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-Organized Maps in tasks of 2-D projection of multi-dimensional symbolic data (e.g., for natural language processing such as language classification, topic extraction, etc.). A conventional approach for this type of problems is to use n-gram statistics as a fixed size representation for an input of Self-Organized Maps. The performance bottleneck with n-gram statistics is that the size of representation and as a result the computation time of Self-Organized 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 much fewer 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 [10] 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 [4],

---

autonomous video surveillance [11]; and identification of trends [14]. 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 if 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.

The paper is structured as follows. The outline of the approach is presented in Section 2. Section 3 describes the related work on recent developments in the SOM domain. Section 4 presents the methods used in this paper. Section 5 reports the results of the experiments on the languages recognition task. The conclusions follow in Section 6.

2   Outline of the proposed approach

The core of the proposed approach is in the use of hyperdimensional computing and distributed data representation. Hyperdimensional computing a bio-inspired computational paradigm in which all computations are done with randomly generated vectors of high dimensionality. Fig. 1 outlines the overall pipeline used in this paper. The details of the methods used in the pipeline are presented in Section 4. First, for the input symbolic data we calculate n-gram statistics. The size of the vector $\mathbf{s}$, 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 $\mathbf{s}$ as an input $\mathbf{x}$ to either train or test the SOM (the dashed red line in Fig. 1). In the proposed approach, before providing $\mathbf{s}$ to the SOM, $\mathbf{s}$ is mapped to a distributed representation $\mathbf{h}$ using hyperdimensional computing, which is then used as an input to the SOM (the solid red line in Fig. 1).

3   Related Work

DK: Related work needs repositioning.
Less emphasis on hardware and efficient SOM modifications
More emphasis on the previous usage of SOMs and n-grams together
Potential references:
Visualization and Integration of Databases using Self-Organizing Map
Online Anomaly Detection by Using N-gram Model and Growing Hierarchical Self-Organizing Maps
Self-organizing maps of symbol strings

EO: Agree. Also this section should be quite compact. Basically after the first generic paragraph describe only related works in the area of SOM application on symbolic data

As explicated above, the SOM algorithm [10] endeavours to develop a non-linear mapping of a high-dimensional input space to a two-dimensional map of nodes using competitive, unsupervised learning. The representative topological information of the output map can then be used for dimension reduction, visualization, clustering, classification, profiling, and prediction.

The original SOM has been effectively extended into multiple variants that overcome functional limitations and application-focused limitations. Among the key developments are the Generative Topographic Mapping based on non-linear latent variable modelling [2], the Growing SOM (GSOM) that addresses the predetermined size constraints [1], the TASOM based on adaptive learning rates and neighborhood sizes [13], the WEBSOM for text analysis [9], and the IKASL algorithm [3] that addresses challenges in incremental unsupervised learning.
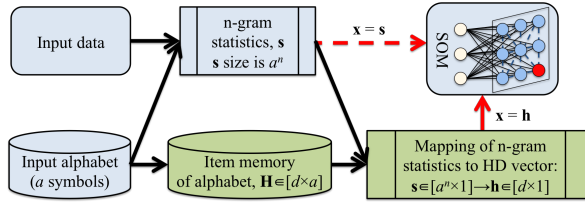

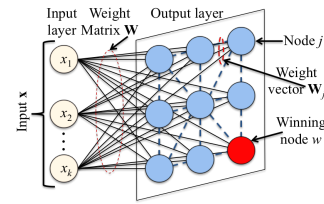**Fig. 1.** Outline of the conventional and proposed approaches.

Fig. 2. Illustration of a SOM with 9 nodes in grid topology.

4   Methods

This section presents the methods used in this paper. First, we describe the basics of the SOM algorithm. Next, we present the process of collecting n-gram statistics. Third, we introduce readers to the basics of hyperdimensional computing,

which are then followed by the mapping of n-gram statistics to the distributed representation using hyperdimensional computing and its operations.

4.1 Self-Organizing Maps

A SOM [10] (see Fig. 2 for an example) 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 following subsections.

Distance Computation and Winner Takes All (WTA) In order to compare $\mathbf{x}$ and $\mathbf{W}_j$, a similarity measure is needed. For this purpose, the SOM uses Euclidian distance $D(\mathbf{x}, \mathbf{W}_j) = \sqrt{\sum_{i=1}^{i=k}(\mathbf{x}_i - \mathbf{W}_{ji})^2}$, where $\mathbf{x}_i$ and $\mathbf{W}_{ji}$ are the corresponding values of ith positions in the input and weight vectors.

