Distributed Representation of n-gram Statistics for Boosting Self-organizing Maps with Hyperdimensional Computing

Chapter · December 2019

DOI: 10.1007/978-3-030-37487-7_6

Denis Kleyko
Örebro University
108 PUBLICATIONS **1,899** CITATIONS
SEE PROFILE

Evgeny Osipov
Luleå University of Technology
121 PUBLICATIONS **1,635** CITATIONS
SEE PROFILE

Urban Wiklund
Umeå University
156 PUBLICATIONS **5,027** CITATIONS
SEE PROFILE

Valeriy Vyatkin
Aalto University
535 PUBLICATIONS **9,600** CITATIONS
SEE PROFILE

Distributed Representation of n-gram Statistics for Boosting Self-Organizing Maps with Hyperdimensional Computing*

Denis Kleyko[1], Evgeny Osipov[1], Daswin De Silva[2], Urban Wiklund[3], Valeriy Vyatkin[1], and Damminda Alahakoon[2]

[1] Luleå University of Technology, Luleå, Sweden
{denis.kleyko, evgeny.osipov, valeriy.vyatkin}@ltu.se
[2] La Trobe University, Melbourne, Australia
{d.desilva, d.alahakoon}@latrobe.edu.au
[3] Umeå University, Umeå, Sweden
urban.wiklund@umu.se

Abstract. This paper presents an approach for substantial reduction of the training and operating phases of Self-Organizing Maps in tasks of 2-D projection of multi-dimensional symbolic data for natural language processing such as language classification, topic extraction, and ontology development. The conventional approach for this type of problem is to use n-gram statistics as a fixed size representation for input of Self-Organizing Maps. The performance bottleneck with n-gram statistics is that the size of representation and as a result the computation time of Self-Organizing Maps grows exponentially with the size of n-grams. The presented approach is based on distributed representations of structured data using principles of hyperdimensional computing. The experiments performed on the European languages recognition task demonstrate that Self-Organizing Maps trained with distributed representations require less computations than the conventional n-gram statistics while well preserving the overall performance of Self-Organizing Maps.

Keywords: Self-Organizing Maps, n-gram statistics, hyperdimensional computing, symbol strings

1 Introduction

The Self-Organizing Map (SOM) algorithm [25, 41] has been proven to be an effective technique for unsupervised machine learning and dimension reduction of multi-dimensional data. A broad range of applications ranging from its conventional use in 2-D visualization of multi-dimensional data to more recent developments such as analysis of energy consumption patterns in urban environments [6, 8], autonomous video surveillance [29], multimodal data fusion [14],

* This work was supported by the Swedish Research Council (VR, grant 2015-04677) and the Swedish Foundation for International Cooperation in Research and Higher Education (grant IB2018-7482) for its Initiation Grant for Internationalisation, which allowed conducting the study.

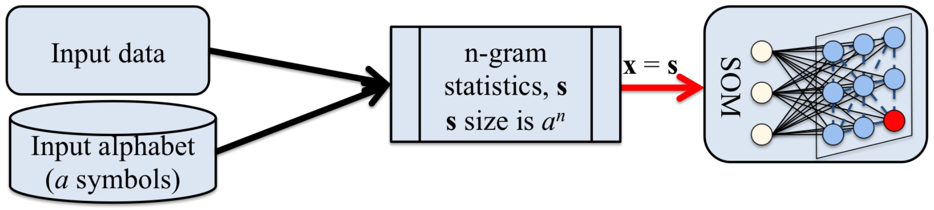

Fig. 1. Outline of the conventional approach.

incremental change detection [28], learning models from spiking neurons [12], and identification of social media trends [3, 7]. The latter use-case is an example of an entire application domain of SOMs for learning on symbolic data. This type of data is typically present in various tasks of natural language processing.

As the SOM uses weight vectors of fixed dimensionality, this dimensionality must be equal to the dimensionality of the input data. A conventional approach for feeding variable length symbolic data into the SOM is to obtain a fixed length representation through n-gram statistics (e.g., bigrams when $n = 2$ or trigrams when $n = 3$). The n-gram statistics, which is a vector of all possible combinations of n symbols of the data alphabet, is calculated during a preprocessing routine, which populates the vector with occurrences of each n-gram in the symbolic data. An obvious computational bottleneck of such approach is due to the length of n-gram statistics, which grows exponentially with n. Since the vector is typically sparse some memory optimization is possible on the data input side. For example, only the indices of non-zero positions can be presented to the SOM. This, however, does not help with the distance calculation, which is the major operation of the SOM. Since weight vectors are dense, for computing the distances the input vectors must be unrolled to their original dimensionality. In this paper, we present an approach where the SOM uses mappings of n-gram statistics instead of the conventional n-gram statistics. Mappings are vectors of fixed arbitrary dimensionality, where the dimensionality can be substantially lower than the number of all possible n-grams.

Outline of the proposed approach

The core of the proposed approach is in the use of hyperdimensional computing and distributed data representation. Hyperdimensional computing is a bio-inspired computational paradigm in which all computations are done with randomly generated vectors of high dimensionality. Fig. 1 outlines the conventional approach of using n-gram statistics with SOMs. First, for the input symbolic data we calculate n-gram statistics. The size of the vector **s**, which contains the n-gram statistics, will be determined by the size of the data's alphabet a and the chosen n. Next, the conventional approach will be to use **s** as an input **x** to either train or test the SOM (the red vertical line in Fig. 1). The approach proposed in this paper modifies the conventional approach by introducing an additional

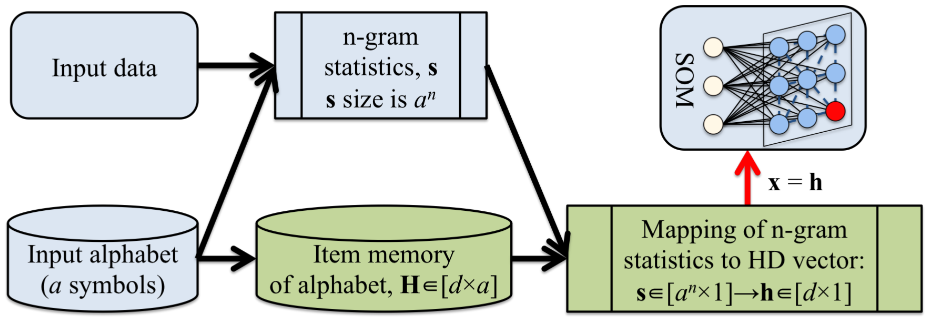
Input data
n-gram statistics, s
s size is a^n
SOM
x = h
Input alphabet (a symbols)
Item memory of alphabet, H∈[d×a]
Mapping of n-gram statistics to HD vector: s∈[a^n×1]→h∈[d×1]

