Projet 1 : Structured data

Antoine Prat - Matias Kondracki - Raphael Attali

28 Mars 2019

Abstract
Sujet : Show and Tell: A Neural Image Caption Generator.
Application sur le Dataset : Google AI Conceptual Captions

1 Analyse du papier

1.1 Contexte

La description automatique d'images recoupe deux branches distinctes de l'intelligence artificielle: une partie Computer Vision pour l'analyse de l'image, et une partie NLP pour permettre de générer la description. Il ne s'agit pas seulement de décrire les objets présents mais aussi de comprendre le contexte de l'image, c'est à dire la manière dont les différents objets interagissent entre eux.
D'un point de vue Bayésien, cela correspond à maximiser la vraisemblance $p(S|I)$ où I représente l'image à décrire et $S = \{S_1, S_2, ...\}$ la séquence de mots issue d'un dictionnaire représentant correctement l'image.

1.2 Model

Il s'agit donc de trouver le paramètre θ tel que:

$$\theta^* = argmax_\theta \Sigma_{(I,S)} log\ p(S|I;\theta)$$

On réécrit $p(S|I;\theta)$ en utilisant les propriétés de factorisation de la loi jointe:

$$log\ p(S|I;\theta) = \Sigma_{t=0}^{N} log\ p(S_t|I, S_0, .., S_{t-1};\theta)$$

Afin de maximiser pour tout t la probabilité $p(S_t|I, S_0, .., S_{t-1};\theta)$ il semble naturel d'utiliser un réseau de neurones récurrent (LSTM). On définit h_t comme l'état du LSTM conditioné par les $t-1$ premiers mots. Cet état est mis-à-jour à chaque fois qu'une nouvelle entrée apparaît : $h_t = f(h_{t-1}, x_t)$. Afin de déterminer cette fonction f, un Long-Short Term Momory(LSTM) sera utlisé dans cet article.
Le LSTM est un réseau récurrent de neurones dont le comportement est contrôle par trois couches de paramètres appelés "gates": l'imput gate, l'output gate et le forget gate.

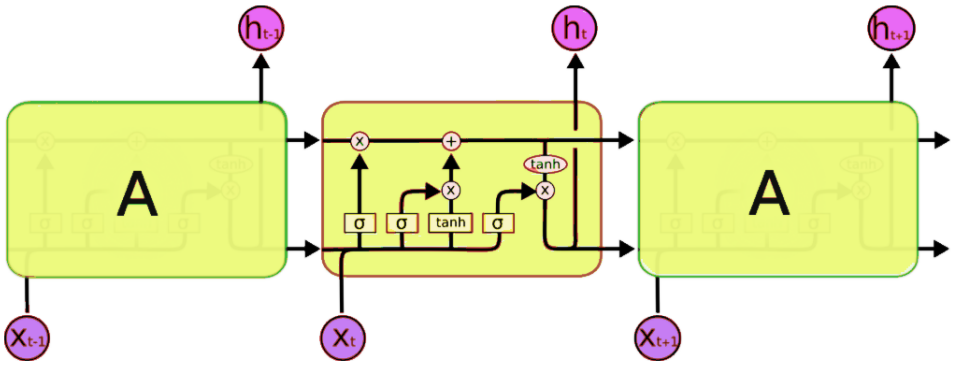

Figure 1: Représentation schématique du LSTM

Ces différentes couches de paramètres vont permettre de réguler par multiplication matricielle, l'impact de la nouvelle entrée x_t sur la sortie h_t ("imput gate"), l'impact de la dernière sortie h_{t-1} sur la nouvelle sortie ("output gate") ainsi que l'impact de la connaissance de toutes les valeurs des cellules précédentes sur la nouvelle sortie ("forget gate").

