HyperEmbed: Tradeoffs Between Resources and Performance in NLP Tasks with Hyperdimensional Computing enabled embedding of n-gram statistics

Anonymous authors
Paper under double-blind review

Abstract

Recent advances in Deep Learning have led to a significant performance increase on several NLP tasks, however, the models become more and more computationally demanding. Therefore, this paper tackles the domain of computationally efficient algorithms for NLP tasks. In particular, it investigates distributed representations of n-gram statistics of texts. The representations are formed using hyperdimensional computing enabled embedding. These representations then serve as features, which are used as input to standard classifiers. We investigate the applicability of the embedding on one large and three small standard datasets for classification tasks using nine classifiers. The embedding achieved on par F_1 scores while decreasing the time and memory requirements by several times compared to the conventional n-gram statistics, e.g., for one of the classifiers on a small dataset, the memory reduction was 6.18 times; while train and test speed-ups were 4.62 and 3.84 times, respectively. For many classifiers on the large dataset, the memory reduction was about 100 times and train and test speed-ups were over 100 times. More importantly, the usage of distributed representations formed via hyperdimensional computing allows dissecting the strict dependency between the dimensionality of the representation and the parameters of n-gram statistics, thus, opening a room for tradeoffs.

1 Introduction

Recent work (Strubell et al., 2019) has brought significant attention by demonstrating potential cost and environmental impact of developing and training state-of-the-art models for Natural Language Processing (NLP) tasks. The work suggested several countermeasures for changing the situation. One of them recommends a concerted effort by industry and academia to promote research of more computationally efficient algorithms. The main focus of this paper falls precisely in this domain.

In particular, we consider NLP systems using a well-known technique called n-gram statistics. The key idea is that hyperdimensional computing (Kanerva, 2009) allows forming distributed representations of the conventional n-gram statistics (Joshi et al., 2016). The use of these distributed representations, in turn, allows trading-off the performance of an NLP system (e.g., F_1 score) and its computational resources (i.e., time and memory). We demonstrate the usefulness of hyperdimensional computing-based embedding, which is highly time and memory efficient. Our experiments on a well-known dataset (Braun et al., 2017) for intent classification show that it is possible to reduce memory usage by ~ 10x and speed-up training by ~ 5x without compromising the F_1 score. Several important use-cases are motivating the efforts towards trading-off the performance of a system against computational resources required to achieve that performance: high-throughput systems with an extremely large number of requests/transactions (the power of one per cent); resource-constrained systems where computational resources and energy are scarce (edge computing); green computing systems taking into account the aspects of environmental sustainability when considering the efficiency of algorithms (AI HLEG, 2019).

2 METHODS

2.1 CONVENTIONAL n-GRAM STATISTICS

An empty vector $\boldsymbol{s}$ stores n-gram statistics for an input text $\mathcal{D}$. $\mathcal{D}$ consists of symbols from the alphabet of size a; ith position in $\boldsymbol{s}$ keeps the counter of the corresponding n-gram $\mathbb{A}_i = \langle \mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_n, \rangle$ from the set $\mathbb{A}$ of all unique n-grams; $\mathcal{S}_j$ corresponds to a symbol in jth position of $\mathbb{A}_i$. The dimensionality of $\boldsymbol{s}$ equals the total number of n-grams in $\mathbb{A}$ and calculated as a^n. Usually, $\boldsymbol{s}$ is obtained via a single pass-through $\mathcal{D}$ using the overlapping sliding window of size n. The value of a position in $\boldsymbol{s}$ (i.e., counter) corresponding to a n-gram observed in the current window is incremented by one. In other words, $\boldsymbol{s}$ summarizes how many times each n-gram in $\mathbb{A}$ was observed in $\mathcal{D}$.

2.2 SUBWORD SEMANTIC HASHING

Subword Semantic Hashing (SemHash) is described in details in Shridhar et al. (2019); Huang et al. (2013). SemHash represents the input sentence in the form of subword tokens using a hashing method reducing the collision rate. These subword tokens act as features to the model and can be used as an alternative to word/n-gram embeddings. For a given input sample text T, e.g., *"I have a flying disk"*, we split it into a list of words t_i. The output of the split would look as follows: *["I", "have", "a", "flying", "disk"]*. Each word is then passed into a prehashing function $\mathcal{H}(t_i)$. $\mathcal{H}(t_i)$ first adds a # at the beginning and at the end of t_i. Then it generates subwords via extracting n-grams (n=3) from $\#t_i\#$, e.g., $\mathcal{H}(have) = [\#ha, hav, ave, ve\#]$. These tri-grams are the subwords denoted as t_i^j, where j is the index of a subword. $\mathcal{H}(t_i)$ is then applied to the entire text corpus to generate subwords via n-gram statistics. These subwords are used to extract features for a given text.