Fig. 2. Outline of the proposed approach.

step, as outlined in Fig. 2. The blocks in green denote the elements of the introduced additional step. For example, the item memory stores the distributed representations of the alphabet. In the proposed approach, before providing **s** to the SOM, **s** is mapped to a distributed representation **h**, which is then used as an input to the SOM (the red vertical line in Fig. 2).

The paper is structured as follows. Section 2 describes the related work. Section 3 presents the methods used in this paper. Section 4 reports the results of the experiments. The conclusions follow in Section 5.

2 Related Work

The SOM algorithm [25] was originally designed for metric vector spaces. It develops a non-linear mapping of a high-dimensional input space to a two-dimensional map of nodes using competitive, unsupervised learning. The output of the algorithm, the SOM represents an ordered topology of complex entities [26], which is then used for visualization, clustering, classification, profiling, or prediction. Multiple variants of the SOM algorithm that overcome structural, functional and application-focused limitations have been proposed. Among the key developments are the Generative Topographic Mapping based on non-linear latent variable modeling [4], the Growing SOM (GSOM) that addresses the predetermined size constraints [1], the TASOM based on adaptive learning rates and neighborhood sizes [40], the WEBSOM for text analysis [17], and the IKASL algorithm [5] that addresses challenges in incremental unsupervised learning. Moreover, recently an important direction is the simplification of the SOM algorithm [2, 19, 39] for improving its speed and power-efficiency.

However, only a limited body of work has explored the plausibility of the SOM beyond its original metric vector space. In contrast to a metric vector space, a symbolic data space is a non-vectorial representation that possesses an internal variation and structure which must be taken into account in computations. Records in a symbolic dataset are not limited to a single value, for instance, each data point can be a hypercube in p-dimensional space or Cartesian product of distribution. In [26], authors make the first effort to apply SOM algorithm to

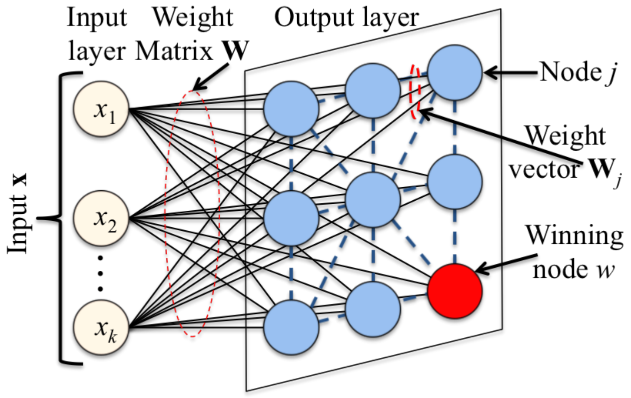

Fig. 3. Illustration of a Self-Organizing Map with nine nodes organized according to the grid topology.

symbol strings, the primary challenges were the discrete nature of data points and adjustments required for the learning rule, addressed using the generalized means/medians and batch map principle. Research reported in [42] takes a more direct approach to n-gram modeling of HTTP requests from network logs. Feature matrices are formed by counting the occurrences of n-characters corresponding to each array in the HTTP request, generating a memory-intensive feature vector of length 256^n. Feature matrices are fed into a variant of the SOM, Growing Hierarchical SOMs [9] to detect anomalous requests. Authors report both accuracy and precision of 99.9% on average, when using bigrams and trigrams. Given the limited awareness and availability of research into unsupervised machine learning on symbolic data, coupled with the increasing complexity of raw data [27], it is pertinent to investigate the functional synergies between hyperdimensional computing and the principles of SOMs.

3 Methods

This section presents the methods used in this paper. We describe: the basics of the SOM algorithm; the process of collecting n-gram statistics; the basics of hyperdimensional computing; and the mapping of n-gram statistics to the distributed representation using hyperdimensional computing.

3.1 Self-Organizing Maps

A SOM [25] (see Fig. 3) consists of a set of nodes arranged in a certain topology (e.g., a rectangular or a hexagonal grid or even a straight line). Each node j is characterized by a weight vector of dimensionality equal the dimensionality of an input vector (denoted as $\mathbf{x}$). The weight vectors are typically initialized at random. Denote a $u \times k$ matrix of k-dimensional weight vectors of u nodes in a

SOM as $\mathbf{W}$. Also denote a weight vector of node j as $\mathbf{W}_j$ and i'th positions of this vector as $\mathbf{W}_{ji}$. One of the main steps in the SOM algorithm is for a given input vector $\mathbf{x}$ to identify the the winning node, which has the closest weight vector to $\mathbf{x}$. Computation of a distance between the input $\mathbf{x}$ and the weight vectors in $\mathbf{W}$, the winner takes all procedure as well as the weight update rule are the main components of SOM logic. They are outlined in the text below.

In order to compare $\mathbf{x}$ and $\mathbf{W}_j$, a similarity measure is needed. The SOM uses Euclidian distance:

$$D(\mathbf{x}, \mathbf{W}_j) = \sqrt{\sum_{i=1}^{i=k} (\mathbf{x}_i - \mathbf{W}_{ji})^2}, \tag{1}$$

where $\mathbf{x}_i$ and $\mathbf{W}_{ji}$ are the corresponding values of ith positions. The winning node (denoted as w) is defined as a node with the lowest Euclidian distance to the input $\mathbf{x}$.

