Geometric Methods in Machine Learning

Poincaré Embeddings

Zakarya Ali
Samir Tanfous

19 mai 2018

Introduction

Les algorithmes d'apprentissage peuvent se nourrir de l'information issues de taches non supervisées pour réaliser des taches supervisées. Dans le contexte du machine learning, une bonne représentation des observations permet de simplifier la tache à réaliser. Le choix de la représentation dépend donc de la tâche à effectuer et dans notre cas, nous voulons étudier les relations hiérarchiques d'un ensemble de mots. Nous avons choisi d'étudier l'article Poincaré Embeddings for Learning Hierarchical Representations de Maximilian Nickel et Douwe Kiela. Le code que nous avons développé pour établir l'ensemble de notre étude est disponible ici : `https://github.com/zakaryaxali/poincare_embeddings`.

Nous allons voir comment la représentation de liens entre des mots par la distance de Poincaré permet de sensiblement améliorer l'interprétation des liens hiérachiques qui les régissent. Dans un premier temps, nous allons présenter l'algorithme utilisé. Puis nous allons présenter son entrainement. Enfin, nous allons analyser les résultats.

1 Algorithme

Comme l'énonce l'article, l'apprentissage de représentation est devenu un enjeu majeur dans le Natural Language Processing. Si bon nombre des algorithmes actuels mettent en avant une représentation utilisant des distances classiques (euclidennes par exemple), ces dernières ne sont pas nécessairement les plus efficaces pour chaque cas. Pour des données avec une structure hiérarchique latente, la méthode de Poincaré Embeddings, basé sur l'optimisation de Riemann semble nettement meilleure.

On cherche à représenter dans un espace la similarité des données grâce à la distance de Poincaré. On cherche à montrer l'homophilie entre les mots selon leur place hiérarchique et leur liens. En géométrie euclidienne, il est très compliqué de représenter ce genres de liens, alors qu'un espace hyperbolique est plus adapté. En effet, cet espace est un analogue continu aux arbres.

On veut représenter les vecteurs dans la boule de Poincaré. On a la distance de Poincaré pour deux vecteurs u et v :

$$d(u,v) = arccosh\left(1 + 2\frac{||u-v||^2}{(1-||u||^2)(1-||v||^2)}\right)$$

L'avantage de prendre une distance de ce type sur la boule de Poincaré est qu'elle varie lentement lorsque les vecteurs sont près de l'origine, mais croît très fortement pour ceux situés sur le bord. Cette représentation multidimensionnelle dans un espace hyperbolique permet de représenter de multiples hiérarchies dans un corpus, comme dans le cas de la taxinomie que nous étudierons après. L'ojectif désormais est de trouver des incorporations dans des données symboliques. Le cas général sera de considérer un problème pour lequel il faudra minimiser une fonction de perte $\mathrm{L}(\Theta)$ qui encourage les objets sémantiquement proches à être proches dans l'espace de représentation :

$$\Theta' \leftarrow \underset{\Theta}{arg\,min}\ L(\Theta)\ s.t.\ \forall \theta_i \in\ \Theta\ :\ \|\theta_i\| < 1.$$

Afin de minimiser cette fonction de perte, nous utilisons une méthode d'optimisation stochastique de Riemann. A chaque itération, on actualise le paramètre θ de cette manière :

$$\theta_{i+1} \ = \ \mathcal{R}_{\theta_t} \ (-\eta_t \nabla_R L(\theta_t)),$$

où $\theta \ \in \ \mathcal{B}^d$, et avec ∇_R le gradient de Riemann de $L(\Theta)$, $\mathcal{R}_{\theta_t}$ une fonction de rétraction sur $\mathcal{B}$ et η_t le pas de gradient au temps t. Comme $\nabla_{\mathcal{R}}$ est le gradient de Riemann, nous ne sommes pas en mesure de le dériver facilement. Ce que nous savons facilement dériver par contre, c'est le gradient euclidien ∇_E pour lequel il suffit d'appliquer une dérivation en chaîne, de la fonction de perte par rapport à la distance de Poincaré d'une part, et de la distance de Poincaré par rapport à θ d'autre part (qui est connue). Il existe une relation entre le gradient de Riemann et le gradient euclidien. Il suffit de faire un rescale par rapport à l'inverse du tenseur de Poincaré :

$$\nabla_E \ = \ g_\theta^{-1} \nabla_{\mathcal{R}},$$

ce qui est aisé à calculer.

