INALCO
INSTITUT NATIONAL DES LANGUES ET CIVILISATIONS ORIENTALES

Analyse et évaluation à grande échelle du moteur de traduction neuronale OpenNMT

Membres

Liza FRETEL

Kenza AHMIA

Alice WALLARD

Enseignant encadrant

Mr. N.SEMMAR

TABLE DES MATIÈRES

Introduction

La traduction automatique neuronale (TAN) s'inscrit dans le cadre plus vaste de la traduction à base de corpus. Les systèmes de traduction à base de corpus s'appuient sur les probabilités et ne cherchent pas à gérer la syntaxe, contrairement aux systèmes à base de règles qui étaient à l'origine des premiers

Alors que les **modèles de traduction statistique** associent des calculs de probabilités issus de modules différents, la **traduction neuronale** repose elle sur **un seul module intégré** dont l'architecture est celle d'un **réseau de neurones** (*Artificial Neural Network* ANN). Dans ce contexte, les modèles de traduction sont des modèles de classification **par apprentissage**, dont la tâche principale est de "prédire" une valeur en sortie après avoir été **entraîné** sur un grand nombre de données. Cet entraînement est une opération délicate qui fait jouer plusieurs paramètres. Il est donc d'autant plus difficile de faire des ajustements si le nombre de paramètres est très important.

Dans le cas de la traduction neuronale, les paramètres sont des poids. Comparé à un système statistique classique, les caractéristiques du réseau de neurones incluent la capacité à ajuster ses poids de manière autonome, la possibilité d'extension en ajoutant des couches de neurones, ainsi que l'intégration de tous les modules et mécanismes au sein du réseau.

Ce qui nous amène à ce projet qui s'inscrit dans le cadre du cours de Traduction automatique de la formation Master 1 en Traitement Automatique des Langues, visant à analyser et évaluer à grande échelle le moteur de traduction neuronale OpenNMT.

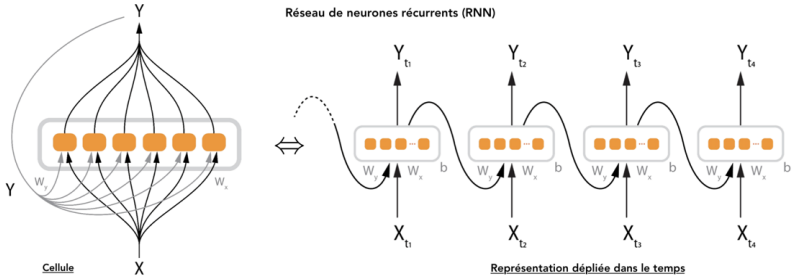
Pour mener à bien ce projet, nous disposons de plusieurs éléments. Tout d'abord, nous avons accès à des corpus de données tels que **Europarl** et **emea**, qui serviront de bases de données pour entraîner et tester notre système de traduction. De plus, nous utilisons le fichier **mosesdecoder**, qui contient les scripts nécessaires à la préparation et l'analyse des données. Enfin, le fichier moses.env est utilisé pour configurer l'environnement d'exécution des scripts.

Le principe de notre projet est de développer une compréhension approfondie du fonctionnement et des performances du moteur de traduction neuronale OpenNMT. Nous étudions les différents aspects de la traduction automatique neuronale, tels que les architectures de réseaux de neurones utilisées, les techniques d'apprentissage automatique, et les méthodes d'évaluation des performances.

OpenNMT est réputé pour sa capacité à générer des traductions de haute qualité grâce à son approche basée sur les réseaux de neurones. Notre projet vise à analyser et à évaluer les avantages et les limitations de cette approche, en tenant compte des spécificités des langues sources et cibles, ainsi que des différents domaines de traduction.

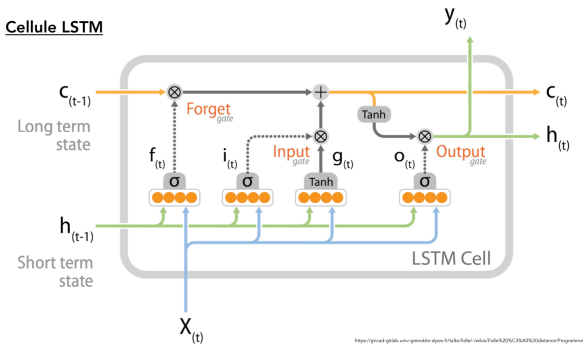
1. Présentation du moteur de traduction neuronale OpenNMT

- LSTM, RNN, Transformers...

