Bridging the Gap between Hyperdimensional Computing and Kernel Methods via the Nyström Method

Quanling Zhao, Anthony Thomas, Ari Brin, Xiaofan Yu, Tajana Rosing

Department of Computer Science and Engineering, UC San Diego
{quzhao,ahthomas,abrin,x1yu,tajana}@ucsd.edu

Abstract

Hyperdimensional computing (HDC) is an approach for solving cognitive information processing and a variety of learning tasks using data represented as high-dimensional vectors. The technique has a rigorous mathematical backing, and is easy to implement in energy-efficient and highly parallel hardware like FPGAs and "processing-in-memory" architectures. The success of HDC based machine learning approaches is heavily dependent on the mapping of raw data to high-dimensional space. In this work, we propose NysHD, a new method for constructing this mapping that is based on the Nyström method from the literature on kernel approximation. Our approach provides a simple recipe to turn any user-defined positive-semidefinite similarity function into an equivalent mapping in HDC. There is a vast literature on the design of such functions for learning problems. Our approach provides a mechanism to import them into the HDC setting, expanding the types of problems that can be tackled using HDC. Empirical evaluation against existing HDC encoding methods shows that NysHD can achieve, on average, 11% and 17% better classification accuracy on graph and string datasets respectively.

Introduction

Biological brains "compute" using representations of data that are intrinsically fault tolerant, amenable to highly-parallel circuitry, and expose complex structure in the environment in a way that is easy to learn from (Hertz 2018). Motivated by these desirable qualities, hyperdimensional computing (HDC) builds on theories of representation from cognitive science (Kanerva 2009; Plate 1995) in an effort to develop a new approach to designing hardware and algorithms for information processing tasks (Kanerva 2009). In HDC, all computation is performed using high-dimensional, low-precision, vector representations of data. These representations can be manipulated using simple, element-wise operators, so as to implement learning or other information processing tasks.

In contrast to deep learning models, training HDC based models can typically be done in a single pass over the training data (Yu et al. 2022) and does not require back-propagation. The operations used in HDC are lightweight and highly parallelizable, making them suitable for implementation on low-energy and highly-parallel hardware platforms. This makes HDC an attractive alternative for implementing learning in resource constrained settings. As a result, HDC has gained significant interest in recent years, especially in Internet of Things (IoT) (Khaleghi et al. 2022; Zhao et al. 2022; Morris et al. 2021); and in the computer hardware community (Dutta et al. 2022; Kang et al. 2022a) such as FPGAs (Salamat et al. 2019), GPUs (Kang et al. 2022b), ASICs (Zhang et al. 2023) and in-memory computing (Dutta et al. 2022; Xu, Kang, and Rosing 2023).

The first stage in any HDC task is encoding, which maps data from its ambient representation $x \in \mathcal{X}$, into a representation $\phi(x)$ that lives in high-dimensional inner-product space $\mathcal{H}$ (Thomas, Dasgupta, and Rosing 2021). HDC uses a pair of operators for addition and multiplication (called 'bundling" and "binding" in the HDC literature), which can be used to build representations of complex structures from simple building blocks or to implement tasks like learning. For instance, classification using HDC represents each class as a bundle (sum) of the encodings of its training data, called a "prototype." Inference can then be performed by finding the closest prototype to a query.

The crucial assumption underlying the success of this technique is that "similar" points in $\mathcal{X}$ are mapped to "similar" regions of $\mathcal{H}$. In practice, this desideratum typically means that dot-products in $\mathcal{H}$ should be reflective of some salient notion of similarity on $\mathcal{X}$. In HDC, one typically builds representations incrementally, by bundling and binding together the embeddings of simpler atoms. In this work, we observe that one can also go in the opposite direction: starting from a known similarity function of interest, it is possible to generate an equivalent—up to some approximation factor—encoding function.

The advantage of this "top down" approach is that there is a vast literature on designing good similarity functions for different kinds of learning problems. In machine learning, similarity functions that work by computing inner-products between high-dimensional embeddings of data are called kernel functions, and are the basis of kernel methods, a vast area of research in theoretical and applied ML (Shawe-Taylor and Cristianini 2004; Smola and Schölkopf 1998). This literature has devoted substantial attention to the problem of designing good similarity functions, which are also

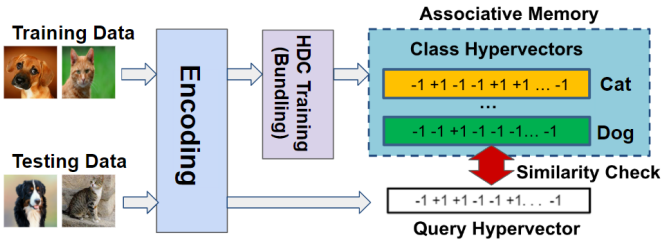

Figure 1: Overview of HDC training and inference for classification tasks.

potentially applicable to the kinds of problems encountered in HDC. In this work, we study an approach, based on the Nyström method from the literature on kernel approximation (Williams and Seeger 2000), which can take a user defined kernel and generate an low-precision, randomized embedding, suitable for use in the kinds of learning algorithms employed in HDC. In a nutshell, the contributions of this paper are as follows:

- We propose NysHD, a new way to generate embeddings for HDC which can turn any user-defined positive-semidefinite similarity function into an equivalent embedding.
- We analyze the similarity-preserving properties of NysHD formally and show that the inner-product between encoded samples preserves normalized kernel values.
- We perform an empirical evaluation against state of the art HDC encoding methods and various neural network architectures and show our method substantially improves the performance of HDC based learning, while preserving efficiency benefits relative to DNNs.

