Geometric Methods project: Sinkhorn embeddings

Gabriel ROMON
gabriel.romon@ensae.fr

Salomé DO
salome.do@ensae.fr

1 Introduction

In many machine learning problems data points live in a high-dimensional metric space $(\mathcal{A}, d_{\mathcal{A}})$ where computations are not tractable. It is often necessary to reduce the dimensionality of the data so that it can be used efficiently in downstream tasks. This can be achieved by **learning an embedding** ϕ that maps each data point into a target metric space $(\mathcal{B}, d_{\mathcal{B}})$ with the constraint that ϕ preserves distances to some extent. In the past, techniques were developed to learn embeddings into Euclidean spaces where $\mathcal{B} = \mathbb{R}^n$ and the distance is given by a p-norm or the cosine similarity. The authors of [1] argue that these target spaces cannot represent complex relationships between data points. Instead, they explore embeddings into Wasserstein spaces.

We begin with a short review of Wasserstein spaces and Sinkhorn divergences. Then we move on to the learning problem, with a focus on word embeddings. Finally we demonstrate our attempt to reproduce the results of [1].

2 Review of some optimal transport topics

2.1 Wasserstein spaces

Let $(\mathcal{X}, d)$ be a metric space and μ, ν be probability measures on $(\mathcal{X}, \mathcal{B}(\mathcal{X}))$. A probability measure P on $\mathcal{X} \times \mathcal{X}$ is said to be a coupling of (μ, ν) if for any $A, B \in \mathcal{B}(\mathcal{X}), P(A \times \mathcal{X}) = \mu(A)$ and $P(\mathcal{X} \times B) = \nu(B)$. Let $\Pi(\mu, \nu)$ denote the set of such couplings. $\Pi(\mu, \nu)$ is clearly non-empty since it contains the product measure $\mu \otimes \nu$.
Let $p \geq 1$. The p-**Wasserstein distance** is defined by

$$\mathcal{W}_p(\mu, \nu) := \left(\inf_{P \in \Pi(\mu,\nu)} \int \int d(x,y)^p dP(x,y) \right)^{1/p}$$

If $(\mathcal{X}, d)$ is Polish, $\mathcal{W}_p$ is a metric on the **Wasserstein space** $\mathcal{W}_p(\mathcal{X})$ of probability measures with finite moments of order p (see [2]).

[1] deals exclusively with the **discrete case**, where couplings have the form

$$P = \sum_{i=1}^{M} \sum_{j=1}^{N} P_{ij} \delta_{(x_i, y_j)}$$

where N, M are fixed integers and $x_1, \dots, x_N, y_1, \dots, y_M$ are called **support points**. One can alternatively constrain μ and ν to have the form $\mu = \sum_{i=1}^{M} u_i \delta_{x_i}$ and $\nu = \sum_{j=1}^{N} v_j \delta_{y_j}$.
Let $D \in \mathbb{R}_+^{M \times N}$ be the matrix of pairwise distances defined by $D_{ij} = d(x_i, y_j)$. In the discrete case, $\mathcal{W}_p$ verifies

$$[\mathcal{W}_p(\mu, \nu)]^p = \min_{\substack{T \in \mathbb{R}_+^{M \times N} \\ T \mathbb{1}_N = u \\ T^\top \mathbb{1}_M = v}} \sum_{i=1}^{M} \sum_{j=1}^{N} D_{ij}^p T_{ij} = \min_{\substack{T \in \mathbb{R}_+^{M \times N} \\ T \mathbb{1}_N = u \\ T^\top \mathbb{1}_M = v}} \operatorname{tr}(D^p T^\top)$$

where D^p denotes entrywise multiplication. The equality on the left shows that the optimization problem at hand is a linear program.

