Supplement to “Statistical embedding: Beyond principal components”

Dag Tjøstheim[1], Martin Jullum and Anders Løland

1. PERSISTENCE DIAGRAMS AND SIMPLICAL COMPLEXES

Assume that we observe a sample $X_1, \ldots, X_n$ drawn from a distribution P supported on a set S, and let us define the empirical distance function

$$\hat{d}(x) = \min_{1 \le i \le n} ||x - X_i||.$$

It should be noted that lower level sets $\hat{L}_\varepsilon$ defined by $\hat{L}_\varepsilon = \{x : \hat{d}(x) \le \varepsilon\}$ are precisely the union of balls described in Equation (13) in the main paper, i.e.,

$$\hat{L}_\varepsilon = \{x : \hat{d}(x) \le \varepsilon\} = \cup_{i=1}^n B(X_i, \varepsilon).$$

The persistence diagram $\hat{D}$ defined by these lower level sets is an estimate of the underlying diagram D.

The empirical distance function is often used for defining the persistence diagram of a data set in computational topology. However, as pointed out by Wasserman (2018), from a statistical point of view this is a poor choice, as it is highly non-robust. Wasserman points out several more robust alternatives. One of them is the so called DTM distance introduced by Chazal, Cohen-Steiner and Mégot (2011) given by

$$\hat{d}_m^2(x) = \frac{1}{k} \sum_{i=1}^{k} ||x - X_i(x)||^2,$$

where $k = [mn]$ is the largest integer less than or equal to mn and with $0 \le m \le 1$ being a scale parameter. Further, $X_j(x)$ denotes the data after re-ordering them so that $||X_1(x) - x|| \le ||X_2(x) - x|| \le \cdots$. This means that $\hat{d}_m^2(x)$ is the average squared distance to the k-nearest neighbors Other alternative references to a robustified distance measure are given in Wasserman (2018).

Actually, in more complicated situations, the persistence diagram is not computed directly from $\hat{L}_\varepsilon$, but from so-called simplical complexes. This approach is particularly interesting since it generalizes the embedding of a point cloud in a graph as described in Sections 3.4 and 3.5 in the main manuscript. We will give a brief description here. Much more details can be found in Chazal and Michel (2021).

First, recall the definition of a simplex: Given a set $\mathbb{X} = \{X_0, \ldots, X_k\} \subset \mathbb{R}^p$ of $k+1$ “affinely independent” (i.e., the vectors $(X_0, X_1, \ldots X_k)$ are linearly independent), the k-dimensional simplex $\sigma = [X_0, \ldots, X_k]$ spanned by $\mathbb{X}$ is the convex hull of $\mathbb{X}$. For instance, for $k = 1$ the simplex is simply given by the line from X_0 to X_1. The points of $\mathbb{X}$ are called the nodes of σ and the simplices spanned by the subsets of $\mathbb{X}$ are called the faces of σ. A geometric simplical complex K in $\mathbb{R}^p$ is a collection of simplices such that (i) any face of

Dag Tjøstheim is Professor Emeritus at the Department of Mathematics, University of Bergen, Bergen, Norway and Professor II at the Norwegian Computing Center, Oslo, Norway (e-mail: Dag.Tjostheim@uib.no). Martin Jullum is Senior Research Scientist at the Norwegian Computing Center, Oslo, Norway (e-mail: Martin.Jullum@nr.no). Anders Løland is Research Director at the Norwegian Computing Center (e-mail: Anders.Loland@nr.no).

[1]Corresponding author.

a simplex of K is a simplex of K, (ii) the intersection of any two simplices of K is either empty or a common face of both.

As seen in Sections 3.4 and 3.5 in the main paper, connecting pairs of nearby data points by edges leads to the standard notion of a neighboring graph from which the connectivity of the data can be analyzed and clustering can be obtained, including non-convex situations, as described in Section 3.4. Using simplical complexes, where simplical complexes of dimension 1 are graphs, one can go beyond this simple form of connectivity. In fact a central idea in TDA is to build higher dimensional equivalents of neighboring graphs by not only connecting pairs but also $(k+1)$-tuples of nearby data points. This enables one to identify new topological features such as cycles and voids and their higher dimensional counterparts. Regarding embedding of networks, as treated in Section 5, such a technique could possibly be used to discover cycles in networks such as criminal rings in fraud detection, say.

Simplical complexes are mathematical objects that have both topological and algebraic properties. This makes them especially useful for TDA There are two main examples of complexes in use. They are the Vietoris-Rips complex and the Čech complex. The Vietoris-Rips complex $V_\varepsilon(\mathbb{X})$ can be introduced in a metric space (M,d). It is the set of simplices $\mathbb{X}=[X_0,\dots,X_k]$ such that $d_{\mathbb{X}}(X_i,X_j)\le\varepsilon$ for all (i,j). The Čech complex $C_\varepsilon(\mathbb{X})$ is defined as the simplices $[X_0,\dots,X_k]$ such that the $k+1$ balls $B(X_i,\varepsilon)$ have a nonempty intersection.

These definitions should be compared to the use of ball-coverings in Section 4 of the main paper and level sets defined in the present subsection. It can in fact be shown that the homology of $\hat{L}_\varepsilon$ is the same as the homology of C_ε. The homology of C_ε can be computed using basic matrix operations. All relevant computations can be reduced to linear algebra. This gives a method of computing homology and persistent homology relating the complexes as ε varies as briefly mentioned in our simple introductory example of chain of circles, or the more involved example involving Ranunculoids, in Section 4.2 of the main paper (see Edelsbrunner and Harer (2010)). In fact, it is computationally easier to work out the algebra for the Vietoris-Rips complex V_ε. It can be shown that the persistent homology defined by V_ε approximates the persistent homology defined by C_ε.