2.3 EMBEDDING n-GRAM STATISTICS INTO AN HD VECTOR

Alphabet's symbols are the most basic elements of a system. We assign each symbol with a random d-dimensional bipolar HD vector. These vectors are stored in a matrix (denoted as $\boldsymbol{H}$, where $\boldsymbol{H} \in [d \times a]$), which is referred to as the item memory, For a given symbol $\mathcal{S}$ its HD vector is denoted as $\boldsymbol{H}_{\mathcal{S}} \in \{-1, +1\}^{[d \times 1]}$. To manipulate HD vectors, hyperdimensional computing defines three key operations[1] on them: bundling[2] (denoted with $+$ and implemented via position-wise addition), binding (denoted with $\odot$ and implemented via position-wise multiplication), and permutation[3] (denoted with ρ).

Three operations above allow embedding n-gram statistics into distributed representation (HD vector) Joshi et al. (2016). First, $\boldsymbol{H}$ is generated for the alphabet. A position of symbol $\mathcal{S}_j$ in $\mathbb{A}_i$ is represented by applying ρ to the corresponding HD vector $\boldsymbol{H}_{\mathcal{S}_j}$ j times, which is denoted as $\rho^j(\boldsymbol{H}_{\mathcal{S}_j})$. Next, a single HD vector for $\mathbb{A}_i$ (denoted as $\boldsymbol{m}_{\mathbb{A}_i}$) is formed via the consecutive binding of permuted HD vectors $\rho^j(\boldsymbol{H}_{\mathcal{S}_j})$ representing symbols in each position j of $\mathbb{A}_i$. For example, the trigram 'cba' will be mapped to its HD vector as follows: $\rho^1(\boldsymbol{H}_{\text{c}}) \odot \rho^2(\boldsymbol{H}_{\text{b}}) \odot \rho^3(\boldsymbol{H}_{\text{a}})$. In general, the process of forming HD vector of an n can be formalized as follows:

$$\boldsymbol{m}_{\mathbb{A}_i} = \prod_{j=1}^{n} \rho^j(\boldsymbol{H}_{\mathcal{S}_j}),$$

where $\prod$ denotes the binding operation when applied to n HD vectors. Once it is known how to get $\boldsymbol{m}_{\mathbb{A}_i}$, embedding the conventional n-gram statistics stored in $\boldsymbol{s}$ (see section 2.1) is straightforward.

[1]Please see Kanerva (2009) for proper definitions and properties of hyperdimensional computing operations.

[2]The bundling operation allows storing information in HD vectors(Kleyko et al. (2016)); if several copies of any HD vector are included (e.g., $2\boldsymbol{H}_{\mathcal{S}_1} + \boldsymbol{H}_{\mathcal{S}_2}$), the resultant HD vector is more similar to the dominating HD vector than to other components. Since this paper does not go into deep analytical details of why HD vectors allow embedding the conventional n-gram statistics, the diligent readers are referred to Frady et al. (2018) for the relevant analysis.

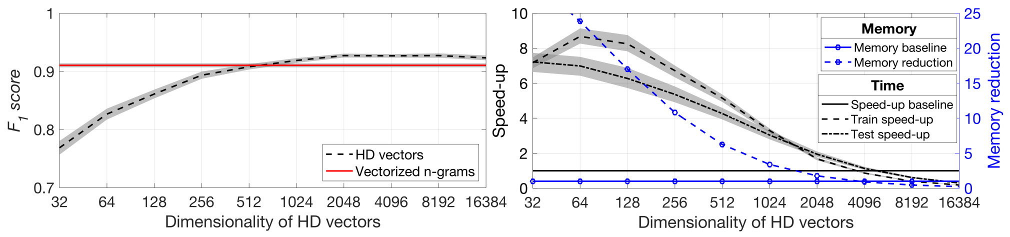
[3]It is convenient to use ρ to bind symbol's HD vector with its position in a sequence.