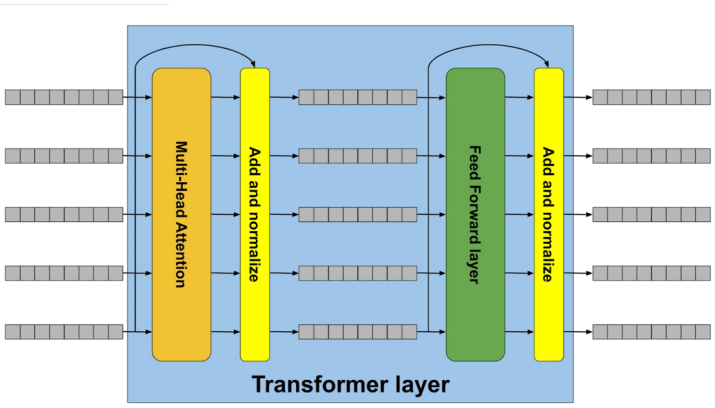


Un neurone récurrent est une extension d'un neurone classique avec l'ajout d'un deuxième poids et de l'état de la sortie précédente. Le problème des réseaux de neurones récurrents classiques est que la

mémoire est vite perdue au bout de plusieurs séquences.
Pour pallier cette faiblesse, les réseaux de neurones **à mémoire court ET long terme** (*Long Short Term Memory* LSTM), ont été développés. La mémoire à long terme y est effacée beaucoup plus doucement.
Dans la cellule, la mémoire rapprochée est le résultat de l'itération précédente, le vecteur de sortie est dupliqué et conservé dans la mémoire courte mais, et c'est là la nouveauté, il y aussi une duplication dans une mémoire à long terme. Les *gates* sont des sortes de vannes pilotées par des couches de neurones, qui vont gérer 3 composantes : la mémoire long terme, la mémoire court terme et l'entrée, selon l'importance qui leur sera accordée. Les cellules GRU sont des versions simplifiées des cellules LSTM.



Le **transformer** est une architecture qui crée un encodage de la position des tokens (positional encoding sur 4 bits), et des fonctions sin et cos (parallélisme). En association avec les embeddings, ce système permet de confronter les mots les uns aux autres en sachant s'ils sont proches ou éloignés et dans quel ordre ils ont été positionnés, et ce, sur toute la taille de la séquence.
Cette architecture permet également de faire un entraînement commun avant la spécialisation par tâches (résumé, traduction, etc.) et d'utiliser plusieurs GPU. Les transformers sont désormais utilisés par tous les modèles d'analyse du langage.
Le **mécanisme d'attention** est une des caractéristiques principales du transformer. Il permet de créer des correspondances entre les différents vecteurs de tokens en créant en sortie une matrice de vecteurs où chaque token est pondéré en fonction de son importance dans la séquence. Il peut être **multi-tête** ce qui signifie que les matrices sont découpées et le réseau calcule davantage de matrices de sortie qui sont ensuite concaténées. Cette étape supplémentaire permet de créer plus d'information et d'empêcher de favoriser certains éléments au détriment d'autres.



- Présentation de la distribution OpenNMT

OpenNMT est une distribution modulaire, présentée par ses auteurs comme "*un écosystème open source pour la traduction automatique neuronale et l'apprentissage des séquences neuronales*", lancée en 2016 par Harvard NLP et SYSTRAN[1]. Elle comprend deux applications : OpenNMT-tf et OpenNMT-py,

[1] https://opennmt.net/

qui s'appuient respectivement sur les frameworks TensorFlow2 et PyTorch. Dans le cadre de notre projet, nous avons utilisé la version 3.0 d'OpenNMT-py.

- Le système de traduction de OpenNMT-py

OpenNMT-py propose différents types d'architectures de RNN. Ces architectures exploitent les mécanismes d'attention, d'auto-attention, les convolutions et les transformers (cf. Annexe 2).
Exemples de paramètres : le nombre de tête du mécanisme d'attention (***--heads, -heads***), réglé à 8 têtes par défaut. Le type d'architecture de l'encodeur et du décodeur est réglé par défaut sur '***rnn'***, il est précisé que les couches ***non_RNN*** sont expérimentales. (***--encoder_type, -encoder_type au choix : rnn, brnn, ggnn, mean, transformer, cnn, transformer_lm***).