The winning node (denoted as w) is defined as a node with the lowest Euclidian distance to the input $\mathbf{x}$.

Neighborhood Selection and Weight Update 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: $\gamma(j, w, t) = e^{-l(j,w)/2\sigma(t)^2}$, 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 $\triangle \mathbf{W}_j = \eta(t)\gamma(j, w, t)(\mathbf{x} - \mathbf{W}_j)$, 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.

4.2 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 ith position in $\mathbf{s}$ corresponds to an n-gram $\boldsymbol{\mathcal{N}}_i = \langle \mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n, \rangle$ 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 $\mathcal{D}$.

4.3 Hyperdimensional computing

Hyperdimensional computing [8] is a family of bio-inspired methods of representing and manipulating concepts and their meanings in a high-dimensional space. 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 evenly distributed across HD vectors positions, therefore, it is said that hyperdimensional computing operates with distributed representations [6]. Distributed representations (i.e., HD vectors) 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.

Usually, HD vectors for the most basic components (atomic HD vectors) of a system (e.g., symbols) are generated randomly. In the scope of this paper, symbols of the alphabet used for extracting n-gram statistics are the most basic components. 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 $\mathbf{H} \in [d \times a]$ and a denotes the number of symbols in the alphabet. 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 ($\mathbf{H}_{\mathcal{S}} \in \{-1, +1\}^{[d \times 1]}$). They are generated randomly 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, e.g., for representing complex structures or sequences of symbols, hyperdimensional computing defines operations and a similarity measure on HD vectors. In this paper, we use the operations defined for the hyperdimensional computing framework called Multiply-Add-Permute (MAP) [5]. The MAP framework uses the cosine similarity for characterizing similarity between two HD vectors. 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:

$$\mathbf{b} = \mathbf{H}_{\mathcal{S}_1} \odot \mathbf{H}_{\mathcal{S}_2}, \tag{1}$$

where the notation $\odot$ for the Hadamard product is used to denote the binding operation because in the MAP framework it is implemented via the positionwise multiplication. 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 $\mathbf{H}_{\mathcal{S}_1}$ is approximately 0.

An alternative approach to binding when there is only one HD vector is to randomly permute (rotate) the positions of the HD vector. It is often 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:

$$\mathbf{r} = \rho(\mathbf{H}_{\mathcal{S}_1}). \tag{2}$$

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:

$$\mathbf{a} = \mathbf{H}_{\mathcal{S}_1} + \mathbf{H}_{\mathcal{S}_2}. \tag{3}$$

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. 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.

The next subsection presents a mapping, which combines atomic HD vectors and all three operations together in order to represent n-gram statistics a distributed representation.

4.4   Mapping of n-gram statistics with hyperdimensional computing

The mapping of n-gram statistics into distributed representation using hyperdimensional computing was first shown in [7]. At the initialization phase, the random item memory $\mathbf{H}$ is generated for the alphabet. A position of symbol $\mathcal{S}_j$ in $\mathcal{N}_i$ 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 $\mathcal{N}_i$ (denoted as $\mathbf{m}_{\mathcal{N}_i}$) is formed via the consecutive binding of permuted HD vectors $\rho^j(\mathbf{H}_{\mathcal{S}_j})$ representing symbols in each position j of $\mathcal{N}_i$. For example, for the trigram 'cba' will be mapped to its HD vector as follows: $\rho^1(\mathbf{H}_c) \odot \rho^2(\mathbf{H}_b) \odot \rho^3(\mathbf{H}_a)$. In general, the process of forming HD vector of an n-gram can be formalized as follows:

$$\mathbf{m}_{\mathcal{N}_i} = \prod_{j=1}^{n} \rho^j(\mathbf{H}_{\mathcal{S}_j}), \tag{4}$$

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:

$$\mathbf{h} = \sum_{i=1}^{a^n} \mathbf{s}_i \mathbf{m}_{\mathcal{N}_i} = \sum_{i=1}^{a^n} \mathbf{s}_i \prod_{j=1}^{n} \rho^j(\mathbf{H}_{\mathcal{S}_j}), \tag{5}$$

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

5 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 [7], 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 [12]. 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 [7] 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