Given a subset $\mathbb{X}$ of a compact metric space (M,d), the families of Vietoris-Rips complexes, $\{V_\varepsilon(\mathbb{X})\}_{\varepsilon\in\mathbb{R}}$ and the family of Čech complexes, $\{C_\varepsilon(\mathbb{X})\}_{\varepsilon\in\mathbb{R}}$ are filtrations, that is, nested families of complexes. As indicated earlier, the parameter ε can be considered as a data resolution level at which one considers the data set $\mathbb{X}$. For example if $\mathbb{X}$ is a point cloud in $\mathbb{R}^p$, the filtration $\{C_\varepsilon\}$ encodes the topology of the whole family of unions of balls $\mathbb{X}^\varepsilon=\cup_{X\in\mathbb{X}}B(X,\varepsilon)$ as ε goes from 0 to ∞.

As in the example in Section 4.2 of the main paper, the homology of a filtration $\{F_\varepsilon\}$ changes as ε increases: new connected components can appear, existing components can merge, loops and cavities may appear or be filled. Persistence homology tracks these changes, identifies the appearing features, and attaches a lifetime to them. The resulting information can be encoded as a set of intervals, the bar-code, or equivalently, as a multiset of points in $\mathbb{R}^2$, where the coordinates of each point is the start and end point of the corresponding interval. In Chazal and Michel (2021) a formal definition of bar-code and persistence diagram is given via the concept of persistence module which again is defined in terms of an indexed family of vector spaces and a doubly-indexed family of linear maps.

1.1 Persistent landscapes, functional spaces and applications

The space of persistence diagrams is not a function space in the sense that it is not a Hilbert space. This may make it more difficult to directly apply methods from statistics and machine learning. For example, the definition of a mean persistence diagram is not obvious and unique (Chazal and Michel, 2021, p. 28). Further, according to Chazal and Michel (2021, p. 29) the highly nonlinear nature of diagrams prevents them from being used as a standard feature of machine learning algorithms. An exception, however, is Obayashi and Hiraoka (2017).

Bubenik (2015) introduced persistence landscapes. The persistence landscape is a collection of continuous linear functions obtained by transforming the points of the persistence diagram into tent functions. This function space can be given a Hilbert space structure (in fact a more general structure of a separable Banach space in Bubenik's original paper). The random structure created by $X_1, \ldots, X_n$ may then be represented by Hilbert space variables, and it becomes meaningful to consider means, variances and a central limit theorem. The vector space structure of persistent landscapes and similar constructions may appear to be more directly extendable to machine learning, in particular to kernel methods, cf. also Section 3.7 in the main paper, in reproducing kernel Hilbert space (see for instance Reininghaus et al. (2015), Kusano and Hiraoka (2016) and Carriere and Oudot (2019)). It can safely be stated that combining TDA and persistence homology with machine learning is becoming an active research direction with results having potential for unsolved practical problems.

Clearly, the bar codes, the persistence diagrams and Betti numbers can also be used directly as feature extractors for classification problems. In particular, these have been used for network characterizations in Cartsens and Horadam (2013). Possibly such features can be used as a supplement to the network embedding and clustering methods presented in Section 5 in the main paper of this survey.

Connections between persistent homology and deep learning has also started to be explored. Umeda (2017) has done this in a time series context. Another application to time series is Ravisshanker and Chen (2019).

For applications to specific problems we refer to references in Wasserman (2018) and Chazal and Michel (2021). Wasserman discusses briefly applications to the cosmic web, images and proteins, Chazal and Michel discuss applications to protein binding configurations and classification of sensor data.

1.2 Statistical inference

A central concept in inference for persistence diagrams is the bottleneck distance. Given two diagrams C_1 and C_2, the bottleneck distance is defined by

$$\delta_\infty(C_1, C_2) = \inf_\gamma \sup_{z \in C_1} ||z - \gamma(z)||_\infty,$$

where γ ranges over all bijections between C_1 and C_2. Intuitively, this is like overlaying the two diagrams and asking how much one has to shift the diagrams to make them the same (Wasserman, 2018). The practical computation of the bottleneck distance amounts to the computation of perfect matching in a bipartite graph for which classical algorithms can be used (Chazal and Michel, 2021).

The bottleneck distance is a natural tool to express stability of persistence diagrams. An alternative distance measure is the Wasserstein distance. The bottleneck distance is also a natural tool in statistical inference on persistent landscapes, cf. Chazal et al. (2015).

The (estimated) persistence diagram $\hat{C}$ is based on a finite collection of random variables $X_1, \ldots, X_n$. One might think of a true persistence diagram C as $n \to \infty$. A central question is then whether there is such a thing as consistency, and is it possible to introduce confidence intervals? Such questions have been considered by Chazal and Michel (2021, Section 5.7; see especially Section 5.7.4) and is based on the bottleneck distance between $\hat{C}$ and C.

For many applications, in particular when the point cloud does not come from a (perturbation of) a geometric structure, the persistence diagram will look quite complicated. In particular, there will be a number of cases where the life time is quite short and consequently with representative points close to the diagonal. The question then arises whether these points can be considered as noise and should therefore be eliminated from the diagram. One needs a concept of statistical significance to make such an evaluation, and again the bottleneck distance can be used as a tool. When estimating a persistence diagram C with an estimator $\hat{C}$ one may look for a quantile type number η_α such that

$$P(d_\infty \geq \eta_\alpha) \leq \alpha, \tag{1}$$

for $\alpha \in (0, 1)$. This can be taken as a point of departure for computation of confidence intervals and significance tests.