1.3 Training

Le LSTM est donc entraîné à prédire chaque mot de la phrase en partant de l'embedding de l'image ainsi qu'en ayant la conaissance des précédents mots de la phrase. On encode les images dans le même espace que celui de l'embedding W_e du vocabulaire et cela grâce à un réseau de neurones convolutif classique. On définit S_0 le "start word" et S_N le "stop word" qui délimitent la phrase générée. La procédure est alors décrite par les équations suivantes:

$$x_{-1} = CNN(I)$$

$$x_t = W_e S_t, t \in \{0...N-1\}$$

$$p_{t+1} = LSTM(x_t)$$

où S_t correspond au vecteur one-hot du mot issue d'une des vraies phrases correspondantes à la description de l'image. p_{t+1} correspond aux prob.
On peut dalors définir une fonction de perte à minimiser:

$$L(I,S) = -\Sigma_{t=1}^{N} log\ p_t(S_t)$$

Les paramètres de l'embedding des mots W_e, les paramètres du LSTM ansi que ceux du CNN encodant l'image peuvent donc être mis-à-jour à chaque itération lors de la phase d'entrainement

1.4 Inférence

Deux méthodes sont proposés afin de générer une phrase. Par ***Sampling***, ce qui consiste à choisir aléatoirement le t-ème mot en fonction de la probabilité p_t jusqu'à chosir un "end-word". Par ***BeamSearch***, où l'on va générer les k meilleurs phrases au temps t comme des candidats pour générer un nouveau mot. On ne retiendra encore que les k meilleurs phrases de taille t+1 générées.

1.5 Critiques

Un tel modèle a apporter des résultats très promettant. Il existe cependant un certain nombre de problèmes qu'il reste à résoudre pour parvenir à de meilleurs résultats.

L'encodage des images se fait grace à un VGG-16 qui n'est aujourd'hui plus le réseau de neurones convolutionnel le plus performant. Afin de capturer le plus d'information possible de l'image à décrire, il serait préférable d'utiliser un CNN plus récent par exemple ResNet ou SSD. Comme nous pouvons l'observer sur la figure 3, l'utilisation d'autres algorithmes peuvent largement améliorer notre score.

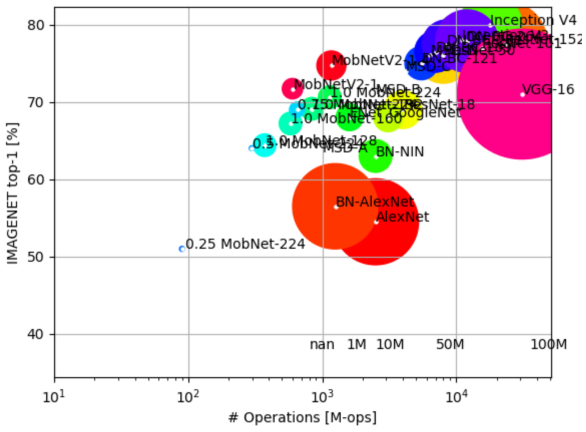

Figure 2: IMAGENET top-1 précision, la taille du rond dépend de la taille du modèle

Le modèle décrit utilise un LSTM unidirectionnel pour générer un mot en connaissance des mots précédents. Il semblerait plus intéressant de générer des mots grâce à la connaissance de tous les autres mots de la phrase, chose que l'on pourrait faire grace à un LSTM bidirectionnel.

Il n'existe dans ce modèle aucun procédé permettant de relier un mot de la phrase décrivant l'image à la région de l'image où se situe l'objet décrit. Il serait utile de connaître la région de l'image dont le mot employé

dans la description se réfère. Il manque donc à notre modèle le mécanisme dit d'*attention*. Ce mécanisme a la capacité d'extraire des informations supplémentaires concernant l'image décrite puisqu'en connaissant la région où se situe l'objet, on peut déterminer son importance dans l'image à décrire. On pourrait penser qu'un objet présent en arrière plan de l'image a en effet moins d'importance qu'un autre situé au premier plan. On pourrait alors proposer un modèle permettant d'afficher la région de l'image où se trouve l'objet comme on peut le voir sur la figure 3.