From a practical standpoint, our work has the potential to expand the scope of tasks that can be effectively addressed using HDC by allowing practitioners to access the large repertoire of kernels that have been designed for them (i.e. graph kernels, string kernels etc.).

Background and Related Work

Learning with HDC

In this work, we focus on using HDC to solve classification problems, which is a common practical application of the technique (Imani et al. 2019; Rahimi et al. 2018; Menon et al. 2022). Fig 1 demonstrates a typical HDC learning workflow for classification (Morris et al. 2021; Nunes et al. 2022; Miranda and d'Aliberti 2022). Let $\mathcal{D} = \{(x_1, y_1), ..., (x_n, y_n)\}$ be a set of training data, where $x_i \in \mathcal{X}$ is an input, and $y_i \in \{1, ..., c\}$ is a class label. The first step is to embed the training data into a d-dimensional inner-product space $\mathcal{H}$ under a map (called the encoding function) $\phi : \mathcal{X} \to \mathcal{H}$. The "training" step then associates each class with a vector in $\mathcal{H}$, which is typically formed by summing (bundling) the training data corresponding to a particular class. That is, one represents class j as $\theta_j = \sum_{i=1}^{n} \alpha_{ij} \phi(x_i)$ where $\alpha_i = 1(y_i = j)$. In practice, one sometimes quantizes θ_j in some fashion to reduce precision, which is beneficial in some hardware settings. The label of a query x is predicted by via: $\hat{y} = \arg\max_{j \in \{1,...,c\}} \langle \phi(x), \theta_j \rangle$ where the operands are sometimes normalized in some fashion if appropriate. In any event, this procedure can be interpreted as associating each class to a linear function in $\mathcal{H}$.

It is also common to fine tune the θ_j's by running the Perceptron algorithm (Rosenblatt 1958) (often referred to as "re-training" in HDC literature) where class prototypes are adjusted iteratively use misclassified samples.

A number of HDC encoding methods have been proposed to encode various types of data. For example, string or text data can be encoded into hypervectors through the N-gram encoding method (Joshi, Halseth, and Kanerva 2017; Imani et al. 2018). Concretely, let $S = (a_1 a_2 a_3 ... a_N)$ be a string of N characters drawn from some alphabet $\mathcal{A}$ (say, the English-language alphabet or $\{\texttt{A}, \texttt{T}, \texttt{G}, \texttt{C}\}$). To encode S, we start by assigning each $a \in \mathcal{A}$ an embedding $\phi(a)$ by sampling uniformly at random from $\{\pm 1/\sqrt{d}\}^d$, after which, one might represent S as $a_1 \circ \rho(a_2) \circ \rho^2(a_3) \circ \cdots \circ \rho^{N-1}(a_N)$ where $\circ$ can be either element-wise addition (Imani et al. 2018) or multiplication (Joshi, Halseth, and Kanerva 2017) and ρ is a permutation operation of the vector coordinates, ρ^n means same permutation applied n times in sequence. If $\circ$ is multiplication, the way N-grams representations are constructed makes them almost mutually orthogonal in $\mathcal{H}$ in the sense that $\mathbb{E}[\phi(S) \cdot \phi(S')] = 1(S = S')$. The deviation from this expectation can be controlled using concentration arguments (Thomas, Dasgupta, and Rosing 2021).

Finally, longer strings that contain multiple N-grams can be treated as a bundle of N-gram vectors. This string encoding scheme can be viewed as a "compressed" version of the bag-of-words model (Harris 1954) in the sense that all N-gram vectors are superimposed into one representation. Intuitively, the inner-product between two such encoded strings can measure how "similar" two strings are, as that value would be large if two strings shared many common N-grams, and vice versa. For general feature vectors (or simple images), techniques based on random projection are popular (Morris et al. 2021; Khaleghi et al. 2022).

The state of the art HDC encoding methods tend to capture fairly simple notions of similarity based on the L1/L2 or angular distance. However, the design of good encoding functions for complex forms of data like graphs and time-series remains an important area of research. For inspiration, we turn to another area of machine learning that has thought extensively about how to measure similarities between data points using high-dimensional vectors.