It is necessary to translate (1) into something that can be computed. This can be done by the bootstrap as in Chazal, Massart and Michel (2016). Let $(X_1^*, \ldots, X_n^*)$ be a sample from the empirical measure defined from the observations $(X_1, \dots, X_n)$. Moreover, let $\hat{C}^*$ be the persistence diagram derived from this sample. One can then take as an estimate of η_α the quantity $\hat{\eta}_\alpha$ defined by

$$P[d_\infty(\hat{C}^*, \hat{C}) > \hat{\eta}_\alpha | X_1, \dots, X_n] = \alpha,$$

where it is straightforward to estimate $\hat{\eta}_\alpha$ by Monte Carlo integration. Chazal, Massart and Michel (2016) have shown that the bootstrap is valid when computing the sub-level sets of a density estimator. Using the bottleneck bootstrap and given a certain significance level, a band can be constructed parallel to the diagonal of the persistence diagram, and such that points in this level are considered as noise. A bootstrap algorithm can also be used to construct confidence bands for landscapes as shown in Chazal, Massart and Michel (2016).

There are a number of problems of interest for statisticians in TDA. Chazal and Michel (2021) in particular mentions four topics:

1. Proving consistency and studying the convergence rates of TDA methods.
2. Providing confidence regions for topological features and discussing the significance of estimated topological quantities.
3. Selecting relevant scales (i.e. selecting ε in the examples discussed above) at which topological phenomenons should be considered as functions of observed data.
4. Dealing with outliers and providing robust methods for TDA.

In addition, one may want to introduce the block bootstrap to take better care of dependence structures There are also recent contributions to hypothesis testing, Moon and Lazar (2020), sufficient statistics, Curry, Mukherjee and Turner (2018), and Bayesian statistics for topological data analysis, Maroulas, Nasrin and Obello (2020).

2. EMBEDDING AND WORD FEATURE REPRESENTATION OF A LANGUAGE TEXT

Sections 5.2 and 5.3 of the main paper describe the importance of embedding of networks and its use in feature extraction, in clustering, characterization and classification for ultra-large data sets. It was pointed out in Section 5.3 that a main methodology for this is the Skip-Gram procedure which was developed in the context of word embedding for a natural language. The purpose of the present section is twofold. First, language processing is of considerable independent interest. Second, it provides more details on the Skip-Gram procedure, its background and its use. Although this material is couched in terms of language analysis, we believe that when read in conjunction with Section 5.3 of the main paper, it will also provide added insight into the details of network embedding.

2.1 A few basic facts of neural nets

The Skip-Gram procedure is based on a neural network with a single hidden layer, and we therefore include a brief summary of neural networks in this supplement.

Neural networks are used for a number of problems in prediction, classification and clustering. The developments perhaps stagnated somewhat in the early seventies, but received renewed interest the last decades, following a massive increase in computational power. Currently, there is an intense activity involving among other things deep learning, where some remarkable results have been obtained. See Schmidhuber (2015) for a relatively recent overview.

Assume that we are given an n-vector x as input. In a neural network approach one is interested in transforming x via linear combinations of its components and possibly a nonlinear transformation of these linear combinations. This transformation constitutes what is called a hidden layer. Then this might be sent through a new transformation of the same type to create a new hidden layer and eventually to an output layer y which should be as close as possible to a target vector t. If there is more than one hidden layer, it is said to be a deep network, its analysis being a base for so-called deep learning. In this supplement, mainly dealing with the background of the Skip-Gram, only the case of one hidden layer will be treated, that, in our context, will be formed by a linear transformation.

Given the input layer, the first step in forming the hidden layer is to form linear combinations

$$h_i = \sum_{j=1}^{n} w_{ij} x_j, \tag{2}$$

where $i = 1, \ldots, m$. Note that implicitly, there may be a constant term by taking x_1, say, equal to 1. (This is sometimes termed the bias term of the linear combination.)

In the case of one hidden layer, the output layer is given by

$$y_j = \sum_{i=1}^{m} w'_{ij} h_i,$$

for $j = 1, \ldots, q$. In subsequent applications for language and network embedding models $q = \dim(y) = \dim(x) = n$.

In a classification problem, y_j may be associated with an unnormalized probability for a class j, which in Section 5.3 of the main paper is the appropriate neighborhood of a node v_j in a network. In such cases the output layer is also transformed. A common transformation is the so-called softmax function given by

$$\text{softmax}(y_j) = \frac{\exp(y_j)}{\sum_{i=1}^{n} \exp(y_i)}. \tag{3}$$

This is recognized (if there is no hidden layer) as the multinomial logistic regression model which is a standard tool in classification.

Using a training set, the coefficients (or weights) w_{ij} and w'_{ij} are determined by a penalty function measuring the distance between the output y and the target vector t, for example measured by the loss function $E = ||y - t||^2$. In a classification and clustering problem the training set consists of input vectors x belonging to known classes i (known words in the vocabulary in the text). The target vector is a so-called "one hot" vector having 1 at the component j for the given target word and zeros elsewhere. The weights are adjusted such that the output vector is as close as possible to this vector, which means that the softmax function should be maximized for this particular component and ideally $\exp(y_i) \approx 0$ for $i \neq j$.

The error function is evaluated for each of the samples coming in as inputs, and the gradient of the error function with respect to y is evaluated with the weights being re-computed and updated in the direction of the gradient by stochastic gradient descent.

The weights w'_{ij} for the output layer is computed first and then w_{ij} by the chain differentiation rule using so-called back propagation. Details are given in e.g. the appendix of Rong (2016). Schematically this may be represented by

$$w_{ij}^{(\text{new})} = w_{ij}^{(\text{old})} - \varepsilon \frac{\partial E}{\partial w_{ij}}$$

and similarly for w'_{ij}. Initial values for the weights can be chosen by drawing from a set of uniform variables. Below the updating scheme will be illustrated on word representation of natural languages, which next can be applied to embedding of networks.