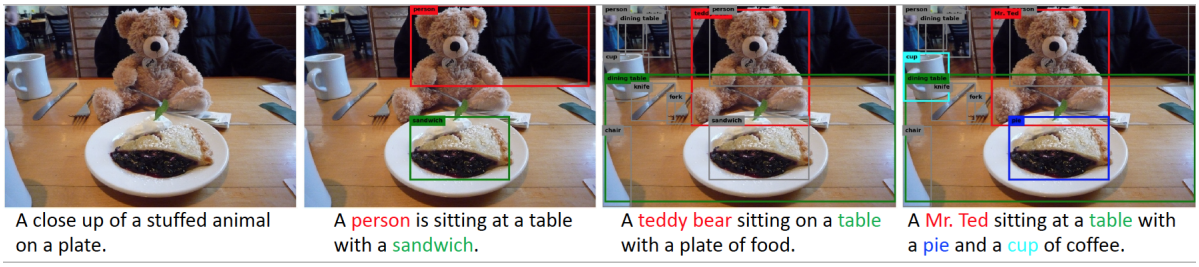

Figure 3: Cartographie de l'image

2 Rapport sur le code

2.1 Notes sur le dataset

Nous avons tout d'abord téléchargé le dataset de Google qui est disponible (avec explications) ici : `https://ai.googleblog.com/2018/09/conceptual-captions-new-dataset-and.html`. Ce dataset est différent de ceux sur lequel l'algorithme a été entrainé sur le papier puisqu'il est plus général que COCO (voir figure 4. Il est ainsi plus "apprécié" par les noteurs.

Model Architecture	Training Dataset	% of Captions with Favorable Human Ratings
RNN	COCO	27.6
T2T Transformer	COCO	36.2
RNN	Conceptual Captions	41.8
T2T Transformer	Conceptual Captions	50.6

Figure 4: Appréciation des commentaires par les humains sur deux datasets

En effet, tous les termes spécifiques ont été changés par des généralités comme on peut le voir dans la photo suivante :

Figure 5: Exemple d'une image

Pour plus d'informations sur le dataset, voir la bibliographie [2].

2.2 Notes sur le code

Nous nous sommes tout d'abord inspiré de la pipeline (et regarder un traitement post prédiction) qui se trouve ici : `https://github.com/Shobhit20/Image-Captioning`. Le reste a été codé par nos soins.

Comment se déroule notre code ?

Nous avons dû tout d'abord télécharger le dataset de Google, correspondant à un fichier tsv avec en première colonne l'annotation de l'image, suivi du lien de l'image. Nous avons donc télécharger les 200,000 premières images pour le training et la validation s'est faite avec 15,000 images. Nous aurions souhaiter télécharger plus d'images encore, mais cela nous prenait environ 15heures pour 100,000 images, nous avons donc du nous restreindre pour ce projet. Le code permettant de télécharger les images se trouve dans le fichier `get_data.py`. Le code va permettre de générer deux fichiers textes. Un fichier contenant les noms de toutes les images appartenant au train_set et test_set, et un contenant le nom de l'image et de son annotation.

Même si ce dataset provient de Google, celui-ci est rempli d'erreur (images n'existant plus, des photos qui se téléchargent sans contenu, des annotations remplis de symboles non compréhensible et des ","empechant de créer des DataFrames). Nous réglons toutes ces erreurs dans `preprocess_data.py`

Nous avons tout ensuite dû implémenter le CNN qui générera l'encodage des image, la feature-map sur laquelle nous utiliserons par la suite le LSTM pour générer la description. Le VGG (qui se trouve dans le fichier `vgg16.py`) n'a pas pu être entraîné sur les images de notre dataset car nous ne disposions pas des objets contenus dans chacune des images ainsi que de leur positionnement; nous avons donc dû effectuer un transfer learning en utilisant les poids de imagenet. Le modèle est directement tiré du Github de FChollet [3]. Comme décrit dans l'article, nous n'avons pas gardé la dernière couche du VGG (la couche permettant la classification des images) et utilisons la couche précédente, de taille 4096, pour générer l'encodage des images. Le fichier `imagenet_utils.py`, lui aussi tiré du Github de FChollet, permet un pre-processing des images pour leur donner les bonnes dimensions.