Kernel Methods

Kernel methods are a wide ranging area of research in statistics and machine learning that shares many similarities with HDC (Shawe-Taylor and Cristianini 2004; Meanti et al. 2020; Hofmann, Schölkopf, and Smola 2008). Just like in HDC, kernel methods work by embedding data into a high-dimensional space wherein similarities are measured using inner-products. That is to say, kernel methods measure similarities between data points $x, x' \in \mathcal{X}$ via a func-

tion $K(x, x') = \langle \psi(x), \psi(x') \rangle$, called a "kernel function," where $\psi : \mathcal{X} \to \mathcal{H}$ is an embedding into an inner-product space. For many types of kernel functions used in practice, it is possible to compute $K(x, x')$ directly on the ambient representation of the data without materializing the embeddings. Notable examples include the Gaussian kernel $K(x, x') \propto \exp(-\|x - x'\|_2^2)$, and the p-th order polynomial kernel $K(x, x') = (1 + x^T x')^p$. Both of these kernels can be evaluated in closed form on the ambient representation of the data, allowing kernel methods to *implicitly* compute a similarity based on a high-dimensional embedding. HDC, however, always explicitly materializes the embeddings, hence the need for an encoding function ϕ.

Kernel based learning methods make predictions using functions taking the form $f(x) = \sum_{i=1}^{n} a_i k(x_i, x)$, where $x_1, ..., x_n$ are training data points, and $\alpha_1, ..., \alpha_n$ are weights that are learned by a training algorithm. Noting that $f(x) = \sum_{i=1}^{n} \alpha_i k(x, x_i) = \psi(x) \cdot \theta$ where $\theta = \sum_{i=1}^{n} a_i \psi(x_i)$, in this way we can interpret such functions as *linear models* in the embedding space associated with the kernel, much like in the previous paragraph on HDC. One significant difference between kernel methods and HDC, is that in the former the embeddings are implicit, and similarities are evaluated using the kernel function. This property is appealing because it allows one to efficiently work with infinite-dimensional embeddings, which can have desirable properties for learning (Steinwart 2001).

Kernel methods and HDC

There is a large theoretical and applied literature on kernel methods that has designed kernels (e.g. similarity functions) that are applicable to many settings of practical interest (Neumann et al. 2016; Leslie, Eskin, and Noble 2001; Shimodaira et al. 2001; Shawe-Taylor and Cristianini 2004). To give a concrete example of how the literature on kernel methods can offer insights for the HDC community, we consider the method for encoding graphs presented in (Nunes et al. 2022). GraphHD (Nunes et al. 2022) proposes to first use the PageRank centrality metric (Brin and Page 1998) to rank the "importance" of each node and assign a hypervector to each node based on its importance, in the sense that nodes in different graphs with the same ranking will be assigned the same hypervector. Edges are represented by multiplying (binding) of two node hypervectors, and an entire graph can be represented as the summation (bundle) of edge hypervectors. This encoding method basic information of graphs is preserved during encoding, while other crucial information such as node labels or node attributes (where each node in the graph is associated with either a feature vector or a label) is not being utilized. Such limitations, however, have been addressed with kernel methods. For example, the propagation kernel (Neumann et al. 2016) is able to work with graphs that have node labels or node attributes. The propagation kernel uses random walk techniques to measure not only the structural difference between graphs but also taking node labels or node attributes into consideration as well, therefore is capable of capturing a potentially richer notion of similarity.

On time-series data, the permutation operation is used in a similar fashion to encode the temporal order (Joshi, Halseth, and Kanerva 2017; Asgarinejad, Thomas, and Rosing 2020). However, such encoding method can fail if the events between two time-series do not align exactly. Since each time step is associated with a unique permutation during encoding, if events in time-series are shifted, even by a small amount, existing HDC encoding methods will map two time-series to nearly orthogonal vectors. In practice, however, if two time-series data reflect the same underlying activity or nature, one would want that similarity to be preserved after the encoding, even if some event misalignment may exist. Here again, the literature on kernel methods suggests a solution: the dynamic-time-warping kernel (Gudmundsson, Runarsson, and Sigurdsson 2008) can handle time-series sequences with dis-alignment or with time-stretching/compression. In this work, our goal is to devise a procedure that can translate any kernel into an equivalent HDC encoding, thereby allowing practitioners to exploit the wealth of kernel functions that have been designed for practical problems while continuing to reap the benefits of computing with distributed representations.

The connection between HDC and kernel approximation is generally well known (Thomas, Dasgupta, and Rosing 2021; Paxon Frady et al. 2021), mostly through the lens of random Fourier features (RFF) (Rahimi and Recht 2007). RFF is a sampling based scheme that generates a vector of features $\phi(x) \in \mathbb{R}^d$ with the property that $\phi(x) \cdot \phi(x') \approx K(x, x')$, where K is a shift-invariant kernel (e.g. the Gaussian kernel, polynomial kernel). Closely related methods arise in the HDC literature under the names "nonlinear-encoding" (Miranda and d'Aliberti 2022; Imani et al. 2020) and "fractional power encoding" (Paxon Frady et al. 2021). Using RFF in the context of HDC means that inner-products in HD space approximate some shift-invariant kernel, usually the Gaussian or Sinc kernels. However, a limitation of RFF is that it can only work with shift-invariant kernels on a Euclidean space, which many useful kernels do not satisfy (i.e. kernels on graphs and strings). NysHD provides a way to generate approximations for a larger class of kernels that do not need to be translation invariant.

