Rapport - Stage Cardiff

Antao Amory

Lundi 10-06 :

Mardi 11-06 :

Mercredi 12-06 :

Jeudi 13-06 :

Vendredi 14-06 :

Matin ça clique. Meeting avec Steven Schockaert, il a recentré le truc sur les transformers.
Prochaine étape : écrire un transformer from scratch, les chemins dans le graphe de connaissance vont être sous la forme de séquences (quoi que ça veuille dire), et après il faudras voir pour les marches aléatoires (où c'est nécessaire, comment l'implémenter, à quoi ça sert ? ?).

Bon pour le transformer, il va falloir le faire from scratch (car c'est tricher si on en utilise un pré-entrainé). Attention, il existe deux principales tâches pour ces transformers : soit on a une phrase et on masque un mot et il faut le retrouver (ça, ça ne nous intéresse pas) ou soit il faut déterminer comment la va être la suite d'une phrase (ça ça nous intéresse !).

Pour m'introduire au sujet des transformers, je vais regarder :
https ://jalammar.github.io/illustrated-transformer/
(et en particulier la traduction française.
On va résumer ça ici :

Le seq2seq et le processus d'attention Un modèle seq2seq est, comme son nom l'indique, un modèle qui prend une séquence (de mots, de lettres, ou de plus ou moins tout ce que l'on veut) et qui en resort une autre.

Le modèle est décomposé en deux parties : un encoder et un décoder.
L'encoder prend la séquence en entrée, traite chaque élément et met le résultat (à priori un vecteur de flottants) de ce traitement dans un vecteur (appelé context), qu'il fait passer au décoder, qui lui décode le context item par item pour réobtenir une séquence.

En pratique, l'encoder et le décoder sont souvent des réseaux de neuronnes récurrents (RNN). Un RNN prend deux entrées à chaque pas de temps, une entrée classique (ex : un élément de notre séquence) et un état caché. De même, il produit une sortie classique et l'état caché suivant (celui que l'on réutilise, d'où le récurrent).
Dans notre cas, le dernier état caché produit par l'encoder va être le context que l'on fourni au décoder.
Dans le contexte des traductions automatique, le problème était qu'avec de longues phrases, les différentes étapes du décoder ne pouvait pas bien prendre en compte les données importantes.

C'est pourquoi le concept d'attention a été créé. Il a permit d'amplifier le signal des parties importantes de la séquence d'entré suivant à quelle étape du décoder on était.
Le modèle nouveau modèle ainsi créé, modèle d'attention, a deux principales différences avec le modèle seq2seq.
1 - L'encoder ne passe plus uniquement que le dernier état caché mais tous les états cachés intermédiaire (beaucoup plus de données).
2 - Le décoder donne un score (un peu mystérieux pour le moment) à chaque état caché puis applique un softmax (pour si le score est s_i alors $\frac{e^{s_i}}{\sum e^{s_j}}$), et prend ce nouveau vecteur comme context.

Ainsi au final, le RNN du décoder fait comme précédemment, sauf que sa sortie n'est pas la sortie définitive. En effet on applique l'étape de l'attention, et au colle le vecteur que l'on obtient à la précédente sortie. On fait passer le tout dans un réseau neuronal feed-forward (entraîné en même temps que le modèle), et cette sortie là est la bonne.

Illustration du word embedding et de word2vec

Illustration du transformer Le transformer est est composé d'une pile d'encoders et d'une pile de décoders de même nombre.
Tous les encoders ont une structure identique : ils sont divisés en deux sous-couches, d'abord la self-attention puis le feed forward.
Les décoders ont la même structure à laquelle on a ajouté une couche d'attention (comme dans le seq2seq) qui est entre la self-attention et le feed forward.

Tout d'abord, chaque mot d'entrée est transformé en un vecteur (embedding).

Tous les mots d'entrée avances selon leurs propre chemin, et sans dépandance lors de la couche de feed forward (on peut donc exécuter les différents chemins en parallèle dans ces couches).

La couche de sefl-attention a pour objectif d'améliorer la compréhension, le sens du mot qu'il traite (comment il est relié sémantiquement aux autres mots), par exemple dans "The animal didn't cross the street because it was too tired", ça a pour vocation de relier le mot "it" à "The animal" (et non à "street" ou a autre chose).
En pratique, la self-attention crée trois vecteurs pour chaque vecteur d'entrée, la query, la key et la value, en multipliant le vecteur d'entrée par trois matrices obtenue par le processus d'entraînement.
Une fois ces trois vecteurs obtenues, on calcule un score pour chaque mot (représentant le degré de concentration à placer sur les autres mots) en prenant le produit scalaire de la query de l'entrée et de la key du mot dont on calcule le score.
On divise ces scores par la racine carré de la dimension des vecteurs, puis on applique une la fonction softmax (ce que l'on obtient correspond à l'importance (entre 0 et 1) d'un mot vis à vis du mot d'entrée).
Pour finir, on somme le produit de la value et du softmax, pour que la value des mots important soit plus pris en compte que celle des autres mots.
Une fois fait cela pour toute les entrées, on a terminé la couche de self-attention.

