Synthesis of the paper "Learning embeddings into entropic Wasserstein spaces"

GHERMI Ridouane
ENSAE Paris
ridouane.ghermi@ensae.fr

1 Introduction

Embeddings are at the core of deep learning. In fact, its success can be (at least partly) explained by the use of learned embeddings instead of handcrafted features as in classical machine learning.

An embedding is a vector representation of the input data that captures the semantic structure of the data. They are used for downstream tasks, such as classification. Typically, embeddings are computed by a neural network and classification is performed by the final layer (often a softmax layer), in which case the model is trained end-to-end by backpropagation.

Examples of successfully used embeddings include word embeddings in NLP (Word2Vec, GloVe, fastText, ELMo), graph embeddings in Graph Machine Learning (node2vec), visual embeddings in Computer Vision (pre-trained ImageNet embeddings, Siamese Networks).

Usually, data is embedded in the Euclidean space where the Euclidean distance (or sometimes angle) tells how similar two embeddings are. The key idea of this paper is to embed data as probability distributions in a Wasserstein space and therefore use an Optimal Transport metric to measure the similarity between two embeddings.

More precisely, several design choices are made in this paper. Data is embedded as discrete distributions supported at a fixed number of points. Instead of computing the Wasserstein distance (which requires to solve a linear program), we use an approximation called Sinkhorn divergence. This distance requires us to solve another transport problem which has the advantage to be differentiable. Thus, we can incorporate the Sinkhorn divergence in any ML pipeline and compute the gradients via automatic differentiation.

In the original paper, this method is tested to create graph embeddings and word embeddings. We will show that this framework also applies for learning unsupervised visual embeddings via Siamese Networks. We will try this method on the MNIST dataset, which is extensively used in the computer vision literature.

2 Preliminaries

2.1 Optimal transport and the Wasserstein distance

Let (X, d) be a metric space and μ, ν be probability measures over $(X, B(X))$. A probability measure π on $X \times X$ is a transport plan of (μ, ν) if for any $A, B \in B(X)$, $\pi(A \times X) = \mu(A)$ and $\pi(X \times B) = \nu(B)$. The set of transport plans is noted $\Pi(\mu, \nu)$.

Let $p \geq 1$. The p-Wasserstein distance between probability distributions μ and ν over a metric space (X, d) is

$$W_p(\mu, \nu) = \left(inf_{\pi \in \Pi(\mu,\nu)} \int_{X \times X} d(x_1, x_2)^p d\pi(x_1, x_2) \right)^{1/p}$$

The Wasserstein distance is the cost of the optimal transport plan matching μ and ν. We now consider the case of discrete distributions supported on finite sets of points in R^n, such as

$$\mu = \sum_{i=1}^{M} u_i \delta(x_i), \nu = \sum_{j=1}^{N} v_j \delta(y_j)$$

where u and v are vectors of nonnegative values summing to 1, and $(x_i)_i$ and $(y_j)_j$ are the support points. Then, a transport plan π becomes discrete and is determined by the matrix $P \in R_+^{M \times N}$ such as

$$\pi = \sum_{i,j} T_{ij} \delta(x_i) \delta(y_j)$$

where $\forall i, \sum_j T_{ij} = u_i$ and $\forall j, \sum_i T_{ij} = v_j$. These constraints can be written in matrix form as $T1 = u$ and $T^T 1 = v$. We note D the matrix of pairwise metric distances, with $D_{ij} = d(x_i, y_j)$. Thus, for discrete distributions, the p-Wasserstein distance is

$$W_p(\mu, \nu)^p = min_{T \geq 0} Tr(D^p T^T) s.t. T1 = u, T^T 1 = v$$

where D^p is taken elementwise. To compute the Wasserstein distance between two discrete probability distributions, we need to solve a linear program.

2.2 Sinkhorn divergence

To simplify this problem, we add an entropic regularization term to the objective function, resulting in a modified version called Sinkhorn divergence:

$$\left(W_p^{\lambda}(\mu, \nu)\right)^p = min_{T \geq 0} Tr(D^p T^T) + \lambda Tr(T(\log(T) - 11^T)^T) s.t. T1 = u, T^T 1 = v$$

where $log(.)$ is applied elementwise and $\lambda \geq 0$ is the regularization parameter. It's called an entropic regularizer because this term enforces T to be a uniform probability.