Pour l'opérateur de rétraction, on utilise $\mathcal{R}_\theta(v) = \theta + v$. Il faut ensuite que les incorporations calculées à chaque itération restent dans la boule de Poincaré, on définit alors un opérateur de projection qui les y projettent :

$$proj(\theta) = \begin{cases} \theta / \|\theta\| - \epsilon \text{ si} \|\theta\| \geq \ 1 \\ \theta \text{ sinon,} \end{cases} \tag{1}$$

Enfin, l'équation finale de mise à jour de θ est :

$$\theta_{t+1} \ \leftarrow proj(\theta_t - \eta_t \frac{(1 - \|\theta_t\|^2)^2}{4} \nabla_E)$$

Le papier de recherche évoque 3 principaux cas d'usage, à savoir les relations taxinomiques, la modélisation de graphes complexes et les implications lexicales. Pour notre implémentation, nous avons travaillé sur les relations taxinomiques, en nous appuyant sur un dataset issu de Wordnet référençant tous les hypernymes du mot "mammal" (mammifère en anglais). Dans le cas de la taxinomie, nous cherchons à minimiser la fonction de perte suivante :

$$\mathcal{L}(\Theta) = \sum_{(u,v) \in \mathcal{D}} log \left(\frac{e^{-d(u,v)}}{\sum_{v \in \mathcal{N}(u) e^{-d(u,v')}}} \right)$$

où les couples (u,v) désignent des pairs de mots pouvant présenter ou non une relation d'hyper ou d'hyponymie.

2 Entrainement

Les données que nous avons récupérées représentent tous les hypernymes du mot "mammal" dans Wordnet. Il y a deux problèmes ici que nous souhaitons résoudre, une reconstruction et une prédiction. Nous allons d'une part évaluer la capacité de reconstruction du

modèle de Poincaré, c'est-à-dire à partir de la construction des vecteurs d'embeddings, retrouver les relations de taxinomie entre les différents mots. D'autre part, nous allons diviser aléatoirement nos données entre entraînement et test, pour entraîner le modèle de Poincaré pour déterminer les vecteurs d'embeddings, et ensuite le tester sur les données de test pour retrouver les relations de taxinomie.

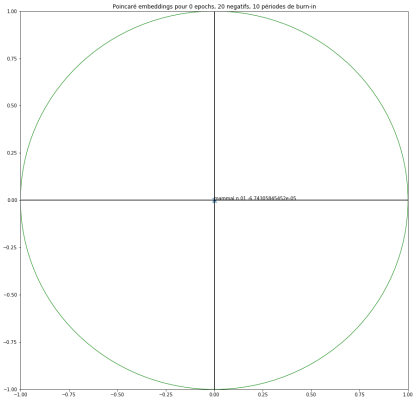
Pour réaliser l'entrainement on suit toujours l'article et on initialise chaque élément des vecteurs selon une loi uniforme $\mathcal{U}[-0.001, 0.001]$ pour optimiser l'entrainement. On ajoute également une période de burn-in, un certains nombre d'epochs de départ pendant lesquels le pas d'apprentissage est réduit (ici divisé par 10). On choisit un nombre fixes de mots négatifs (mots qui n'ont pas de lien avec le mot étudié) de 20. De plus, lors du choix des mots négatifs, ces derniers sont choisis aléatoirement, mais avec un poids proportionnel à leurs nombre de connexions.

Les implémentations sont disponibles dans notre Notebook.