Cependant, les calculs avec les vecteurs sont long, donc en pratique on traite avec des matrices pour gagner en efficacité.
Pour ce faire, on concatène toutes les entrées en une matrice, que l'on multiplie par les matrices que l'on a entraîné pour calculer la value, la key et la query (qui sont donc maintenant des matrices, et non plus des vecteurs). On peut alors rassembler toutes les étapes restantes en une seule.
Si, si Q est la query, K la key, V la value et d la dimension des vecteurs, on calcule la sortie Z par $Z = \text{softmax}(\frac{Q \times K^T}{\sqrt{d}})V$.
De plus, on peut encore améliorer cela en considérer plusieurs têtes (8). Le principe est que l'on crée dans chaque tête une matrice Q, une matrice K et une matrice V (qui sont différente, puisque on considère que les matrices permettant des les obtenirs sont choisie aléatoirement au départ). Il ne suffit plus que de les refusionner en une seule matrice (pour la sortie), ce que l'on fait en concaténant toutes les sorties puis en les multipliant par une matrice W^0 entraîné parallèlement.

D'autre choses en vrac, important en pratique mais pas pour comprendre. Il est utile de rajouté en entrée un encodage positionnel. Il y a une formule, c'est assez bizarre. De plus, entre chaque couche il y a une partie de normalisation.

Pour le décoder, tout est assez similaire même si rien n'est pareil...

Le décoder renvoie un vecteur de flottant. Il faut donc le traiter et pour se faire on a deux couches finales : linéaire et softmax. La couche linéaire est un réseau neuronal qui projette le vecteur de sortie dans un vecteur de logits, aussi large que tout le vocabulaire que l'on connait, et qui va lui donner un score, que l'on transforme en probabilité par la couche softmax (et le mot obtenu est celui de probabilité la plus grande).

Pour entraîner le modèle, l'on doit déjà connaître l'ensemble de notre vocabulaire. Une fois connu, on peut créer un vecteur correspondant à chaque mot (le vecteur est le one-hot, équivalent de l'indicatrice).
On peut alors faire tourner le modèle pour obtenir un vecteur de sortie, et faire du rétro-pédalage dans le but de modifier les poids pour être plus proche de la réalité.

Lundi 17-06 :

Implémentation d'un transformer : Je suis un tuto youtube.
Les mots que je veux c'est une suite de relation formant un chemin dans notre KG.
L'objectif est de trouver le mot suivant dans la séquence.
J'implémente, ça ne marche pas hyper bien.
Je regarde un autre tuto, sur une page, pour un translater d'anglais vers kannada.
Je suis pas à pas pour comprendre les étapes.
Ça marche quasiment directement (seulement quelques debug, je suis content.

Mardi 18-06 :

Je continu sur le traducteur. En réalité ça ne marche pas tout à fait bien.
Je l'ai fait tourner sur le supercalculateur la nuit et j'ai eu des erreurs.
Je cherche et je trouve un problème avec les tokens (ils étaient tous codé par "").
Après je fais tourner et ça marche bien. Le problème c'est que je ne connais pas le kannada.
Je vais donc chercher des datas pour la traduction anglais-français (kaggle).
Je traite les données pour avoir ce que je veux, et je modifie légèrement le code.
Je lance et ça marche plutôt bien.
Là je suis content :).
Ok mais ce que je veux moi c'est pas un traducteur mais un truc sur les graphes de connaissance.
Je reprend le transformer-traducteur comme base.
En vocabulaire je met la liste des relations et la liste des noeuds (important).
Pour se faire je doit traiter les données.

Après j'importe mes données de trainning et je fais une marche aléatoire pour obtenir une liste de chemin entre des points.
Première version : complètement aléatoire mais ça n'a aucune chance de donner quoi que ce soit.
Second version : j'utilise networkx et pour mettre une distance entre deux points et pouvoir pondérer mes probabilités.
Je fais en fonction du plus court chemin jusqu'au noeud d'arrivé (avec la fonction nx.shortest_path_length).
Je normalise après avec un sorfmax (où le poids est multiplié par deux car desfois il y a vraiment beaucoup de voisins).
Le random walk marche très bien, mais en réalité c'est tout bonnement inenvisageable car beaucoup trop long à calculer sur de grandes entrées.
Après je fais une fonction pour obtenir des random walk entre point aléatoire.
Et je vais dormir.

Mercredi 19-06 :

Je continu de faire le transformer pour le KG.
J'ai créer le dataset, et je suis en train de coriger les erreurs dans le transformer pour pouvoir lancer les epoch comme il faut. Pour le moment tout le reste marche comme il faut.
Pas grand chose à dire car c'est pas mal de débuggage mais ça avance bien donc c'est cool.
Il faut notamment bien faire gaffe à la taille des input et des output.
En effet je veux que les input soit de taille 2 (noeud de départ et noeud d'arrivé) tandis que les output sont des séquences de relations, de taille variable (normalisé à 200 car c'est bien mieux si elles ont toutes les même tailles).
Autre chose tricky à géré c'est lorsqu'il y a plusieurs relations différentes entre les mêmes noeuds. Ça m'a pris pas mal de temps.
J'ai géré en choisissant aléatoirement quel relation je prennais. Peut être d'autre manière où les probas sont pondéré (ex : si une relation est beaucoup plus présente dans le graphe que l'autre) mais on verra plus tard si jamais.
Autre problème d'avoir les inputs et les outputs de taille différente c'est pour la MultiHeadCrossAttention

Jeudi 20-06 :

J'ai tout mis au propre dans un github, c'est mieux !
J'ai séparer mes fichier pour avoir un module_transformer et un truc avec les données, c'est mieux.
Je trouves quelques bug, c'est chiant, ça prend du temps, mais ils se corrigent.
J'ai rajouter une fonction de prédiction.
Meeting avec Akash : Je dois rajouter les relations inverse.

