Large scale ordinal embedding: training neural networks with structure-free inputs

Leena C Vankadara [*1 2] **Siavash Haghiri** [*3] **Faiz Ul Wahab** [4] **Ulrike von Luxburg** [1]

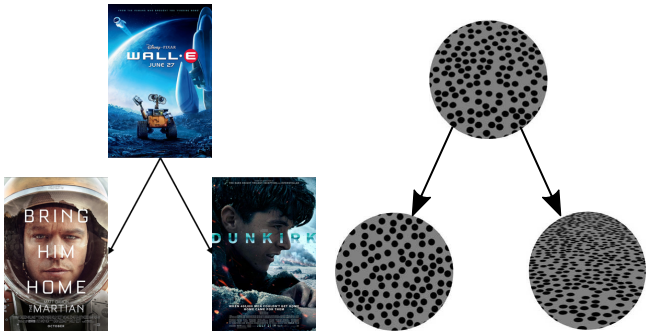
Abstract

In this paper, we discuss the fundamental problem of representation learning when no explicit representation of the input items (for example, RGB images) is accessible. All we are provided with are the answers to triplet comparisons of the following form: Is item A closer to item B than to item C? Existing approaches to this problem, which is also called ordinal embedding, are painfully slow and cannot embed more than an order of 1000 items in a reasonable amount of time. We use a feedforward network architecture as a basis of an ordinal embedding method that works on any given set of triplet comparisons. Our algorithm is significantly faster than the existing state of the art approaches and to date is the only approach that can scale to large real-world datasets. Our paper also features a somewhat unconventional way to use neural networks in a discrete setup: we do not use any input representation beyond the index of the item, yet achieve compelling results.

1. Introduction

We investigate the problem of representation learning when no meaningful input representation or explicit similarity information is available. Instead, we assume that we are provided with a set of triplets (x_i, x_j, x_k), which encode the following relationship: x_i is closer to x_j than to x_k. This problem is formally referred to as ordinal embedding(OE) (Kruskal, 1964; Agarwal et al., 2007; McFee & Lanckriet, 2009; Jamieson & Nowak, 2011; van der Maaten & Weinberger, 2012; Kleindessner & von Luxburg, 2014; Terada & von Luxburg, 2014; Jain et al., 2016a). The utility of such comparison based settings is ubiquitous in many machine learning applications.

Lack of meaningful input representation. The comparison-based setting can be particularly desirable when the items of interest are abstract and it is unclear how to obtain meaningful input representations or similarity values between pairs of items. For instance, consider learning a representation for a large set of movies. There does not exist an obvious informative low-level representation that can be used in a straight forward manner, e.g, as input to a neural network, yet one can extract triplet information relatively easily from user ratings. Such use-cases also arise in more scientific contexts. Consider, for instance, cancer medications with highly complex molecular structures. It is non-trivial to find vectorized representations for such structures. Moreover, if we do represent them with, say, graphs, it is still a cumbersome task to compare them and come up with accurate similarity values between them. However, in many cases, experts can make relative judgments fairly easily.



(a) (b)

Figure 1: Illustration of general triplet comparisons: Is the item on the top closer to the item on the left or to the item on the right?

Data gathering. The availability of abundant labeled data is paramount to the success of many modern machine learning algorithms and gathering such information can be highly inaccurate and/or expensive. Gathering triplet information, on the other hand, can be considerably easier and more accurate. There are studies, in various fields ranging from psychology to computer graphics, that suggest human beings are better in making relative judgments than providing quantitative (numerical) feedback (Miller, 1956; Stewart et al., 2005; Demiralp et al., 2014; Li et al., 2016). For

[*]Equal contribution [1]University of Tübingen, [2]International Max Planck research school for intelligent systems(IMPRS-IS), [3]Zalando SE, [4]Nyris GmbH,. Correspondence to: Leena C Vankadara <leena.chennuru-vankadara@uni-tuebingen.de>, Siavash Haghiri <siyavash.haghiri@gmail.com>.

instance, Figure 1a shows an example of a triplet question to gather information on the similarity of movies from users of a movie database. Likewise, Figure 1b depicts an example of a triplet question used in a psychophysical study (Wichmann & Jäkel, 2018).

Implicit triplets. Use-cases also arise in applications where deriving triplets from the available data is fairly straightforward but inferring an accurate similarity function is not. For instance, consider click-through rates of web-pages in a search engine (Schultz & Joachims, 2003). If three pages X, Y, Z appear in the search results and X, Y have, on average, higher click-through rates, it can be deduced that page X is more similar to Y than to Z.

Finally, qualitative comparisons can also be helpful in the context of privacy, where acquiring explicit representations of data may be undesirable. A whole sub-community is dedicated to the area of machine learning based on such qualitative comparisons (Agarwal et al., 2007; van der Maaten & Weinberger, 2012; Amid & Ukkonen, 2015; Ukkonen et al., 2015; Balcan et al., 2016; Haghiri et al., 2017; 2018; Kleindessner & von Luxburg, 2014; Kleindessner & Luxburg, 2015; Kleindessner & von Luxburg, 2017). One approach that enables the application of machine learning methods to such data is through ordinal embedding. The ordinal embedding procedure results in a Euclidean representation of items, which subsequently can be used as the input to traditional machine learning approaches.

Our contributions. We provide a new ordinal embedding method based on a 3-layer, feedforward neural network architecture. Our algorithm is significantly faster than the state of the art approaches and can scale to large real-world datasets. We can embed 0.6 million items or 600 million triplets in under 25 minutes and yet achieve high-quality embeddings (see Section 4). To put this in context, existing approaches can barely manage to embed $2,000$ points in under 30 minutes.

Our approach features a couple of unconventional ways of using neural networks. i) Since there is no input representation for the data, we merely use the index of the item as an input to the network. ii) In our approach, the role of the neural network is merely that of a non-convex optimization solver. We only care about embedding the training data as well as possible, generalization is not the primary goal. Using neural networks in this manner, particularly in unsupervised machine learning methods, is not the standard, and our approach could provide some direction to similar use cases of neural networks.

2. Background and related work

Assume we are given a set of items $X = \{x_1, x_2, \ldots, x_n\}$, with neither an explicit representation of the items nor a similarity function over pairs of items. Instead, we only see triplets $t = (i, j, k)$ which encode the relationship: "Item x_i is closer to item x_j than item x_k. Learning from such comparison-based data can be challenging. A fundamental approach used for analyzing such data is the **ordinal embedding** procedure.

Assuming that a set of triplet comparisons $T = \{t_1, t_2, \ldots t_m\}$ is given, the aim is to find a d-dimensional representation $y_1, y_2, \ldots y_n \in \mathbb{R}^d$ such that the Euclidean distances between these points satisfy as many triplets as possible. This can be expressed by the following optimization problem:

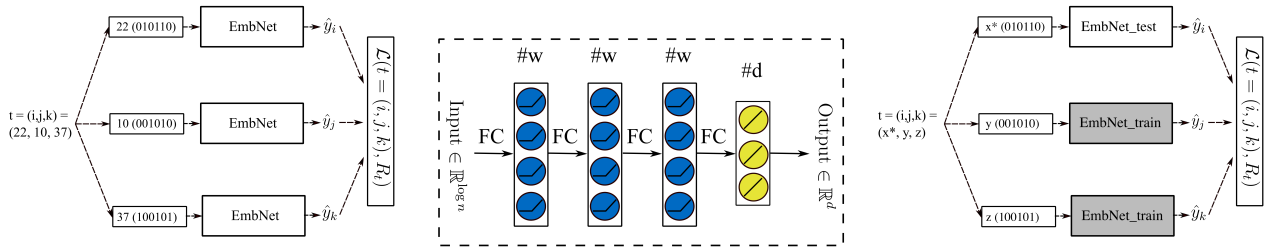
$$\min_{y_1,\ldots,y_n \in \mathbb{R}^d} \sum_{(i,j,k)\in T} \mathbb{1}_{\|y_i - y_j\|^2 > \|y_i - y_k\|^2}. \tag{1}$$

where $\mathbb{1}_E$ is the indicator function, which equals one, if the condition E is true and zero otherwise. The performance of ordinal embedding methods is typically evaluated by the *triplet error*. Given an estimated representation $\{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n\}$ of the data items and a set T of triplets, the triplet error (TE) simply counts how many triplets are satisfied by the given representation:

$$\text{TE} = \frac{1}{|T|} \sum_{(i,j,k)\in T} \mathbb{1}_{\|\hat{y}_i - \hat{y}_j\|^2 > \|\hat{y}_i - \hat{y}_k\|^2}. \tag{2}$$