2.2 Word feature representation of natural languages

Consider a natural language text. We start with a set of input vectors x_i, $i = 1, \ldots, n$, where n is the number of words in the vocabulary of the text, and x_i represents word i in the vocabulary. Each vector is of dimension n, where x_i has a one in position i of the vector and zeros elsewhere ("one-hot" encoded vector). Let m be the dimension of the desired word embedding feature representation. The dimension may be quite large. Common choices are in the range $100 - 1000$. Let the one-hot vector for the word w_i, word number i in the vocabulary, be x_i. Further, consider a $n \times m$ weight matrix $\mathbf{W}$. Define the m-dimensional hidden units h_i, $i = 1, \ldots, m$ (without a nonlinear transformation) by

$$h_i = \mathbf{W}^T x_i \doteq v_{w_i}^T, \tag{4}$$

which is essentially copying the m-dimensional ith row of $\mathbf{W}$ to h_i. The vector v_{w_i} is the input word representation vector for word number i in the vocabulary, or the feature vector f_i of the word w_i . This means that the link (activation) function of the hidden layer units is simply *linear*. The weights, i.e., the vector word representation can then be learned by the neural network given appropriate targets and a penalty function.

An obvious question is whether a nonlinear transformation is needed. Bengio et al. (2003), in their pioneering paper suggest an added nonlinearity, whereas the approach of Mikolov et al. (2013a,b) is entirely linear, but using the softmax transformation henceforth. The latter papers also have some other ingredients which have made them extremely influential.

An essential feature of the papers by Mikolov et al. (2013a,b) and related papers is that they have found clever approximations to simplify and speed up the calculations of Bengio et al. (2003).

2.3 The Mikolov et al. approach: word2vec

We have already presented the input linear representation of word vectors as rows of the weight matrix $\mathbf{W}$, see (4). The output layer should consist of conditional probabilities of words in the vocabulary as in Bengio et al. (2003), but Mikolov et al. has a purely linear transformation to the output layer prior to the softmax transformation.

As a further simplification we assume that we have a window passing over a given text with the window consisting of just two words w_t, w_{t-1} in position t and $t-1$ of the text. Here, w_t is the target word of the text w_O, w_{t-1} is the input word w_I, and the conditional probability $P(w_t|w_{t-1})$ can also be written $P(w_O|w_I)$. This means that there is only one context word w_I for the output word, whereas in the case of Bengio et al. (2003) there were $l-1$ context words. (Note that in Skip-Gram, and the use of it in network embedding, the context words are more naturally being thought of as target words belonging to the output.) To describe the transition from the hidden layer to the output layer we introduce a new $m \times n$ dimensional weight matrix $\mathbf{W}' = \{w'_{ij}\}$. Let v'_{w_j} be the jth column of the matrix $\mathbf{W}'$ (it has dimension m). It is the output vector representation of word number j in the vocabulary. Then the n-dimensional output vector is defined by

$$y = (\mathbf{W}')^T h,$$

where $h = v_{w_I}$. Component y_j is given by

$$y_j = (v'_{w_j})^T h, \ j = 1, \ldots, n. \tag{5}$$

To obtain the posterior distribution one uses softmax as defined in (3),

$$P(w_j|w_I) \doteq u_j = \frac{\exp(y_j)}{\sum_{i=1}^n \exp(y_i)}, \tag{6}$$

where now u_j is the transformed output of the jth unit in the output layer. By substitution, one obtains

$$P(w_j|w_I) = \frac{\exp\Big((v'_{w_j})^T v_{w_I}\Big)}{\sum_{i=1}^n \exp\Big(w'_i)^T v_{w_I}\Big)}. \tag{7}$$

It should be noted that one gets two distinct word representations v_w and v'_w for each word w in the vocabulary, one input and one output word vector. The output vector is the relevant one in the sense that the context relations are baked into it. Since the system is completely linear, there are no extra parameters to be learned from the network, "just" the matrices $\mathbf{W}$ and $\mathbf{W}'$.

The network is trained by stochastic gradient descent as in Bengio et al. (2003) and most other neural network applications. Given the input word w_I and the output word w_O, one is interested in maximizing the conditional probability $P(w_O|w_I)$; i.e., finding the index $j = j^\star$ and the corresponding probability u_j in the output layer so that, using (6),

$$\max u_j = \max P(w_O|w_I) \quad \text{or} \quad \max \log u_j = y_{j^\star} - \log \sum_{i=1}^n \exp(y_i). \tag{8}$$

By taking derivatives one gets the update equation

$$(w'_{ij})^{(\text{new})} = (w'_{ij})^{(\text{old})} - \eta e_j h_i,$$

or

$$(v'_{w_j})^{(\text{new})} = (v'_{w_j})^{(\text{old})} - \eta e_j h_i. \tag{9}$$

for $j = 1, \dots, n$, where $\eta > 0$ is the learning rate and $e_j = u_j - t_j$ with $t_j = 1(j = j^\star)$. One has to go through every word in the vocabulary, check its output probability u_j, and compare u_j with its targeted output, either 0 or 1.

Going through the same exercise for the transition between the input and the hidden layer, one obtains (see Rong (2016) for details) for the update equation if $w_I = w_i$

$$v_{w_i}^{(\text{new})} = v_{w_i}^{(\text{old})} - \eta F,$$

where F is the vector whose ith component, using back propagation, is given by $\sum_{j=1}^n e_j w'_{ij}$. Recall that $v_{w_I}^T$ is a row of $\mathbf{W}$, the "input word vector" of the only context word $w_I = w_i$, and it is the only row of $\mathbf{W}$ whose derivative is non-zero. All the other rows will remain unchanged after this iteration, since their derivatives are zero.

The generalization from a one word context to a context with several words is quite straightforward in the Mikolov et al. (2013a,b) set-up. They distinguish between two ways of doing this, the CBOW and the Skip-Gram model.