HDC Encoding via Nyström Approximation

In this section we describe our new encoding algorithm, NysHD, for HDC using Nyström method for kernel approximation. We also show formally that the inner-product between encoded samples preserves normalized kernel values.

Given a suitable kernel function K, our goal is to generate an encoding function $\phi : \mathcal{X} \to \mathcal{H}$ such that $\langle \phi(x), \phi(x') \rangle \approx K(x, x') \quad \forall x, x' \in \mathcal{X}$, while ensuring that the representations continue to be amenable to the kinds of noisy and highly parallel hardware used in practical applications of HDC (Thomas, Dasgupta, and Rosing 2021; Kleyko et al. 2023). Having embeddings that approximate a particular kernel is important as it enables HDC learning algorithms to exploit more useful similarities captured by the kernel.

Nyström method

Fitting kernel machines commonly requires storing all pairwise evaluations of the kernel function in a large matrix

K defined element-wise by $K_{ij} = K(x_i, x_j)$, problematic when n is large. The Nyström method is a low-rank matrix approximation technique widely employed to speed up kernel machines by avoiding the need to evaluate the entire kernel matrix (Williams and Seeger 2000; Drineas, Mahoney, and Cristianini 2005; Kumar, Mohri, and Talwalkar 2012). In a way, Nyström method is similar to RFF since they are both sampling based schemes and can be used to approximate kernel functions. The key difference between RFF and Nyström method is that Nyström method can work with a larger class of kernels than RFF, many of which are useful for applications involving discrete structures such as strings, graphs and more.

Intuitively, the Nyström method works by sub-sampling the kernel matrix and reconstructing the full kernel matrix from the sampled one. This is possible because the kernel matrix is typically close to low-rank in practice. Concretely, suppose we have a dataset $\mathcal{D} = \{x_1, x_2, ..., x_n\}$, from which we sample a set of landmarks $\mathcal{Z} = \{z_1, ..., z_s\}$ uniformly at random, where $s \ll n$. Let $G \in \mathbb{R}^{n\times n}$ be the full kernel matrix defined element-wise by $G_{ij} = K(x_i, x_j)$, and let $H_{\mathcal{Z}} \in \mathbb{R}^{s\times s}$ be the sub-sampled kernel matrix defined element-wise by $(H_{\mathcal{Z}})_{ij} = K(z_i, z_j)$. The Nyström method yields the following approximation (Drineas, Mahoney, and Cristianini 2005):

$$\hat{G} = C H_{\mathcal{Z}}^{+} C^T \approx G \tag{1}$$

Where C is an $n \times s$ matrix such that $C_{ij} = K(x_i, z_j)$ for some kernel function K, and H_Z^+ denotes the pseudo-inverse of $H_{\mathcal{Z}}$. That is, let Q and Λ be the eigenvectors and eigenvalues of $H_{\mathcal{Z}}$ then:

$$H_Z^+ = Q\Lambda^{-1}Q^T \quad \text{(symmetric eigen-decomposition)}$$

To further decompose Eq. (1), let $C^{(i)}$ denotes i^{th} row of C in a column vector, we can have embedding for each data point such that:

$$\begin{aligned}\hat{G_{ij}} &\approx \phi_{nys}(x_i)^T \phi_{nys}(x_j) \\ &= \left(\Lambda^{-\frac{1}{2}} Q^T C^{(i)}\right)^T \left(\Lambda^{-\frac{1}{2}} Q^T C^{(j)}\right)\end{aligned} \tag{2}$$

In general, the embeddings generated by Nyström method do not need to be high-dimensional to have the similarity preserving property, however, they may lack other desiderata of HDC like noise robustness, and low-precision. Our method rectifies this issue by composing the features extracted using the Nyström method with another encoding technique that preserves angular similarities which we discuss in detail in the next section .

Due to the data-dependent nature of the Nyström method, the quality of its approximation and computational complexity is dependent on the quality and size of $\mathcal{Z}$. As an initial step, in this paper we simply use uniform sampling without replacement as our sampling strategy. However, more sophisticated strategies such as ensemble and adaptive sampling (Kumar, Mohri, and Talwalkar 2012) for constructing landmark sets can potentially yield better performance, and it would be of interest to explore these in future work.

Encoding Process

To achieve the aforementioned goals (generate HDC embeddings such that their inner-products approximate some useful kernel) we compose random projection with Nyström method. Alg. 1 details the generating process of Nyström random projection. After which, the encoding of a data point can be done through Alg. 2.