In the SOM, a neighborhood $\mathcal{M}$ of nodes around the winning node w is selected and updated; the size of the neighborhood progressively decreases:

$$\gamma(j, w, t) = e^{-l(j,w)/2\sigma(t)^2}, \tag{2}$$

where $l(j, w)$ is the lateral distance between a node j and the winning node w on the SOM's topology; $\sigma(t)$ is the decreasing function, which depends of the current training iteration t. If a node j is within the neighborhood $\mathcal{M}$ of w then the weight vector $\mathbf{W}_j$ is updated with:

$$\triangle \mathbf{W}_j = \eta(t)\gamma(j, w, t)(\mathbf{x} - \mathbf{W}_j), \tag{3}$$

where $\eta(t)$ denotes the learning rate decreasing with increasing t. During an iteration t, the weights are updated for all available training inputs $\mathbf{x}$. The training process usually runs for T iterations.

Once the SOM has been trained it could be used in the operating phase. The operating phase is very similar to that of the training one except that the weights stored in $\mathbf{W}$ are kept fixed. For a given input $\mathbf{x}$, the SOM identifies the winning node w. This information is used depending on the task at hand. For example, in clustering tasks, a node could be associated with a certain region. In this paper, we consider the classification task, and therefore, each node would have an assigned classification label.

3.2 n-gram statistics

In order to calculate n-gram statistics for the input symbolic data $\mathcal{D}$, which is described by the alphabet of size a, we first initialize an empty vector $\mathbf{s}$. This vector will store the n-gram statistics for $\mathcal{D}$, where the ith position in $\mathbf{s}$ corresponds to an n-gram $\boldsymbol{\mathcal{N}}_i = \langle \mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_n, \rangle$ from the set $\boldsymbol{\mathcal{N}}$ of all unique n-grams; $\mathcal{S}_j$ corresponds to a symbol in jth position of $\boldsymbol{\mathcal{N}}_i$. The value $\mathbf{s}_i$ indicates the number of times $\boldsymbol{\mathcal{N}}_i$ was observed in the input symbolic data $\mathcal{D}$. The dimensionality of

$\mathbf{s}$ is equal to the total number of n-grams in $\mathcal{N}$, which in turn depends on a and n (size of n-grams) and is calculated as a^n (i.e., $\mathbf{s} \in [a^n \times 1]$). The n-gram statistics $\mathbf{s}$ is calculated via a single pass through $\mathcal{D}$ using the overlapping sliding window of size n, where for an n-gram observed in the current window the value of its corresponding position in $\mathbf{s}$ (i.e., counter) is incremented by one. Thus, $\mathbf{s}$ characterizes how many times each n-gram in $\mathcal{N}$ was observed in $\mathcal{D}$.

3.3 Hyperdimensional computing

Hyperdimensional computing [16, 31, 33, 34] also known as Vector Symbolic Architectures is a family of bio-inspired methods of representing and manipulating concepts and their meanings in a high-dimensional space. Hyperdimensional computing finds its applications in, for example, cognitive architectures [10], natural language processing [20, 38], biomedical signal processing [22, 35], approximation of conventional data structures [23, 30], and for classification tasks [18], such as gesture recognition [24], physical activity recognition [37], fault isolation [21]. Vectors of high (but fixed) dimensionality (denoted as d) are the basis for representing information in hyperdimensional computing. These vectors are often referred to as high-dimensional vectors or HD vectors. The information is distributed across HD vectors positions, therefore, HD vectors use distributed representations. Distributed representations [13] are contrary to the localist representations (which are used in the conventional n-gram statistics) since any subset of the positions can be interpreted. In other words, a particular position of an HD vector does not have any interpretable meaning – only the whole HD vector can be interpreted as a holistic representation of some entity, which in turn bears some information load. In the scope of this paper, symbols of the alphabet are the most basic components of a system and their atomic HD vectors are generated randomly. Atomic HD vectors are stored in the so-called item memory, which in its simplest form is a matrix. Denote the item memory as $\mathbf{H}$, where $\mathbf{H} \in [d \times a]$. For a given symbol $\mathcal{S}$ its corresponding HD vector from $\mathbf{H}$ is denoted as $\mathbf{H}_{\mathcal{S}}$. Atomic HD vectors in $\mathbf{H}$ are bipolar ($\mathbf{H}_{\mathcal{S}} \in \{-1, +1\}^{[d \times 1]}$) and random with equal probabilities for $+1$ and -1. It is worth noting that an important property of high-dimensional spaces is that with an extremely high probability all random HD vectors are dissimilar to each other (quasi orthogonal).

In order to manipulate atomic HD vectors hyperdimensional computing defines operations and a similarity measure on HD vectors. In this paper, we use the cosine similarity for characterizing the similarity. Three key operations for computing with HD vectors are bundling, binding, and permutation.

The binding operation is used to bind two HD vectors together. The result of binding is another HD vector. For example, for two symbols $\mathcal{S}_1$ and $\mathcal{S}_2$ the result of binding of their HD vectors (denotes as $\mathbf{b}$) is calculated as follows:

$$\mathbf{b} = \mathbf{H}_{\mathcal{S}_1} \odot \mathbf{H}_{\mathcal{S}_2}, \tag{4}$$

where the notation $\odot$ for the Hadamard product is used to denote the binding operation since this paper uses positionwise multiplication for binding. An important property of the binding operation is that the resultant HD vector $\mathbf{b}$ is dissimilar to the HD vectors being bound, i.e., the cosine similarity between $\mathbf{b}$ and $\mathbf{H}_{\mathcal{S}_1}$ or $\mathbf{H}_{\mathcal{S}_2}$ is approximately 0.

An alternative approach to binding when there is only one HD vector is to permute (rotate) the positions of the HD vector. It is convenient to use a fixed permutation (denoted as ρ) to bind a position of a symbol in a sequence to an HD vector representing the symbol in that position. Thus, for a symbol $\mathcal{S}_1$ the result of permutation of its HD vector (denotes as $\mathbf{r}$) is calculated as follows:

$$\mathbf{r} = \rho(\mathbf{H}_{\mathcal{S}_1}). \tag{5}$$

Similar to the binding operation, the resultant HD vector $\mathbf{r}$ is dissimilar to $\mathbf{H}_{\mathcal{S}_1}$.