En entrée j'ai pas d'entitéx, mais juste une relation car le but c'est de trouver une séquence de relation qui donne la même chose. Exemple :
relation : Parle_langue.
prédiction : Habite, Langue_officielle.
Ça permet de sortir la règle :
Habite(X,Y) $\wedge$ Langue_officielle(Y,Z) $\rightarrow$ Parle_langue(X,Z).x

Je suppose donc que nos random walk prennent deux noeuds en relations et fond des walks un peut autour avant de revenir. Va falloir que je regarde comment faire ça...

Ok j'ai essayer de me rebaser sur nn.Transformer mais c'est vraiment bizarre.
J'ai viré les noeuds en entré pour prendre juste des relation.
Je modifie en conséquence tout mes trucs.
J'accroche pas mal au niveau des masques.
À priori un masque est de taille batch $\times$ tailleTxt $\times$ tailleTxt.
Pour le cross, je sais pas si c'est le plus petit ou le plus gros texte.
Pour les autres déjà :
Après embedding, les données sont de taille : batch $\times$ tailleTxt $\times$ d_model
Notre masque est lui de taille : batch $\times$ tailleTxt $\times$ tailleTxt.
Ok j'ai la ref le masque n'est utilisé que dans le calcul avec les tenseur key, query et value.

Ok à priori j'ai de bon masques. Je me suis pas mal aidé visuellement de cette question dans un forum :
https ://datascience.stackexchange.com/questions/126187/cross-attention-mask-in-transformers

Maintenant il faut que je modifie encore deux trois trucs dans le modèle de transformer et après ça sera OK je pense.
Ah et la marche aléatoire aussi.

Vendredi 21-06 :

J'ai commencé par terminer le modèle du transformer pour prendre en entrée une relation au lieu de deux noeuds (potentiellement éloignées).
J'ai aussi rajouter les relations inverse dans la liste des relations.
Ça marche plutôt bien, je suis pas mal content, je vois assez bien comment ça marche maintenant.
Maintenant il faudrait peu être que je m'attaque à l'histoire des random walk. Je ne suis pas exactement comme faire encore.
L'ojectif est de rentrer une relation (prise aléatoirement), de choisir (toujours aléatoirement) deux entités liés par cette relation, puis trouver un chemin reliant ces deux entités.

Je vais déjà tester l'implémentation hyper basique. Full aléatoire mais chemins limité à genre 4 de longueur (en vérifiant bien que je ne mette pas la relation de base (et potentiellement éviter aussi, si la relation est r, d'avoir un chemin x - x-1 - r)).
Ok ça ne marchait pas vraiment. Je garde les probababilités en fonction de la distance à l'arrivé, avec un softmax, mais je fais attention à empêcher la relation d'être prise en premier.
De plus j'ai stocker mes grosses données, le graphe de train, les index, dans des fichier.pickel.
Ça semble fonctioner mais c'est vraiment hyper lent, surtout en mode débuggage.
Ok j'ai trouver comment faire avec networkx. Je peut créer un générateur avec nx.all_simpls_paths puis prendre un chemin aléatoire (jepeux même limiter la taille).
Réunion avec Steven Schockaert.
C'est pas trop mal, ça avance. Là je fini avec mon transformer.
Si je teste sur (h,r,t) il faut que j'empêche de prendre cette relation dans le chemin.
De plus, il faut que j'empêche d'avoir un truc du style (n, r, n') suivit de (n', r-1, n) (mais si c'est une autre relation c'est ok et si les entités sont différentes c'est ok aussi).
Il faudra ensuite que je crée un autre transformer pour classifier mes chemins et voir les quels modélisent réellement une relation et lesquels sont beaucoup mais sûr.
J'ai aussi changé de data, ça devrait être bien plus rapide.
Mon job pour la prochaine semaine est de finir ça (le random walk qui marche bien, avec les contraintes souhaités) puis de voir quels doivent être les bons hyperparamètres pour le transformer (nombre de têtes, de layers, ...).
Ça risque d'être un poil frustrant mais ça avante.
Je devrais ensuite tester mon transformer (sur une partie des chemins aléatoire), puis créé le second comme dit précédemment.
Si j'ai du temps / de la motivation, je pourrais aussi regarder les méthodes LSTM.
Et puis il sera l'heure de s'attaquer aux règles de productions. Potentiellement avec AnyBURL, potentiellement avec autre chose.