2. Evaluation du moteur de traduction neuronale OpenNMT sur un corpus en formes fléchies

- Le corpus d'apprentissage et d'évaluation

Le corpus d'apprentissage est constitué de 100 000 phrases issues du corpus Europarl aléatoirement ainsi que de 10 000 phrases issues du corpus EMEA aléatoirement. Ces phrases ont toutes 80 mots ou moins. Le corpus Europarl est issu de discours du Parlement européen, tandis que le corpus EMEA est issu de documents de l'Agence européenne des médicaments. Il s'agit de deux corpus de domaines totalement différents, ce qui a influencé l'apprentissage et la qualité de la traduction finale.

- Le score BLEU

Le score BLEU mesure la ressemblance entre deux textes, en s'appuyant sur les précisions n-grammes et une pénalité de concision. Il a donc une valeur indicative de ressemblance et non pas de qualité de traduction. Certains affirment qu'il a été démontré que les scores BLEU coïncident avec ce qu'un humain peut penser de la qualité d'une traduction[2]. Il est en tout cas très utile dans le cadre de l'évaluation d'un système de traduction. Il est exprimé par un chiffre de 0 à 1 ou bien un pourcentage.

- Tableau des résultats

Corpus Europarl seul (10k)	**Corpus Europarl (100k) + EMEA (10k)**	
Train perplexity: 1.83705	Train perplexity: **4.61314**	
Train accuracy: 87.9025	Train accuracy: **66.0961**	
Validation perplexity: 1294.76	Validation perplexity: **8.8166**	
Validation accuracy: 40.4095	Validation accuracy: **60.4041**	
Score BLEU: **12.25760** (après 35k époques d'apprentissage)	Score BLEU **Europarl** (général): **29.551395**	Score BLEU **EMEA** (spécifique): **23.26700854905571**

La perplexité et la précision de validation du corpus Europarl de 10 000 phrases sont mauvaises. En revanche, la perplexité et la précision de l'entraînement sont très bonnes. Ceci est dû au fait que le

[2] https://cloud.google.com/translate/automl/docs/evaluate?hl=fr#bleu

corpus est très petit, alors le modèle apprend par cœur les données, il y a sur-apprentissage. Le score BLEU moins élevé confirme qu'avec un corpus plus grand, comme celui de la colonne de droite, on obtient des traductions de meilleure qualité.

3. Evaluation du moteur de traduction neuronale OpenNMT sur un corpus en lemmes

- Le lemmatiseur NLTK pour l'anglais

Le lemmatiseur NLTK pour l'anglais est WordNetLemmatizer. La version lemmatisée des mots est presque identique en anglais, alors il n'y a pas beaucoup d'erreurs.

- Le lemmatiseur NLTK pour le français

Le lemmatiseur pour le français est FrenchLefffLemmatizer. Il présente quelques erreurs quant aux lemmes. En effet, sans utiliser les parties du discours, il est difficile de connaître le lemme de certains mots, par exemple, le lemme du mot "avions" (verbe avoir) a été lemmatisé en "avion". Il faudrait fournir au lemmatiseur la liste des POS associés aux formes des mots, avec la commande : lemmatize(mot, POS.

- Tableau des résultats

Apprentissage	**Test sur le corpus général (Europarl)**	**Test sur le corpus spécifique (EMEA)**
Train perplexity: 7.52	Score BLEU: **31.300787339398234**	Score BLEU : **24.940690004913076**
Train accuracy: 60.08		
Validation perplexity: 8.43		
Validation accuracy: 60.32		

Les résultats de l'apprentissage sont très similaires à ceux de l'apprentissage précédent. Nous pouvons constater une légère amélioration du score BLEU. Cependant, la traduction produite est en lemmes, ce qui signifie qu'il faudrait la retranscrire en formes.

4. Points forts, limitations et difficultés rencontrées

L'évaluation du moteur de traduction neuronale OpenNMT a révélé plusieurs points forts, limitations et difficultés rencontrées.

