $$L(y, d) = \frac{1}{2} \cdot y \cdot d^2 + \frac{1}{2} \cdot (1 - y) \cdot (\max(0, m - d))^2$$

  où y est le label pour la paire, d est la distance entre les représentations des deux exemples, et m est la marge minimale entre les distances des paires similaires (fixée à 1).

Hyperparamètres d'apprentissage (Grid Search) Nous avons effectué une Grid Search en faisant varier :

- Le taux d'apprentissage (learning rate)
- La taille du batch (batch size)
- Le nombre d'epochs
- L'optimiseur (Adam, SGD, RMSprop, etc.)

Hyperparamètres de l'architecture (Grid Search) Les seuls paramètres fine-tuné du modèle ont été le nombre de neurones dans la couche cachée Bi-LSTM et le Dropout (qui désactive temporairement certains neurones pour contrer l'overfitting). Toutefois, on aurait pu également optimiser le nombre de couches (voire le type de couches).

Finalement, les paramètres/hyperparamètres retenus sont:

- Nombre de neurones dans la couche LSTM: 128 neurones
- Dropout de 0.1
- Taux d'apprentissage: 0.001
- Optimiseur: Adam
- Taille du batch: 32
- Nombre d'epochs: 3
- Fonction de coût: Binary cross-entropy

2.2 Résultats

Il est évident que, pour évaluer la qualité des modèles, nous avons pris en compte plusieurs métriques, notamment l'accuracy sur les différents échantillons. Cependant, nous avons également examiné les fonctions de coût, telles que la binary cross-entropy, pour comprendre comment les modèles convergaient lors de l'apprentissage.

Résultat du modèle Les résultats du modèle siamois Bi-LSTM obtenus sur l'échantillon de validation ont montré une précision encourageante de 0.84, indiquant que le modèle a la capacité de prédire avec précision les deux classes sur cet échantillon et n'overfitte que très légèrement. Cependant, les performances sur l'échantillon de test ont été très médiocres, avec une précision de seulement 0,23. Plusieurs raisons peuvent expliquer ces résultats, notamment le surapprentissage du modèle sur les deux échantillons et le déséquilibre des classes des données de test, en faveur des paraphrases (100% de paraphrases sur l'échantillon). Ce déséquilibre est d'autant plus préoccupant que nous avons constaté que certaines paires de phrases étaient étiquetées comme des paraphrases alors qu'elles ne semblaient pas l'être (Exemple: "What is the difference between phrasal verbs and prepositional verbs?" and "Do you ever notice jealous wives or girlfriends?" sont considérées comme paraphrase). Ces résultats suggèrent que la qualité des données pourrait être améliorée, en particulier en veillant à un équilibre entre les classes dans l'échantillon et un étiquetage plus réaliste.

Résultat du modèle concurrent Comme expliqué précédemment, nous avons souhaité comparer les performances de notre modèle avec celles d'un modèle basé sur les Transformers, connus pour être très performants en NLP. Pour cela, nous avons utilisé un modèle pré-entraîné tel que BERT, puis l'avons fine-tuné pour notre tâche précise de classification. L'accuracy s'est avéré plus élevé que dans le premier cas puis qu'elle était de 0.89 et le modèle n'overfitte pas (0.89 sur l'échantillon de train également). Sur l'échantillon de test, les performances sont très bonnes puisque l'accuracy est de 0.95.

Les résultats peuvent être résumé dans le tableau suivant:

Modèle	Train accuracy	Validation accuracy	Test accuracy
Siamois Bi-LSTM	0.88	0.84	0.23
BERT fine-tuné	0.89	0.89	0.95

Table 1: Résumé des performances des deux modèles sur les différents échantillons.

Ces résultats confirment l'efficacité des modèles basés sur les Transformers en NLP, ainsi que l'intérêt des techniques de Transfer Learning pour améliorer les performances des modèles. Toutefois, il convient de rappeler que ces résultats ne sont pas généralisables sans disposer d'un échantillon de test adéquat.