Algorithm 1: Generate Nyström Random Projection

Require: kernel K over $\mathcal{X}$, dataset $\mathcal{D}$, number of landmarks $s > 0$, HDC dimension $d > 0$
$\mathcal{Z} \leftarrow$ uniform sampling of s data points from $\mathcal{D}$
/*Landmarks*/
$(H_{\mathcal{Z}})_{ij} = K(z_i, z_j) \quad \forall \quad 0 \le i, j \le s$
/*Partial kernel Matrix over landmarks*/
$Q\Lambda Q^T = H_{\mathcal{Z}}$
/*Symmetric Eigen-decomposition*/
$P_{rp} = [w_1, w_1, \cdots, w_d]^T \in \mathbb{R}^{d\times s}$
/*w_i sampled from s dimensional unit sphere*/
$P_{nys} = P_{rp}\Lambda^{-\frac{1}{2}} Q^T$
return P_{nys} , $\mathcal{Z}$

Algorithm 2: Encode one data point from $\mathcal{X}$

Require: $x_i \in \mathcal{X}$, Projection P_{nys}, Landmarks $\mathcal{Z}$ and kernel function K
$C^{(i)} = [K(x_i, z_1) \quad K(x_i, z_2) \quad \cdots \quad K(x_i, z_s)]^T \in \mathbb{R}^s$
return $\sqrt{\frac{\pi}{2d}}$ sign $\left(P_{nys} C^{(i)}\right)$

The rows of $P_{rp} \in \mathbb{R}^{d\times s}$ are sampled from the uniform distribution over the s dimensional unit sphere. This allows us to use the following result regarding sign-thresholded random projection - suppose $v, v' \in \mathbb{R}^n$ are unit vectors, with respect to randomness in the sampling of P_{rp}, it is well known that the following result holds (see for instance (Charikar 2002)):

$$\mathbb{E}\left[\frac{1}{d}\text{sign}(Pv)\cdot\text{sign}(Pv')\right] = (1 - 2\cos^{-1}(v\cdot v')/\pi) \tag{3}$$

NysHD generates hypervectors for which the similarity induced by the kernel K is preserved by dot product between encodings in $\mathcal{H}$. We summarize this result in the following theorem:

Theorem 1. *Define $\Theta_i = \Lambda^{-\frac{1}{2}} Q^T C^{(i)}$ and $\Theta_i \in \mathbb{R}^n$. Based on HDC encoding algorithm presented in alg. 1 and alg. 2, let K be an arbitrary positive semi-definite kernel, the encoding function $\phi : \mathcal{X} \to \mathcal{H}$ can be writen as following:*

$$\phi(x_i) = \sqrt{\frac{\pi}{2d}} sign(P_{rp}\Theta_i) \tag{4}$$

$$\mathbb{E}\left[\langle \phi(x_i), \phi(x_j)\rangle\right] \approx \frac{\hat{G}_{ij}}{\sqrt{\hat{G}_{ii}\hat{G}_{jj}}} \quad \textit{i.e. normalized kernel} \tag{5}$$

Where $\hat{G}_{ij}$ is estimated kernel value between x_i and x_j produced by Nyström method and the expectation is taken with respect to randomness and orthogonality in P_{rp}.

Proof: Let $\hat{\Theta}_i \in \mathbb{R}^s$ denote $\frac{\Theta_i}{||\Theta_i||}$. Noting that $\text{sign}(cv) = \text{sign}(v)$ for any $v \in \mathbb{R}^s$ and $c > 0$:

$$\begin{aligned}\langle \phi(x_i), \phi(x_j) \rangle &= \left(\sqrt{\frac{\pi}{2d}} \text{sign}\left(P_{rp}\Theta_i\right)^T \sqrt{\frac{\pi}{2d}} \text{sign}\left(P_{rp}\Theta_j\right) \right) \\ &= \frac{\pi}{2d} \text{sign}\left(P_{rp} \frac{\Theta_i}{||\Theta_i||} \right)^T \text{sign}\left(P_{rp} \frac{\Theta_j}{||\Theta_j||} \right) \\ &= \frac{\pi}{2} \left(\frac{1}{d} \text{sign}\left(P_{rp}\hat{\Theta}_i \right)^T \text{sign}\left(P_{rp}\hat{\Theta}_j \right) \right)\end{aligned} \tag{6}$$

Using result regarding sign-thresholded random projection in equation. 3, we have:

$$\mathbb{E}\left[\langle \phi(x_i), \phi(x_j) \rangle\right] \approx \frac{\pi}{2} \left(1 - \frac{2\cos^{-1}\left(\hat{\Theta}_i^T \hat{\Theta}_j\right)}{\pi} \right) \tag{7}$$

Since $||\hat{\Theta}_i|| = ||\hat{\Theta}_j|| = 1$, it follows $-1 \leq \hat{\Theta}_i^T \hat{\Theta}_j \leq 1$, which is within the domain of $\cos^{-1}$. Use first order Taylor series of $\cos^{-1}(x)$ that $\cos^{-1}(x) \approx \frac{\pi}{2} - x$:

$$\mathbb{E}\left[\langle \phi(x_i), \phi(x_j) \rangle\right] \approx \frac{\pi}{2} \left(1 - \frac{2\left(\frac{\pi}{2} - \hat{\Theta}_i^T \hat{\Theta}_j\right)}{\pi} \right) \tag{8}$$