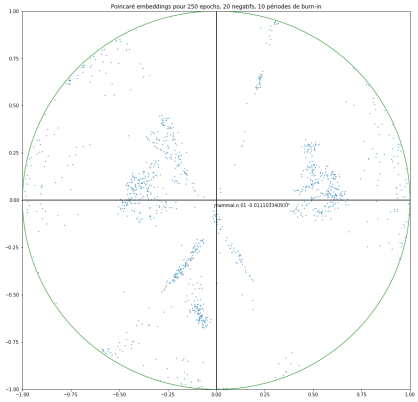
3 Résultats

3.1 Taxinomie - Représentation 2D

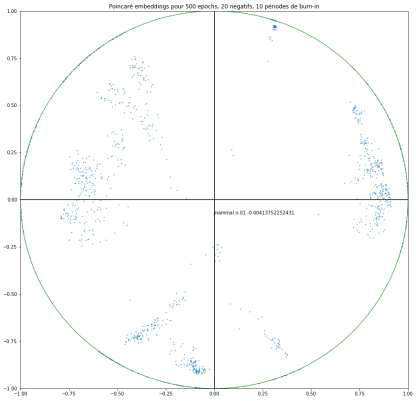
Nous avons essayé de représenter les données dans un espace en 2 dimensions. La figure 1 montre nos résultats obtenus sur différents epochs. On constate que plus le nombre d'epochs est grand plus certains points se rapprochent du cercle unité. Le mot mammal quant à lui reste toujours proche du centre. Pour l'epoch 1000 (figure 3) on converge bien et on peut voir que mammal est le seul mot au centre, tandis que les mots liés sont répartis autour du cercle avec les mots non liés asez écartés. Nos résultats sont similaires à l'article à ceci près que les liens hiérarchiques sont plus décelés à repérer (exemple : carnivore n'est pas beaucoup plus proche de mammal que cannine).



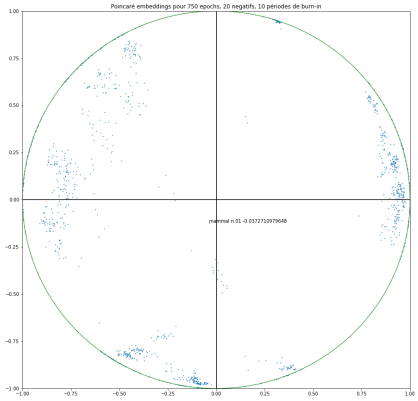
(a) epoch 0



(b) epoch 250

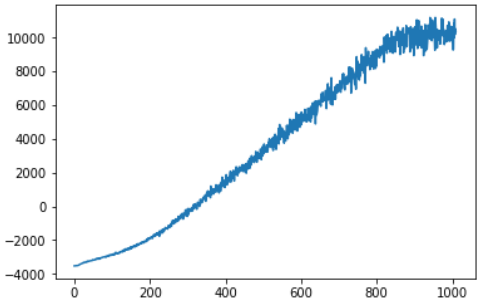


(c) epoch 500

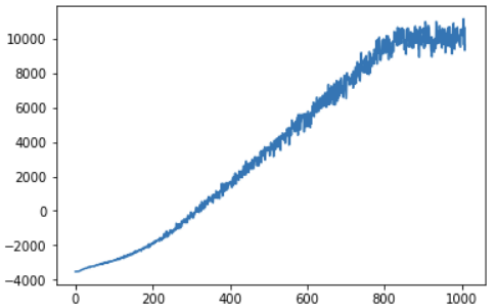


(d) epoch 750

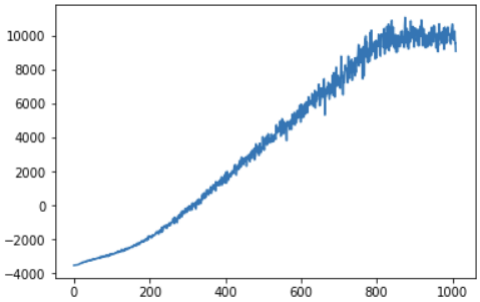
FIGURE 1: Evolution de l'embeddings pour des vecteurs de dimension 2



(a) dimensions = 10



(b) dimensions = 50



(c) dimensions

FIGURE 2: Fonction de perte pour différentes dimensions

Nous avions dans un premier temps déterminé les vecteurs d'embeddings pour 250 epochs, mais en constatant en dimension 2 que les points étaient très proches de l'origine, nous avions décidé d'afficher la perte. Elle ne convergeait pas au bout de 250 époch, donc nous sommes passés à 1000 époch. Les résultats obtenus sont ci-dessus, on peut voir la perte se stabiliser à partir d'environ 900 époch, bien que l'on constate l'apparition d'un bruit. Cela est très certainement dû au pas de gradient qui doit être réduit à partir d'un certain nombre d'époch. Nous avions choisi de travailler avec un pas de gradient constant, et il aurait été judicieux, comme piste d'amélioration, de le diminuer afin de nous rapprocher plus facilement du minimum de la fonction.