We can show that the solution of the regularized problem takes the form $T^* = Diag(r) \exp(-D^p/\lambda) Diag(c)$, where $r \in R_+^M$ and $c \in R_+^N$. Therefore, we can optimize directly on (r, c) instead of T. This can be solved via matrix balancing, starting from an initial matrix $K = \exp(-D^p/\lambda)$ and alternately projecting onto the marginal constraints until convergence:

$$r \leftarrow u./Kc, c \leftarrow v./K^T r$$

where ./ is the elementwise division. These are called Sinkhorn iterations. The main advantage is that these iterations are differentiable. We can then incorporate a fixed number of Sinkhorn iterations in any machine learning pipeline and then perform backpropagation (with automatic differentiation provided by libraries such as TensorFlow or PyTorch for example).

3 Learning Wasserstein embeddings

We aim to recover a pairwise distance or similarity relationship $r : C \times C \to R$ where C is a collection of objects (e.g. words, symbols, images, nodes of a graph, etc.). To do that, we have training samples $S = (u_i, v_i, r_i)$ with $r_i = r(u_i, v_i)$. Our goal is to find a map $\phi : C \to W_p(X)$ such that $W_p(\phi(u), \phi(v)) \simeq r(u, v)$. We reach that by considering a parametric mapping ϕ_θ (such as a Neural Network) and minimizing a loss function $L(W_p(\phi(u), \phi(v)), r)$.

We give some examples. For word embeddings, (u_i, v_i) are words (integers identifying them in a vocabulary) and $r_i \in \{0, 1\}$ such that $r_i = 1$ if v_i is in the context of u_i. The loss function is then a contrastive loss function defined as

$$L(d_i, r_i) = r_i d_i^2 + (1 - r_i) max(0, m - d_i)^2$$

where $d_i = d(\phi(u_i), \phi(v_i))$ and m is a fixed margin. If r_i is equal to 1, we want the distance between the two embeddings to be as close as possible. Otherwise, we want the two embeddings to be far from each other but no more than a given margin.

We have a very similar situation for facial recognition system. (u_i, v_i) are images representing two faces. $r_i \in \{0, 1\}$ is equal to 1 if these are two pictures of the same person, and 0 otherwise. The mapping ϕ corresponds to a feature extraction, where visual features are relevant for the task. In the case of facial recognition, features of pictures of the same person should be close together, and features of two different persons should be distant from each other. That's why we also use the contrastive loss in computer vision. Embeddings (often computed by a CNN) are therefore invariant to transformations such as illumination, rotation of the face, wearing glasses or not, etc.

This framework also encompasses metric embeddings, where r is a distance metric we wish to approximate by the distance between our embeddings in the Wasserstein space (this is also called metric learning), and graph embeddings, where (u_i, v_i) are nodes of a graph and $r_i \in \{0, 1\}$ is equal to 1 if they are connected.

Two things need to be taken into account for solving this problem. First, the authors consider discrete probability distributions in R^k with finite support of M points and uniform weights. Hence the mapping ϕ needs to output matrix of size $M \times k$. Second, as said previously, we compute the Wasserstein distance between two discrete distributions via a fixed number of (differentiable) Sinkhorn iterations. These iterations are added to the pipeline and gradients are computed by automatic differentiation.

4 Empirical study

We focus our attention on unsupervised visual embeddings, which is very similar to word embeddings in its optimization process: we use a neural network to embed the data and we minimize a contrastive loss.

4.1 Siamese Networks

Siamese networks is a training procedure leading to a model able to extract visual embeddings from any image, these embeddings being relevant for a given task.

They are extensively used in facial recognition, where the task is to extract embeddings which are invariant to the identity of the individual. It means that two embeddings from two different pictures of the same person should be close to each other (in the embedding space, typically the Euclidean space). On the other hand, pictures of two different persons should conduct to embeddings far away from each other. These embeddings should thus be invariant to transformations such as illumination, rotation of the face, wearing glasses or not, having a beard or not, etc. These embeddings should on the contrary be sensitive to skin color, eye color, details on the face, etc.

In practice, siamese networks are very similar to word embeddings (in particular word2vec). The dataset is composed of triplet (u_i, v_i, r_i) where u_i and v_i are two pictures of a face, and $r_i = 1$ if these pictures are of the same person, $r_i = 0$ otherwise. The training is as follows: we pass both image through the same CNN (one after the other) and get a visual embedding for each of them, denoted $\phi(u_i)$ and $\phi(v_i)$. If $r_i = 1$, we want to minimize the distance between these two embeddings; if $r_i = 0$, we want to maximize the distance between these two embeddings (up to a maximum distance m, called the margin, beyond which we consider the objective achieved). This is summarized by the following contrastive loss function

$$L(\phi(u_i), \phi(v_i), r_i) = r_i d(\phi(u_i), \phi(v_i)) + (1 - r_i) \max(0, m - d(\phi(u_i), \phi(v_i)))$$

where $d(., .)$ is a distance on the embedding space. In the basic setting, the embedding space is R^d and d is the Euclidean distance between two vectors.