Recall Nyström method in equation 2: ${\Theta_i}^T \Theta_j = (\Lambda^{-\frac{1}{2}} Q^T C^{(i)})^T (\Lambda^{-\frac{1}{2}} Q^T C^{(j)}) = \hat{G}_{ij}$ and $||\Theta_i|| = \sqrt{{\Theta_i}^T \Theta_i} = \sqrt{\hat{G}_{ii}}$:

$$\hat{\Theta}_i^T \hat{\Theta}_j = \frac{{\Theta_i}^T \Theta_j}{||\Theta_i|| \cdot ||\Theta_j||} = \frac{\hat{G}_{ij}}{\sqrt{\hat{G}_{ii}\hat{G}_{jj}}} \tag{9}$$

Finally:

$$\begin{aligned}\mathbb{E}\left[\langle \phi(x_i), \phi(x_j) \rangle\right] &\approx \frac{\pi}{2} \left(1 - \frac{2\left(\frac{\pi}{2} - \frac{\hat{G}_{ij}}{\sqrt{\hat{G}_{ii}\hat{G}_{jj}}}\right)}{\pi} \right) \\ &= \frac{\hat{G}_{ij}}{\sqrt{\hat{G}_{ii}\hat{G}_{jj}}}\end{aligned} \tag{10}$$

As shown above, NysHD preserves the kernel in $\mathcal{H}$ in the sense that the inner-product between any pair of encoded data points approximates the normalized kernel value of some user defined kernel K. Since HDC uses inner-product based metric in $\mathcal{H}$ for inference as explained in section "Learning with HDC" , the similarity metric kernel K captures is also preserved, and is be beneficial for any subsequent HDC learning algorithms.

Encoding Complexity

The encoding method we are proposing here differs from existing HDC encoding methods in the sense that in order to generate embeddings using Nyström method, a partial kernel matrix needs to be computed. For that reason, similar to kernel machines, the efficiency of our encoding algorithm is highly dependent on the number of kernel computations, which is related to the size of landmarks s.

In this section we provide an asymptotic runtime analysis of the encoding algorithm. Assuming evaluating the kernel function K takes $\mathcal{O}(m)$, that is to say the complexity of kernel function is linear in the dimesntion of the input (e.g. gaussian kernel), the complexity of running the encoding algorithm over a dataset of n samples with s landmarks to d dimensional embedding is $O(\max(s^3, snm, snd))$. The first term corresponds to the symmetric eigen-decomposition of the matrix $H_{\mathcal{Z}}$, the second term reflects the number of kernel evaluations, and the last term represents sign-thresholded random projection as discussed previously. In a typical Nyström kernel approximation setting where $s \ll n$, depending on the nature of the kernel function being used, the second term is likely to be the dominant term, which is why the choice of s and K is crucial. Moreover, the analysis we provided here assumes the complexity of the kernel is linear in the number of features (for example, the Gaussian kernel), however, this is not always the case for kernel functions. For example, the spectrum kernel (Leslie, Eskin, and Noble 2001) which computes the number of unique N-grams between two input strings, has $\mathcal{O}(m \log(m))$ in the length of the input sequences.

Evaluation

We conduct an empirical evaluation of NysHD. We first provide an empirical validation of Theorem 1, we then study the practicality (both accuracy and efficiency) of our method against both HDC and non-HDC baselines.

Datasets and kernel functions

The encoding method we propose is general-purpose and can be applied to a wide range of data and tasks as long as a suitable kernel function exists. To confirm its versatility, we conduct assessments across two distinct tasks: **Graphs** and **Strings** classification. In total, we have chosen 8 graph datasets from TUDataset (Morris et al. 2020), a well-known standardized repository for graph classification bench-marking datasets; and 4 string classification datasets on bio-sequence and text classification. More information on datasets can be found in Table. 1.

The main advantage of our method is that it generate embeddings for HDC learning algorithms that preserve any user-defined positive-semidefinite kernel function. As such, the choice of kernel function is crucial for achieving the best accuracy and efficiency. For our method, we use the gappy (Leslie, Kuang, and Bennett 2004) kernel for string classification, which is a variant of the spectrum kernel (Leslie, Eskin, and Noble 2001), that computes the number of common N-grams (allowing a small amount of gap) between two input sequences. Propagation kernel (Neumann