Traditional text classification is based solely on frequencies in the text of words in the vocabulary. This is the bag of words (BOW) approach. Mikolov et al. (2013a,b) take context into account resulting in a continuous bag of words (CBOW). We are then essentially back to the situation in Bengio et al. (2003) where there are $C = l - 1$ context words and we want to maximize $P(w_O | w_1, \dots, w_C)$, but Mikolov et al. assume linearity in the concatenated C words in such a way that the concatenated word vector corresponding to $[w_1, \dots, w_C]$ is simply given by the average $\frac{1}{C}(v_{w_1} + \cdots + v_{w_C})$ of the individual pairwise word vectors. The hidden layer is then given by

$$\begin{aligned} h &= \frac{1}{C}\mathbf{W}^T(x_1 + x_2 + \cdots x_C) \\ &= \frac{1}{C}(v_{w_1} + \cdots + v_{w_C}). \end{aligned} \tag{10}$$

This is the CBOW assumption. With this assumption one is more or less back to the one-context word updates. The loss function can be written (cf. (5) and (8)),

$$\begin{aligned} E &= -\log P(w_O | w_1, \cdots w_C) \\ &= -y_{j^\star} + \log \sum_{i=1}^n \exp(y_i) = -(v'_{w_O})^T h + \log \sum_{i=1}^n \exp((v'_{w_i})^T h), \end{aligned} \tag{11}$$

which is the same as (8), the objective of the one-word context model, except that h is different, being defined as in (10) instead of in (4). This leads to an update equation for the output words which is identical to (9), whereas the update equation for input words has to be updated separately for every word w_c, $c = 1, \dots, C$, namely

$$v_{w_c}^{(\text{new})} = v_{w_c}^{(\text{old})} - \frac{1}{C}\eta F,$$

where F is defined as before.

2.4 The Skip-Gram model

The Skip-Gram model is in a sense the opposite of the CBOW model, and this is the situation considered in the network embedding in Section 5.3. It is also different from the Bengio model. For a window centered at the word w_I, the window contains $C/2$ (with C being an even number) words before the center word w_I and $C/2$

word after the center word, so that in the notation of Bengio et al. (2003) the window consists of the words $[w_{t+C/2}, \dots, w_t, \dots, w_{t-C/2}]$. Sliding the window, the objective is to predict each of the C context words (i.e. maximize the conditional probability) $[w_{t+C/2}, \dots, w_{t+1}, w_{t-1}, \dots, w_{t-C/2}]$ given the input word $w_I = w_t$. Here, conditional independence is assumed, so that the conditional probability for each context word is maximized separately.

For the input word representation the derivation in the two word case is the same as the present situation for the input word and with the same definition of the hidden layer h, so that we still have $h_I = v_{w_I}^T$. Instead of outputting one (multinomial) distribution, we are outputting C (multinomial) distributions. But, importantly, each output is computed using the same matrix $\mathbf{W}'$ mapping the hidden layer into the output layer. (This means that the *sequencing* of the context words does not matter, only *which* words are there in the window). Moreover,

$$P(w_{c,j}|w_I) = \frac{\exp(y_{c,j})}{\sum_{i=1}^n \exp(y_i)},$$

where $w_{c,j}$, $c = 1, \dots, C$, $j = 1, \dots, n$, and where the index j is referring to the number in the vocabulary of the word $w_{O,c}$. Further for $h = v_{w_i}$,

$$y_{c,j} = (v'_{w_j})^T h,$$

for $c = 1, \dots, C$, where v'_{w_j} is the output vector for the jth word w_j in the vocabulary, and also v'_{w_j} is taken from the jth column of weight matrix $\mathbf{W}'$ transforming the hidden layer to the output layer.

The derivations of the parameter update equations are similar to the one-word context. Assuming conditional independence, the loss function in (11) is changed to

$$E = -\log P(w_{O,1}, \dots, w_{O,C}|w_I) = -\sum_{c=1}^{C} (v'_{w_c})^T v_{w_I} + C \log \sum_{i=1}^{n} \exp\{(v'_{w_i})^T v_{w_I}\}.$$

The updating equations can be derived by taking derivatives similarly to the CBOW case, and we refer to Rong (2016) for details.

In spite of the relatively simple linear structure of CBOW and Skip-Gram, it makes for some quite astonishing properties that goes beyond simple syntactic regularities. This is obtained using just very simple algebraic operations in the word representation space $\mathbb{R}^m$, such that for example the embedded word vector(“King”)-word vector(“Man")+word vector(“Woman”) has a high probability of having the word vector(“Queen”) as its closest word vector, as measured by cosine distance in word feature space $\mathbb{R}^m$. Several similar examples are given in Mikolov et al. (2013a,b), and they have also examined quite systematically the capabilities of CBOW and Skip-Gram compared to other word representation routines in solving such tasks.

2.5 The computational issue

For all of the word models presented so far, there is a computational issue. As the size of the vocabulary and the size of the training text set increase, they are heavy to update. For the two-word, the CBOW and the Skip-Gram models there are two vector representations for each word in the vocabulary: the input vector v_w and the output vector v'_w. Learning the input vectors is cheap, but learning the output vectors is expensive. From the update equations (6), (7), (8) and (9) it is seen that to update v'_w for each training instance, one has to iterate through every word w_j in the vocabulary, compute y_j, the prediction error e_j and finally use the prediction error to update the output vector v'_{w_j}.

Such kind of computations makes it difficult to scale up to large vocabularies or large training corpora. The obvious solution to circumvent this problem is to limit the number of output vectors that must be updated per training instance. There are two main approaches for doing this, hierarchical softmax and negative sampling. Both approaches optimize *only* the computation for updates for output vectors.

Hierarchical softmax is an efficient way of computing softmax (Morin and Bengio, 2005; Mnih and Hinton, 2008). With this method the frequency of words appearing in texts is taken into account. In hierarchical softmax the list of words from word 1 to word n is replaced by a binary Huffman encoded tree with the n words appearing at the leaves (outer branches) of the tree. The probability of the occurrence of a word given an input word is computed from a probability path from the root of the tree to the given word. This reduces the number of operations in an update from n to $\log_2 n$, e.g. for $n = 1$ million $= 10^6$, the number of operations are reduced to $6\log_2 10 \approx 20$. We refer to Morin and Bengio (2005) and Mnih and Hinton (2008) for a detailed description of hierarchical softmax.