The last operation is called bundling. It is denoted with $+$ and implemented via positionwise addition. The bundling operation combines several HD vectors into a single HD vector. For example, for $\mathcal{S}_1$ and $\mathcal{S}_2$ the result of bundling of their HD vectors (denotes as $\mathbf{a}$) is simply:

$$\mathbf{a} = \mathbf{H}_{\mathcal{S}_1} + \mathbf{H}_{\mathcal{S}_2}. \tag{6}$$

In contrast to the binding and permutation operations, the resultant HD vector $\mathbf{a}$ is similar to all bundled HD vectors, i.e., the cosine similarity between $\mathbf{b}$ and $\mathbf{H}_{\mathcal{S}_1}$ or $\mathbf{H}_{\mathcal{S}_1}$ is more than 0. Thus, the bundling operation allows storing information in HD vectors [11]. Moreover if several copies of any HD vector are included (e.g., $\mathbf{a} = 3\mathbf{H}_{\mathcal{S}_1} + \mathbf{H}_{\mathcal{S}_2}$), the resultant HD vector is more similar to the dominating HD vector than to other components.

3.4 Mapping of n-gram statistics with hyperdimensional computing

The mapping of n-gram statistics into distributed representation using hyperdimensional computing was first shown in [15]. At the initialization phase, the random item memory $\mathbf{H}$ is generated for the alphabet. A position of symbol $\mathcal{S}_j$ in $\boldsymbol{\mathcal{N}}_i$ is represented by applying the fixed permutation ρ to the corresponding atomic HD vector $\mathbf{H}_{\mathcal{S}_j}$ j times, which is denoted as $\rho^j(\mathbf{H}_{\mathcal{S}_j})$. Next, a single HD vector for $\boldsymbol{\mathcal{N}}_i$ (denoted as $\mathbf{m}_{\boldsymbol{\mathcal{N}}_i}$) is formed via the consecutive binding of permuted HD vectors $\rho^j(\mathbf{H}_{\mathcal{S}_j})$ representing symbols in each position j of $\boldsymbol{\mathcal{N}}_i$. For example, for the trigram 'cba' will be mapped to its HD vector as follows: $\rho^1(\mathbf{H}_\mathrm{c}) \odot \rho^2(\mathbf{H}_\mathrm{b}) \odot \rho^3(\mathbf{H}_\mathrm{a})$. In general, the process of forming HD vector of an n-gram can be formalized as follows:

$$\mathbf{m}_{\boldsymbol{\mathcal{N}}_i} = \prod_{j=1}^{n} \rho^j(\mathbf{H}_{\mathcal{S}_j}), \tag{7}$$

where $\prod$ denotes the binding operation (positionwise multiplication) when applied to n HD vectors.

Once it is known how to map a particular n-gram to an HD vector, mapping the whole n-gram statistics $\mathbf{s}$ is straightforward. HD vector $\mathbf{h}$ corresponding to

$\mathbf{s}$ is created by bundling together all n-grams observed in the data, which is expressed as follows:

$$\mathbf{h} = \sum_{i=1}^{a^n} \mathbf{s}_i \mathbf{m}_{\mathcal{N}_i} = \sum_{i=1}^{a^n} \mathbf{s}_i \prod_{j=1}^{n} \rho^j(\mathbf{H}_{\mathcal{S}_j}), \tag{8}$$

where $\sum$ denotes the bundling operation when applied to several HD vectors. Note that $\mathbf{h}$ is not bipolar, therefore, in the experiments below we normalized it by its ℓ_2 norm.

4 Experimental results

This section describes the experimental results studying several configurations of the proposed approach and comparing it with the results obtained for the conventional n-gram statistics. We slightly modified the experimental setup from that used in [15], where the task was to identify a language of a given text sample (i.e., for a string of symbols). The language recognition was done for 21 European languages. The list of languages is as follows: Bulgarian, Czech, Danish, German, Greek, English, Estonian, Finnish, French, Hungarian, Italian, Latvian, Lithuanian, Dutch, Polish, Portuguese, Romanian, Slovak, Slovene, Spanish, Swedish. The training data is based on the Wortschatz Corpora [32]. The average size of a language's corpus in the training data was $1,085,637.3 \pm 121,904.1$ symbols. It is worth noting, that in the experiments reported in [15] the whole training corpus of a particular language was used to estimate the corresponding n-grams statistics. While in this study, in order to enable training of SOMs, each language corpus was divided into samples where the length of each sample was set to $1,000$ symbols. The total number of samples in the training data was $22,791$. The test data is based on the Europarl Parallel Corpus[4]. The test data also represent 21 European languages. The total number of samples in the test data was $21,000$, where each language was represented with $1,000$ samples. Each sample in the test data corresponds to a single sentence. The average size of a sample in the test data was 150.3 ± 89.5 symbols.

The data for each language was preprocessed such that the text included only lower case letters and spaces. All punctuation was removed. Lastly, all text used the 26-letter ISO basic Latin alphabet, i.e., the alphabet for both training and test data was the same and it included 27 symbols. For each text sample the n-gram statistics (either conventional or mapped to the distributed representation) was obtained, which was then used as input $\mathbf{x}$ when training or testing SOMs. Since each sample was preprocessed to use the alphabet of only $a = 27$ symbols, the conventional n-gram statistics input is 27^n dimensional (e.g., $k = 729$ when $n = 2$) while the dimensionality of the mapped n-gram statistics depends on the dimensionality of HD vectors d (i.e., $k = d$). In all experiments reported in this paper, we used the standard SOMs implementation, which is a part of the Deep Learning Toolbox in MATLAB R2018B (Mathworks Inc, Natick, Ma.)

[4] Available online at http://www.statmt.org/europarl/.