Points forts

- L'aspect modulaire du système OpenNMT-py et les tutoriels clairs de Yasmin Moslem ont facilité l'exploration des différentes fonctionnalités et tâches offertes par le moteur de traduction.
- OpenNMT-py offre une grande flexibilité pour personnaliser et ajuster les modèles de traduction en fonction des besoins spécifiques du projet.

- La qualité des traductions générées par OpenNMT est souvent élevée, et ce grâce à l'utilisation des Transformers, technologie la plus avancée à ce jour pour les tâches de TAL.

Limitations

- Sur MacOS, l'utilisation du GPU pour l'entraînement n'est pas possible, ce qui limite les ressources disponibles et peut entraîner des temps d'entraînement plus longs.
- L'entraînement du modèle demande beaucoup de temps et d'efforts, ce qui peut entraîner une surchauffe de la machine lors de longues sessions d'entraînement. Cela peut affecter les performances globales de la machine et entraîner des ralentissements ou des pannes.

Difficultés rencontrées

- L'utilisation de ressources limitées sur certains systèmes, comme l'absence de GPU sur MacOS, a posé des défis pour l'entraînement efficace des modèles et a nécessité des ajustements et des optimisations supplémentaires.
- La gestion de la surchauffe de la machine lors de longues sessions d'entraînement a nécessité une surveillance constante et des mesures pour maintenir la stabilité du système.
- Les temps d'entraînement prolongés ont exigé une planification minutieuse pour s'assurer que les ressources informatiques soient disponibles pendant de longues périodes et que les éventuelles pannes ou interruptions soient minimisées.
- Au niveau de l'apprentissage, nous avons commis quelques erreurs, notamment celle d'avoir mis un nombre de samples (paramètre n_sample) trop petit (seulement 10k) lors de la constitution du vocabulaire (avec la commande onmt_build_vocab), ce qui a fait que le modèle ne reconnaissait pas suffisamment de mots et traduisait beaucoup de phrases par <unk>. Pour régler ce problème, nous avons mis **n_sample à 110 000**, soit la taille du corpus d'apprentissage. Autre erreur commise : les deux corpus d'entraînement étaient en anglais dans le fichier de configuration, ce qui a fait que beaucoup de <unk> apparaissaient dans la traduction finale. Enfin, un problème matériel est survenu après 7 000 itérations lors de l'entraînement du modèle sur les phrases lemmatisées, et ce à chaque fois que nous lancions l'apprentissage. Après avoir redémarré la machine, nous avons pu reprendre l'apprentissage à partir d'un checkpoint avec la commande :

```
onmt_train -config lemmaEuroEmea/Europarl.yaml -train_from lemmaEuroEmea/run/model_step_7500.pt -continue
```

5. Organisation

Afin d'assurer une bonne entente et un travail d'équipe harmonieux, nous avons choisi d'utiliser GitHub comme plateforme de collaboration pour notre projet. GitHub offre un environnement propice à un travail de groupe autonome, permettant à chaque membre de contribuer au projet, de le modifier et de visualiser les modifications apportées par les autres membres.

Chaque membre du groupe a participé activement au projet en travaillant individuellement et en accomplissant toutes les étapes requises, tout en bénéficiant de l'entraide mutuelle.

Étant donné la taille importante des fichiers, Liza, qui disposait d'une carte graphique NVIDIA, s'est volontiers chargée des entraînements nécessitant plus de ressources. Pendant ce temps, Kenza et Alice, qui utilisaient des machines Mac, et pour qui l'entraînement prenait environ 20 heures pour atteindre 10 000 étapes. Elles ont donc choisi de se concentrer sur les étapes réalisées sur le petit corpus afin de mieux comprendre la méthodologie et le fonctionnement d'OpenNMT-py, tout en suivant de près les

progrès réalisés par Liza. Finalement, grâce à cette approche collaborative, chaque membre du groupe a réussi à contribuer de manière significative au projet en accomplissant toutes les tâches requises.

La rédaction du projet s'est faite en groupe, offrant à chacun la liberté de modifier ou d'ajouter des éléments. Cette approche collaborative a favorisé l'échange d'idées, la consolidation de nos connaissances et la création d'un travail final qui reflète les contributions de chaque membre.