On j'ai trouver une méthode pour le random walk. Je n'utilise pas d'algo compliqué de netorkx, mais juste les trucs basique. C'est surement pas ouf mais ça semble marche plus ou moins et ça va assez vite.
J'ai tout bon maintenant.
Je lance.
Et pouf, mes predictions sont toutes à zero, je ne sais pas trop pourquoi.
Bon il est quasi 19h.
J'arête, j'aurais peu être la motivation de continuer ce weekend.

Lundi 24-06 :

J'ai re regarder mon code, eu pas mal de problèmes avec les masques.
Ça a commencé à ma saouler et j'ai décidé de repartir sur des bases saines, à savoir le traducteur qui marchais bien.
J'ai modifié deux trois trucs que pour ça marche bien, et ça semble un peu mieux qu'avant.
Ok j'ai testé avec 3000 chemins aléatoires. Ça marche, pas d'erreurs et on voit que l'on arrive petit à petit à quelque chose de cohérent. Ça apprend.
Maintenant, comme l'a dit Schockaert, tout le problème est de trouver les bons hyperparamètres.
Je suis en train de faire de petits tests. Une question me vient à l'esprit : pour la partie induction, comment on limite à 4 la séquence ? On le force, on limite à 6 mais sans forcément de <END> auquel cas on ne le prend pas en compte ou en ne limite pas et on s'arrête que quand on aura un <END> ?
Autre question : comment j'évalue si ma prédiction est bonne ?
Ok j'ai des données de test spécifique, non vu à l'entrainement, mais une fois que j'ai ma prédiction comment je sais si elle est correcte ou pas ?

Mardi 25-06 :

J'ai rajouté les relations inverse dans le graphes (comme ça accessible dans le random walk).
J'ai trouvé un autre problème : comment faire pour obtenir plusieurs prédictions pour une même relation (avec degrès de confiance, ça serait pas mal) ?
Potentiellement utiliser le beam search ou le top-k sampling.
J'ai rajouté le multi-translate (je crois avec le beam search, plus ou moins).
J'ai aussi mis en pickel le transformer carrément, ça ira beaucoup plus vite.
J'ai créé le Makefile pour clean les pickels.
Je crois qu'il y a un soucis dans mon implémentation. En effet, qu'importe la relation que j'évalue en imput, j'obtiens la même prédiction, en quand je regarde les 10 première prédictions, elles sont toute équivalente, ce qui n'est pas censé être le cas.
Je pense que ça fait de l'entraînement, je ne saurais pas exactement dire quoi mais c'est ce que je pense.
Vendredi dernier on avait parler de l'entré comme une séquence avec SOURCE | séparateur | DÉBUT DE PRÉDICTION.
Je crois que je n'ai pas bien implémenté, et je regarde dans spécifiquement les transformer pour de la prédiction mot à mot. Ça devrait être mieux je l'espère, car j'ai assez perdu de temps là.
OK journée pas hyper productive, la fatigue n'a pas aidé.
Il faudra faire mieux demain.

Mercredi 26-06 :

Ok je change de plan.
Je pense que ce que j'avais fait pour la traduction n'était pas adapté.
Je suis reparti voir un exemple de transformer from scratch mais cette fois-ci pour de la génération de texte.
J'ai suivi ce tuto : https ://debuggercafe.com/text-generation-with-transformers/.
Après quelques ajustement (notamment le masque pour les PAD) ça semble fonctionner très bien !
J'ai mis mes données sous la forme RELATION | SEP | START | PATH | END | PAD.
Et j'ai modifié le vocabulaire pour avoir une différence entre les relations de RELATION et celles de PATH.
Ça semble bien marcher. Là je vais faire la multiprédiction puis je ferais tourner sur un gros exemple.
Pour la multiprédiction, j'ai besoin d'une fonction que calcule le degré de confiance en une prédiction.
On repart sur le produit des probabilité d'avoir les tokens, ça reste ok.
Là ça tourne sur un "gros" exemple (juste 5000 chemins). Après cela, j'ai plusieurs choses à faire :
- Trouver les bons hyperparamètres pour le transformer.
- Créer un autre transformer pour classifier les chemins (et voir s'ils modélisent comme il faut la relation initiale).
- Tester sur d'autres données pour voir si ça marche bien.

J'ai regardé pour les hyperparamètres. C'est possible de le faire avec un grid search (long), un random search (un peu plus rapide) ou un bayesian search (le plus rapide).
Les modules hyperopt ou optuna semble pouvoir faire cela. Je ne m'y suis pas encore trop penché dessus mais ça ne semble pas non plus trivial.
Objecti : m'en occuper demain.

Jeudi 27-06 :

J'ai commencé par regarder pour les hyperparamètres.
J'ai utilisé hyperopt.
Ça semble marcher (au moins ça tourne).
J'essaye d'optimiser les paramètres suivant :
epochs, batch_size, optimizer, num_layers, num_heads, embed_dim, learning_rate
Ça tourne pas trop mal, mais c'est vachement long.
Il va falloir que je le fasse tourner sur ARCAA, mais j'ai déjà les 100000 chemins qui tourne.
D'ailleurs je ne sais pas si c'est utile que je le fasse tourner sur beaucoup de chemin si si un truc sur 1000 comme ce que je fais là est nécessaire.