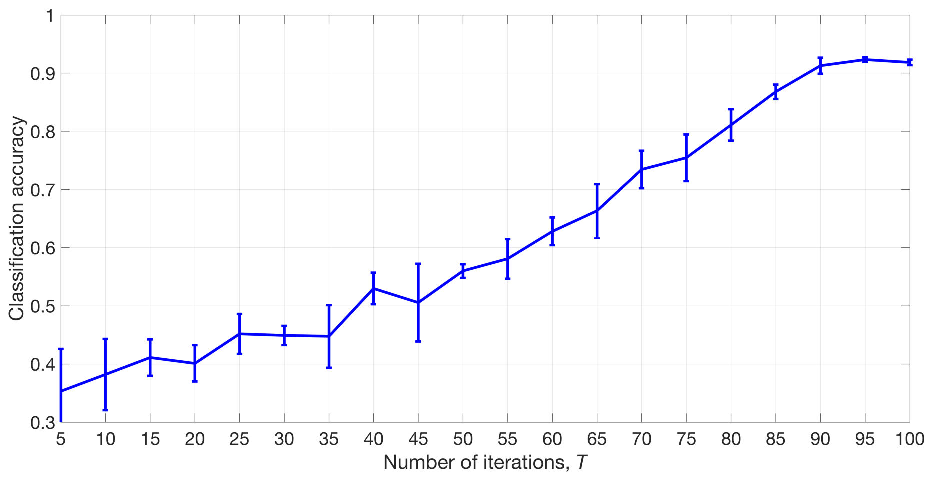

Fig. 4. The classification accuracy of the SOM trained on the conventional bigram statistics ($n = 2$; $k = 729$) against the number of training iterations T. The grid size was set to ten ($u = 100$). T varied in the range $[5, 100]$ with step 5.

During the experiments, certain parameters SOM were fixed. In particular, the topology of SOMs was set to the standard grid topology. The initial size of the neighborhood was always fixed to ten. The size of the neighborhood and the learning rate were decreasing progressively with training according to the default rules of the used implementation. In all simulations, a SOM was trained for a given number of iterations T, which was set according to an experiment reported in Fig. 4. All reported results were averaged across five independent simulations. The bars in the figure show standard deviations.

Recall that SOMs are suited for the unsupervised training, therefore, an extra mechanism is needed to use them in supervised tasks such as the considered language recognition task, i.e., once the SOM is trained there is still a need to assign a label to each trained node. After training a SOM for T iterations using all $22,791$ training samples, the whole training data were presented to the trained SOM one more time without modifying $\mathbf{W}$. Labels for the training data were used to collect the statistics for the winning nodes. The nodes were assigned the labels of the languages dominating in the collected statistics. If a node in the trained SOM was never chosen as the winning node for the training samples (i.e., its statistics information is empty) then this node was ignored during the testing phase. During the testing phase, $21,000$ samples of the test data were used to assess the trained SOM. For each sample in the test data, the winning node was determined. The test sample then was assigned the language label corresponding to its winning node. The classification accuracy was calculated using the SOM predictions and the ground truth of the test data. The accuracy was used as the main performance metric for evaluation and comparison of different SOMs. It is worth emphasizing that the focus of experiments is not on achieving the highest

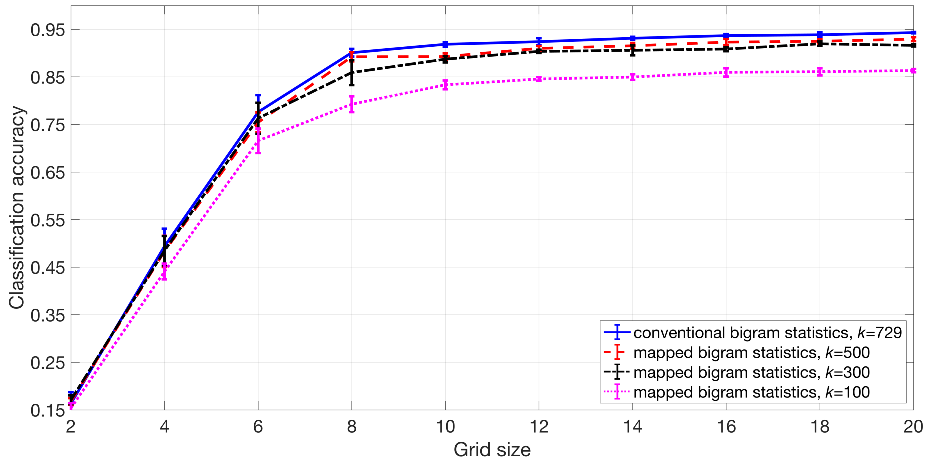

Fig. 5. The classification accuracy of the SOM against the grid size for the case of bigram statistics. The grid size varied in the range $[2, 20]$ with step 2.

possible accuracy but on a comparative analysis of SOMs with the conventional n-gram statistics versus SOMs with the mapped n-gram statistics with varying d. However, it is worth noting that the accuracy, obtained when collecting an n-gram statistics profile for each language [15, 36] for $n = 2$ and $n = 3$ and using the nearest neighbor classifier, was 0.945 and 0.977 respectively. Thus, the results presented below for SOMs match the ones obtained with the supervised learning on bigrams when the number of nodes is sufficiently high. In the case of trigrams, the highest accuracy obtained with SOMs was slightly (about 0.02) lower. While SOMs not necessarily achieve the highest accuracy compared to the supervised methods, their important advantage is data visualization. For example, in the considered task one could imagine using the trained SOM for identifying the clusters typical for each language and even reflecting on their relative locations on the map.

The experiment in Fig. 4 presents the classification accuracy of the SOM trained on the conventional bigram statistics against T. The results demonstrated that the accuracy increased with the increased number T. Moreover, for higher values of T the predictions are more stable. The performance started to saturate at T more than 90, therefore, in the other experiments the value of T was fixed to 100.

The grid size varied in the range $[2, 20]$ with step 2, i.e, the number of nodes u varied between 4 and 400. In Fig. 5 the solid curve corresponds to the SOM trained on the conventional bigram statistics. The dashed, dash-dot, and dotted curves correspond to the SOMs trained on the mapped bigram statistics with $k = d = 500$, $k = d = 300$, and $k = d = 100$ respectively.