HD vector $\boldsymbol{h}$ corresponding to $\boldsymbol{s}$ is created by bundling together all n-grams observed in the data:

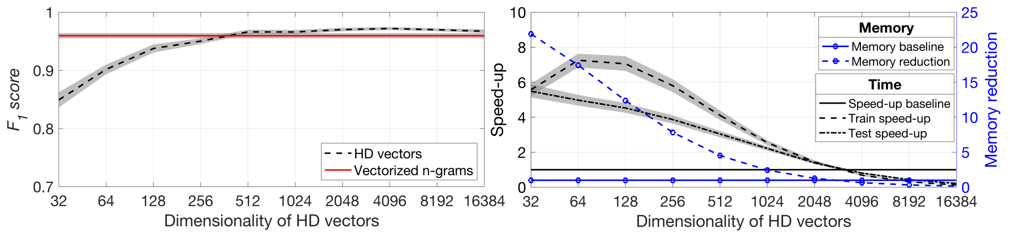
$$\boldsymbol{h} = \sum_{i=1}^{a^n} \boldsymbol{s}_i \boldsymbol{m}_{\mathbb{A}_i} = \sum_{i=1}^{a^n} \boldsymbol{s}_i \prod_{j=1}^{n} \rho^j(\boldsymbol{H}_{\mathcal{S}_j}),$$

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

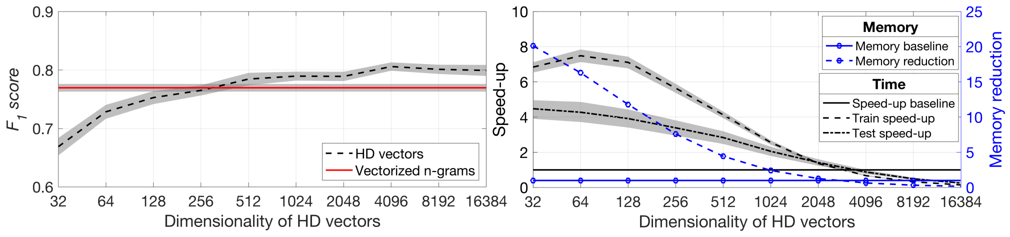
3 Empirical evaluation



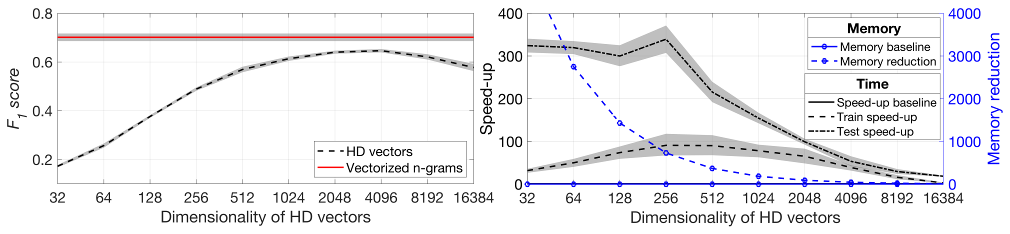
(a)



(b)



(c)



(d)

Figure 1: MLP results vs. the dimensionality of HD vectors on: (a) the AskUbuntu dataset. (b) the Chatbot dataset. (c) the WebApplication dataset. (d) the 20NewsGroups dataset.

3.1 Results

First, we report the results of the MLP classifier on all datasets as it represents a widely used class of algorithms – neural networks. The goal of the experiments was to observe how the dimensionality of HD vectors embedding n-gram statistics affects the F_1 scores and the computational resources. Figures 1a-1d present the results for the AskUbuntu, Chatbot, WebApplication, and 20NewsGroups

Table 1: Performance of all classifiers for the AskUbuntu dataset.

	F_1 score			Resources: SH vs. HD			Resources: SH vs. BPE		
Classifier	SH	BPE	HD	Tr.	Ts.	Mem.	Tr.	Ts.	Mem.
MLP	0.92	0.91	0.91	4.62	3.84	6.18	1.67	1.61	1.72
Passive Aggr.	0.92	0.93	0.90	4.86	3.07	6.31	2.19	2.14	1.76
SGD Classifier	0.89	0.89	0.88	4.66	3.50	6.31	1.94	2.16	1.76
Ridge Classifier	0.90	0.91	0.90	3.91	4.74	6.31	1.63	1.62	1.76
KNN Classifier	0.79	0.72	0.82	2.11	4.53	8.48	1.56	1.79	1.76
Nearest Centroid	0.90	0.89	0.90	1.66	3.41	6.32	1.35	1.87	1.76
Linear SVC	0.90	0.92	0.90	1.18	2.39	6.29	0.91	1.91	1.76
Random Forest	0.88	0.90	0.86	0.91	1.09	6.11	1.15	0.96	1.75
Bernoulli NB	0.91	0.92	0.85	2.30	3.72	6.34	1.96	2.42	1.76

datasets, respectively. The dimensionality of HD vectors varied as 2^k, $k \in [5, 14]$. All figures have an identical structure. Shaded areas depict 95% confidence intervals. Left panels depict the F_1 score while right panels depict the train and test speed-ups as well as memory reduction. Note that there are different scales (y-axes) in the right panels. A solid horizontal line indicates 1 for the corresponding y-axis, i.e., the moment when both models consume the same resources.