2.6 Negative sampling

The idea of negative sampling is far more straightforward than hierarchical softmax. It is sampling-based, and for each updating instance, only a sample of output vectors are used. This seems to be an, perhaps *the*, essential idea that makes Skip-Gram work so well.

Obviously the output words; i.e. w_O in CBOW and each of the words $w_{O,c}$ for $c = 1, \ldots, C$ in the Skip-Gram procedure should be included in the updating sample. They represent the ground truth and are termed positive samples. In addition, a certain number k of word vectors (noise or negative samples) are updated, such that $k = 5 - 20$ are useful for small training sets, whereas for large training sets, $k = 2 - 5$ may be sufficient (Mikolov et al., 2013b). The sampling is carried out via a probability mechanism where each word is sampled according to its frequency $f(w_i)$ in the text. In addition, Mikolov et al. recommend from empirical experience that each word is given a weight equal to its frequency (word count) raised to the $3/4$ power. The probability for selecting a word (vector) is just its weight divided by the sum of weights for all words, i.e.,

$$P_n(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=1}^{n} f(w_j)^{3/4}}.$$

In addition, in word2vec, instead of using the loss functions (8) and (11) constructed from multinomial distributions, the authors argue that the following simplified training objective is capable of producing high-quality word embeddings:

$$E = -\log\sigma((v'_{w_O})^T h) - \sum_{w_j \in \mathcal{W}_{\text{neg}}} \log\sigma(-(v'_{w_j})^T h), \tag{12}$$

where $\sigma(u)$ is the logistic function given by $\sigma(u) = 1/(1+\exp(-u))$ and $\mathcal{W}_{\text{neg}}$ is the collection of negative samples for the given update. Further, w_O is the output word (the positive sample), v'_{w_O} is the output vector; h is the value of the hidden layer with $h = \frac{1}{C}\sum_{c=1}^{C} v_{w_c}$ in the CBOW model and $h = v_{w_I}$ in the Skip-Gram model. Note that Mikolov et al. write (12) as

$$E = -\log\sigma((v'_{w_O})^T h) - \sum_{i=1}^{k} E_{w_i \sim P_n(w)} \log\sigma(-(v'_{w_i})^T h).$$

To obtain the update equations we again use the chain rule of differentiation. First, the derivative of E with respect to $(v'_{w_j})^T h$ is computed as

$$\frac{\partial E}{\partial((v'_{w_j})^T h)} = \left\{ \begin{array}{ll} \sigma((v'_{w_j})^T h) - 1 & \text{if } w_j = w_O \\ \sigma((v'_{w_j})^T h) & \text{if } w_j \in \mathcal{W}_{\text{neg}} \end{array} \right\},$$

which results in the derivative being equal to $\sigma((v'_{w_j})^T h) - t_j$ where t_j is the label of word w_j such that $t_j = 1$ if w_j is a positive sample, and 0 otherwise. Next, we take the derivative of E with regard to the output vector of the word w_j,

$$\frac{\partial E}{\partial v'_{w_j}} = \frac{\partial E}{\partial((v'_{w_j})^T h)} \frac{\partial((v'_{w_j})^T h)}{\partial v'_{w_j}} = \Big(\sigma((v'_{w_j})^T h) - t_j\Big) h.$$

This results in the following update equation for the output vector

$$v'^{(\text{new})}_{w_j} = v'^{(\text{old})}_{w_j} - \varepsilon\Big(\sigma((v'_{w_j})^T h) - t_j\Big)h,$$

which only needs to be applied to $w_j \in \{w_O\} \cup \mathcal{W}_{\text{neg}}$ instead of every word in the vocabulary. This equation can be used both for CBOW and the Skip-Gram model. For the Skip-Gram model, the equation has to be applied for one context word at a time.

To back-propagate the error to the hidden layer and thus update the input vectors of words, it is necessary to take the derivative of E with regard to the hidden layer's output, obtaining

$$\begin{aligned}\frac{\partial E}{\partial h} &= \sum_{w_j \in \{w_O\} \cup \mathcal{W}_{\text{neg}}} \frac{\partial E}{\partial (v'_{w_j})^T h} \frac{\partial (v'_{w_j})^T h}{\partial h} \\ &= \sum_{w_j \in \{w_O\} \cup \mathcal{W}_{\text{neg}}} \Big(\sigma((v'_{w_j})^T h) - t_j\Big) v'_{w_j} \doteq F.\end{aligned}$$

Using this, one can obtain update equations for the input vectors of the CBOW and Skip-Gram models.

2.7 Some results

There are a number of results for variously structured text data sets in Mikolov et al. (2013a,b), where it is seen that CBOW and Skip-Gram perform well compared to other methods and that with negative sampling or hierarchical softmax the methods can be applied to vocabularies in the millions and text samples in the billions of words. Choices of parameters such as the number of context words (not much greater than 10), sample size of negative samples, and dimension of word vectors are discussed. Further, there are several experiments analyzing the sensitivity of the results on applications to empirical data. The Skip-Gram is a slightly heuristic method when combined with negative sampling (such as a sudden shift from one objective function to another one, raising the empirical frequencies to an exponent of 3/4). The authors justify this from the empirical results obtained, which are quite impressive. There are several papers attempting to simplify and complement the rather brief description in the papers by Mikolov et al. (2013a,b), and trying to give it a firmer mathematical basis. We have found Rong (2016) useful. The shift of objective function is sought explained in Goldberger and Levy (2014).