The experiment presented in Fig. 5 studied the classification accuracy of the SOM against the grid size for the case of bigram statistics. Note that the

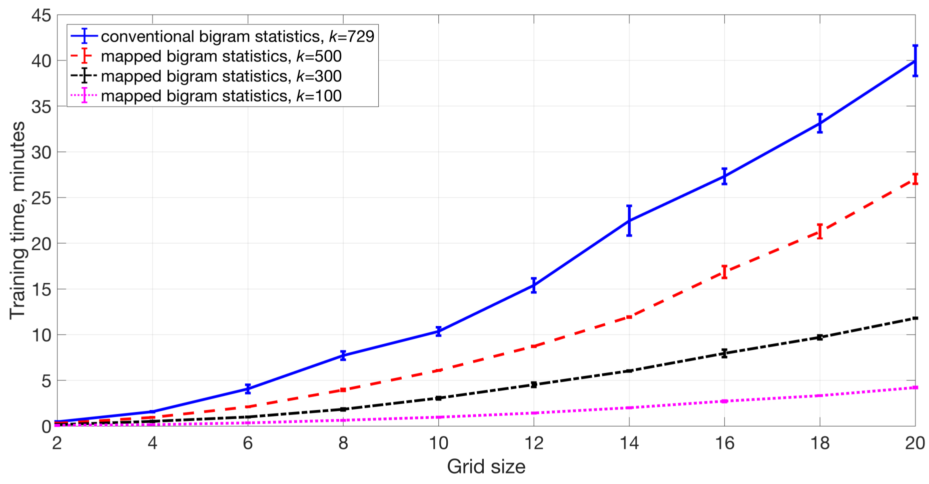

Fig. 6. The training time of the SOM against the grid size for the case of bigram statistics. The grid size varied in the range $[2, 20]$ with step 2.

number of nodes u in the SOM is proportional to the square of the grid size. For example, when the gris size equals 2 the SOM has $u = 4$ nodes while when it equals 20 the SOM has $u = 400$ nodes. The results in Fig. 5 demonstrated that the accuracy of all considered SOMs improves with the increased grid size. It is intuitive that all SOMs with grid sizes less than five performed poorly since the number of nodes in SOMs was lower than the number of different languages in the task. Nevertheless, the performance of all SOMs was constantly improving with the increased grid size, but the accuracy started to saturate at about 100 nodes. Moreover, increasing the dimensionality of HD vectors d was improving the accuracy. Note, however, that there was a better improvement when going from $d = 100$ to $d = 300$ compared to increasing d from 300 to 500. The performance of the conventional bigram statistics was already approximated well even when $d = 300$; for $d = 500$ the accuracy was just slightly worse than that of the conventional bigram statistics.

It is important to mention that the usage of the mapped n-grams statistics allows decreasing the size of $\mathbf{W}$ in proportion to d/a^n. Moreover, it allows decreasing the training time of SOMs. The experiment in Fig. 6 presents the training time of the SOM against the grid size for the case of bigram statistics. Fig. 6 corresponds to that of Fig. 5. The number of training iterations was fixed to $T = 100$. For example, for grid size 16 the average training time on a laptop for $k = d = 100$ was 2.7 minutes (accuracy 0.86); for $k = d = 300$ it was 8.0 minutes (accuracy 0.91); for $k = d = 500$ it was 16.9 minutes (accuracy 0.92); and for $k = a^n = 729$ it was 27.3 minutes (accuracy 0.93). Thus, the usage of the mapping allows the trade-off between the obtained accuracy and the required computational resources.

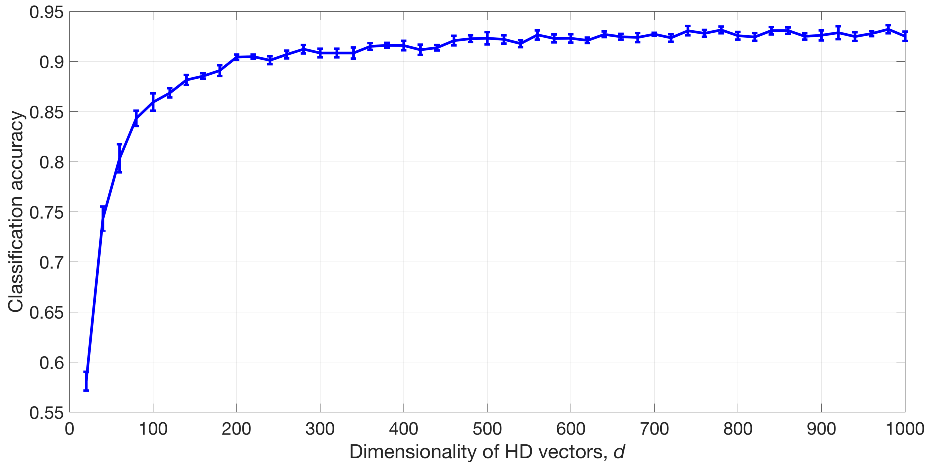

Fig. 7. The classification accuracy of the SOM trained on the mapped bigram statistics ($n = 2$) against the dimensionality of HD vectors d ($k = d$). The grid size was set to 16 ($u = 256$). The number of training iterations T was fixed to 100.

In order to observe a more detailed dependency between the classification accuracy and the dimensionality of distributed representations d of the mapped n-gram statistics, an additional experiment was done. Fig. 7 depicts the results. The dimensionality of distributed representations d varied in the range $[20, 1000]$ with step 20. It is worth mentioning that even for small dimensionalities ($d < 100$), the accuracy is far beyond random. The results in Fig. 7 are consistent with the observations in Fig. 5 in a way that the accuracy was increasing with the increased d. The performance saturation begins for the values above 200 and the improvements beyond $d = 500$ look marginal. Thus, we experimentally observed that the quality of mappings grows with d, however, after a certain saturation point increasing d further becomes impractical.

The last experiment in Fig. 8 is similar to Fig. 5 but it studied the classification accuracy for the case of trigram statistics ($n = 3$). The grid size varied in the range $[2, 20]$ with step 2. The solid curve corresponds to the SOM trained on the conventional trigram statistics ($k = 27^3 = 19,683$). The dashed and dash-dot curves correspond to the SOMs trained on the mapped trigram statistics with $k = d = 5,000$ and $k = d = 1,000$ respectively. The results in Fig. 8 are consistent with the case of bigrams. The classification of SOMs was better for higher d and even when $d < a^n$ the accuracy was approximated well.