In real world settings, it is typically not possible to satisfy all the input triplets: the underlying similarity might not be Euclidean or the triplets might be noisy. The above optimization problem is discrete, non-convex and NP-hard (Bower et al., 2018; Pardalos & Vavasis, 1991). There exist various methods that employ relaxations of this optimization problem: Generalized non-metric multidimensional embedding (GNMDS; Agarwal et al., 2007), stochastic triplet embedding (STE; van der Maaten & Weinberger, 2012) and local ordinal embedding (LOE; Terada & von Luxburg, 2014) are among the most popular methods in this category. However, they are computationally very demanding, and none of them can scale beyond a few thousand input points. The only somewhat scalable approach is the one by Anderton & Aslam (2019), based on explicit geometric constructions and landmarks. However, this approach requires an active access to triplet answers: rather than just receiving a bag of triplet answers, the algorithm needs to repeatedly ask very specific questions to an oracle. Our approach works in the more general passive setting.

Another notable area of related work is in the field of contrastive representation learning (Wang et al., 2014; Hoffer & Ailon, 2015; Cheng et al., 2016; Ge, 2018; Arora et al., 2019), where similarity information is provided in terms of contrastive triplets of points (x, x^+, x^-): for a given point x, the point x^+ and x^- are specifically chosen data points that are similar/dissimilar to x. Such approaches



(a) (b) (c)

Figure 2: (a) The architecture of Ordinal Embedding Neural Network (OENN). As example a triplet (22, 10, 37) and its answer R_t are fed to the architecture. (b) The EmbNet neural network, which is used as a building blocks of ordinal embedding architecture. (c) Inference architecture - x^* denotes an item from the test set, y, z denote items from the train set. The parameters of $\text{EmbNet}_{\text{train}}$are frozen.

have been extremely successful for image and text data, and they are particularly elegant if the similarity information can be extracted from the raw data in some unsupervised manner. Examples are the case of word embeddings (Mikolov et al., 2013), where words are considered similar if they occur within the same local neighborhood, or text representations (Logeswaran & Lee, 2018), where two subsequent sentences are considered similar, or in computer vision (Wang & Gupta, 2015) where pairs of image patches in subsequent frames of videos are considered similar.

3. Proposed method: ordinal embedding neural network (OENN)

Our proposed architecture is inspired by the recent line of work on contrastive learning (Wang et al., 2014; Schroff et al., 2015; Cheng et al., 2016; Hoffer & Ailon, 2015). Figure 2a shows a sketch of our proposed network architecture. The central sub-module of our architecture is what we call the embedding network (***EmbNet***): one such network takes a certain encoding of a single data point x_i as input (typically, an encoding of its index i, see below) and outputs a d-dimensional representation $\hat{y_i}$ of data point x_i. The EmbNet is replicated three times with shared parameters. The overall OENN network now takes the ***indices*** (i, j, k) corresponding to a triplet (x_i, x_j, x_k) as an input. It routes each of the indices i, j, k to one of the copies of the EmbNet, which then return the d-dimensional representations $\hat{y_i}, \hat{y_j}, \hat{y_k}$, respectively (cf. Figure 2a). The three sub-modules are trained jointly using the triplet hinge loss, as described by the following objective function: $\mathcal{L}(T) =$

$$\frac{1}{|T|} \sum_{(i,j,k)\in T} \max\left\{\|\hat{y_i} - \hat{y_j}\|^2 - \|\hat{y_i} - \hat{y_k}\|^2 + 1, 0\right\} \quad (3)$$

Note that this optimization problem is not a relaxation to the OE problem. Rather, its an equivalent one (Bower et al., 2018). Meaning that every global optima of this optimization problem is a feasible solution to the ordinal embedding problem and vice versa (up to a scale).

On the choice of input representation: Since we do not have access to any informative low-level input representations, the choice of input representations presents a challenge to this approach. However, we leverage the expressive power of neural networks (Leshno et al., 1993; Barron, 1993) and their ability to fit random labels to random inputs (Zhang et al., 2016) to motivate our choice of input encoding. Since our main goal is to find representations that minimize the training objective, we believe that completely arbitrary input representations are a viable choice.

One such input representation could be the one-hot encoding of the index (where point i is encoded by a string $\hat{x}_i$ of length n such that $\hat{x}_i(l) = 1$ if $l = i$ and 0 otherwise). The advantage of choosing such a representation is that it is memory efficient in the sense that there is no need to additionally store the representations of the items. However, under this choice of representation the length of the input vectors grows linearly with the number n of input items. As one of the central contributions of our work is to perform ordinal embedding in large scales, we consider a more efficient way: we represent each item by the binary code of its index, leading to a representation length of $\log n$. Such a representation retains the memory efficiency of the one-hot encoding (in the sense as discussed above) but improves the length of the input representation from n to $\log n$. As we will see below, this representation works well in practice. However, note that there is nothing peculiar about this choice of binary code. Our simulations (see Subsection 4.1) suggest that unique representations for items generated uniformly, randomly from a unit cube in $\mathbb{R}^{\alpha=\Omega(\log n)}$ can be used as the input encoding.

Structure of EmbNet: Figure 2b shows the schematic of the EmbNet. We propose a simple network with three fully-connected hidden layers and ReLu activation function. The final layer is a linear layer that takes the output of third hidden layer and produces the output embedding. The input

	t-STE			LOE			OENN		
	TE_{train}	TE_{test}	Time	TE_{train}	TE_{test}	Time	TE_{train}	TE_{test}	Time
MNIST($\sim$ 1K)	0.16	0.17	2000	0.003	0.07	5500	0.05	0.07	36
CHAR ($\sim$ 1.8K)	0.15	0.16	1600	-	-	-	0.02	0.04	400
USPS ($\sim$ 8K)	0.14	0.14	14000	-	-	-	0.04	0.05	1500
Uniform ($\sim$ 10K)	0.04	0.07	3000	0	0.04	4500	0.04	0.06	300
FMNIST ($\sim$ 60K)	-	-	-	-	-	-	0.07	0.08	1600
Covtype ($\sim$ 0.6M)	-	-	-	-	-	-	0.03	0.03	1500

Table 1: Comparison of TE_{train} and TE_{test} and training time Time across t-STE, LOE and OENN for all datasets. A run-time limit of 4 hours is set for all the experiments.

size to the network is $\lceil \log n \rceil$ and each hidden layer contains w nodes. The output layer has d nodes to produce embeddings in $\mathbb{R}^d$. The input and output size, $\lceil \log n \rceil$ and d, are pre-determined by the task. Thus the only independent parameter of the network is the width w of hidden layers. Our experiments (see Subsection 4.1) demonstrate that the hidden layer-width w should grow logarithmically to the number of items n to produce good embedding outputs.

Generalization to new data points: While the primary goal of embedding methods is to find a representation of the given training set, it is sometimes desirable to be able to extend the representation to new data points. A naive approach would be to re-train the OENN with the appropriate parameter settings (see Section 4.1) with the combined set of training and test triplets. However, our approach features an elegant way to infer embeddings of a set of test items. Let $X' = \{x_1^*, \cdots x_k^*\}$ denote a set of test items observed via a set of triplets T' of the form (x, y, z) where at least one of the items in each triplet comes from X' and the rest could arise either from X or X'. Such a triplet encodes the usual triplet relationship: x is closer to y than z.