6. Annexes

- Captures d'écran des résultats

- Exemple ayant échoué (corpus Europarl 10k)

```
[2023-05-25 04:25:18,792 INFO] Step 9900/10000; acc:  80.65; ppl:  2.04; xent: 0.71; lr: 1.00000; 302/336 tok/s;  58093 sec;
[2023-05-25 04:30:06,027 INFO] Step 9950/10000; acc:  80.26; ppl:  2.07; xent: 0.73; lr: 1.00000; 308/348 tok/s;  58380 sec;
[2023-05-25 04:34:39,944 INFO] Step 10000/10000; acc:  80.13; ppl:  2.08; xent: 0.73; lr: 1.00000; 307/346 tok/s;  58654 sec;
```

→ Il s'agit d'une **précision d'apprentissage**. Sur un petit corpus, si la précision augmente considérablement, cela signifie que le réseau a réussi à mémoriser les exemples et qu'il pourrait être mauvais lorsqu'il est confronté à de nouvelles données. C'est pourquoi nous "testons" les réseaux sur des données de test qui diffèrent des données d'apprentissage (et d'ailleurs, nous calculons le score BLEU sur les données de test).

- Expérimentations supplémentaires

Tentative d'adaptation des scripts car impossibilité d'utiliser le gpu sur Mac OS

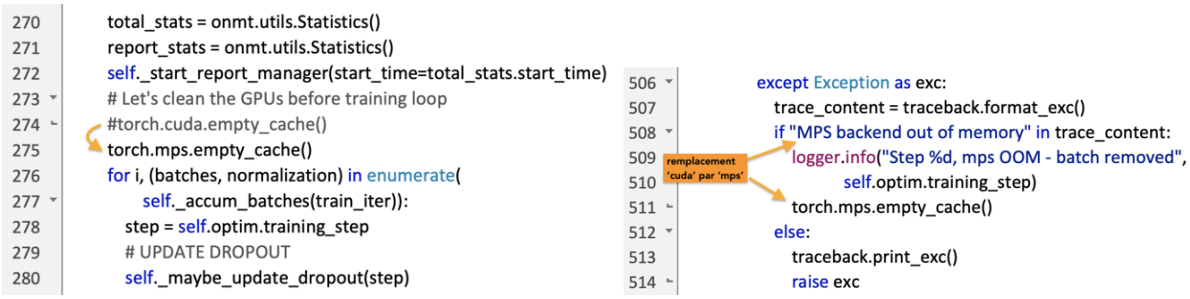
`AttributeError: module 'torch._C' has no attribute '_cuda_setDevice'

Cuda est un driver qui permet d'envoyer des calculs au processeur graphique pour délester le processeur principal. OpenNmt a été développé avec Cuda. Cependant, les dernières versions de Mac OS (jusqu'à 10.13) ne sont plus compatibles avec ce driver. Utiliser PyTorch sans cet accès à la carte graphique est dommage notamment dans le contexte de calculs traitant de gros volumes de données, et qui prennent, de ce fait, **énormément de temps**. D'autant que l'intérêt des transformers est de pouvoir utiliser les gpu.

Plus récemment, des avancées ont permis d'**utiliser PyTorch avec la carte graphique installée sur les Mac**[3], mais cela n'est possible qu'avec les dernières versions. Concernant les versions de mac OS inférieure à Ventura (13.), il est possible d'utiliser le **module Pytorch MPS backend**[4]. En pratique, après installation de l'API torch.mps, le device devient mps au lieu de cuda : torch.device("mps") au lieu de torch.device("cuda"). On peut envisager de le remplacer dans les scripts. Cependant, on se heurte à des insuffisances : le module MPS backend est encore en phase de prototypage et les scripts du package Opennmt-py sont écrits pour le gpu cuda, ou pour l'unité centrale de calcul (cpu), il est donc **difficile de les adapter** entièrement. Après un moment on butte sur des incompatibilités, par exemple le module mps ne prend pas en charge le calcul de la *mixed precision* avec la fonctionnalité *pytorch.amp*.