5 Conclusions

This paper presented an approach for the mapping of n-gram statistics into vectors of fixed arbitrary dimensionality, which does not depend on the size of n-grams n. The mapping is aided by hyperdimensional computing a bio-

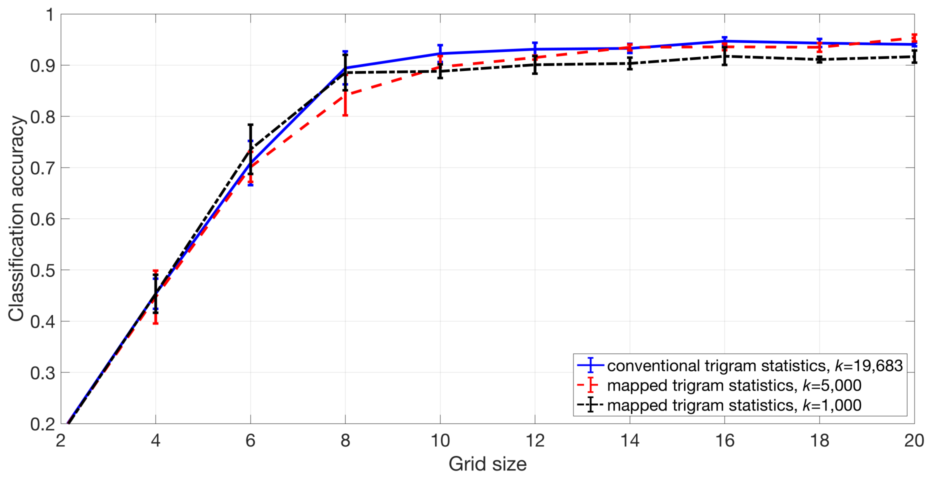

Fig. 8. The classification accuracy of the SOM against the grid size for the case of trigram statistics ($n = 3$). The number of training iterations T was fixed to 100.

inspired approach for computing with large random vectors. Mapped in this way n-gram statistics is used as the input to Self-Organized Maps. This novel for Self-Organized Maps step allows removing the computational bottleneck caused by the exponentially growing dimensionality of n-gram statistics with increased n. While preserving the performance of the trained Self-Organized Maps (as demonstrated in the languages recognition task) the presented approach results in reduced memory consumption due to smaller weight matrix (proportional to d and u) and shorter training times. The main limitation of this study is that we have validated the proposed approach only on a single task when using the conventional Self-Organized Maps. However, it is worth noting that the proposed approach could be easily used for other modifications of the conventional Self-Organizing Maps such as Growing Self-Organizing Maps [1], where dynamic topology preservation facilitates unconstrained learning. This is in contrast to a fixed-structure feature map as the map itself is defined by the unsupervised learning process of the feature vectors. We intend to investigate distributed representation of n-gram statistics in structure-adapting feature maps in future work.

References

1. Alahakoon, D., Halgamuge, S., Srinivasan, B.: Dynamic Self-Organizing Maps with Controlled Growth for Knowledge Discovery. IEEE Transactions on Neural Networks **11**(3), 601–614 (2000)
2. Appiah, K., Hunter, A., Dickinson, P., Meng, H.: Implementation and Applications of Tri-State Self-Organizing Maps on FPGA. IEEE Transactions on Circuits and Systems for Video Technology **22**(8), 1150–1160 (2012)

3. Bandaragoda, T.R., De Silva, D., Alahakoon, D.: Automatic Event Detection in Microblogs using Incremental Machine Learning. Journal of the Association for Information Science and Technology **68**(10), 2394–2411 (2017)
4. Bishop, C.M., Svensén, M., Williams, C.K.: GTM: The Generative Topographic Mapping. Neural computation **10**(1), 215–234 (1998)
5. De Silva, D., Alahakoon, D.: Incremental Knowledge Acquisition and Self Learning from Text. In: International Joint Conference on Neural Networks (IJCNN). pp. 1–8. IEEE (2010)
6. De Silva, D., Alahakoon, D., Yu, X.: A Data Fusion Technique for Smart Home Energy Management and Analysis. In: Annual Conference of the IEEE Industrial Electronics Society (IECON). pp. 4594–4600 (2016)
7. De Silva, D., Ranasinghe, W., Bandaragoda, T., Adikari, A., Mills, N., Iddamalgoda, L., Alahakoon, D., Lawrentschuk, N., Persad, R., Osipov, E., Gray, R., Bolton, D.: Machine Learning to Support Social Media Empowered Patients in Cancer Care and Cancer Treatment Decisions. PloS One **13**(10), 1–10 (2018)
8. De Silva, D., Yu, X., Alahakoon, D., Holmes, G.: A Data Mining Framework for Electricity Consumption Analysis from Meter Data. IEEE Transactions on Industrial Informatics **7**(3), 399–407 (2011)
9. Dittenbach, M., Merkl, D., Rauber, A.: The Growing Hierarchical Self-Organizing Map. In: International Joint Conference on Neural Networks (IJCNN). vol. 6, pp. 15–19 (2000)
10. Eliasmith, C.: How to Build a Brain. Oxford University Press (2013)
11. Frady, E.P., Kleyko, D., Sommer, F.T.: A Theory of Sequence Indexing and Working Memory in Recurrent Neural Networks. Neural Computation **30**, 1449–1513 (2018)
12. Hazan, H., Saunders, D.J., Sanghavi, D.T., Siegelmann, H.T., Kozma, R.: Unsupervised Learning with Self-Organizing Spiking Neural Networks. In: International Joint Conference on Neural Networks (IJCNN). pp. 1–6 (2018)
13. Hinton, G., McClelland, J., Rumelhart, D.: Distributed Representations. In: Rumelhart, D., McClelland, J. (eds.) Parallel Distributed Processing. Explorations in the Microstructure of Cognition. Volume 1. Foundations. pp. 77–109. MIT Press (1986)
14. Jayaratne, M., Alahakoon, D., De Silva, D., Yu, X.: Bio-Inspired Multisensory Fusion for Autonomous Robots. In: Annual Conference of the IEEE Industrial Electronics Society (IECON). pp. 3090–3095 (2018)
15. Joshi, A., Halseth, J., Kanerva, P.: Language Geometry Using Random Indexing. In: Quantum Interaction (QI). pp. 265–274 (2016)
16. Kanerva, P.: Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors. Cognitive Computation **1**(2), 139–159 (2009)
17. Kaski, S., Honkela, T., Lagus, K., Kohonen, T.: WEBSOM–Self-organizing maps of document collections1. Neurocomputing **21**(1-3), 101–117 (1998)
18. Kleyko, D., Osipov, E.: Brain-like Classifier of Temporal Patterns. In: International Conference on Computer and Information Sciences (ICCOINS). pp. 104–113 (2014)
19. Kleyko, D., Osipov, E., De Silva, D., Wiklund, U., Alahakoon, D.: Integer Self-Organizing Maps for Digital Hardware. In: International Joint Conference on Neural Networks (IJCNN). pp. 1–8 (2019)
20. Kleyko, D., Osipov, E., Gayler, R.: Recognizing Permuted Words with Vector Symbolic Architectures: A Cambridge Test for Machines. Procedia Computer Science **88**, 169–175 (2016)