Dans le fichier `model_image.py` se trouve le code permettant de prendre le fichier text contenant tous les noms d'images appartenant au train_set pour obtenir en sortie un fichier `image_encodings.p` contenant l'encoding de toutes les images du training set.

Nous avons implémenté à travers l'API de Keras, le modèle d'annotation dans le fichier `model_text.py`. Celui ci contient une classe qui charge en entrée les images encodées, les poids s'ils existent, et entraîne le modèle. Les différentes fonctions permettent de créer le vocabulaire en fonction des valeurs d'entraînement, et de gérer le format des annotations. Le modèle a été entrainé directement à partir des annotations de notre dataset puisque qu'aucun poids n'a été fourni par l'article.

Le fichier `train.py` crée le modèle LSTM et l'entraîne, on sauvegarde les poids et le modèle dans des fichiers h5 pour faire le test après. On donne en entrée, le nombre d'epochs que nous souhaitons.
Nous rajoutons l'option de re-train l'algorithme à partir de poids déjà existants. Exemple de commande : `train.py 10 0` pour 10 epochs en entraînant depuis le début, et `train.py 10 1` en prenant des poids déjà existants.

Pour le fichier `test.py`. Nous lançons la fonction test, qui renvoie ses prédiction pour les images que nous avons défini dans la fonction text(). Il y a un post_processing des annotations dans le fichier `test_postprocess.py`. Ce fichier va choisir la meilleure annotation parmi les "beam_size" qu'on a généré. Il coupe aussi l'annotation à la première balise '<end>' qu'il rencontre.
Pour faire marcher le programme, il faut d'abord lancer `get_data.py` (très long, on va mettre les données directement dans le drive.) Il faut ensuite preprocess les datas avec `preprocess_data.py`
Ensuite `train.py 60 0`, Et finalement `test.py` qui va créer un fichier dans data/Text avec les prédictions des images de validations et le score bleu de celle-ci.

2.3 Résultats

Nous avons tout d'abord comparé (pour un même nombre d'epochs) les résultats entre notre nouveau dataset contenant 200,000 images peu génériques, et les datasets présents dans le papier. L'algorithme présenté dans le papier est pas très peu robuste et n'arrive pas, sur nos images synthétiques, à capturer les informations utiles. En effet, les prédictions sont à peu près toutes les mêmes. Pour réduire la difficulté nous avons alors réduit notre dataset à 100,000 image. Les images restent très différentes et la variance ne permet donc pas pour cet algorithme de bien apprendre. Pour réellement réussir nous aurions dû : Soit choisir 100,000 images similaires, soit intégrér les 3 millions d'images du dataset (donc plus de 300 heures pour les télécharger). L'algorithme marche cependant très bien sur des datasets avec moins de variances comme ceux présent dans l'article [1].

Une baseline comme m-RNN ne marche pas non plus, cependant nous pensons qu'un algorithme comme Neural Baby Talk [4] qui capture les objets présent et s'en sert pour créer les captions.

Vous pourrez trouver le code sur ce github : `https://github.com/Antolivprat/NIC-ImageCaptionning`

References

[1] Oriol Vinyals, Alexander Toshev, Samy Bengio, Dumitru Erhan *Show and Tell: A Neural Image Caption Generator*. Google

[2] Piyush Sharma, Nan Ding, Sebastian Goodman, Radu Soricut *Conceptual Captions: A Cleaned, Hypernymed, Image Alt-text Dataset For Automatic Image Captioning*. Google AI

[3] François Chollet `https://github.com/fchollet/deep-learning-models/blob/master/vgg16.py`. Code de `vgg16.py`

[4] Jiasen Lu, Jianwei Yang, Dhruv Batra, Devi Parikh *Neural Baby Talk* `http://openaccess.thecvf.com/content_cvpr_2018/CameraReady/0205.pdf`.

[5] Bert Moons, Marian Verhelst *Embedded Deep Learning - Algorithms, Architectures and Circuits for Always-On Neural Network Processing* `https://github.com/BertMoons/Comparing-CNN-Architectures`.