---

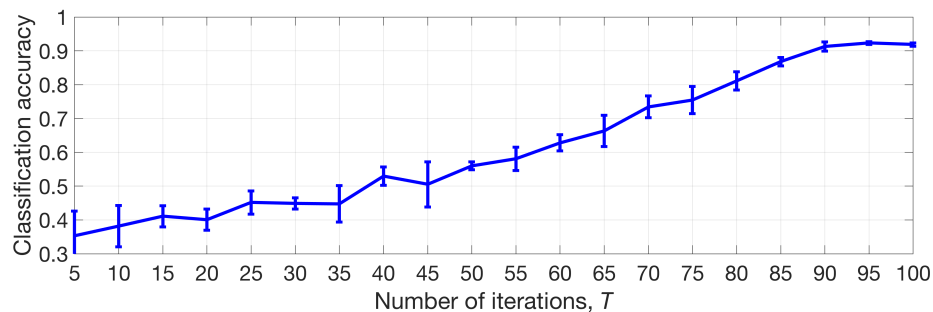[4] Available online at http://www.statmt.org/europarl/.

Fig. 3. 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 ($u = 100$). The results were averaged across five independent runs. The bars show standard deviations.

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.)[5].

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. In the experiments below, the labeling process was automated. After training a SOM for T iterations using all $22,791$ samples from the training data, the whole training data were presented to the trained SOM one more time without modifying the weight matrix $\mathbf{W}$. The supervised information for the training data was used to collect the statistics information 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[6].

---

[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. 3 .

[6] It is worth emphasizing that the focus of experiments is not on achieving the highest 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. The accuracy obtained when collecting a single n-gram statistics per language for $n = 2$ and $n = 3$ was 0.945 and 0.977 respectively.
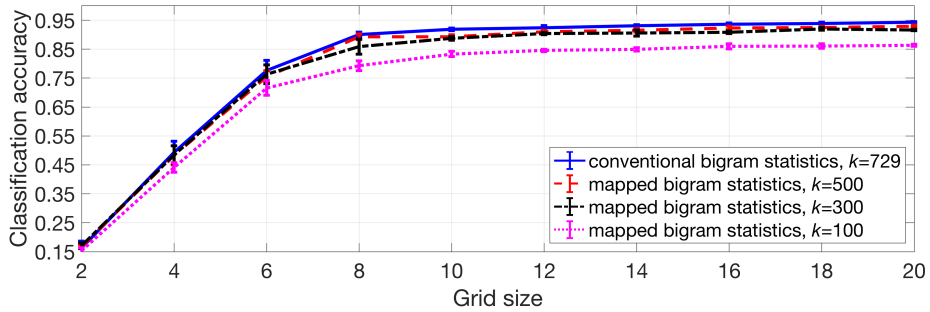
Fig. 4. The classification accuracy of the SOM against the grid size for the case of bigram statistics. Note that the number of nodes u in the SOM is proportional to the square of the grid size. For example, when the gris size equals 2 the SOM has $u = 4$ nodes while when it equals 20 the SOM has $u = 400$ nodes. The results were averaged across five independent runs. The bars show standard deviations.

The experiment in Fig. 3 presents the classification accuracy of the SOM trained on the conventional bigram statistics ($n = 2$; $k = 729$) against the number of training iterations T. The number of iterations T varied in the range $[5, 100]$ with step 5. 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 experiment presented in Fig. 4 studied the classification accuracy of the SOM against the grid size for the case of bigram statistics. The number of training iterations 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. 4 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 results in Fig. 4 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 **W** in proportion to d/a^n. Moreover, it allows de-
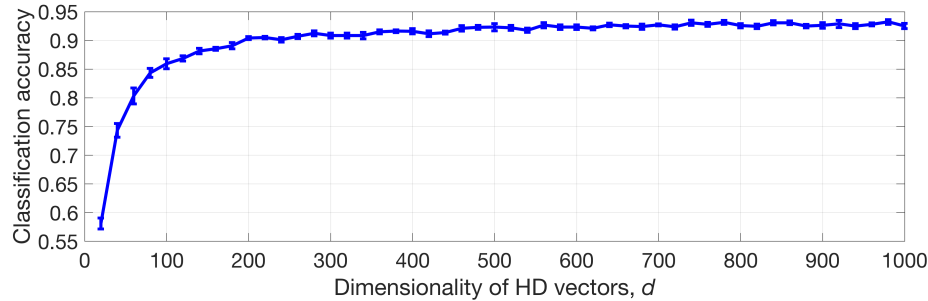
Fig. 5. The classification accuracy of the SOM trained on the mapped bigram statistics ($n = 2$) against the dimensionality of HD vectors d ($k = d$). The grid size was set to 16 ($u = 256$). The number of training iterations T was fixed to 100. The results were averaged across five independent runs. The bars show standard deviations.