Aussi, il faut voir si j'optimise les paramètres un par un ou tous en même temps.
Quelques résultats : Best hyperparameters found : 'batch_size' : 32.0, 'embed_dim' : 30.0, 'epochs' : 70.0, 'learning_rate' : 0.002796658974208622, 'num_heads' : 3.0, 'num_layers' : 4.0
Pour une recherche dans les intervales :
'epochs' : hp.quniform('epochs', 10, 100, 10),
'batch_size' : hp.quniform('batch_size', 8, 128, 8),
'num_layers' : hp.quniform('num_layers', 1, 8, 1),
'num_heads' : hp.quniform('num_heads', 1, 8, 1),
'embed_dim' : hp.quniform('embed_dim', 10, 100, 10),
'learning_rate' : hp.uniform('learning_rate', 0.0001, 0.01)
Un training sur 1000 chemins, un nombre de trials de 20 et avec l'algo tpe.

Je vais lancer la même chose mais cette fois ci avec l'algo rand :
Best hyperparameters found : 'batch_size' : 40.0, 'embed_dim' : 60.0, 'epochs' : 90.0, 'learning_rate' : 0.000951295752507925, 'num_heads' : 4.0, 'num_layers' : 6.0
Sur 5000 chemins : Best hyperparameters found : 'batch_size' : 32.0, 'embed_dim' : 40.0, 'epochs' : 50.0, 'learning_rate' : 0.0006883701650054103, 'num_heads' : 6.0, 'num_layers' : 8.0

Bilan sur les questions que je me pose :
- Lors de la marche aléatoire, je dois sélectionner un triplet aléatoirement et essayer de trouver un chemin entre les deux entités, ou je dois m'assurer que chaques relation est représenté proportionnellement à sa présence dans le graphe, ou je dois mettre un nombre minimum pour chaque relation ?.
- Lors de la vérification sur un autre jeux de donnée (graphe différent mais mêmes relations), ce que je dois vérifier c'est que losque la prédiction me donne un chemin, je dois le trouver dans le nouveau graphe (à pourcentage de confiance pré) ?
- Sur combien de chemin aléatoire je dois tabler ?
- Quelles relations j'extrais ? Pour chaque relation j'extrais les 10 première plus probable, je supprime celles trop improbable (moins de 1% de chance par exemple) puis je passe dans le second transformer et si je trouve la bonne relation initiale, je le garde ? - Comment utiliser la perplexité pour la validation ?

Fini pour aujourd'hui.
J'ai un classifier qui marche théoriquement, je fais les tests demains matin avant la réunion.

Vendredi 28-06 :

Ok à priori j'ai tout qui marche : import des données, transformer pour générer les chemins, classifier pour vérifier s'il sont bons.
J'ai même l'optimisation des hyperparamètres qui marche convenablement (c'est terriblement long mais ça fonctionne).
Je lance un test global sur 5000 chemins (générations des chemins puis classification) pour voir mon taux de réussite (pour combien de relations parmis les 237 j'arrive à prédire un chemin qui se rattache à me relation, dans le top 5).
C'est pas exactement une bonne mesure de la qualité du modèle, mais ça m'aidera à y voir plus clair déjà.
Ensuite il va falloir que je me penche sur ce que ça veux dire exactement d'utiliser les données de la validation.
Pour mon test global, je trouve la bonne relation pour 87% d'entre elles, et quand je trouve c'est en moyenne à la place 1.7.
54% la première place, 15% la seconde, 9% la troisième, 6% la quatrième, 2% la cinquième et 13% en dehors du top 5.
Réunion avec Steven Schock3aert.
J'ai fais un peu de la merde : l'optimisation des hyperparamètres n'est pas correcte, je dois la faire sur le jeux de validation.
Pour ça, je dois créer les chemins via les données de validations, et pour chaque chemin, calculer la probabilité qui arrive pour calculer la perplexité : $-\sum_i \log p_i$ si p_i est la probabilité d'avoir la relation r_i, du chemin $r_1 \dots r_n$, en rentrant $r < SEP > r_1 \dots r_{i-1}$.
L'objectif est de minimiser cette perplexité.
De plus, pour les chemins il en faut un nombre similaire pour chaque relation.
Ok ça ma saouler d'attentre à chaque fois de construire les chemins, et puis je crois qu'il en faut vraiment beaucoup plus que ce que je faisais, j'ai donc décider de stocker tous les chemins existant dans un fichier en .pickle.
Problème de mémoire, je dépasse largement les 30Gio. Je stocke donc dans un fichier différent pour chaque relation.
C'est gros mais j'espère que ça va passer.
J'ai des relation avec plus de d'un million de chemins. Et je ne suis qu'avec les données de validations, donc pas les plus grosse encore.

Mardi 02-07 :

J'ai lancer l'algo pour la création de la liste de tous les chemins, cette fois-ci sur les données d'entraînement.
J'ai aussi créer un fichier transformer_validation, pour calculer la perplexité.
D'ailleurs je suis pas hyper au fait du calcul de la perplexité (moyenne de la somme des log, exponentielle, ...).
Ok j'ai testé ça me renvoie bien une valeur.