Table 1: Summary of tasks and datasets

Task	dataset	# training	# testing	# class	Description
Graph	**ENZYMES** (Borgwardt et al. 2005)	480	120	6	Graph with attributed nodes
	NCI1 (Wale, Watson, and Karypis 2008)	3288	822	2	Graph with labeled nodes
	D&D (Dobson and Doig 2003)	943	235	2	Graph with labeled nodes
	BZR (Sutherland, O'brien, and Weaver 2003)	324	81	2	Graph with attributed nodes
	MUTAG (Debnath et al. 1991)	150	38	2	Graph with labeled nodes
	COX2 (Sutherland, O'brien, and Weaver 2003)	373	94	2	Graph with attributed nodes
	NCI109 (Wale, Watson, and Karypis 2008)	3301	826	2	Graph with labeled nodes
	Mutagenicity (Riesen and Bunke 2008)	3469	868	2	Graph with labeled nodes
String	**Protein** (Selvaraj et al. 2023)	721	181	6	Protein sequence
	SMS (Almeida and Hidalgo 2012)	4459	1115	2	Nautral Language
	Splice (mis 1992)	2552	628	3	DNA sequence
	Promoter (Harley and Noordewier 1990)	84	22	2	DNA sequence

et al. 2016) is used for graph classification for its ability to work with labeled or attributed graphs, as discussed in section "kernel methods" . We chose the aforementioned kernels for their relatively low computation complexity and effectiveness on respective tasks.

Experimental Setup and Baselines

We benchmark our method against the state of the art HDC encoding baselines within each domain: **Graph classification** using graph encoding scheme introduced in GraphHD (Nunes et al. 2022) and **String classification** use N-gram HDC encoding approach (Joshi, Halseth, and Kanerva 2017). The details of both HDC encoding methods are discussed in the background section.

In addition to HDC baselines, we also include comparisons with popular state of the art deep neural network architectures. For graph classification, we use **DGCNN** (Zhang et al. 2018), **GCN** (Chen, Bian, and Sun 2019), **GIN** (Xu et al. 2018), **GIUNet** (Amouzad et al. 2024). On string datasets, following the recent trend of applying large models for bio-sequence and language modeling (Qiu et al. 2020; Raffel et al. 2020), we fine-tune pre-trained large models as DNN baselines for string classification. Large protein model (**ESM-2-8m** (Rives et al. 2021)) is used for bio-sequence datasets, and large language model (**BERT** (Devlin et al. 2018)) is used for natural language dataset.

For our method, we set the number of landmarks as $s = max$(300, 2% of training data) on each dataset, and kernel specific hyperparamters are chosen empirically. To ensure the fairness of comparison, we use an identical HDC learning pipeline adapted from (Hernández-Cano et al. 2021) when evaluating different HDC encoding methods. The Perceptron algorithm (Rosenblatt 1958) is used to fine-tune class prototypes. In all of our experiments, 20 epochs of fine-tuning have been found to be sufficient.

All experiments were run on an Intel i5-11400 CPU (except for Large model fine-tune method for string datasets, for which a Nvidia RTX 3050 GPU is used due to long training time on CPU). We evaluate different methods by their training efficiency (time in seconds) and classification accuracy. Each experiment was run 10 times, and we report the mean and standard deviation of the results.

Approximating normalized kernel matrix

The accuracy performance of HDC based models hinge on the capability of encodings to capture some salient notion of similarity via inner product. One straight-forward way to make sure the HDC encodings produced by our method do indeed preserve the kernel value of interest is to compare the spectral norm between the actually normalized kernel matrix computed via the kernel function and the pairwise inner product between our encodings. Using different percentages of the training samples as landmarks, the results of selected datasets (ENZYMES, NCI1, Protein, Promoter) are shown in Fig. 2. In line with Theorem.1, the method we propose preserves the normalized kernel value, whose quality is positively correlated with the number of landmarks (a less noisy approximation can be achieved with a larger number of landmarks). The results presented here indicate our method is effective in transferring the vast repertoire of similarity functions in kernel methods into HDC settings.

Accuracy and efficiency results

The accuracy results on graph and string datasets are summarized in Table 2 and Table 3. In comparison with the GraphHD, NysHD is able to achieve, on average, 11% better accuracy on graph datasets, the improvement is especially significant on the ENZYMES dataset (38% accuracy improvement), as the previous HDC encoding method on graph (Nunes et al. 2022) could not utilize node attributes in fully attributed graphs but could be done with our method. On string datasets, our method again consistently achieves better accuracy when compared with N-gram based HDC encoding (on average 17% better in accuracy). Efficiency-wise, our method is comparable to previous HDC encoding methods (slightly slower than GraphHD (Nunes et al. 2022) on graph datasets due to kernel computation).

Despite the efficiency and implementation simplicity of HDC learning algorithms, there remains an accuracy gap between HDC based models and DNN models on more complex classification tasks. We hope our work can potentially reduce that gap. To that end, we also include several state of the art DNN approaches on both graph and string classifications in our comparison. On graph datasets, NysHD achieved the best accuracy in 3 out of 8 graph datasets. On average, NysHD outperforms DGCNN by 3%,