2.2 Sinkhorn divergences

Solving this linear program is computationally intensive and regularization techniques have been developed to approximate solutions by leveraging the abilities of GPU to perform matrix operations, thus providing a significant speed boost.
Let $\lambda \geq 0$ denote the regularization parameter. The **Sinkhorn divergence** $\mathcal{W}_p^\lambda$ is defined by

$$[\mathcal{W}_p^\lambda(\mu, \nu)]^p = \min_{\substack{T \in \mathbb{R}_+^{M \times N} \\ T\mathbb{1}_N = u \\ T^\top \mathbb{1}_M = v}} \sum_{i=1}^{M} \sum_{j=1}^{N} \left[D_{ij}^p T_{ij} + \lambda(T_{ij}(\log(T_{ij}) - 1)) \right]$$

Let $K \in \mathbb{R}_+^{M \times N}$ be the kernel matrix defined by $K_{ij} = \exp\left(-\frac{D_{ij}^p}{\lambda}\right)$. In [3] it is shown that the solution of the regularized problem has the form $T = \operatorname{diag}(r) K \operatorname{diag}(c)$ where $r \in \mathbb{R}_+^M$, $c \in \mathbb{R}_+^N$. Plugging this back into the constraints yields

$$\begin{cases} \operatorname{diag}(r) K c = u \\ \operatorname{diag}(c) K^\top r = v \end{cases} \iff \begin{cases} r \odot (Kc) = u \\ c \odot (K^\top r) = v \end{cases}$$

By optimizing directly on $(r, c) \in \mathbb{R}_+^M \times \mathbb{R}_+^N$ instead of $T \in \mathbb{R}_+^{M \times N}$, we get a **matrix scaling** problem which can be solved iteratively with the simple **Sinkhorn iterations**:

$$\begin{cases} r \leftarrow \frac{u}{Kc} \\ c \leftarrow \frac{v}{K^\top r} \end{cases}$$

where the quotients are taken entrywise.

As explained in [3], the algorithm may lead to numerical overflows when λ is small compared to the entries of D^p. Section 4.4 of the book explains how this can be avoided by carrying out computations in log-space.

3 The learning problem

3.1 Generalities

Learning Wasserstein embeddings can be framed in the following way. Let $\mathcal{C}$ be a set of data points (words, images, ...), let $r : \mathcal{C} \times \mathcal{C} \to \mathbb{R}$ be a target relationship that measures the degree of similarity between two points and suppose we are given training samples $(u^{(i)}, v^{(i)}, r(u^{(i)}, v^{(i)}))_{i \in [\![1, N]\!]}$. Note that r is known only through its values on the samples.

The goal is to learn a mapping $\phi : \mathcal{C} \to \mathcal{W}_p(\mathcal{X})$ such that $\mathcal{W}_p\left(\phi(u^{(i)}), \phi(v^{(i)})\right)$ is a good proxy for $r(u^{(i)}, v^{(i)})$. Ideally, there should exist some increasing function $\psi : \mathbb{R}_+ \to \mathbb{R}_+$ such that $\mathcal{W}_p(\phi(\cdot), \phi(\cdot)) \approx \psi \circ r(\cdot, \cdot)$.

The authors of [1] constrain their embeddings to the **discrete case**, with a **fixed number of support points** M and **uniform weights**. For an arbitrary space $(\mathcal{X}, d)$, the embedding mapping in the corresponding Wasserstein space $\mathcal{W}_p(\mathcal{X})$ is therefore of the form

$$\phi : c \mapsto \sum_{i=1}^{M} \frac{1}{M} \delta_{x_i}$$

and the only elements to be learned are the locations $(x_1, \ldots, x_M) \in \mathcal{X}^M$ which depend on $c \in \mathcal{C}$.

Since the regularized transport problem is much more computationally tractable, it makes sense to replace $\mathcal{W}_p$ with $\mathcal{W}_p^\lambda$ for a fixed $\lambda > 0$. Given a **problem-specific loss function** $\mathcal{L}$ that measures how close $\mathcal{W}_p^\lambda\left(\phi(u^{(i)}), \phi(v^{(i)})\right)$ is to $r(u^{(i)}, v^{(i)})$ and $\mathcal{H}$ a class of mappings, the learning problem is formulated in [1] as

$$\arg\min_{\phi \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(\mathcal{W}_p^\lambda\left(\phi(u^{(i)}), \phi(v^{(i)})\right), r(u^{(i)}, v^{(i)})\right)$$

$\mathcal{H}$ can be a parametric family of models indexed by a parameter-vector $\theta \in \mathbb{R}^p$: $\mathcal{H} = \{\phi_\theta, \theta \in \mathbb{R}^p\}$. In order to solve the minimization problem, one must be able to compute the gradient of $\mathcal{L}\left(\mathcal{W}_p^\lambda\left(\phi_\theta(u^{(i)}), \phi_\theta(v^{(i)})\right), r(u^{(i)}, v^{(i)})\right)$ with respect to θ, and the only hurdle for that matter is $\mathcal{W}_p^\lambda$. Luckily, Sinkhorn iterations are each differentiable with respect to the locations of the point clouds that are passed as inputs, hence $\mathcal{W}_p^\lambda$ is amenable to **automatic differentiation**.