Our inference procedure (shown in Figure 2c) to obtain embeddings from T' is as follows. Recall that our architecture is composed of 3 identical EmbNet components with shared weights. Consider a OENN trained on T and denote its (trained) EmbNet component as EmbNet*. We define $\text{EmbNet}_{\text{test}}$ to be an instance of EmbNet* with randomly re-initialized weights and $\text{EmbNet}_{\text{train}}$ to be an instance of EmbNet* with weights un-touched. The inference architecture, is then constructed dynamically for any triplet from T' in the following manner: items from X' in the triplet are provided as inputs to $\text{EmbNet}_{\text{test}}$, while items from X are provided as inputs to $\text{EmbNet}_{\text{train}}$. We then proceed to train the network in the same fashion as before over T' except that we freeze the parameters of $\text{EmbNet}_{\text{train}}$. Note that the training examples are unaffected in the inference procedure. The procedure produces embeddings of test items that satisfy the triplet constraints containing items both from X and X'. Our experiments indicate that the quality of embeddings provided by this procedure is quite satisfactory. We evaluate this using the triplet error. Since, the generalization performance of our method is tangential to the central arguments of the paper, due to space constraints, we present the experimental setup and the results of this experiment in the supplementary.

Difference to previous contrastive learning approaches: As described earlier, our architecture is inspired by Wang et al. (2014); Hoffer & Ailon (2015). However, there are fundamental differences: 1) we have no access to representations for the input items $x_1, .., x_n$ and our network takes completely arbitrary representations for the input items. 2) In the previous work, triplets are always contrastive, which is of the form (x, x^+, x^-) while simply gather triplets involving arbitrary sets of three points. Indeed, obtaining contrastive triplets requires additional information, either the class labels or more explicit similarity information between objects.

4. Experiments

We run an extensive set of simulations, on the one hand, to find good rules of thumb to determine the parameters of the network architecture and the input encoding, and on the other hand to evaluate the performance of OENN in terms of triplet error and compare it to alternative approaches.

Note: It is important to remember that, in all of our experiments, the input representation of the data or the actual distances between the data points is **never** used for training. Moreover, we also **never** use the label information of any dataset in either training or the triplet generation process. The labels, when available, are only used for visualization.

4.1. Choice of parameters and hyper-parameters

Our network architecture depends on a few parameters: the number of layers l, the width of the hidden layers w, length of the input encoding α and the dimension of the embedding space d. To reduce the number of independent parameters of the network, we simply fix the number of layers to 3,

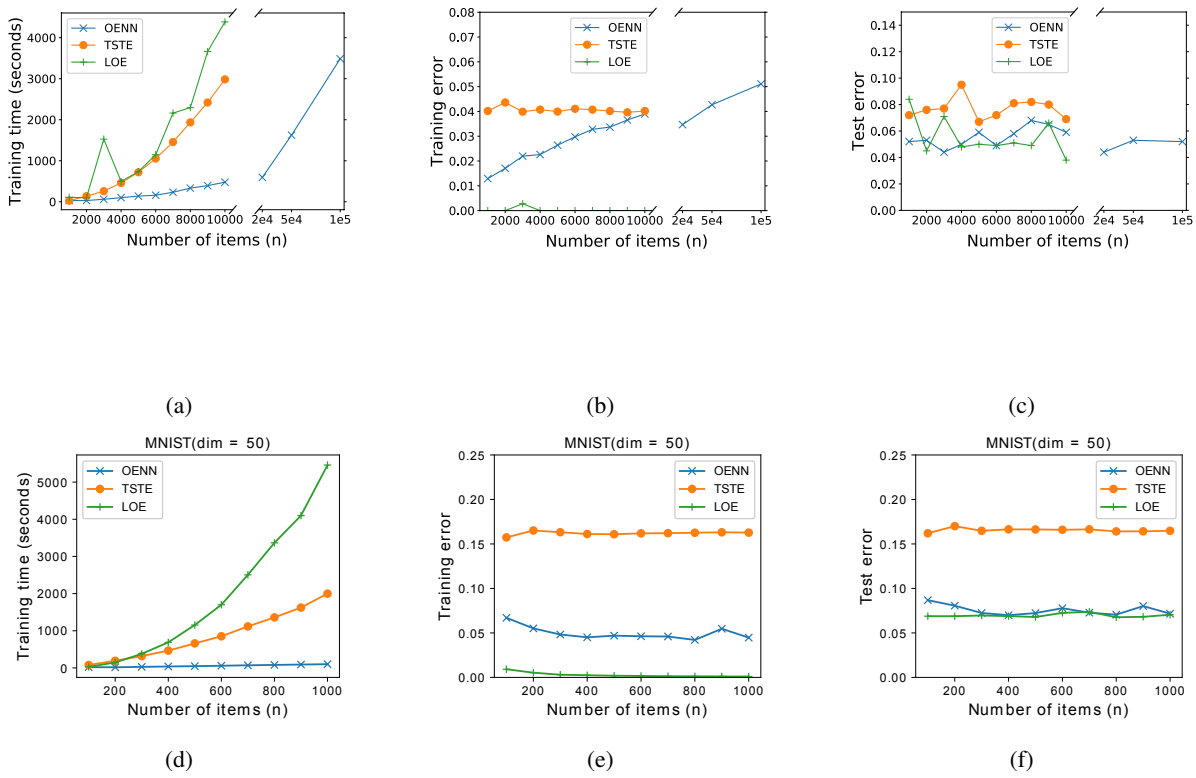


Figure 3: First row: the performance comparison of three methods on the two-dimensional uniform dataset (a) Training time (embedding time) of the three methods in seconds. LOE and t-STE did not scale to data sets larger than 10000 points. Note that the right part corresponds to the extended range of items. (b) Training triplet error with varying number of items for three different ordinal embedding methods. (c) Test triplet error for varying number n of items. In addition, the x-axis on the right part is exponential, thus the slope of the curve is not comparable to the left part of the plot. Second row: the performance comparison of the three methods on a random subsample of MNIST dataset. The embedding dimensions are chosen as $d = 50$ for all methods. (d) Training time (embedding time) of the three methods in seconds. (e) Training triplet error with varying number of items for three different ordinal embedding methods. (f) Test triplet error for varying number n of items.

where the number of units is the same in each of these layers. The embedding dimension is determined by the given task. Additionally, the training procedure requires the setting of a few hyper-parameters: choice of the optimizer, batch size, and learning rate. We use ADAM Kingma & Ba (2014) to train our network and use a batch size of $\min(\#\text{ triplets}, 50,000)$ and a learning rate of 0.005 for all the experiments.

On the width of the hidden layers w: The width w of the hidden layers depends on the input dimension d and the number of items n. To investigate this dependence we use toy datasets where the dimension of the input space could be controlled. We ran simulations on 5 different datasets, specifically, we used the d principal components of MNIST(Lecun & Cortes, 1998), USPS(Hull, 1994), CHAR(Dua & Graff, 2017), Mixture of Gaussian in $\mathbb{R}^d$ and data sampled uniformly from a unit cube in $\mathbb{R}^d$.

We perform two sets of experiments. First, we fix the number of points and generate datasets in an increasing number of dimensions, d. We generate random triples from these datasets and use our model with increasing w to embed them back into $\mathbb{R}^d$. Our simulations demonstrate that the width of the hidden layers needs to grow linearly with the embedding dimension. Similarly, fixing the input and the embedding dimension, our simulations demonstrate that w needs to grow logarithmically with n. Therefore, in all of our experiments, we set,

$$w = \max(120, 2d \log n).$$

The detailed experimental setups, as well as the results from

these experiments, are provided in the supplementary.

On the choice of the length of input encoding α. The input to the OENN, as described earlier, can be chosen as a triplet of arbitrary encoding of the items. We ran simulations to establish the dependence of the length of such encoding with the number of items. These simulations show that the size of the input encoding needs to grow logarithmically with the number of items. In all the experiments, we set,

$$\alpha = \lceil \log n \rceil.$$

For more details on the parameters of our simulations and the corresponding plots, please refer to our supplementary.

