Poincarre Embeddings Implementation

Charles Dognin
charles.dognin@ensae.fr

Robin Beaudet
robin.beaudet@ensae.fr

Abstract

This project consists in summarizing and implementing a recent method for learning hierarchical representations called Poincare embeddings. This state of the art method was developed by Maximilian Nickel and Douwe Kiela, from Facebook AI Research, in the paper "Poincare Embeddings for Learning Hierarchical Representations"[1].

1 Introduction

Embedding is a major task in machine learning that aims at representing various objects as vectors. For instance, in Natural Language Processing (NLP), one may want to embed words, that is, represent words as vectors in order to use these vectors for various tasks, such as machine translation (e.g. translate Chinese text to English), semantic analysis (e.g. What is the meaning of query statement?), or question answering (e.g. answering questions). Embedding graphs is another example as it may be of interest to detect communities in social networks.

To be useful, the embedding space should be of relatively low dimensionality. Thus, in NLP, a word is often represented as a large vector of dimensionality V - where V is the size of the entire vocabulary - filled with 0's, except for a 1 at the position corresponding to this word in the vocabulary. Embedding techniques such as the Skip-Gram model[2] represent each word of the vocabulary as a vector of dimensionality $D << V$, allowing to fill models with such inputs.

Naturally, a notion of similarity has to be introduced to measure the quality of the embedded vectors. Indeed, one would expect that the word vectors "cat" and "dog" should be more similar than the word vectors "bread" and "school". Notions of geometry even arise : it is captured within the word vectors and for instance, we can have "queen king = actress actor".

Though, classic embedding techniques do not take into account the complete structure of the objects they aim to embed as they are limited by the dimensionality of the embedding space. Said simply, the bigger the dimensionality, the better the modeling of the patterns but the worse the computational feasibility. The WordNet dataset[3] (a large lexical database of English nouns, verbs, adjectives and adverbs that are grouped into sets of cognitive synonyms, each being interlinked by means of semantic and lexical relations) gives us a hint of the type of latent hierarchical structure

[1]Maximilian Nickel and Douwe Kiela. "Poincare embeddings for learning hierarchical representations", 2017.

[2]Mikolov, T., Chen, K., Corrado, G., and Dean, J. "Efficient estimation of word representations in vector space". CoRR, abs/1301.3781. (2013)

[3]https://wordnet.princeton.edu/

we wish to recover (see Figure 1).

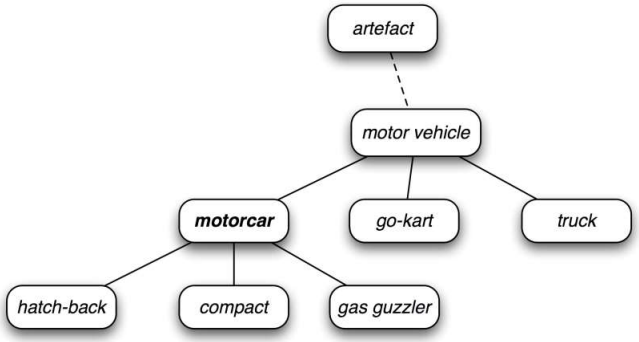

Figure 1: Hierarchy among words

Poincar embeddings allows to overcome this difficulty by considering a embedding space which does not use Euclidean geometry but rather hyperbolic geometry. In the next sections, we present the model, giving the intuition on how hyperbolic geometry can help us capturing more complex patterns in the data than Euclidean geometry.

2 Hyperbolic geometry for embeddings

We make the hypothesis that the data contain an underlying hierarchical structure, though we do not have prior knowledge of this latent structure. The embeddings should then be able to recover the similarity of the data as well as their hierarchy. Therefore, the approach follows a complete unsupervised paradigm which justifies the needs for a specific geometry.

As the authors point out, hyperbolic spaces can be thought as continuous versions of trees, which are well adapted to model hierarchical representations. Table 1 below compares some properties of Euclidean and hyperbolic geometries.

Property	Euclidean geometry	Hyperbolic geometry
Curvature K	0	< 0
Parallel lines	1	∞
Triangles are	normal	thin
Sum of triangle angles	π	$< \pi$
Circle length	$2\pi r$	$2\pi \sin h\xi r$
Disk area	πr^2	$2\pi(\cos h\xi r - 1)$

Table 1: Euclidean and Hyperbolic geometries. "Parallel lines" are the number of lines that are parallel to a line and that go through a point not belonging to this line. $\xi = \sqrt{K}$.