The authors of [1] apply their framework to graph embeddings and word embeddings. We put our focus on word embeddings because of personal preference and our goal was to replicate the results of the paper.

3.2 Word embeddings

The authors choose $\mathbb{R}^k$ with the Euclidean norm $\|\cdot\|_2$ as the ground metric space. k is taken in $\{2, 3, 4\}$ and the Wasserstein metric exponent p is fixed to 1.

In order to deal with words, it suffices to design a relationship r and a suitable loss $\mathcal{L}$. The authors define a context-based r by fixing some window size $l = 2$ and, given a sentence $(\mathbf{x}_0, \ldots, \mathbf{x}_n)$ they let $r(\mathbf{x}_i, \mathbf{x}_j) = \mathbb{1}_{|i-j| \leq l}$. The loss for two words $\mathbf{x}_i, \mathbf{x}_j$ in the same sentence is defined as

$$r(\mathbf{x}_i, \mathbf{x}_j)\left(\mathcal{W}_1^\lambda(\phi(\mathbf{x}_i), \phi(\mathbf{x}_j))\right)^2 + (1 - r(\mathbf{x}_i, \mathbf{x}_j))\left[\mathrm{ReLU}\left(m - \mathcal{W}_1^\lambda(\phi(\mathbf{x}_i), \phi(\mathbf{x}_j))\right)\right]^2$$

The global loss is obtained by summing through all sentences and all pairs of words in each sentence.

The mappings ϕ are constrained to be neural networks with a specific architecture detailed below.

4 Implementation

4.1 Comments on the paper

The previous section only gave a vague outline of the coding process. Note that the autors did not release any source for their models. More importantly, there are **a number of technical details essential to the implementation that remain unclear or missing, even after reading the paper**.

1. The authors used the corpus `text8` which is a collection of Wikipedia articles. This corpus has ≈ 17 million words, but it is just one line of continuous text. There are **no sentence boundaries** or punctuation marks. However, since context windows are defined for a given sentence, the way the corpus is divided into sentences matters. The NLP module `gensim` has a built-in function which iterates through `text8` by creating sentences with 10.000 words (default value). We assume the authors split the corpus in a similar fashion, but they did not report the length of their sentences. We chose a value of 1000 words per sentence in our implementation.
2. The authors **did not provide the values of some important hyperparameters**: the value of the margin parameter m in the loss is unknown. We fixed it to $m = 1$. The number of Sinkhorn iterations in the computation of W_1^λ is unknown as well. We fixed it to 20.
3. The **architecture for the neural network is also unclear**. The authors state they use "a Siamese architecture [...] with negative sampling [...] for selecting words outside the context. The network architecture in each branch consists of a linear layer with 64 nodes followed by our point cloud embedding layer."
    (a) A Siamese network is theoretically made up of two subnetworks that share the exact same weights. However, in practice, there is **no need to define subnetworks or branches**. The same network can be applied during the forward pass on the two sample points to be compared, and gradients will automatically be accumulated.
    (b) **Negative sampling** is mentioned in relation to Word2Vec, which is **quite vague** for the setting of the paper. In Mikolov's paper negative sampling was used to limit the number of weight updates done after each forward pass. For a given sentence and a

given word $\mathbf{x}_i$, the loss may be rewritten as

$$\sum_{\mathbf{x}_j \in \text{context}(\mathbf{x}_i)} \left(\mathcal{W}_1^\lambda(\phi(\mathbf{x}_i), \phi(\mathbf{x}_j))\right)^2 + \sum_{\mathbf{x}_j \notin \text{context}(\mathbf{x}_i)} \left[\text{ReLU}\left(m - \mathcal{W}_1^\lambda(\phi(\mathbf{x}_i), \phi(\mathbf{x}_j))\right)\right]^2$$

The second sum is much heavier to compute than the first one. In our setting, it is reasonable to assume that negative sampling simply means **computing the second sum over a few randomly sampled** $\mathbf{x}_j \notin \text{context}(\mathbf{x}_i)$ (and not over all such $\mathbf{x}_j$). Since the paper mentions a sampling rate of 1 per positive we sample uniformly as many negatives as there are words in the context of $\mathbf{x}_i$ (which may be less than 4 when x_i is at the boundaries of the sentence).