On the choice of number of triplets m. The number of input triplets to the embedding algorithm in our experiments is always chosen around

$$m = nd \log n$$

which is a known theoretical lower bound on the number of active triplets required to reconstruct the original embedding up to similarity transformations (Jain et al., 2016b). Note that if we used less number of triplets, then the problem of ordinal embedding would be highly under-determined and would typically not lead to a unique solution. We illustrate this both visually as well as by measuring the generalization triplet error of embeddings' output by the OENN that is trained with decreasing number of triplets. The results of this experiment are included in the supplementary.

4.2. Experiments on scalability

Choice of datasets: To demonstrate the scalability of our approach, we report the results of our experiments conducted on 5 real-world datasets and one synthetic dataset. Specifically, we use data points randomly sampled from MNIST ($\sim 1K$), CHAR ($\sim 1.7K$), USPS ($\sim 8K$), FashionMNIST ($\sim 60K$, Xiao et al. (2017)), Covtype ($\sim 600K$, Dua & Graff (2017)), data points randomly, uniformly sampled from a unit cube in $\mathbb{R}^2$ ($\sim 10K$). These datasets vary between $1,000$ to 0.6 million items and between 2 to 784 dimensions across real and synthetic datasets.

Algorithms. We compare the performance of the proposed OENN method with two competitors, namely local ordinal embedding (LOE) (Terada & von Luxburg, 2014) and t-distributed stochastic triplet embedding (t-STE) (van der Maaten & Weinberger, 2012). The optimization parameters of LOE and t-STE are set to the default values. We use the author's implementation of LOE in R[1], and the Python implementation of t-STE[2]. Note that the Python implementation is consistently faster than the author's MATLAB implementation.

[1]https://cran.r-project.org/web/packages/loe/index.html

[2]https://github.com/gcr/t-STE-theano

Hardware. All experiments are performed on the same machine with Intel XEON CPU E5-2620 processor, with 4 GeForce GTX 1080-Ti GPU's and 20 GB RAM. The proposed OENN method takes advantage of fast GPU computations, while the other methods run solely on the CPU.

Embedding dimension d. For datasets in $\mathbb{R}^2$, the embedding dimension is chosen as $\mathbb{R}^2$. For all the high dimensional datasets except MNIST, the embedding dimension is chosen as 25. For MNIST, since it is our baseline dataset on which we run experiments with high dimensional embedding space, the embedding dimension is chosen as 50.

Triplet generation. For each dataset, we generate around $nd \log n$ triplets based on Euclidean distance, where d refers to the dimension of the embedding space. To generate a triplet, we sample 3 items x_i, x_j, x_k uniformly from the dataset and evaluate if $d(x_i, x_j) < d(x_i, x_k)$ or vice versa. Note that we do not consider an active setting where the learning algorithm has access to an oracle that answers specific triplet questions (as studied in Anderton & Aslam, 2019).

Test triplet error (TE_{test}). It measures the triplet error on a set of β triplets drawn independent of the set of training triplets. β is heuristically chosen as $10K$ for toy, $2D$ datasets and $100K$ otherwise.

Scalability of OENN. For each dataset, we use t-STE, LOE and OENN (our approach) to find embeddings of the items that satisfy the set of generated triplets. We limit the training time of the experiments to 4 hours. In Table 1, For each experiment, we report the training triplet error TE_{train}, training time as Time and the test triplet error TE_{test}.

As shown in Table 1, our approach is significantly faster than LOE and t-STE. Notably, the Covertype dataset consists of around 0.6 million items and we generate $2nd \log n \approx 600$ million triplets based on Euclidean data. OENN takes ≈ 1500 seconds to achieve a test triplet error of 0.03. Considering that the other approaches cannot perform ordinal embedding for more than an order of 1000 points in a reasonable time, it is clear that our approach offers a hugely significant computational advantage over the existing approaches.

Training time vs n.

We perform two simulations to observe the relationship between the training time and the number of items for all three approaches. In the first experiment, the items are sampled from a uniform distribution on the unit square in $\mathbb{R}^2$. We compare running times for the number of items in the range $n \in \{1000, 2000, \ldots, 10000\}$. Because the running time for LOE and t-STE grows very quickly with n, we could not run these two algorithms with larger n, while our method is still tractable. Therefore, we run our method alone on an

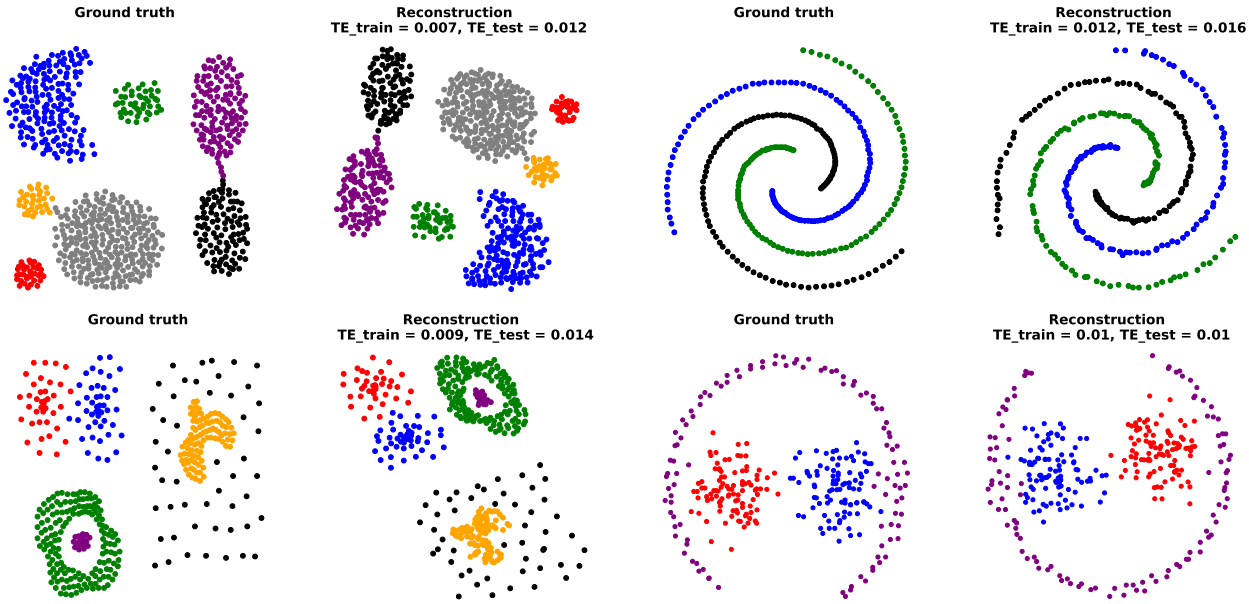


Figure 4: In each block, the image on the left corresponds to the true embedding and the one on the right corresponds to the embedding output of OENN trained using triplets. It can be seen that the embedding output closely matches the ground truth.