Mercredi 03-07 :

Les relations 294, 296 et 290 posent problèmes.
Je les sépare en plusieurs fichier.
C'est vraiment une méthode de merde. Je devrais tester avec une BDD mais je ne les métrise pas.
Je pense que juste sur le principe c'est nul comme méthode, mais j'ai pas trouver de random walk qui soit pertinente pour trouver un chemin aléatoire entre deux relations.
Mais bon juste si je passe de 4 à 5 relations autorisé c'est finito cette méthode, impossible à mettre en place sur des gros graphes.
Mais bon j'ai déjà fait une grosse partie du travail pour implémenter ça, je ne vais pas renoncer maintenant, mais faudrait vraiment que je trouve une autre méthode.
Mais déjà il faudrait que ça marche avant la réunion avec Steven.
En découpant les fichiers, ça marche effectivement. Ça ne prend donc plus trop de temps pour avoir des chemins aléatoires.
Pour le moment j'en prend 500 par relations, mais je pense que ça va prendre du temps à entraîner le transformer avec ça.
On verra, c'est le prochain truc que je fais dès que tous les chemins sont générés.
Ensuite il faudra vérifier que la validation (avec le calcul de la perplexité) fonctionne, puis on pourra lancer l'optimisation des hyperparamètres.
Pour information, mon pickle le plus lourd fait 3.3G (en vrai j'ai plus de 35.6 Go pour la relation 296, mais séparé en 15 fichiers), et je l'ouvre en utilisant environ 72% de la mémoire de mon ordinateur.
Ok ça à fini de compiler. J'ai juste eu à rajouter les token START, END et PAD dans les chemins et le train du transformer est en cours.
C'est pas rapide, mais c'est pas trop long non plus. Une epoch (2 têtes, 2 layers, 500 chemins par relations) prend environ 45s.
Pour les vrais test, si ce temps de bouge pas trop après avoir optimisé les hyperparamètres, je pourrais surement montrer à 5000 chemins par relation, ce qui me fera 2 370 000 données.
J'ai lancée la validation, je suis toujours avec 500 chemins par relation.
Ils me disent qu'il y a 200 000 test. Je pense que j'ai du oublier de faire les avec les relations inverses en tant que source, car ça me paraît peu.
Mais bon encore une fois c'est pas court mais c'est pas trop long, je dois être à 0.5s pour tester 100 chemins.
Ça me fait de l'ordre de 1000s, soit un peu plus d'un quart d'heure.
C'est OK quand on le fait une fois mais pour l'optimisation des hyperparamètres ça va être long, surtout que ce calcul devrait rester constant (bon avec 2 fois plus de chemins donc 30min) mais pour le trainning à chaque fois tout dépendra des hyperparametres.
Peu être que je dois diminuer à seulement 10 chemins par relation, pour que ça soit plus rapide.
Surtout qu'OK c'est bien beau ARCAA mais j'ai toujours pas la ref de si c'est vraiment plus rapide. Le seul truc sûr c'est que ça tourne tout seul, et que ça

ne fais ni souffler ni chauffer mon pc.
Pour le premier test (juste 200000 chemins) j'ai une perplexité de 29.8.
20 epoch, 128 batch_size, 2 layers, 2 heads, 100 embed_dim, 0.001 learning_rate.
Ah non ouai j'ai dit n'imp, 200 000 chemins c'est bon, il y a bien 400 relations (avec les inverses) et 500 chemins par relations.
Ok c'est un poil long en fait.
Je vais déjà baisser à 50 chemins par relations pour la validation.
Ok je vais diminuer à 50 pour le train aussi.
Je ferais les vrais tests plus tard.
Autre technique, je vais arrêter de supprimer les list_paths, juste les renommer, pour les reprendre sans devoir tout re-calculer quand j'en ai besoin.
Bon tout ça ça va être vraiment long. Surtout que je suis cencé avoir beaucoup de trials. Genre entre 100 et 500...
Je vais lancer sur ARCAA entre aujourd'hui et demain. Ça va faire environ 60h de calculs. Je sais pas si c'est suffisant pour disons 200 trials.
On verra demain.

Jeudi 04-07 :

J'ai pas réussi à lancer sur ARCAA. J'avais d'abord une erreur pickle. J'ai virré le git clone et juste copié les fichiers, ça ne changeait rien donc j'ai virré le plus possible les pickle (tout ceux non nécessaire) et ça passait.
Mais j'ai ensuite eu une erreur de taille de tensor. Et je suis rentré chez moi.
Finalement j'ai pu corriger l'erreur ce matin (je dois juste avoir num_head ou embed_dim_multiplier pair sinon ça crée un problème pour l'embedding).
Donc ça tourne sur ARCAA, et plutôt pas mal (je peux suivre la progression vie le fichier output).
Ok d'ailleurs j'ai découvert vsc, nano (qui est assez intuitif) mais je pense que vim peut aussi être avoir une plus value.
J'ai donc commencer à regarder vimtutor.
Pour l'instant sur les résultats de l'optimisation des hyperparamètres, je vois que ça marche vien (la perplexité baisse petit à petit). Ça prend sont temps mais bon du temps j'en ai avec ARCAA, je le ferais tourner en boucle pendant une semaine et on verra des résultats.
Seul point de questionnement, je ne devrais pas re-changer mes données de validation à chaque fois, pour éviter que le modèle s'adapte à ces données ?
Ou alors en prenant de grands jeux de données ça suffit (aka les 200 000 chemins pour la validation).
Aussi autre intérogation : le jeu de validation sert uniquement pour l'optimisation des hyperparamètres, mais à quoi sert le jeu de test ?
Réunion avec Akash :
Prendre le même nombre de relation par triplet plutôt que par relation.
Faire la validation lors du training : une epoch, une évaluation, et je garde le meilleur modèle en format pth.