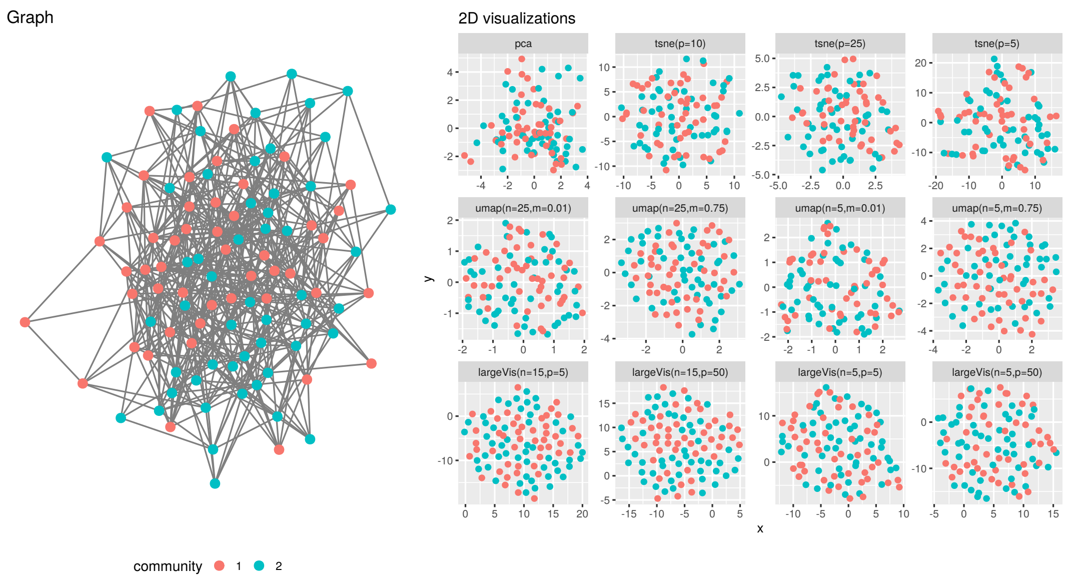
There are extensions to classification of text extending the context of word-vector to the concept of paragraph-vector in Le and Mikolov (2014), but it is very concisely written. There is also a paper on machine translation by Mikolov, Le and Sutskever (2013). Software is easily available for all of the algorithms described in this section.

3. A MORE INVOLVED ILLUSTRATING EXAMPLE

Fig. 1 contains more challenging variants of the graphs in Fig. 5 in the main paper. The homogeneous graph in Fig. 1a is simulated from a stochastic block model with 2 communities, 100 nodes, average node degree $d = 10$ and ratio of between-community edges over within-community edges $\beta = 0.75$, i.e. it is generated from the same model as Fig. 5a in the paper, except that β has increased form 0.4 to 0.75. As for Fig. 5 in the main paper, embeddings with dimension 64 were computed using node2vec with 30 nodes in each walk with 200 walks per node, and a word2vec window length of 10 where all words are included. The accompanying 2-dimensional visualizations of the embeddings are done with PCA and t-SNE, UMAP and LargeVis, all with different tuning parameters.

Compared to Fig. 5a in the main paper, the PCA is far inferior to the three other embedded visualizations for this more involved example. Similarly to Fig. 5b in the main paper, the heterogeneous graph in Fig. 1b is simulated from three stochastic block models (three subgraphs $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$, each with 2 communities:

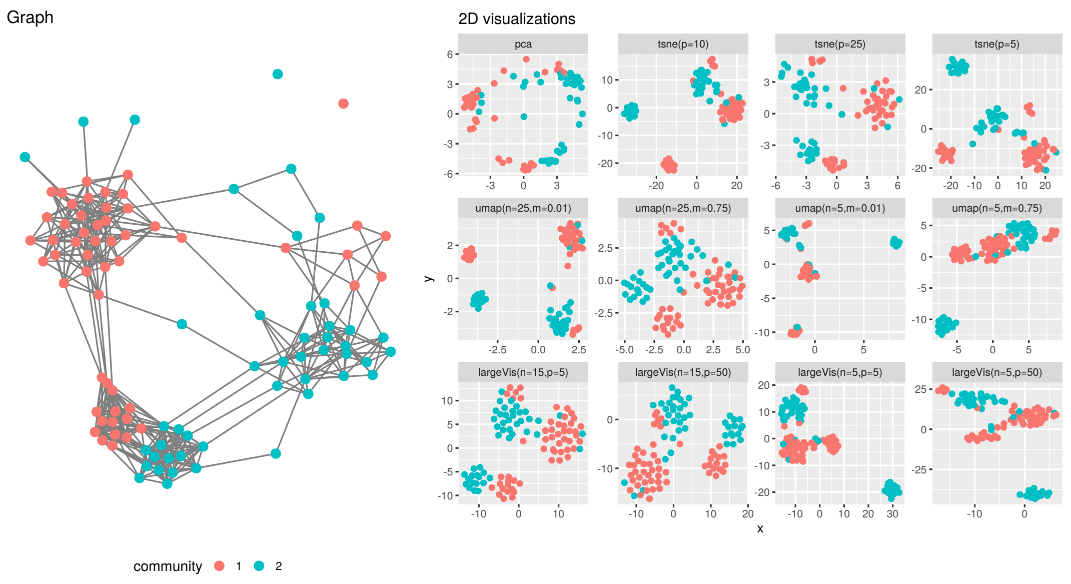
Graph a: 30 nodes, average node degree $d = 7$, ratio of between-community edges over within-community edges $\beta = 0.1$



Classification scores

	org_embedding	pca	tsne(p=10)	tsne(p=25)	tsne(p=5)	umap(n=25,m=0.01)	umap(n=25,m=0.75)	umap(n=5,m=0.01)	umap(n=5,m=0.75)	largeVis(n=15,p=5)	largeVis(n=15,p=50)	largeVis(n=5,p=5)	largeVis(n=5,p=50)
prop community correct	0.562	0.472	0.48	0.536	0.518	0.508	0.522	0.504	0.512	0.554	0.572	0.484	0.542
prop majority vote correct	0.58	0.42	0.45	0.51	0.58	0.57	0.56	0.54	0.55	0.6	0.62	0.47	0.58

(a) Homogeneous graph from the stochastic block model.