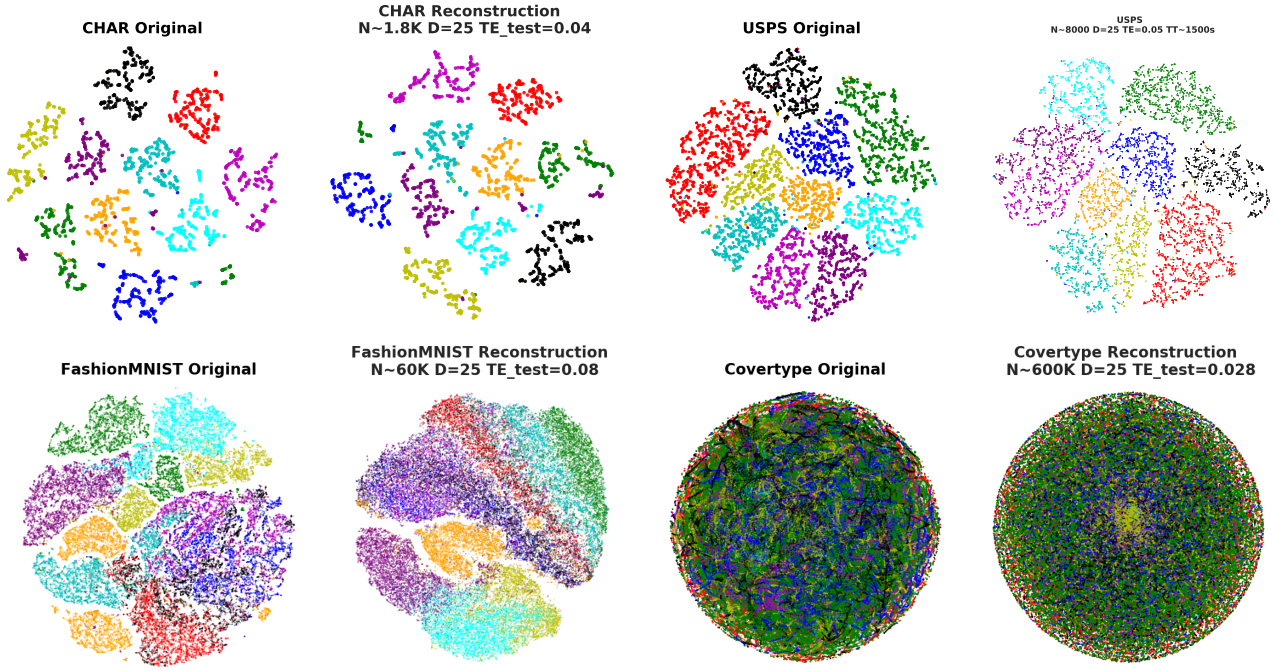


Figure 5: In each block, the image on the left corresponds to a 2D visualization of the original dataset and the one on the right corresponds to a 2D visualization of the embedding output of OENN trained using triplets. On top of this image, we report the number of items in each dataset (n), the number of dimensions (Dim) the dataset is embedded into using OENN, and the test triplet error, (TE_{test}) of the corresponding experiment. The colors encode the label information (only used for visualization). In each of the cases, the embeddings output by the OENN faithfully respect the structure present in the original dataset.

extended range of items: $n \in \{20000, 50000, 100000\}$. We generate $nd \log n$ triplets which are fed to the ordinal embedding algorithms as input. The methods are compared based on three evaluation criteria: training triplet error (TE_{train}),

test triplet error (TE_{test}), and embedding time in seconds.

We depicted the training triplet error and test triplet error of the methods in Figure 3b and Figure 3c respectively. LOE can perfectly fit the embedding to the input triplet answers, thus we observe almost zero training error. Our method (OENN) and t-STE both perform reasonably well, showing less than 5% error on the training set. On the test set, all three methods have a similar performance. The difference becomes apparent when we turn to embed time of the three methods, see Figure 3a. As the number of items grows, the traditional methods (LOE and t-STE) become painfully slow. In the case of $n = 10000$, traditional methods require about 1 hour to obtain the embedding, while the OENN requires about 5 minutes. The slope of the curve already shows that ordinal embedding in large scales is essentially impossible with the traditional methods.

The second experiment aims to show the relation of training time and n in higher-dimensional data. We used a subsample of MNIST with $n \in \{100, 200, \ldots, 1000\}$ data points. We fix the embedding dimension $d = 50$, and $nd \log n$ triplets are generated. We report the same three evaluation criteria in Figure 3d,3e,3f. Similarly, in the case of high dimensional data, the conventional methods require significantly more time to obtain the embedding. Moreover, the triplet errors show that t-STE performs poorly for high dimensional data.

4.3. Reconstructions

Our previous results demonstrate that our approach is significantly faster than the state of the art methods for OE. In this section, we show that the quality of the embedding generated by OENN is also quite high.

Experiments on synthetic data. In order to validate the ability of OENN to reconstruct the true embedding[3], we first use several 2-dimensional toy datasets, generate triplets as usual and use our algorithm to embed the triplets back into $\mathbb{R}^2$. We depict the original dataset and the corresponding embedding output by our proposed algorithm in Figure 4. Recall that the algorithm neither observes the ground truth data nor the magnitude of distances between the input points. Considering that the algorithm only has access to relative distance comparisons in the form of triplets, it is quite remarkable to see that, in all of the datasets, the visual similarity between the output embedding and the ground truth is very high up to similarity transformations. In addition, we also report the training error (TE_{train}) and test error (TE_{test}) of the embedding in Figure 4. The training error and the test error are always close to 0.05, showing that the algorithm is able to reconstruct the original embedding.

[3]It has been shown that an ordinal embedding can be uniquely determined up to an isometric transform — if the items are sampled from a compact subset of the Euclidean space and the number of items grows to infinity (Kleindessner & von Luxburg, 2014)

Figure 6: 60 images depicted on their actual embedded location in two dimensions. 20 images from each concept are chosen at random.

Visualization of embeddings (real-world data). Consider the embeddings generated by OENN on real-world datasets. Note that to obtain a low triplet error, one clearly needs to choose an embedding dimension that is larger than 2 for most high-dimensional datasets. To demonstrate that the embeddings output by OENN faithfully preserve the structure present in the original data, we hence resort to visualizing both the original dataset as well as the obtained embeddings in 2 dimensions using the t-SNE algorithm (van der Maaten & Hinton, 2008). Visualizations corresponding to 4 of the datasets are provided in Figure 5. The rest appear in the supplementary. In each of the dataset, observe that t-SNE visualizations of the original dataset and that of the embedding produced by OENN closely resemble each other. This is also quantitatively shown by the test triplet error in Figure 5. Our approach trained only using the triplet information achieves a faithful representation of the original data.

4.4. Crowd-sourcing experiment

In this section, we perform ordinal embedding on a dataset of triplets generated via crowd-sourcing. We ran a study on Amazon's mechanical Turk (MTurk) platform. The study aims to collect answers to triplet comparisons of natural images. We used a subsample of the Imagenet dataset (Deng et al., 2009) as a basis of our experiment. We chose the three high-level concepts - animals, food, and vehicles. We then gathered 120,000 triplet answers from the MTurk workers, which is somewhat less than the $nd \log n \approx 190,000$ triplets that would be required for high-quality recovery according to the theoretical lower bound for an embedding in $d = 2$ dimensions (Jain et al., 2016b). We show the

2-dimensional embedding produced by OENN for a subsample of 60 images in Figure 6 (to generate the figure, we randomly chose 20 images from each of the three concepts). We can see that three concepts can be fairly well distinguished from each other.

5. Discussion

Over the last decade, the ordinal embedding problem has been analyzed exhaustively, and a number of methods have been proposed. However, ordinal embedding for more than thousands of items has been beyond the reach of algorithms. In this paper, we propose a novel, scalable ordinal embedding algorithm that features the use of neural networks for structure-free inputs. Our proposed method shows an outstanding improvement over the state of the art in terms of computation time. It is the first method that can work with passively collected triplet answers and scale to large real-world datasets.

The singular advantage of OENN lies in its ability to scale. However, note that the number of network parameters in our model scales as $\Omega((d \log n)^2)$, this could potentially affect scalability in high dimensional embedding spaces. It is not necessarily a downside of our particular method since the hardness of ordinal embedding surely increases with the number of dimensions as also demonstrated in Figure 3.

In addition to desirable performance for the ordinal embedding task, our approach is novel in the way that it utilizes neural networks. First, The input to the network solely contains the index of items. Secondly, the network is used to only to overfit to the input triplets. We believe that our work can provide some direction to solve similar optimization problems with neural networks.

References

Agarwal, S., Wills, J., Cayton, L., Lanckriet, G., Kriegman, D., and Belongie, S. Generalized non-metric multidimensional scaling. In *AISTATS*, pp. 11–18, 2007.

Amid, E. and Ukkonen, A. Multiview triplet embedding: Learning attributes in multiple maps. In *ICML*, pp. 1472–1480, 2015.

Anderton, J. and Aslam, J. Scaling up ordinal embedding: A landmark approach. In *ICML*, pp. 282–290, 2019.

Arora, S., Khandeparkar, H., Khodak, M., Plevrakis, O., and Saunshi, N. A theoretical analysis of contrastive unsupervised representation learning. *arXiv preprint arXiv:1902.09229*, 2019.

Balcan, M., Vitercik, E., and White, C. Learning combinatorial functions from pairwise comparisons. In *COLT*, 2016.

Barron, A. R. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, 1993.