Je jeu de test est que pour le rewiever, pas pour moi, et mon jeu de validation c'est sur le même graphes.
Les hyperparamètres je les faits à la mains plutôt qu'avec la machine, elle est pas hyperintelligente.
J'ai donc modifié l'importation des données pour avoir le bon jeux de validation et fixer le nombre de chemins que je veux par triplets.
J'ai aussi insérer l'évaluation entre chaque epoch, comme mentionné.

Vendredi 05-07 :

Ok ça à tourné toute la nuit pour ré-importer les données de training (k chemins par triplets), mais maintenant c'est fait !
J'ai lancé du training avec évaluation pour voir si ça fonctionne bien et ça semble assez OK, juste deux trois problèmes à régler à la fin.
Ça marche bien. Je regarde progressivement dans quel sens améliorer les paramètres.
Ça prendra un peu de temps mais OK.
J'ai aussi modifié en prévision le classifier pour avoir l'évaluation au milieu du training et prendre en compte les bonnes données pour la validation.
Les prochaines étapes sont, je pense : finir d'optimiser les paramètres. Faire la même chose pour le classifier. Passer aux méthodes basées sur les règles.
Réunion avec Steven Schockaert.
Ça avance. Il va falloir faire tout ça sur ARCAA, mais c'est pas trop mal.
J'ai trouver deux trois bug dans l'importation des données, c'est reparti pour les 150h...

Lundi 08-07 :

J'ai lancer sur 20 chemins par triplets. C'est long mais ça marche mieux.
Pour le nombre de têtes et de layers, ça bouge relativement à la marge.
Je vais attendre que ça se finisse, lancer un 100 chemins par triplets, sur ARCAA, puis regarder les résultats avec le classifier.
Vraissemblablement, à partir de 20 c'est ok, pas la peine de faire 100, on gagne quasiment rien.
J'avais un soucis dans le calcul de la perplexité (je mettais pas le start et le end).
J'ai regarder avec le classifier et ça donne un truc pas trop dégeu : 67% en top 1 et seulement 15% pas trouvé.
Je vais regarder la perplexité sur le jeu de test.
Ça me donne une perplexité ok je pense (8.7 contre 5.3 au mieux).

Mardi 09-07 :

Je teste les derniers hyperparamètres.
Leraning rate : 0.01 ne convient pas, 0.0001 est légèrement moins bien que 0.001, je teste entre les deux.
0.005 semble prometteur sur la première epoch.
Entre les deux, ça teste 0.0075. J'obtient (pour le moment) du 5.24, ce qui est mieux que tout le reste !
Je regarde aussi avec 30 chemins et avec 10 chemins.
30 chemins ça n'améliore pas à priori (avec les mêmes paramètres en tout cas).
Je regarde pour 10.
Ça marche bien, avec un LR de 0.0075 j'obtiens encore une fois un 5.24 !
Ce soir je lancerais une recherche random d'hyperparamètre, en prenant en compte le nombre de chemins, je pense.
Je modifie un peu les layers. Avec 4 j'obtiens (pour le moment) une perplexité de 5.12.
J'en ai aussi profité pour regarder sur les données de test.
2 layers, 2 head, 0.0075 LR, 10 chemins, donne une perplexité de 10.26. Pas sûr que ça soit ouffissime.
4 layers, 2 head, 0.0075 LR, 10 chemins, donne une perplexité de 9.02.
Maintenant, faudrait voir ce que je fait. À priori il faut ajoute tout plein de relation, puisque c'était le but pour pouvoir faire une méthode basé sur les règles plus efficacement.
Faut-il sortir les X chemins les plus probable par relation, pondérer (voir comment) avec le classifier, puis regarder chaque triplet et faire tourner les proba.
Faut aussi faire gaffe si le chemin est déjà présent.
À priori ça rajouterait pas mal de relation, mais je ne sais pas si ça sera assez.
Peut être ajouter plusieurs relation par triplet, je ne sais pas.
Et après ça sera ouvert pour les règles.

Mercredi 10-07 :

Choix et ajout des nouvelles relations en utilisant les chemins. Long. Il y en a vraiment beaucoup ! Discussion avec Akash.
En fait il faut comparer avec LSTM et N-gram pour avoir une point intéressant à dire. Développer pourquoi on a utiliser les transformers.

Jeudi 11-07 :

Implémentation du LSTM, hyperparamètres, ça marche assez bien (juste un petit bug à corriger).
Implémentation du N-gram, toujours des erreurs, notamment avec les masques car la taille des input et celle des outputs n'est pas la même.

Vendredi 12-07 :

Réunion avec Steven Schockaert : Pas trop mal. Il peut être intéressant de regarder le fine tuning, et potentiellement augmenter à des chemins de taille 6 pour vior la différence.