Classification scores

	org_embedding	pca	tsne(p=10)	tsne(p=25)	tsne(p=5)	umap(n=25,m=0.01)	umap(n=25,m=0.75)	umap(n=5,m=0.01)	umap(n=5,m=0.75)	largeVis(n=15,p=5)	largeVis(n=15,p=50)	largeVis(n=5,p=5)	largeVis(n=5,p=50)
prop community correct	0.912	0.876	0.912	0.926	0.904	0.934	0.892	0.894	0.902	0.872	0.882	0.894	0.886
prop majority vote correct	0.95	0.9	0.94	0.95	0.94	0.96	0.96	0.94	0.94	0.93	0.93	0.93	0.92

(b) Heterogeneous graph from a combination of three stochastic block models.

Fig 1: Graphs, visualizations and classification results with a k-nearest neighbors algorithm with $k = 5$.

Graph b: 30 nodes, average node degree $d = 15$, ratio of between-community edges over within-community edges $\beta = 0.2$

Graph c: 40 nodes, average node degree $d = 7$, ratio of between-community edges over within-community edges $\beta = 0.1$, and an unbalanced community proportion; a probability of 3/4 for community 1 and a probability of 1/4 for community 2

To link graphs a, b and c, some random edges are added between nodes from the same community[1]. The results are somewhat similar to those of Fig. 5b of the main paper. Again, PCA is inferior to the three other methods, but it is closer than in Fig. 1b.

REFERENCES

BENGIO, Y., DUCHARME, R., VINCENT, P. and JAUVIN, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research* **3** 1137-1155.

BUBENIK, P. (2015). Statistical topological data analysis using persistence landscapes. *Journal of Machine Learning Research* **16** 77-102.

CARRIERE, M. and OUDOT, S. (2019). Sliced Wasserstein kernel for persistence diagrams. arXiv 1803.07961v1.

CARTSENS, C. J. and HORADAM, K. J. (2013). Persistent homology of collaboration networks. *Mathematical Problems in Engineering* 1-7.

CHAZAL, F., COHEN-STEINER, D. and MÉGOT, Q. (2011). Geometric inference for probability measures. *Foundations of Computational Mathematics* **11** 733-751.

CHAZAL, F., MASSART, P. and MICHEL, B. (2016). Rates of convergence for robust geometric inference. *Electronic Journal of Statistics* **10** 2243-2286.

CHAZAL, F. and MICHEL, B. (2021). An introduction to topological data analysis: fundamental and practical aspects for data scientists. *Frontiers in Artificial Intelligence: Machine Learning and Artificial Intelligence* **4** 1-28.

CHAZAL, F., FASY, B. T., LECCI, F., RINALDO, A. and WASSERMAN, L. (2015). Stochastic convergence of persistence landscapes and silhouettes. *Journal of Computational Geometry* **6** 140-161.

CURRY, J., MUKHERJEE, S. and TURNER, K. (2018). How many directions determine a shape and other sufficiency results for two topological transforms. arXiv:1805.09782.

EDELSBRUNNER, H. and HARER, J. (2010). *Computational Topology: An Introduction*. American Mathematical Society.

GOLDBERGER, Y. and LEVY, O. (2014). word2vec explained: Deriving Mikolov et al.'s negative-sampling word-embedding method. arXiv: 1402.3722v.1.

KUSANO, G. and HIRAOKA, Y. (2016). Persistence weighted Gaussian kernel for topological data analysis. Proceedings of the 33rd International Conference on Machine Learning, New York.

LE, Q. and MIKOLOV, T. (2014). Distributed representations of sentences and documents. arXiv:1405.4053v2.

MAROULAS, V., NASRIN, F. and OBELLO, C. (2020). A Bayesian framework for persistent homology. *SIAM Journal of Mathematical Sciences* **2**.

MIKOLOV, T., LE, V. and SUTSKEVER, I. (2013). Exploiting similarities among languages for machine translation. arXiv:1309.4168.

MIKOLOV, T., CHEN, K., CORRADO, G. and DEAN, J. (2013a). Efficient estimation of word representations in vector space. CoRR, abs/1301,3781.

MIKOLOV, T., SUTSKEVER, I., CHEN, K., CORRADO, G. and DEAN, J. (2013b). Distributed representation of words and phrases and their composability. In Advances in Neural Information Processing Systems 26: Proceedings Annual 27th Conference on Neural Information Processing Systems. Lake Tahoe, Nevada, USA.

MNIH, A. and HINTON, G. (2008). A scalable hierarchical distributed language model. NIPS Proceedings 2008.

MOON, C. and LAZAR, N. A. (2020). Hypothesis testing for shapes using vectorized persistence diagrams. arXiv:2006.0n46.

MORIN, F. and BENGIO, Y. (2005). Hierarchical probabilistic neural network language model. AISTATS.

OBAYASHI, I. and HIRAOKA, Y. (2017). Persistence diagrams with linear machine learning models. arXiv preprint 1706.10082.

RAVISSHANKER, N. and CHEN, R. (2019). Topological data analysis (TDA) for time series. arXiv: 1909.10604v1.

REININGHAUS, J., HUBER, S., BAUER, U. and KWITT, R. (2015). A stable multi-scale kernel for topological machine learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.

RONG, X. (2016). word2vec parameter learning explained. arXiv:1411.2738v4.

SCHMIDHUBER, J. (2015). Deep learning in neural networks: An overview. *Neural Networks* **61** 85-117.

UMEDA, Y. (2017). Time series classification via topological data analysis. *Transactions of the Japanese society for Artificial Intelligence* **32** 1-12.

WASSERMAN, L. (2018). Topological data analysis. *Annual Review of Statistics and its Applications* **5** 501-532.

[1]For each pair of nodes between a pair of graphs, say Graph a and c, a new link is randomly sampled with a probability of 0.01, and links connecting two nodes from the same community are kept.