At the end of the training, we get a neural network which is able to extract embeddings invariant for the relation represented by r. This can be seen as a pre-training where, in order to succeed in the given task, the model needs to be able to extract relevant features of an individual's face. We can then extract embeddings from pictures of previously unseen individuals and compare them between each other, creating a classification model (or facial recognition system) only from a few images of an individual.

4.2 Unsupervised visual embeddings in entropic Wasserstein space

To improve upon the basic setting of siamese networks, which uses Euclidean distance between vectors in R^d as a similarity measure, we use the idea presented in the paper about Sinkhorn embeddings.

This is readily done by projecting images into a Wasserstein space, which is here approximated by a discrete probability space in R^k with M support points. Thus, the network outputs a matrix of dimension $d = k \times M$. We will take $k = 2$ in order to visualize this probability distribution on a 2D plane (as a point cloud).

On the other hand, the contrastive loss function stays the same, except for the distance which is now the Wasserstein distance between discrete probability distributions. This distance is approximated by adding an entropic term, resulting in a fixed number of steps of (differentiable) Sinkhorn iterations added to the pipeline.

4.3 Experiments & Results

We implement our solution using Python and the PyTorch library for automatic differentiation. The model is a convolutional neural networks (CNN), classically used to extract features from an image. Sinkhorn iterations are added as an intermediate step in the computation of the loss function, the gradients are thus computed via automatic differentiation.

We apply our method on the basic MNIST dataset, which includes images of handwritten numbers. We consider 10 classes (handwritten numbers from 0 to 9), and a triplet (u_i, v_i, r_i) from the dataset is created by randomly taking images of the different classes and assigning $r_i = 1$ or $r_i = 0$ according to the situation. As they are far more "negative" samples (samples where $r_i = 0$) than "positive" ($r_i = 1$) ones, we make sure to balance each mini-batch sampled from the dataset.

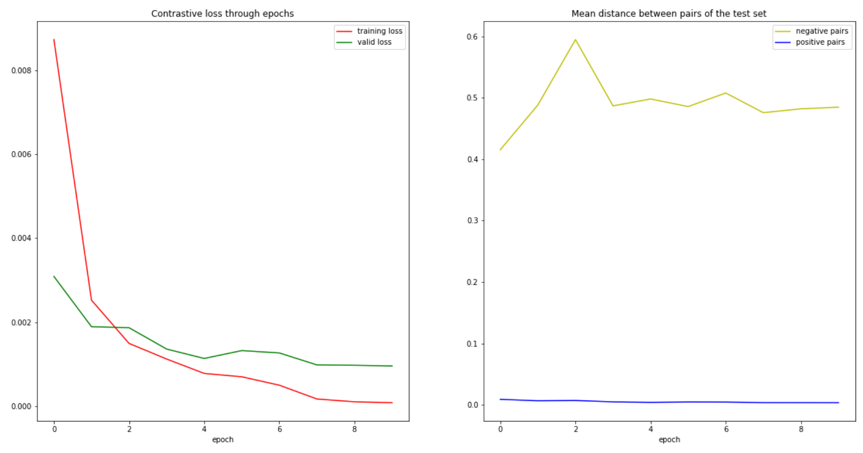

Figure 1: Metrics throughout training. (Left) Loss functions. (Right) Mean distance between pairs of the test set. Average distance between positive pairs decreases while average distance between negative pairs tends to raise.

Figure 1 shows the evolution of contrastive loss functions on both the training set and the test set throughout training. It allows us to monitor learning. To get a better metric, we also check the mean distance between positive/negative pairs, and see that positives tend to be closer together and negatives distant from each other as training goes on. The distance used to monitor training is the Sinkhorn divergence.

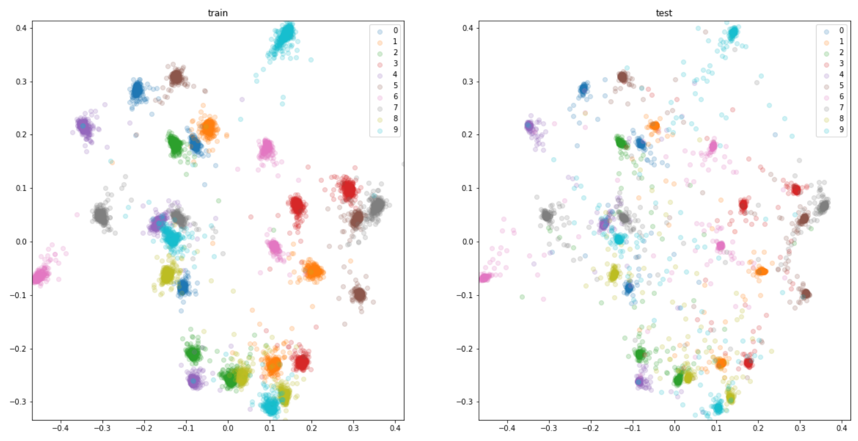

Figure 2: Visualization of the whole MNIST dataset as point clouds. Each color represents a class. We can note that each class fills 3 distinct regions of the space. A single data point corresponds to a point cloud, i.e. three points with each of them in the three parts of the class distribution.