2.3 Conclusion

En conclusion, notre étude a permis de construire un modèle de classification de paraphrases basé sur un réseau de neurones siamois Bi-LSTM. Nous avons réussi à obtenir des performances encourageantes sur l'échantillon de validation, avec une accuracy de 0.84. Cela suggère que le modèle prédit généralement bien les deux classes sur cet échantillon et qu'il n'overfitte que légèrement. Néanmoins, lorsqu'il s'agit de la capacité à se généraliser, le modèle BERT est nettement meilleur puisque il prédit en moyenne 95% les bonnes classes.

En somme, notre étude souligne l'importance de la qualité des données et de l'évaluation rigoureuse des performances des modèles de NLP. De plus, nos résultats suggèrent que les modèles basés sur les Transformers, tels que BERT, associés à des techniques de Transfer Learning, peuvent être particulièrement efficaces pour résoudre des problèmes de classification de paraphrases.

Perspectives Pour aller plus loin dans l'analyse, on pourrait tenter d'utiliser d'autres élements:

- Il aurait été intéressant (voire nécessaire de tester sur un échantillon de test adéquat).
- Stratifier l'échantillon par rapport à la target pour tenter d'éviter le problème d'imbalanced data comme dans le cas du set de test.
- Tenter d'autres modèles pré-entraînés d'Embedding tel que Word2Vec, Glove, voire même BERT.
- Les features synthaxiques, en utilisant des Part-of-Speech. Nous avons tenté rapidement d'intégrer ces features mais cela n'a pas été particulièrement pertinent à notre sens.
- Utiliser davantage des méthodes de régularisation pour éviter l'overfitting.
- Utiliser des méthodes de fine-tuning de réseaux de neurones qui tentent d'optimiser raisonnablement et permettent la généralisation des modèles

2.4 Contributions des membres du groupe

Ghiles Au sein de ce projet, j'ai géré la partie sur la préparation des données en effectuant des opérations de preprocessing et en choisissant un Embedding adapté. J'ai pris conscience de l'importance de cette tache notamment en adaptant les inputs du modèle d'après les retours de mes camarades selon la problématique. En outre, j'ai contribué à améliorer la lisibilité et la clarté du code dans son ensemble.

Yanis Au cours de ce projet, je me suis occupé de l'étude les méthodes d'identification de paraphrases et réalisé l'hypertuning des modèles. J'ai exploré la littérature sur les réseaux de neurones siamois et les Transformers, évaluant leurs avantages et inconvénients. J'ai également pu utilisé la Random Grid Search pour optimiser les hyperparamètres, car cette méthode offre un bon équilibre entre efficacité et performances tout en réduisant le temps et les ressources de calcul. Ainsi, j'ai développé une compréhension approfondie des modèles de traitement du langage naturel et de l'optimisation des hyperparamètres.

Armand Pendant ce projet, j'ai pu m'occuper de créer l'architecture du modèle Siamois LSTM ainsi que de choisir la distance appropriée pour notre problématique. Cela m'a permis de renforcer mes connaissances en matière de réseaux de neurones et de comprendre concrètement comment une architecture comme LSTM, peut être utilisée pour résoudre un problème de NLP.

Lucie Pour ma part, j'ai travaillé sur la création d'un modèle concurrent utilisant le Transfer Learning avec BERT. Ce projet a été particulièrement intéressant car j'ai eu l'opportunité d'expérimenter l'une des dernières technologies en matière d'IA, les Transformers. Étant donné que je travaille sur divers projets de NLP dans le cadre de mon alternance, notamment en clustering, j'ai trouvé particulièrement utile de compléter ces expériences par une approche supervisée. J'ai également tenté d'autres modèles pré-entraînés mais il me paraissait plus logique d'utiliser un modèle très générique puis tenter de le fine-tune réellement à notre problématique.