Bower, A., Jain, L., and Balzano, L. The landscape of non-convex quadratic feasibility. In *ICASSP*, 2018.

Cheng, D., Gong, Y., Zhou, S., Wang, J., and Zheng, N. Person re-identification by multi-channel parts-based cnn with improved triplet loss function. In *CVPR*, pp. 1335–1344, 2016.

Demiralp, Ç., Bernstein, M. S., and Heer, J. Learning perceptual kernels for visualization design. *IEEE transactions on visualization and computer graphics*, 20(12): 1933–1942, 2014.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, 2009.

Dua, D. and Graff, C. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Ge, W. Deep metric learning with hierarchical triplet loss. In *ECCV*, pp. 269–285, 2018.

Haghiri, S., Ghoshdastidar, D., and von Luxburg, U. Comparison-based nearest neighbor search. In *AISTATS*, pp. 851–859, 2017.

Haghiri, S., Garreau, D., and Luxburg, U. Comparison-based random forests. In *ICML*, pp. 1866–1875, 2018.

Hoffer, E. and Ailon, N. Deep metric learning using triplet network. In *International Workshop on Similarity-Based Pattern Recognition*, pp. 84–92. Springer, 2015.

Hull, J. J. A database for handwritten text recognition research. *IEEE Transactions on pattern analysis and machine intelligence*, 16(5):550–554, 1994.

Jain, L., Jamieson, K. G., and Nowak, R. Finite sample prediction and recovery bounds for ordinal embedding. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 2711–2719, 2016a.

Jain, L., Jamieson, K. G., and Nowak, R. Finite sample prediction and recovery bounds for ordinal embedding. In *NeurIPS*, 2016b.

Jamieson, K. G. and Nowak, R. D. Low-dimensional embedding using adaptively selected ordinal data. In *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 1077–1084, 2011.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kleindessner, M. and Luxburg, U. Dimensionality estimation without distances. In *AISTATS*, pp. 471–479, 2015.

Kleindessner, M. and von Luxburg, U. Uniqueness of ordinal embedding. In *COLT*, 2014.

Kleindessner, M. and von Luxburg, U. Kernel functions based on triplet comparisons. In *Advances in Neural Information Processing Systems*, pp. 6807–6817, 2017.

Kruskal, J. B. Nonmetric multidimensional scaling: A numerical method. *Psychometrika*, 29(2):115–129, 1964.

Lecun, Y. and Cortes, C. The MNIST database of handwritten digits, 1998. URL `http://yann.lecun.com/exdb/mnist/`.

Leshno, M., Lin, V. Y., Pinkus, A., and Schocken, S. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

Li, L., Malave, V. L., Song, A., and Yu, A. Extracting human face similarity judgments: Pairs or triplets? In *CogSci*, 2016.

Logeswaran, L. and Lee, H. An efficient framework for learning sentence representations. *arXiv preprint arXiv:1803.02893*, 2018.

McFee, B. and Lanckriet, G. Partial order embedding with multiple kernels. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 721–728, 2009.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In *NeurIPS*, pp. 3111–3119, 2013.

Miller, G. A. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956.

Pardalos, P. M. and Vavasis, S. A. Quadratic programming with one negative eigenvalue is np-hard. *Journal of Global Optimization*, 1(1):15–22, 1991.

Schroff, F., Kalenichenko, D., and Philbin, J. Facenet: A unified embedding for face recognition and clustering. In *CVPR*, pp. 815–823, 2015.

Schultz, M. and Joachims, T. Learning a distance metric from relative comparisons. In *Advances in Neural Information Processing Systems (NIPS)*, 2003.

Stewart, N., Brown, G. D., and Chater, N. Absolute identification by relative judgment. *Psychological review*, 112 (4):881, 2005.

Terada, Y. and von Luxburg, U. Local ordinal embedding. In *ICML*, pp. 847–855, 2014.

Ukkonen, A., Derakhshan, B., and Heikinheimo, H. Crowdsourced nonparametric density estimation using relative distances. In *HCOMP*, 2015.

van der Maaten, L. and Hinton, G. Visualizing data using t-SNE. *JMLR*, 9:2579–2605, 2008.

van der Maaten, L. and Weinberger, K. Stochastic triplet embedding. In *MLSP*, 2012. Code available on `http://homepage.tudelft.nl/19j49/ste`.

Wang, J., Song, Y., Leung, T., Rosenberg, C., Wang, J., Philbin, J., Chen, B., and Wu, Y. Learning fine-grained image similarity with deep ranking. In *CVPR*, pp. 1386–1393, 2014.

Wang, X. and Gupta, A. Unsupervised learning of visual representations using videos. In *ICCV*, pp. 2794–2802, 2015.

Wichmann, F. A. and Jäkel, F. Methods in psychophysics. In *The Stevens' Handbook of Experimental Psychology and Cognitive Neuroscience.*, volume V. Methodology. Wiley, 4th edition, 2018.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding deep learning requires rethinking generalization. *ICLR*, 2016.

A. On the choice of the loss function

In our method, we use the hinge loss as the choice of our loss function. In what follows, we justify this choice by showing that the resulting optimization by using the hinge-loss does not constitute a relaxation to the ordinal embedding problem. Rather it's an equivalent one.

The problem of ordinal embedding — finding an embedding $X = \{x_1, x_2, .., x_n\} \in \mathbb{R}^d$ that satisfies a set of given triplets, $\mathcal{T}$ - can be phrased as a quadratic feasibility problem (Bower et al., 2018) as shown in Equation 4.

$$\text{find } X \text{ subject to } X^T P_{i,j,k} X > 0 \; \forall (i,j,k) \in \mathcal{T}. \quad (4)$$

Each matrix $P_{i,j,k}$ corresponds to a triplet constraint that satisfies,

$$||x_i - x_j||^2 > ||x_i - x_k||^2 \iff X^T P_{i,j,k} X > 0$$

Every feasible solution to 4 is a valid solution to the problem of ordinal embedding.

An equivalent way to solve equation 4 (i.e., find a feasible solution of 4) is to find the global optima of the constrained optimization problem (Bower et al., 2018) given by equation 5:

$$\min_{X \in \mathbb{R}^{nd}} \sum_{(i,j,k) \in \mathcal{T}} \max\left\{0, 1 - X^T P_{i,j,k} X\right\} \quad (5)$$

This it true because every feasible solution to (4) can be scaled to attain global optima of (5) and every global optima of (5) is a feasible solution of (4) (Bower et al., 2018). Moreover, in optimization (1), any positive scaling of a feasible point X is a solution to (1) as well, whereas in optimization (2) this effect is eliminated.

The hinge loss, therefore, satisfies the nice property that using the hinge loss to solve the ordinal embedding problem is not a relaxation but rather an equivalent problem.

B. Generalization to new items

In this section, we present the performance of our method on out of sample extensions. To conduct this experiment, we choose the datasets USPS and CHAR. The USPS dataset has already a pre-defined training and test set. In the case of CHAR, we create the test set by randomly splitting the data into a train (80%) and a test set (20%). Subsequently, we generate $2nd \log n$ triplets by drawing points uniformly at random from the train set $X = \{x_1, x_2, \ldots, x_n\}$ of items. These triplets are then used to train the training network $\text{OENN}_{\text{train}}$. Now we are going to study the out-of-sample extension. To this end, we randomly generate 2 million triplets of items (x_i^*, x_j, x_k), where x_i^* is always chosen from the test set $X^* = \{x_1^*, x_2^*, \ldots, x_k^*\}$ and x_j, x_k are drawn from the train set. The input representation of an item x_i^* from the test set is chosen as the binary encoding of the index i in length $\lceil \log n \rceil$. The input representations for an item x_j from the train set is chosen as the binary encoding of the index j in length $\lceil \log n \rceil$. Note that, the same indices are allowed to appear in the test set as well as the training set since different EmbNet components are used to generate the embeddings of items from the train set and the test set. Let T^* denote the set of all triplets containing items from X^*. The generalization network $\text{OENN}_{\text{test}}$ is then trained on T^* (while keeping $\text{OENN}_{\text{train}}$ fixed as explained in the main paper). The results are demonstrated in Figure 8 by reporting both the fraction of triplets in T^* that are not satisfied by $\text{OENN}_{\text{test}}$ and also by visualizing the output embeddings of both the training data and the test data in 2 dimensions. The t-SNE algorithm is used to obtain the 2D visualization of the combined set of embeddings of both the train and the test items. The results clearly show that, in addition to satisfying all the triplets in T^*, the quality of the obtained embeddings is high: the train and test embeddings look pretty much the same, and the triplet error is very low.