After training, we investigate the model obtained. We recall that the final model (the trained neural network) allows us to project an image into a point cloud of 2-dimensional points (which is equivalent to a discrete probability distributions in R^2 with M support points). Here, $M = 3$. In Figure 2, we display the whole MNIST dataset as point clouds. Each color represents a class. We notice that classes are well clustered, and they have components in three very distinct regions of the space. A single image corresponds to a point cloud, i.e. a set of 3 points (each of them in a distinct part of the space corresponding to the class distribution).

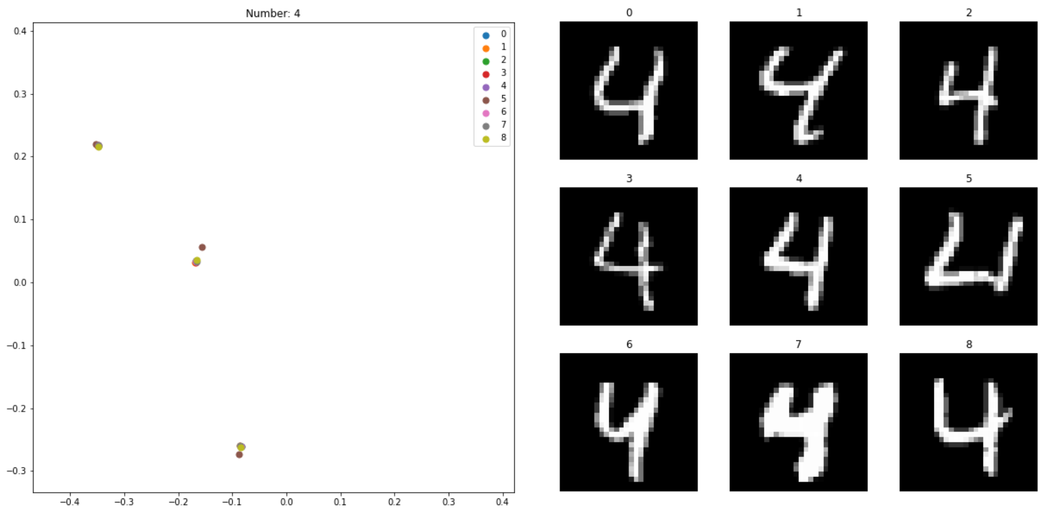

Figure 3: Embeddings visualization of 5 images all of which represent the number eight. Each color corresponds to an image, embedded as a point cloud (3 points). We notice that very different images still have the same point cloud representation in this 2D space.

In Figure 3, we plot embeddings of 9 different images, all of them representing the number four. Each color corresponds to a single image, embedded as a point cloud (3 points in the 2D space). We notice that very different images still have the same point cloud representation in the 2D space. Indeed, all point clouds have coordinates in the same subregions of the space. These 3 subregions, corresponding to the class distribution of the number four in the latent space, do include 9 points, one for each image. In addition, an embedding seems to differ a little bit from the others. This corresponds to the fifth image, which is a poorly written number.

Finally, we evaluate our embeddings on a downstream task: we train a K-Nearest Neighbor classifier using the embeddings generated by the model as features and the Euclidean distance (to simplify). The classifier reaches 99.41% accuracy on the test set, which is equivalent to the best models on the MNIST dataset. Even though MNIST is a very basic dataset, it is a baseline which demonstrate that our method could work on more complex images.

5 Conclusion

In conclusion, this paper proposes to embed data into a probability space and use an optimal transport metric as a distance. By adding a regularization term, we can compute this distance, called the Sinkhorn divergence, with a fixed number of differentiable iterations, hence easily coupled with ML pipelines. In the original paper, this method is applied on word and graph embeddings. In this review, we have shown that this can be also used in the context of visual embeddings, using the framework of siamese networks.

This application is very relevant because a lot of work in the computer vision community is now dedicated to unsupervised learning of visual features. And the main results (CPC, SimCLR) are obtained by models based on the basic idea of siamese networks, which embed data in the Euclidean space. We've proved that Sinkhorn embeddings perform at least as good as other embeddings methods on the MNIST dataset, it would be promising to apply this method to natural images, such as ImageNet.

Bibliography

Frogner et al. (2019) Learning embeddings into entropic Wasserstein spaces.