(c) In the OpenReview comment [4] the authors provide some clarification on the architecture: the **inputs** to the networks are **one-hot encoded vectors**, and the network is made of a **fully-connected linear layer** with 64 nodes, **followed by another fully-connected linear layer** (referred to in the paper as the "point cloud embedding layer"). This last layer outputs the locations of the support points, so it is reasonable to assume that its output has size kM, which can then be reshaped as a cloud of M points in $\mathbb{R}^k$. Besides, the authors provide some information that is **absent from the paper**: the output of the point cloud layer "is normalized to lie within the unit ball". We did not implement this unit normalization.

(d) Regarding the training itself, the authors do not describe their pipeline, nor do they mention the **batch size** they used (if any). However, they do specify that they used Adam optimizer, the value of the entropic parameter is $\lambda = 0.05$ and they trained for 3 epochs.

4.2 Our implementation

We split the `text8` dataset into sentences of 1000 words each, so there is a total of ~ 17.000 sentences. As in [1] we only keep the 8000 most frequent words. A dictionary is built to match each word in the vocabulary with a unique numeric identifier. This makes it easy to build one-hot vectors for each word.

We used Pytorch as a linear algebra and automatic differentiation framework. Given the originality of the task it was not possible to use Pytorch dataloaders out of the box, so we coded our own. We created the function `loader(bsize)` which generates batches. Each batch is made of `bsize` sentences, and for each sentence, all the contexts and negative words are already precomputed (before even calling `loader`). The training loop is detailed in Algorithm 1 below. Note that the original order between sentences was not kept in the batches.

Algorithm 1 Training loop

```
1:  Initialize n_epochs and n_batches
2:  for epoch in n_epochs do
3:      for batch in n_batches do
4:          for sentence in batch do
5:              for word in sentence do
6:                  Get context words and negatives
7:                  Compute embedding of word (forward pass through the network)
8:                  for positive in context words do
9:                      Compute embedding of positive (forward pass through the network)
10:                     Loss += $\left(\mathcal{W}_1^\lambda(\phi(\texttt{word}), \phi(\texttt{positive}))\right)^2$
11:                 for negative in negatives do
12:                     Compute embedding of negative (forward pass through the network)
13:                     Loss += $\left[\text{ReLU}\left(m - \mathcal{W}_1^\lambda(\phi(\texttt{word}), \phi(\texttt{negative}))\right)\right]^2$
14:         Perform backpropagation
```

Our implementation of the Sinkhorn algorithm is done with vanilla iterations, but for the sake of numerical stability we should have used the version in log-space. As stated before we arbitrarily set the number of iterations in the Sinkhorn to 20. We tested higher iteration values (~ 100) and we noticed that it significantly slowed down the backpropagation step (it went from a few seconds to ~ 40 seconds), which is expected since the computational graph becomes more complex.

Since we were primarily interested in visualizing our embeddings, we chose $k = 2$ (a two-dimensional ground metric space) and we set the number of support points to $M = 10$, so that the point clouds are easily observable without resorting to kernel density estimation as in [1].

We ran our experiments on a Google Cloud instance with 8 CPU cores, 16 GO of RAM and a P100 GPU. We could not run the training loop with a batch size larger than 50 because the RAM would max out and the instance would crash. We cannot explain this behavior. To avoid RAM issues we performed training with a batch size of 1. The training loop went through ~ 3000 sentences in 4 hours and we interrupted the program prematurely. Even though the first epoch was far from over, the network we get still provides some interesting results.