Intuitively, a hyperbolic space is well adapted for representing hierarchical data as it shares similar properties with a hierarchical tree. For instance, the number of children nodes of the latter grows exponentially with the level ℓ of the tree. It appears that with hyperbolic geometry, nodes that are ℓ levels below the root are located on a sphere with radius inferior to ℓ, whereas parent nodes to these nodes are located within the sphere.

In other words, such a geometry ensures that the learnt embeddings place "lower nodes" far from the origin, capturing the hierarchical patterns of the data. The reason can be derived from Table 1

where we see that with hyperbolic geometry, disk area as well as circle length grow exponentially with the radius, whereas it only grows linearly and quadratically with the radius for Euclidean geometry.

In a sphere within a hyperbolic space, two points that seem close (in the sense that they are close for the human eye) on the boundary are in reality much more far away than two points closer to the origin.

These nice properties are sufficient to justify the choice of hyperbolic geometry. Indeed, one does not need to increase the dimensionality of the embedding space to capture underlying hierarchical patterns as it would be the case with Euclidean geometry, leading to an increase in the computational cost as well as greater risks of overfitting.

Nickel and Kiela focus on the Poincar ball model $\mathcal{B}^d$ which is defined as follow :

$$\mathcal{B}^d = \{x \in R^d : ||x|| < 1\} \tag{1}$$

where $||.||$ is the Euclidean norm, along with the metric

$$g_x = \left(\frac{2}{1 - ||x||^2}\right)^2 g^E \tag{2}$$

where g^E is the Euclidean metric tensor.

The distance between two points u and v is given by the Poincar distance

$$d(u, v) = arcosh\left(1 + 2\frac{||u - v||^2}{(1 - ||u||^2)(1 - ||v||^2)}\right) \tag{3}$$

In addition to our previous remarks, we can note that according to equation (3), the Poincar distance increases much more quickly when they are closer to the boundary. But two points close from each other and to the boundary have a relatively low Poincar distance, which explains why children nodes of a same parent will be grouped together on a small arc on the boundary.

To conclude this section, we simply note that hyperbolic geometry should allow us to get the following properties :

- Similar data should be close together in the embedding space ;
- Connected nodes should be placed together in the embedding space ;
- Non-connected nodes should be placed far apart in the embedding space ;
- Lower nodes in the hierarchy should be far away from the origin ;
- Higher nodes in the hierarchy should be closer to the origin ;
- Computing cost should not be too high compared to Euclidean geometry.

3 The optimization process

Like many machine learning problems, we get the embeddings by solving an optimization problem. Consider a set of data $S = \{x_i\}_{i=1}^n$ which we wish to embed, for example, say that each x_i is a

word. Said differently, we want to find $\Theta = \{\theta_i\}_{i=1}^n$, $\theta_i \in \mathcal{B}^d$ where d is the dimension of the embedding space. The optimization problem is

$$\Theta' \leftarrow \Theta \mathcal{L}(\Theta) \quad s.t \quad \forall \theta_i \in \Theta : ||\theta_i|| < 1 \tag{4}$$

The choice of the loss function depends on the problem, but should penalize low distances between unconnected nodes and high distances between connected nodes.

The authors use the loss function

$$\mathcal{L}(\Theta) = \sum_{(u,v)\in\mathcal{D}} \log \frac{e^{-d(u,v)}}{\sum_{v'\in\mathcal{N}(u)} e^{-d(u,v')}} \tag{5}$$

but as mentioned by Jayant Jain in a post on Rare-Technologies.com[4], it could be of interest to have a loss function similar to a softmax by defining

$$\mathcal{L}(\Theta) = \sum_{(u,v)\in\mathcal{D}} \log \frac{e^{-d(u,v)}}{e^{-d(u,v)} + \sum_{v'\in\mathcal{N}(u)} e^{-d(u,v')}} \tag{6}$$

Here, $\mathcal{D} = \{(u,v)\}$ denotes the set of observed hypernymy[5] relations between noun pairs, and $\mathcal{N}(u) = \{v|(u,v) \notin \mathcal{D}\} \cup \{u\}$ are negative examples for u (as in the Skip-Gram model).