The results in all figures are consistent in a way that up to a certain point F_1 score was increasing with the increasing dimensionality. For the small datasets even small dimensionalities of HD vectors (e.g., $32 = 2^5$) led to the F_1 scores, which are far beyond random. For example, for the AskUbuntu dataset, it was $84\,\%$ of the conventional n-gram statistics F_1 score. For the values above 512 the performance saturation begins. Moreover, the improvements beyond 2048 are marginal. The situation is more complicated for the 20NewsGroups dataset where for 32-dimensional HD vectors F_1 score is fairly low though still better than a random guess (0.05). However, it increases steeply until 1024 and achieves its maximum at 4096 being $92\,\%$ of the conventional n-gram statistics F_1 score. The dimensionalities above 4096 showed worse results.

When it comes to computational resources, there is a similar pattern for all the datasets. The train/test speed-ups and memory reduction are diminishing with the increased dimensionality of HD vectors. At the point when the dimensionality of HD vectors equals the size of the conventional n-gram statistics, both approaches consume approximately the same resources. These points in the figures are different because the datasets have different size of n-gram statistics: 3729, 2753, 2734, and 192652, for the AskUbuntu, Chatbot, WebApplication, and 20NewsGroups datasets, respectively. Also, for all datasets, the memory reduction is higher than the speed-ups. The most impressive speed-ups and reductions were observed for the 20NewsGroups dataset (e.g., 186 times less memory for 1024-dimensional HD vectors). This is due to its large size it contains a huge number of n-grams resulting in large size of the n-gram statistics. Nevertheless, even for small datasets, the gains were noticeable. For instance, for the WebApplication dataset at 256 F_1 score was $99\,\%$ of the conventional n-gram statistics while the train/test speed-ups and the memory reduction were 5.6, 3.4, and 7.6, respectively.

Thus, these empirical results suggest that the quality of embedding w.r.t. the achievable F_1 score improves with increased dimensionality, however, after a certain saturation or peak point increasing dimensionality further either does not affect or worsen the classification performance and arguably becomes impractical when considering the computational resources.

Tables 1-3[4] report the results for three small datasets[5] when applying all the considered classifiers. Due to the space limitation, a fixed dimensionality of HD vectors is reported only: 512 for small datasets in Tables 1-3 and 2048 for the 20NewsGroups dataset in Table **??**. These dimensionalities were chosen based on the results in Figures 1a-1d as the ones allowing to achieve a good approximation of F_1 score while providing substantial speed-up/reduction. We also performed experiments when using the BPE instead of the SemHash before extracting n-gram statistics. Throughout the

[4]The notations Tr., Ts., Mem. in the tables stand for the train speed-up, test speed-up, and the memory reduction for the given classifier, respectively. SH stands for SemHash.

[5]Due to the page limit, results for the 20NewsGroups dataset are presented in the Appendix (see Table **??**).

Table 2: Performance of all classifiers for the Chatbot dataset.

	F_1 score			Resources: SH vs. HD			Resources: SH vs. BPE		
Classifier	SH	BPE	HD	Tr.	Ts.	Mem.	Tr.	Ts.	Mem.
MLP	0.96	0.94	0.96	3.42	2.62	4.58	1.86	1.52	1.86
Passive Aggr.	0.95	0.91	0.94	4.40	2.38	4.72	2.29	2.22	1.92
SGD Classifier	0.93	0.93	0.92	3.16	2.06	4.72	1.88	1.84	1.92
Ridge Classifier	0.94	0.94	0.92	2.88	2.22	4.72	1.67	1.38	1.92
KNN Classifier	0.75	0.71	0.83	1.66	3.59	6.51	1.43	1.79	1.92
Nearest Centroid	0.89	0.94	0.84	1.41	2.13	4.73	1.17	1.61	1.92
Linear SVC	0.94	0.93	0.94	0.52	1.57	4.72	1.28	1.66	1.92
Random Forest	0.95	0.95	0.91	0.95	1.10	4.61	1.16	0.98	1.91
Bernoulli NB	0.93	0.93	0.82	1.92	2.60	4.73	1.53	1.72	1.92

Table 3: Performance of all classifiers for the WebApplication dataset.