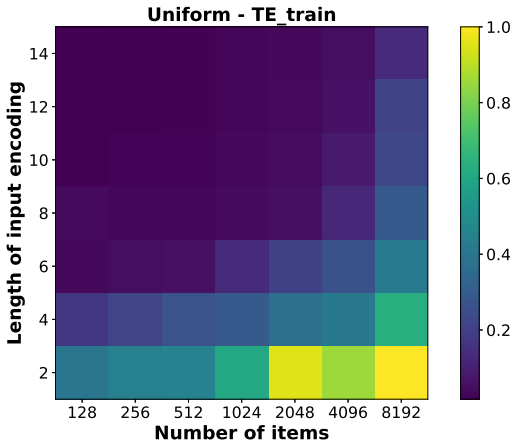


Figure 7: Triplet error (encoded by the heat map) with varying number of items and length of the input encoding. The x and y axes correspond to the number of items (n) and the length of the input encoding (q) respectively. Note that $x-$axes grows exponentially, while the $y-$axes increases linearly. The color of the heat map represents the triplet error obtained in the training procedure.

C. On the width of the hidden layers w

In this section, we describe the setup and the results of the simulations conducted to determine the width of the hidden layers w. To conduct the experiments, we use datasets where the dimension of the input space could be controlled. We run simulations on 5 different datasets: MNIST, USPS, and CHAR projected onto their d principal components, a Mix-

ture of two Gaussians in $\mathbb{R}^d$, and data sampled uniformly from a unit cube in $\mathbb{R}^d$.

C.1. Dependence of the layer width w on the number n of items

Recall that in our setting, the neural network is used to solve a memorization task. The goal of the network is to fit completely arbitrary inputs — the index of the item — to outputs in $\mathbb{R}^d$. Therefore, it is reasonable to expect that with an increasing number n of items, a larger network is required to address the complexity of the task. We can either increase the representational power of the network by adding more layers or by increasing the width of layers. We fix the number of layers to 3 and investigate the dependence of layer width w on n.

For each dataset, we choose an exponentially increasing number of items (n) in $\mathbb{R}^2$. Given a sample of n points, we generate $nd \log n$ triplets (where $d = 2$ is the embedding dimension). The embedding network (EmbNet) is constructed with 3 hidden layers, each having w fully-connected neurons with ReLU activation functions. The width w of the layers (= number of neurons in each layer) is increased linearly. For a fixed network and a fixed set of triplets, the experiment is executed 5 times in order to examine the average behavior of the model.

Figure 9 shows the training triplet error (TE_{train}) of the ordinal embedding with varying number of items and width of the hidden layers for 3 different datasets. The error is reported by means of heat-maps, where warmer colors denote higher triplet error. Note that the $x-$axis increases exponentially and the $y-$axis increases linearly. The plots clearly establish logarithmic dependence of the hidden layer width on the number of items n.

C.2. Dependence of the layer width w on the embedding dimension d

Besides the number of items, the embedding dimension is another factor that influences the complexity of ordinal embedding. We expect that the required layer width needs to grow with the embedding dimension. To investigate this dependence, as earlier, we sample $n = 1024$ items in $\mathbb{R}^d$ from each dataset with varying input dimension d. We generate $nd \log n$ triplets based on the Euclidean distances of items. To construct a network, the width w of the layers is chosen from a linearly increasing set. For a fixed network and a fixed set of triplets, the experiment is executed 5 times in order to examine the average behavior of the model.

Figure 10 shows the training triplet error (TE_{train}) of the ordinal embedding with varying dimensions and width of the hidden layers for 3 different datasets. There is again a clear line of transition between low and high error regions. In all of the datasets, it can be observed that the layer width has a linear dependence on the embedding dimension.

D. On the length of input encoding q

Our algorithm takes random encoding of triplets of items as input and learns a transformation from the input encoding to vectors in Euclidean space of a dimension specified by the task. We run simulations with random input representations of a fixed length q where we choose each component of the vector uniformly at random from the unit interval. Our simulations show that the size of the input encoding needs to grow logarithmically with an increasing number of items and using arbitrary representations of length $\log n$ suffices to achieve a small training triplet error (5%).

To conduct the simulations, we sampled an exponentially increasing number of items n uniformly from a unit square in R^2. For a fixed set of n items, we generate $nd \log n$ triplets (where $d = 2$ is the dimension). To isolate the effect of the length of the input encoding, we construct our EmbNet by fixing the width of the hidden layer ($w = 100$).

Figure 7 shows the training triplet error (TE_{train}) of the ordinal embedding with varying number of items and the size of the input encoding. The result shows that logarithmic growth of the size of the input encoding with respect to n suffices to obtain desirable performance.

E. On the choice of the number of triplets

In this section, we illustrate the effect of the number of chosen triplets on the quality of the reconstructions. Even in an active learning approach, the number of triplets that is required to reconstruct the original embedding is lower bounded by $nd \log n$ (Jain et al., 2016b). In Figure 11 we report results of embeddings that have been constructed with a different number of triplets. The left column shows the true datasets. The number of triplets is chosen as $r \cdot nd \log n$, where $r \in \{0.5, 1, 2, 4\}$. This number, written on top of each plot, increases from left to right. We only use these generated triplets to train an OENN. We can see that even though the loss and the training triplet error are both nearly steady in the regime of $|T|$ considered in the plots, the geometric quality (resemblance of embedding to ground truth) of the embedding increases with the number of input triplets. In particular, with $|T| = 2nd \log n$ many triplets we achieve both a low error and a nice reconstruction of the data, so this is what we use for $|T|$ in all our experiments.