21. Kleyko, D., Osipov, E., Papakonstantinou, N., Vyatkin, V.: Hyperdimensional Computing in Industrial Systems: The Use-Case of Distributed Fault Isolation in a Power Plant. IEEE Access **6**, 30766–30777 (2018)
22. Kleyko, D., Osipov, E., Wiklund, U.: A Hyperdimensional Computing Framework for Analysis of Cardiorespiratory Synchronization During Paced Deep Breathing. IEEE Access **7**, 34403–34415 (2019)
23. Kleyko, D., Rahimi, A., Gayler, R., Osipov, E.: Autoscaling Bloom Filter: Controlling Trade-off Between True and False Positives. Neural Computing and Applications pp. 1–10 (2019)
24. Kleyko, D., Rahimi, A., Rachkovskij, D., Osipov, E., Rabaey, J.: Classification and Recall with Binary Hyperdimensional Computing: Trade-offs in Choice of Density and Mapping Characteristic. IEEE Transactions on Neural Networks and Learning Systems **29**(12), 5880–5898 (2018)
25. Kohonen, T.: Self-Organizing Maps. Springer Series in Information Sciences (2001)
26. Kohonen, T., Somervuo, P.: Self-Organizing Maps of Symbol Strings. Neurocomputing **21**(1-3), 19–30 (1998)
27. Kusiak, A.: Smart Manufacturing must Embrace Big Data. Nature News **544**(7648), 23 (2017)
28. Nallaperuma, D., De Silva, D., Alahakoon, D., Yu, X.: Intelligent Detection of Driver Behavior Changes for Effective Coordination Between Autonomous and Human Driven Vehicles. In: Annual Conference of the IEEE Industrial Electronics Society (IECON). pp. 3120–3125 (2018)
29. Nawaratne, R., Bandaragoda, T., Adikari, A., Alahakoon, D., De Silva, D., Yu, X.: Incremental Knowledge Acquisition and Self-Learning for Autonomous Video Surveillance. In: Annual Conference of the IEEE Industrial Electronics Society (IECON). pp. 4790–4795 (2017)
30. Osipov, E., Kleyko, D., Legalov, A.: Associative Synthesis of Finite State Automata Model of a Controlled Object with Hyperdimensional Computing. In: Annual Conference of the IEEE Industrial Electronics Society (IECON). pp. 3276–3281 (2017)
31. Plate, T.A.: Holographic Reduced Representations: Distributed Representation for Cognitive Structures. Stanford: Center for the Study of Language and Information (CSLI) (2003)
32. Quasto, U., Richter, M., Biemann, C.: Corpus Portal for Search in Monolingual Corpora. In: Fifth International Conference on Language Resources and Evaluation (LREC). pp. 1799–1802 (2006)
33. Rachkovskij, D.A.: Representation and Processing of Structures with Binary Sparse Distributed Codes. IEEE Transactions on Knowledge and Data Engineering **3**(2), 261–276 (2001)
34. Rahimi, A., Datta, S., Kleyko, D., Frady, E.P., Olshausen, B., Kanerva, P., Rabaey, J.M.: High-dimensional Computing as a Nanoscalable Paradigm. IEEE Transactions on Circuits and Systems I: Regular Papers **64**(9), 2508–2521 (2017)
35. Rahimi, A., Kanerva, P., Benini, L., Rabaey, J.M.: Efficient Biosignal Processing Using Hyperdimensional Computing: Network Templates for Combined Learning and Classification of ExG Signals. Proceedings of the IEEE **107**(1), 123–143 (2019)
36. Rahimi, A., Kanerva, P., Rabaey, J.: A Robust and Energy Efficient Classifier Using Brain-Inspired Hyperdimensional Computing. In: IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED). pp. 64–69 (2016)
37. Rasanen, O., Kakouros, S.: Modeling Dependencies in Multiple Parallel Data Streams with Hyperdimensional Computing. IEEE Signal Processing Letters **21**(7), 899–903 (2014)

38. Recchia, G., Sahlgren, M., Kanerva, P., Jones, M.: Encoding Sequential Information in Semantic Space Models. Comparing Holographic Reduced Representation and Random Permutation. Computational Intelligence and Neuroscience pp. 1–18 (2015)
39. Santana, A., Morais, A., Quiles, M.: An Alternative Approach for Binary and Categorical Self-Organizing Maps. In: International Joint Conference on Neural Networks (IJCNN). pp. 2604–2610 (2017)
40. Shah-Hosseini, H., Safabakhsh, R.: TASOM: a New Time Adaptive Self-Organizing Map. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) **33**(2), 271–282 (2003)
41. Vesanto, J., Alhoniemi, E.: Clustering of the Self-Organizing Map. IEEE Transactions on Neural Networks **11**(3), 586–600 (2000)
42. Zolotukhin, M., Hamalainen, T., Juvonen, A.: Online Anomaly Detection by using n-gram Model and Growing Hierarchical Self-Organizing Maps. In: International Wireless Communications and Mobile Computing Conference (IWCMC). pp. 47–52 (2012)