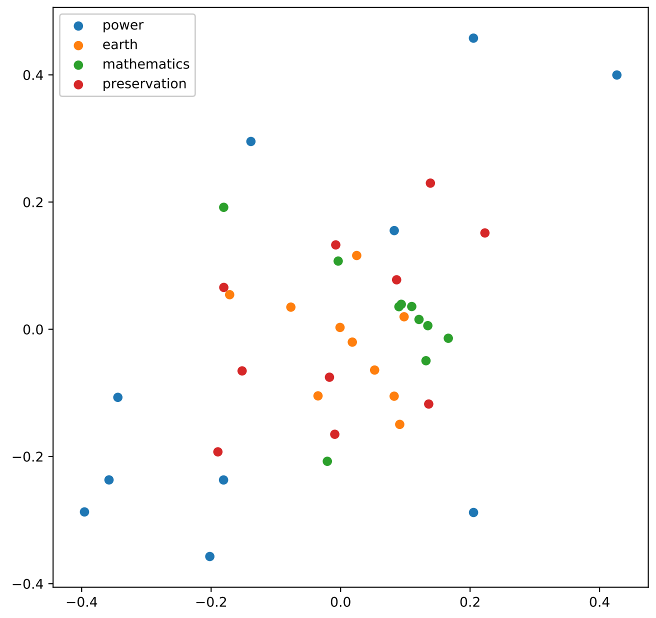

Figure 1: Embeddings obtained for 4 words

In Figure 1 we show the embeddings for the words `power`, `earth`, `mathematics` and `preservation`. The cloud for `mathematics` is remarkably tightly clustered. The clouds for `earth` and `preservation` show a lot of overlapping, most likely because both words tend to co-occur in Wikipedia articles on the environment. The embedding of `power` is split in two distant subclouds. This is expected because this word is polysemic: it has different meanings in politics and physics.

In Figure 2 we show the embeddings for the words `football`, `basketball`, `soccer` and `rugby`. The locations of the support points for the 4 clouds are very similar. This proves that the embeddings from our model capture similarity between words.

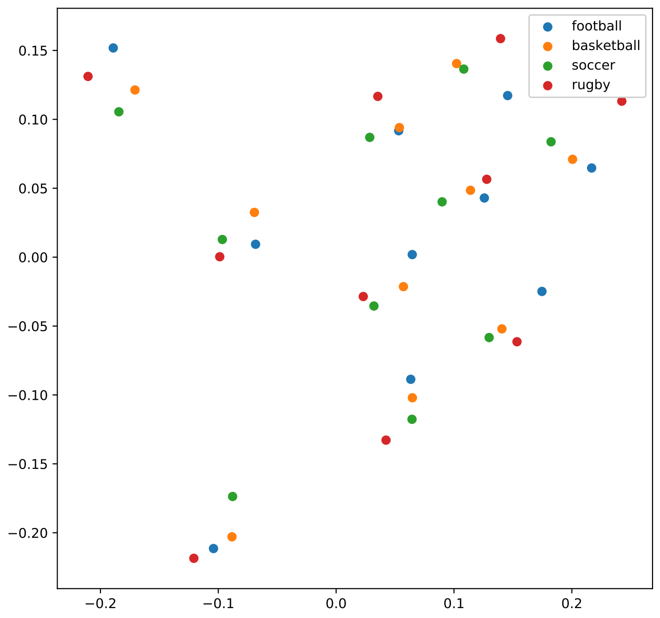

Figure 2: Embeddings for 4 words related to sports

5 Conclusion

Using discrete optimal transport to learn embeddings is an interesting idea. The paper could be extended by removing the constraint that the weights in each cloud be uniform.

Regarding reproducibility of the experiments, the paper lacks clarity in some places. Our attempt to reproduce the results on word embeddings is very promising given that the training was done only on 3000 out of the 17.000 sentences. If we had had the time and resources needed to complete a full epoch (or 3 as in the paper) we surely could have reproduced all their results.

References

[1] Charlie Frogner, Farzaneh Mirzazadeh, and Justin Solomon. Learning Embeddings into Entropic Wasserstein Spaces. *arXiv preprint arXiv:1905.03329*, 2019.

[2] Cédric Villani. *Topics in optimal transportation*. Number 58. American Mathematical Soc., 2003.

[3] Gabriel Peyré, Marco Cuturi, et al. Computational optimal transport. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019.

[4] OpenReview comment. `https://openreview.net/forum?id=rJg4J3CqFm&noteId=S1xCCk9GfN`.