	F_1 score			Resources: SH vs. HD			Resources: SH vs. BPE		
Classifier	SH	BPE	HD	Tr.	Ts.	Mem.	Tr.	Ts.	Mem.
MLP	0.77	0.77	0.79	3.10	2.00	4.43	1.74	1.44	1.73
Passive Aggr.	0.82	0.80	0.80	3.73	1.45	4.33	1.86	1.32	1.75
SGD Classifier	0.75	0.74	0.73	3.01	1.87	4.33	1.62	1.32	1.75
Ridge Classifier	0.79	0.80	0.80	1.66	2.40	4.34	0.71	1.09	1.75
KNN Classifier	0.72	0.75	0.76	1.16	2.76	5.96	1.14	1.51	1.76
Nearest Centroid	0.74	0.73	0.77	1.42	1.79	4.34	1.13	1.21	1.75
Linear SVC	0.82	0.80	0.80	1.04	1.48	4.29	0.47	1.18	1.75
Random Forest	0.87	0.85	0.72	0.95	1.26	4.11	1.05	1.12	1.73
Bernoulli NB	0.74	0.75	0.64	1.51	2.08	4.38	1.19	1.49	1.75

tables, the BPE demonstrated F_1 scores comparable to that of the SemHash while showing the train/test speed-ups and memory reduction at about 2. This is because the usage of the BPE resulted in smaller sizes of the n-gram statistics, which were 2176, 1467, and 1508 for the AskUbuntu, Chatbot, and WebApplication datasets, respectively.

In the case of HD vectors, the picture is less coherent. For example, there is a group of classifiers (e.g., MLP, SGD, KNN) where F_1 scores are well approximated (or even improved) while achieving noticeable computational reductions. In the case of Linear SVC, F_1 scores are well-preserved and there is $4-6$ memory reduction but test/train speed-ups are marginal (even slower for training the Chatbot). This is because Linear SVC implementation benefits from sparse representations (conventional n-gram statistics) while HD vectors in this study are dense. Last, for Bernoulli NB and Random Forest F_1 scores were not approximated well (cf. 0.93 vs. 0.82 for Bernoulli NB in the case of the Chatbot). This is likely because both classifiers are relying on local information contained in individual features, which is not the case in HD vectors where information is represented distributively across the whole vector. The slow train time of Random Forest is likely because in the absence of well-separable features it tries to construct large trees.

Last, due to the difference in the implementation (the official implementation of FastText only uses a linear classifier), we were not able to have a proper comparison of computational resources with the FastText.[6] However, we obtained the following F_1 scores with auto hyperparameter search: 0.91, 0.97, 0.76 for the AskUbuntu, Chatbot, and WebApplication datasets, respectively. These results indicate that for the considered datasets there is no drastic classification performance improvement (even worse for the WebApplication) when using the learned representations of n-grams.

[6]We could have implemented the algorithm ourselves but it can be claimed unfair to compare the required memory and time, if we do not use the best practices, which are unknown to us.

REFERENCES

AI HLEG. *High-Level Expert Group on Artificial Intelligence. Ethics Guidelines for Trustworthy AI.* 2019.

D. Braun, A. Hernandez-Mendez, F. Matthes, and M. Langen. Evaluating Natural Language Understanding Services for Conversational Question Answering Systems. In *Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, pp. 174–185, 2017.

E.P. Frady, D. Kleyko, and F.T. Sommer. A Theory of Sequence Indexing and Working Memory in Recurrent Neural Networks. *Neural Computation*, 30:1449–1513, 2018.

P. Huang, X.He, J. Gao, L. Deng, A. Acero, and L. Heck. Learning Deep Structured Semantic Models for Web Search using Clickthrough Data. In *ACM international conference on Information and Knowledge Management (CIKM)*, pp. 2333–2338, 2013.

A. Joshi, J.T. Halseth, and P. Kanerva. Language Geometry Using Random Indexing. In *Quantum Interaction (QI)*, pp. 265–274, 2016.

P. Kanerva. Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors. *Cognitive Computation*, 1(2):139–159, 2009.

Denis Kleyko, Evgeny Osipov, Alexander Senior, Asad I Khan, and Yaşar Ahmet Şekerciogğlu. Holographic graph neuron: A bioinspired architecture for pattern processing. *IEEE transactions on neural networks and learning systems*, 28(6):1250–1262, 2016.

K. Shridhar, A. Dash, A. Sahu, G. Grund Pihlgren, P. Alonso, V. Pondenkandath, G. Kovacs, F. Simistira, and M. Liwicki. Subword Semantic Hashing for Intent Classification on Small Datasets. In *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–6, 2019.

E. Strubell, A. Ganesh, and A. McCallum. Energy and Policy Considerations for Deep Learning in NLP. *arXiv:1906.02243*, pp. 1–6, 2019.