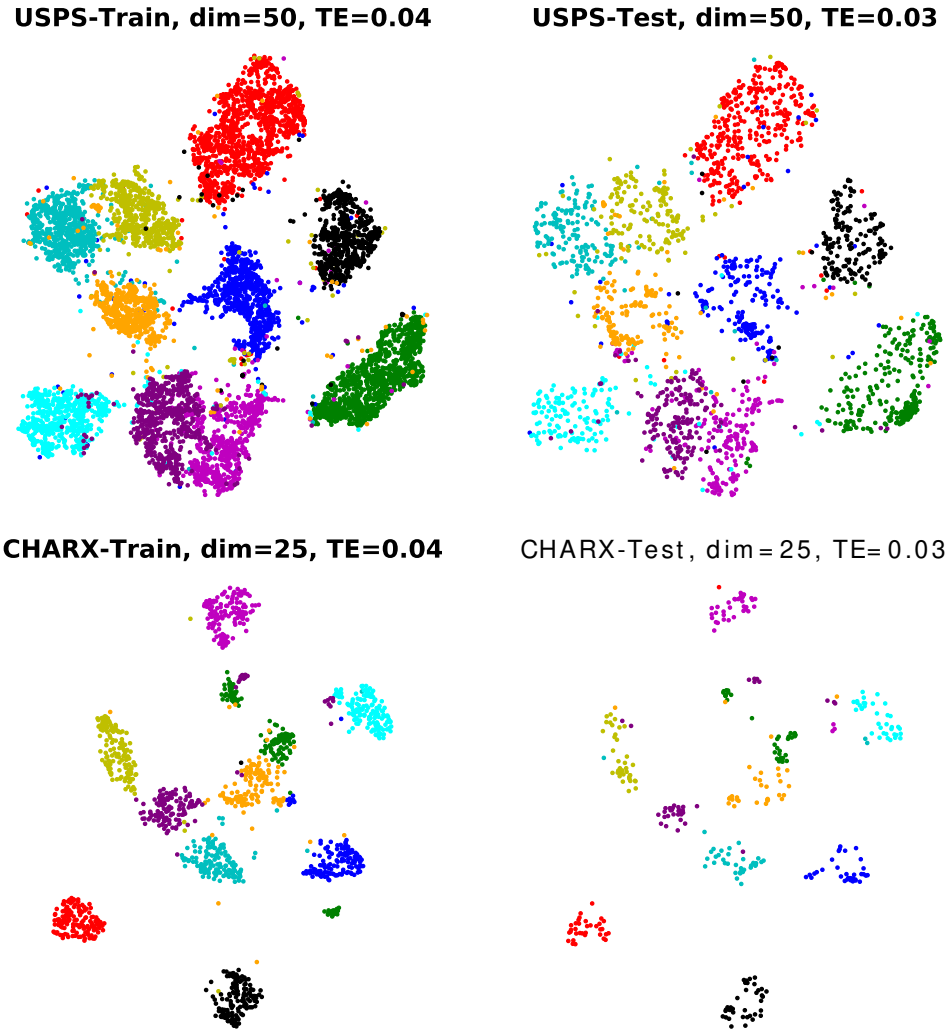


Figure 8: On the left, we show the embeddings generated by $\text{OENN}_{\text{train}}$ on the training set (visualized using t-SNE). On the right, we show the corresponding embeddings generated by $\text{OENN}_{\text{test}}$. The title of each figure reports the dataset used, the embedding dimension, and the fraction of triplets that are violated (TE). Colors indicate the labels of the items (which are only used for visualization, not for training or testing).

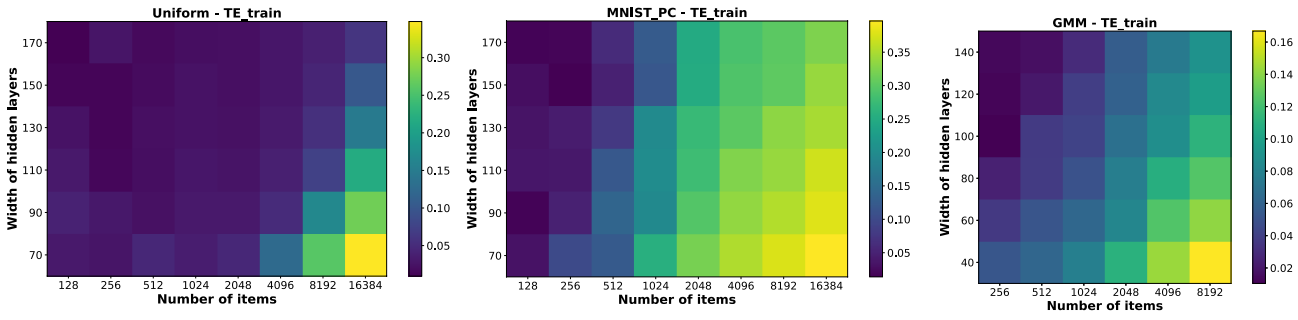


Figure 9: Triplet error (encoded by the heat map) with varying number of items and hidden layer size. The x and y axes correspond to the number of items (n) and the hidden layer size (w) respectively. The datasets used for the experiments is reported on the top of each figure. Note that $x-$axes grows exponentially, while the $y-$axes increases linearly.

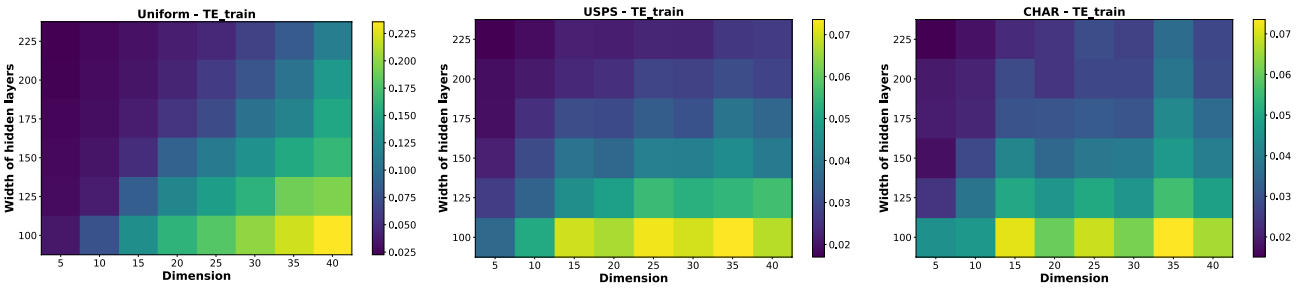


Figure 10: Triplet error (encoded by the heat map) with varying dimensions and hidden layer size. The x and y axes correspond to the number of dimensions (d) and the hidden layer size (w) respectively. The datasets used for the experiments is reported in the title of each figure. Note that both the axes scale linearly.

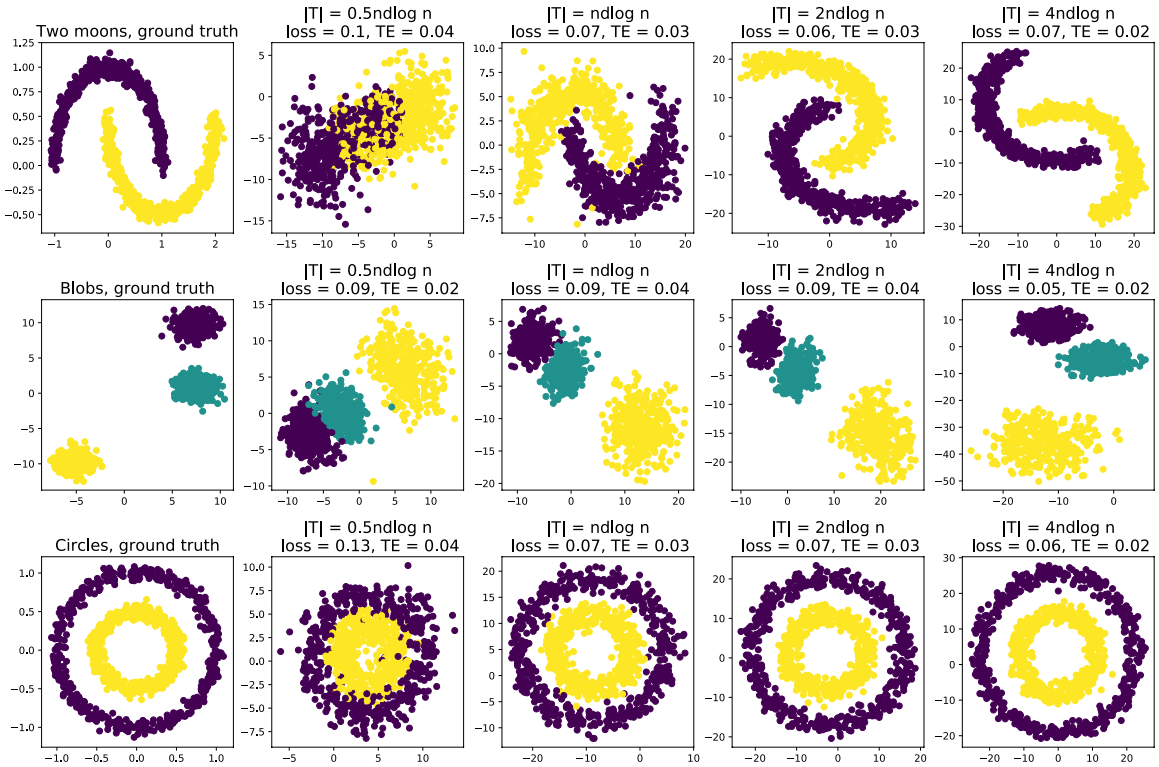


Figure 11: The reconstruction of three toy datasets with a various number of input triplets. Each column corresponds to one dataset. In each row, the first plot (from left) depicts the original dataset. The four next plots show the embedding output of OENN with increasing number of triplet inputs. The number of triplet inputs ($|T|$), training loss, and the triplet error (TE) are reported in the titles of the plots. Colors represent the labels for the items (which are only used for visualization).