This loss function induces that similar objects are closer than objects for which we didnt observe a relationship (u,v).

The choice of the Poincar ball model is adapted for gradient optimization. Denote ∇_E the Euclidean gradient of $\mathcal{L}(\theta)$, which is equal to

$$\nabla_E = \frac{\partial \mathcal{L}(\theta)}{\partial d(\theta,x)} \frac{\partial d(\theta,x)}{\partial \theta} \tag{7}$$

where the first term is assumed to be known. Therefore, to perform an update, we need to differentiate the second term :

$$\frac{\partial d(\theta,x)}{\partial \theta} = \frac{4}{\beta\sqrt{\gamma^2-1}} \left(\frac{||x||^2 - 2\langle\theta,x\rangle + 1}{\alpha^2}\theta - \frac{x}{\alpha} \right)$$

where $\gamma = 1 + \frac{2}{\alpha\beta}||\theta - x||^2$, with $\alpha = 1 - ||\theta||^2$ and $\beta = 1 - ||x||^2$.

The update for each embedding is then

$$\theta_{t+1} \leftarrow proj\Big(\theta_t - \eta_t \frac{(1-||\theta_t||^2)^2}{4} \nabla_E\Big) \tag{8}$$

where $proj(\theta) = \frac{\theta}{||\theta||} - \epsilon$ if $||\theta|| \geq 1$, and θ otherwise. Here, the term ϵ only accounts for numerical stability, and η_t is the learning rate.

To get a clearer view of the previous optimization algorithm, the process consists in :

1. Parametrize each item in the Poincar ball via $proj(x)$
2. Optimize them by Riemannian optimization under metric $g_x = \left(\frac{2}{1-||x||^2}\right)^2 g^E$

[4]Jayant Jain. "Implementing Poincar Embeddings". https://rare-technologies.com/implementing-poincare-embeddings/

[5](From Wikipedia) A hyponym is a word or phrase whose semantic field is included within that of another word, its hypernym. A hyponym is in a type-of relationship with its hypernym. For example, pigeon, crow, eagle and seagull are all hyponyms of bird which is their hypernym.

The second step is decomposed as follow :

1. Compute Euclidean gradient ∇_E
2. Correct the metric via $g_\theta^{-1}\nabla_E$, with g_θ the Poincar ball metric tensor
3. Apply gradient descent $\theta \leftarrow \eta_t g_\theta^{-1}\nabla_E$
4. Project onto space $\theta \leftarrow \text{proj}(\theta)$

For training, the authors randomly initialize the embeddings close to the origin and find that they have better results when training during an initial "burn-in" period with a reduced learning rate $\frac{\eta}{c}$, for example $c = 10$.

4 Python implementation

The algorithm has been implemented in Python. In this section, we give more details about the implementation if the reader wishes to read the code.

The code depends on the pre-processing we apply to the data. In our example, we focused on the **WordNet** dataset.

> *WordNet is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets). Synsets are interlinked by means of conceptual-semantic and lexical relations. [...] WordNet groups words together based on their meanings. [...] The main relation among words in WordNet is synonymy, as between the words shut and close or car and automobile. Synonyms–words that denote the same concept and are interchangeable in many contexts–are grouped into unordered sets (synsets).*

First of all, we define a dictionnary `graph` of the form {`word:[list of hyponyms]`} and a dictionnary `levels` of the form {`word (node): level of node in the graph`}. Finally, we also define a dictionnary `embeddings` which will contain our embedded words in the following way : {`word: embedded vector`}.

In our class, we dedicate several functions to this preprocessing step.

Some other functions are :

- `dist` : computes the Poincare distance between two vectors
- `proj` : computes the projection of a vector in the Poincare disk ball
- `update` : performs the update of a vector $\theta_{t+1} \leftarrow proj\Big(\theta_t - \eta_t \frac{(1-||\theta_t||^2)^2}{4}\nabla_E\Big)$
- `pdr` : computes $\frac{\partial d(\theta,x)}{\partial \theta}$ with θ being a positive or negative sample
- `pdl` : computes $\frac{\partial \mathcal{L}(\theta)}{\partial d(\theta,x)}$ only when considering negative samples for u (otherwise it is equal to -1)

where the gradients were computed with respect to the loss $\mathcal{L} = -d(u,v) - \log\Big(\sum_{v'} e^{-d(u,v')}\Big)$.