creasing the training time of SOMs, e.g., for grid size 16 and $T = 100$ the average training time on a laptop for $k = d = 100$ was 10.6 minutes (accuracy 0.86); for $k = d = 300$ it was 23.7 minutes (accuracy 0.91); for $k = d = 500$ it was 33.2 minutes (accuracy 0.92); and for $k = a^n = 729$ it was 41.9 minutes (accuracy 0.93). Thus, the usage of the mapping allows the trade-off between the obtained accuracy and the required computational resources.

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. 5 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. 5 are consistent with the observations in Fig. 4 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. 6 is similar to Fig. 4 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. 6 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.

6 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
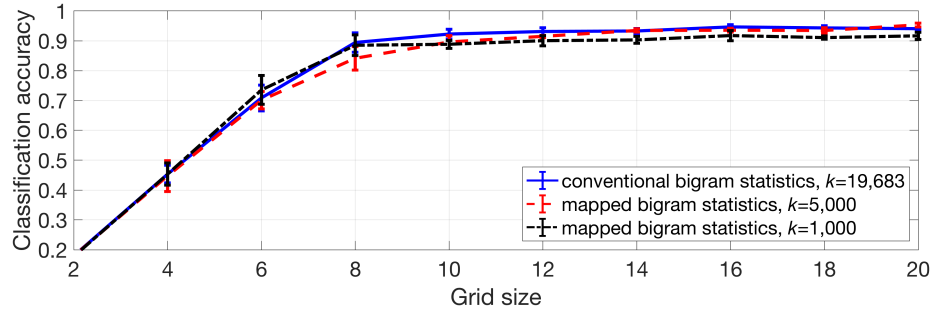
Fig. 6. The classification accuracy of the SOM against the grid size for the case of trigram statistics ($n = 3$). The number of training iterations T was fixed to 100. The results were averaged across five independent runs. The bars show standard deviations.

of n-grams n. The mapping is aided by hyperdimensional computing a bio-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.

References

1. Alahakoon, D., et al.: Dynamic Self-Organizing Maps with Controlled Growth for Knowledge Discovery. IEEE Transactions on Neural Networks **11**(3), 601–614 (2000)
2. Bishop, C.M., et al.: GTM: The generative topographic mapping. Neural computation **10**(1), 215–234 (1998)
3. De Silva, D., Alahakoon, D.: Incremental knowledge acquisition and self learning from text. In: International Joint Conference on Neural Networks. pp. 1–8 (2010)
4. De Silva, D., et al.: A data mining framework for electricity consumption analysis from meter data. IEEE Transactions on Industrial Informatics **7**(3), 399–407 (2011)
5. Gayler, R.W.: Multiplicative binding, representation operators & analogy. In: Advances in analogy research. pp. 1–4 (1998)
6. Hinton, G., et al.: Distributed Representations. In: Parallel Distributed Processing. pp. 77–109 (1986)
7. Joshi, A., et al.: Language Geometry Using Random Indexing. In: Quantum Interaction (QI). pp. 265–274 (2016)
8. Kanerva, P.: An introduction to computing in distributed representation with high-dimensional random vectors. Cognitive Computation **1**(2), 139–159 (2009)
9. Kaski, S., et al.: WEBSOM–Self-organizing maps of document collections. Neurocomputing **21**(1-3), 101–117 (1998)
10. Kohonen, T.: Self-Organizing Maps. Springer Series in Information Sciences (2001)

11. Nawaratne, R., et al.: Incremental knowledge acquisition and self-learning for autonomous video surveillance. In: Annual Conference of IEEE Industrial Electronics Society. pp. 4790–4795 (2017)
12. Quasto, U., et al.: Corpus portal for search in monolingual corpora. In: International Conference on Language Resources and Evaluation. pp. 1799–1802 (2006)
13. Shah-Hosseini, H., Safabakhsh, R.: TASOM: a new time adaptive self-organizing map. IEEE Transactions on Systems, Man, and Cybernetics **33**(2), 271–282 (2003)
14. Silva, D.D., et al.: Machine learning to support social media empowered patients in cancer care and cancer treatment decisions. PloS one **13**(10), 1–10 (2018)