[3] https://towardsdatascience.com/installing-pytorch-on-apple-m1-chip-with-gpu-acceleration-3351dc44d67c
[4] Accelerated PyTorch training on Mac - Metal - Apple Developer

Exemple de tentative d'adaptation de la fonction *empty_cache()* **dans le script *trainer.py*:**



- Comment optimiser l'apprentissage du modèle

- Après avoir rencontré un problème de sur-apprentissage au bout d'un certain nombre d'époques, il serait pratique de pouvoir visualiser en temps réel l'**évolution des quatre variables (précision et perplexité) sur le corpus d'apprentissage et de validation**. Cela permettra de choisir la meilleure version du modèle, à savoir celle qui minimise l'erreur à la validation. Il existe une option TensorBoard pour visualiser la progression de l'apprentissage : https://forum.opennmt.net/t/running-tensorboard/4242
- Pour accélérer l'apprentissage, il faudrait considérer de faire **varier le Learning Rate de façon décrémentale**, de sorte à ce que le Learning Rate soit plus grand au début de l'apprentissage et diminue au fur et à mesure afin d'affiner avec plus de précision les poids des neurones (**fine-tuning**).

- Tableau des paramètres

PARAMÈTRES OpenNMT-py v3

(S'il y a plusieurs paramètres, leur nombre est entre parenthèses)

Build : build_vocab.py	Train : train.py	Translate : translate.py
21 types de paramètres	34 types de paramètres	30 types de paramètres
total : 87 paramètres	total : 147 paramètres	total : 123 paramètres
	Model :	Beam Search (2)
	Embeddings (8)	Random Sampling (4)
	Embedding Features (3)	Penalties (5)
	Task	Decoding tricks (7)
	Encoder-Decoder (27)	Efficiency (4)
	Attention (8)	
	Alignement (4)	
	Optimization :	
	Type (25)	
	Rate (6)	
	Generator (13)	
	General (16)	
	Initialization (8)	
	Dynamic data (4)	
	Pruning (2)	
	Embeddings (4)	
	Logging (14)	

Paramètres commmuns aux trois étapes :

Paramètres de configuration:
Configuration (2 chemins)
Data (11)
Vocab (5 dont 2 chemins)
Features (2)
Transform:
Filter (2 src et tgt)
Prefix (2 src et tgt)
Suffix (2 src et tgt)
FuzzyMatching (7 dont 1 chemin)
InferFeats
InlineTags (7 dont 1 chemin)
Normalize (7)
BART (8) pas pour build_ vocab
Docify (2)
SwitchOut pas pour build_ vocab
Token_Drop
Token_Mask
Uppercase
Subword/Common (10)
Subword/ONMTTOK (5)
Clean (9)
Reproducibility

Bibliographie

F. Barbin. 2020. "La traduction automatique neuronale, un nouveau tournant ? ". *Palimpseste*. Sciences, humanités, sociétés. 4. pp.51-53.
URL : https://shs.hal.science/halshs-03603588/file/palim_4_WEB_Barbin%20(1).pdf

G. Klein et al. 2017. "OpenNMT: Open-Source Toolkit for Neural Machine Translation".
URL: https://www.aclweb.org/anthology/P17-4012/

G. Klein, F. Hernandez, V. NGuyen, J. Senellart, *The OpenNMT Neural Machine Translation Toolkit: 2020 Edition*, in *Proceedings of the 14th Conference of the Association for Machine Translation in the Americas October 6 - 9, 2020.*
URL: https://aclanthology.org/2020.amta-research.9.pdf

M. Benjamin. Mémoire de stage Master M2, *Amélioration a posteriori d'une traduction automatique par recherche locale*, LIMSI - CNRS, Équipe Traitement du Langage Parlé.

T. Lam, J. Demange, J. Longhi. Jan 2021. Attribution d'auteur par utilisation des méthodes d'apprentissage profond. *EGC 2021 Atelier "DL for NLP : Deep Learning pour le traitement automatique des langues"*. Montpellier, France.
URL : https://hal.science/hal-03121305/

Sitographie

Le site officiel d'OpenNMT : https://opennmt.net

GitLab Fidle : supports de la formation "Introduction au DEEPLearning" :
https://gricad-gitlab.univ-grenoble-alpes.fr/talks/fidle/-/wikis/home

Github de Lilian Weng : https://lilianweng.github.io/posts/2018-06-24-attention/

Histoire de la traduction automatique :
https://www.freecodecamp.org/news/a-history-of-machine-translation-from-the-cold-war-to-deep-learning-f1d335ce8b5/

Comprendre le score BLEU : https://cloud.google.com/translate/automl/docs/evaluate?hl=fr#bleu

Mécanismes d'attention : Formation Fidle https://www.youtube.com/watch?v=L3DGgzIbKz4