At each iteration, we update simultaneously u, v and the negative examples $[v_1, ..., v_k]$ where k is the number of negative samples. We thus need to compute the partial derivatives of the loss with respect to these vectors :

- $\frac{\partial \mathcal{L}}{\partial u} = \frac{\partial \mathcal{L}}{\partial d(u,v)}\frac{\partial d(u,v)}{\partial u}$

- $\frac{\partial \mathcal{L}}{\partial v} = \frac{\partial \mathcal{L}}{\partial d(u,v)} \frac{\partial d(u,v)}{\partial v}$

- $\frac{\partial \mathcal{L}}{\partial v'} = \frac{\partial \mathcal{L}}{\partial d(u,v')} \frac{\partial d(u,v')}{\partial v'}$ (put into a list of partial derivatives for negative examples)

4.1 Improvements

In our implementation, the negative examples are chosen randomly from the vocabulary. Even if the vocabulary contains each word once, some of them appear more frequently in written documents. In their paper "Distributed Representations of Words and Phrases and their Compositionality", Mikolov et al. take into account this non-uniform distribution of words and proceed to select negative samples according to the following distribution :

$$P(\omega_i) = \frac{f(\omega_i)}{\sum_{j=1}^{n}(f(\omega_j))^{3/4}} \tag{9}$$

where $f(\omega_i)$ is the weight of word ω_i.

To fully understand, take an example :
- is: $0.9^{3/4} = 0.92$
- constitution: $0.09^{3/4} = 0.16$
- bombastic: $0.01^{3/4} = 0.032$
"Bombastic" is now 3 times more likely to be sampled while "is" only went up marginally.

5 Results

We tested our implementation on a hierarchical graph whose origin node is "mammal". The depth of the graph was set between 3 and 7, and the number of negative examples to be sampled was set between 4 and 10. In order to plot the results of the final embedding, we decided to set the embedding dimension to 2.

We obtained the following results :

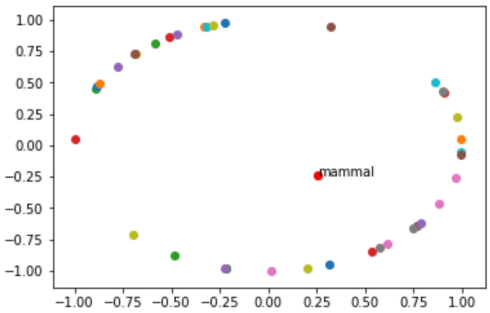

Figure 2: 10000 epochs, 3 levels of hierarchy, Learning Rate of 0.0008, 6 negative samples, -0.05;0.05 for initialization

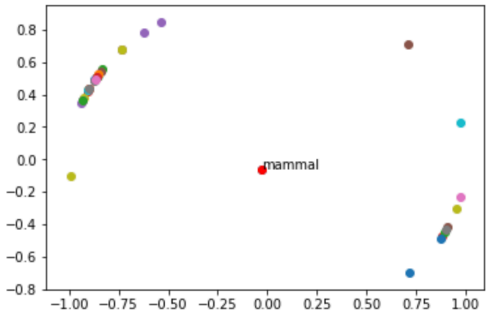

Figure 3: 1000 epochs, 3 levels of hierarchy, Learning Rate of 0.01, 4 negative samples, [-0.05;0.05] for initialization

You will also find on the notebook graphs representing the evolution of the embeddings'learning.

6 Conclusion

We eventually obtained the right results in terms of the embedding of the hierarchical vocabulary. Indeed, we observe that the lower nodes in the graphs are well spread across the 2-dimensional sphere. Also the mammal word clearly stands out in the middle. We observed that results heavily depend on the initialization of the embeddings as well as the learning rate and the number of epochs. In particular, when the learning rate is too small, the embeddings cannot move away from the center and a very large number of epochs would be needed for convergence.

7 References

[1] Jayant, J. ”Implementing Poincar Embeddings.” Rare Technologies. (2017)

[2] Mikolov, T., Chen, K., Corrado, G., and Dean, J. ”Efficient estimation of word representations in vector space.” CoRR, abs/1301.3781. (2013)

[3] Nickel, T. and Kiela, D. ”Poincar embeddings for learning hierarchical representations.” arXiv:1705.08039v2 [cs.AI]. (2017).