J'ai terminer d'implémenter le N-gram, ça marche pas si mal, moins bien que les deux autres boug mais pas si mal !
L'hyperparamètrage est lancé.
Le bug résolu (le "loss" est important en fait).
Les trois vont tourner en continu sur mon PC ce week-end, on va bien se marrer.

Lundi 15-07 :

Mensonge, ils n'ont pas tourné. Ça va commencer aujourd'hui.
Ok ça à tourné mais j'ai vraiment pas fait grand chose de plus... Un poil de fine-tuning et c'est tout.
Faudra faire mieux les prochains jours.

Mardi 16-07 :

J'ai regardé pour le fine-tuning.
Ça pose quelques questions de comment je le met en place réélemment. Je verrais avec Akash demain normalement.
Optimisation du LSTM. Ça tourne bien.

Mercredi 17-07 :

Encore de l'optimisation. Ce soir j'aurais fini le LSTM normalement.
J'ai implémenter le fine-tuning aussi, mais il faut que je regarde plus attentivement car lors du fine-tuning, ma perplexité semble "bloqué". Il faudra que je passe un coup au débugger.
Ce qui est marrant c'est que globalement les meilleurs hyperparamètres sont proches les uns des autres pour les trois modèles différents.

Jeudi 18-07 :

L'optimisation pour LSTM est bien terminé. Je vais faire pour N-gram. Ça ne devrait pas prendre trop de temps. Ensuite je verrais bien...
J'ai toujours pas de vraies infos sur le fine-tuning.
Je vais peut être regarder en autorisant 6 relations par chemins. Mais pas hyper évident à capturer les chemins dans ces conditions.

Vendredi 19-07 :

J'ai fini l'optimisation pour le N-gram.
Résultats cohérents. J'ai regarder aussi avec les données de tests entre N-gram, LSTM et Transformer et tout est logique.
Je suis en train de faire les 6 relations. C'est bad long.
Ok rendez-vous avec Steven Schockaert.
Pour le fine tuning je vais obtenir un modèle par relation et lors du calcul de la perplexité je ferais appel au bon model correspondant.
Il va falloir regarder pour les 6 relations.
Si je suis motivé je peux aussi faire le input/no_input.
Regarder un fine tuning from scratch aussi mais ça peut faire du overfitting.
Après il va falloir regarder les règles et comparer avec AnyBURL.
J'ai déjà mon classifier.
Il faudra que j'optimise les hyperparamètres.
Ensuite Akash va me donner un format pour la sortie et je pourrais comparer avec les règles d'AnyBURL.
Le format c'est : X X prob rel0(X,Y) <= rel1(X,A), R2(A,B), R3(B,Y).

Lundi 22-07 :

Formatage effectué.
Je vais essayer de faire le fine-tuning là, avant d'envoyer mes résultats.
L'entraînement est fait, avec la création des 438 modèles.
Je regarde la perplexité avec les données de validation et les données de tests.
Ok j'ai des résultats moins bon que ceux attendu. Je suppose que j'ai un problème quelque part (nonobstant pas sur à 1000 %, faut peut être que je fasse un tour du model basique pour ne pas le remplacer quand j'ai un truc moins bon).
Ah ouais, c'est surement ça, avec tous les triplets avec 0 chemins.
Je regarde avec ça. Autre problème que j'ai, je comprend pas trop pourquoi le calcul de perplexité n'est pas déterministe.
C'est beaucoup mieux mais encore juste derrière.

Mardi 23-07 :

Je regarde relation par relation le pourquoi de la perplexité est plus haute. Pour cela calcule en même temps les perplexité pour le modèle classique et pour le fine-tunned.
J'attendrais à ce que FT soit plus petit pour chaque relation, mais ça ne va pas être le cas et je vais investiger pourquoi.
Ok je crois que j'ai le problème. Quand mon fine-tuning est moins performant que le modèle basique, ça m'enregistre tout de même le fine tuning, je ne sais exactement pourquoi. Il faut que je modifie ça. Je vais essayer d'enregistrer de

base le modèle et de remplacer seulement si c'est mieux.
J'ai trouvé. Comme j'introduisais le modèle de base une seule fois, ça devait le mettre à jour avec les copie (pourquoi ?) et donc ça faisais tout dérégler mon calcul de perplexité, qui devenais ahurissant. Maintenant je le réouvre à chaque cas, donc plus de soucis de ce sôté là. Je fais tourner puis ça devrait être pas si mal.

Mercredi 24-07 :

Ça hyperparamètre le transformer pour finalise tout ok.
Ça fait pas grand chose d'autre.
Les JO ont commencé :)

Jeudi 25-07 :

J'ai relancer un fine-tuning, c'est long mais ça sera assez bon à la fin.
Je regarde en parallèle la prédiction des chemins. Le soucis c'est que le score d'un schemin, actuellement, c'est le produit des probabilités des tokens.
Ça me donne plein de petites probabilité, et la somme est de 1, alors que moi je veux seulement la probabilité d'avoir un chemin correct, même si il ne sort pas souvent.
Je regarde une log probabilité, mais je ne pense pas que ça va changé énormément de choses.