FIGURE 3: epoch 1000

3.2 Reconstruction

L'article reconstruit l'ensemble de WordNet, mais nous n'avons essayé que les mots ayant un lien avec mammal. Nos résultats sont dans le tableau 1. En comparant avec les résultats de l'article, on constate qu'on a un meilleur Mean rank que le meilleur obtenu par la distance euclidienne du papier. En revanche, notre Mean Average précision est très en deça. Cela pourrait s'expliquer par un problème dans notre code, il est possible qu'au niveau de l'update on ait une erreur. Ce sentiment est confirmé par le peu d'évolution dans les scores en fonction du nombre de dimensions supplémentaire dans l'embeddings. En effet, les scores sont très similaires et le MAP semble même régresser. Il faut souligner que ces résultats sont ceux obtenus pour 250 epochs.

	Dimensionnalité					
	5	10	20	50	100	200
MAP	0.0127	0.0132	0.0116	0.0133	0.0129	0.0129
Mean Rank	730.008	726.27	749.49	721.223	722.109	723.69

TABLE 1: Score obtenus pour la reconstruction, avec **250** epochs, 10 périodes de burn-in et 20 nodes négatives

En poussant plus loin l'entrainement, jusqu'à 1000 epochs on trouve des résultats beaucoup plus intéressants. En effet, le tableau 2 montre des résultats du même ordre de grandeur que ceux exposés dans l'article pour la méthode de Poincaré. Malheureusement, on a toujours un soucis avec le fait que les nombre croissant de dimensions ne change pas significativement les scores obtenus.

	Dimensionnalité					
	5	10	20	50	100	200
MAP	0.183	0.176	0.186	0.187	0.189	0.19
Mean Rank	588.21	588.87	588.76	587.96	588.17	587.70

TABLE 2: Score obtenus pour la reconstruction, avec **1000** epochs, 10 périodes de burn-in et 20 nodes négatives

3.3 Link prediction

Pour la prédiction des liens, on effectue également le test sur les mots liés à mammal. Les résultats figure 3 sont similaires à la reconstruction. Le fait qu'on n'entraine sur un si petit jeu de données est surement un facteur limitant l'apprentissage.

A nouveau, on va jusqu'à 1000 epochs pour obtenir une meilleur convergence et des résultats se rapprochant de l'article. On note qu'avec 4 fois plus d'epochs on

	Dimensionnalité					
	5	10	20	50	100	200
MAP	0.0258	0.0268	0.0066	0.0186	0.0161	0.0151
Mean Rank	786.081	786.908	804.216	781.558	799.58	800.831

TABLE 3: Score obtenus pour la prédiction de liens, avec **250** epochs 10 périodes de burn-in et 20 nodes négatives

	Dimensionnalité					
	5	10	20	50	100	200
MAP	0.314	0.330	0.345	0.298	0.189	0.304
Mean Rank	657.41	653.08	652.84	671.67	588.17	671.71

TABLE 4: Score obtenus pour la prédiction de liens, avec **1000** epochs 10 périodes de burn-in et 20 nodes négatives

Conclusion

Nous avons travaillé avec un petit dataset de l'ordre du kilo-octet contenant les hypernymes d'un seul terme, et déjà le temps computationnel pour 250 epochs prend environ 20 minutes sur une machine. On imagine que pour des jeux de données plus importants, l'aspect computationnel devient primordial, et comme l'article l'évoque, la parallélisation devient un choix important. L'article fait notamment référence à la méthode de Hogwild pour paralléliser une descente de gradient stochastique, dans l'hypothèse où les mises à jour concernent un petit nombre d'embeddings, ce qui est le cas dans notre cadre. Nous avons aussi choisi de mettre à jour le gradient pour une batch-size égale à 1, car certains blogs sur la méthode de Poincaré parlent d'une amélioration des prédictions pour des tailles de batchs plus petites. On aurait pu cependant tenter de travailler avec des batchs un peu plus gros, de l'ordre de 10 ou 20.