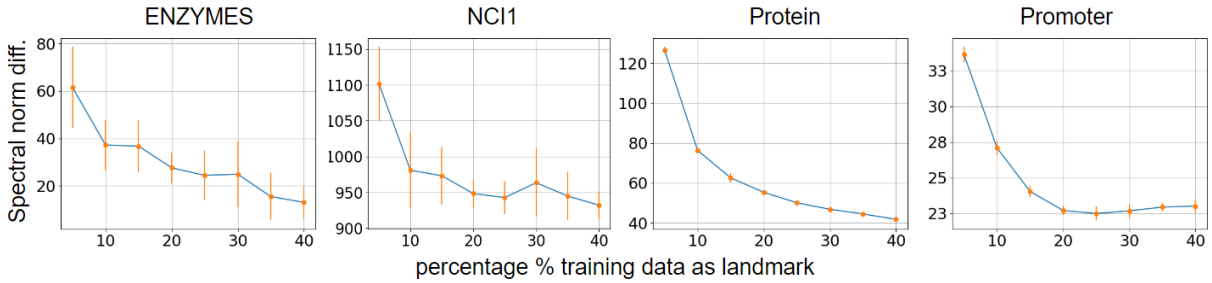

Figure 2: Numerical difference of spectral norm between normalized kernel matrices compute directly from kernel function and approximated kernel matrices with our encoding.

Table 2: Experimental results. The best accuracy result for each dataset are **highted** and second best are underlined.

Method		NCI1	ENZYMES	D&D	BZR	MUTAG	COX2	NCI109	Mutagenicity
DGCNN	Acc.	70.2 ± 2.2%	36.9 ± 4.6%	74.5 ± 3.4%	81.5 ± 4.4%	82.9 ± 4.1%	78.3 ± 2.7%	71.1 ± 2.2%	75.0 ± 1.3%
	Time	*37.5 ± 1.2s*	*5.6 ± 0.5s*	*59.2 ± 3.0s*	*4.0 ± 0.0s*	*1.0 ± 0.0s*	*4.1 ± 0.3s*	*36.2 ± 0.9s*	*44.8 ± 1.3s*
GCN	Acc.	**79.9 ± 1.1%**	60.7 ± 2.1%	74.8 ± 1.6%	**84.2 ± 2.2%**	**85.5 ± 3.8%**	**83.1 ± 3.7%**	**80.2 ± 1.1%**	**79.9 ± 1.0%**
	Time	*74.3 ± 0.6s*	*15.9 ± 0.5s*	*262.3 ± 9.2s*	*7.9 ± 0.3s*	*3.0 ± 0.0s*	*11.0 ± 0.0s*	*73.5 ± 0.7s*	*75.3 ± 0.5s*
GIN	Acc.	73.4 ± 1.7%	28.8 ± 4.2%	67.5 ± 5.9%	76.4 ± 8.4%	76.8 ± 9.6%	78.1 ± 4.3%	70.8 ± 2.6%	78.0 ± 1.7%
	Time	*66.8 ± 1.0s*	*9.0 ± 0.0s*	*63.6 ± 0.7s*	*6.0 ± 0.0s*	*2.0 ± 0.0s*	*7.2 ± 0.4s*	*67.1 ± 0.5s*	*70.5 ± 0.5s*
GIUNet	Acc.	72.4 ± 1.4%	29.9 ± 4.0%	63.4 ± 6.9%	78.3 ± 10.7%	85.0 ± 4.7%	77.4 ± 7.0%	68.8 ± 4.3%	76.5 ± 0.8%
	Time	*89.0 ± 0.4s*	*13.0 ± 0.0s*	*96.2 ± 0.7s*	*9.0 ± 0.0s*	*3.0 ± 0.0s*	*11.0 ± 0.0s*	*89.5 ± 0.5s*	*94.6 ± 0.5s*
GraphHD	Acc.	60.0 ± 2.1%	23.2 ± 2.3%	67.6 ± 1.3%	74.9 ± 2.1%	85.3 ± 10.2%	81.9 ± 3.9%	59.9 ± 2.1%	59.8 ± 1.7%
	Time	*30.0 ± 0.7s*	*6.0 ± 0.0s*	*32.9 ± 0.3s*	*3.0 ± 0.0s*	*1.0 ± 0.0s*	*3.0 ± 0.0s*	*30.0 ± 0.0s*	*31.3 ± 0.4s*
NysHD	Acc.	73.8 ± 2.2%	**61.3 ± 2.5%**	**76.2 ± 2.8%**	82.0 ± 3.3%	**85.5 ± 3.4%**	74.6 ± 3.7%	71.8 ± 2.0%	75.2 ± 0.9%
	Time	*43.8 ± 0.8s*	*7.2 ± 0.6s*	*35.4 ± 1.6s*	*3.4 ± 0.5s*	*2.0 ± 0.0s*	*5.5 ± 0.8s*	*44.4 ± 0.8s*	*35.7 ± 1.0s*

Table 3: Accuracy and training time on string datasets

Method	Protein	SMS	Promoter	Splice
LM	96 ± 0%	**99 ± 0%**	82 ± 2.7%	**92 ± 1.4%**
Finetune	*2003 ± 193s*	*2846 ± 31s*	*6 ± 3.0s*	*142 ± 1.9s*
N**-gram**	96 ± 0.8%	97 ± 0.3%	52 ± 4.0%	43 ± 4.9%
HDC	*13 ± 0.0s*	*43 ± 0.4s*	*1 ± 0.0s*	*25 ± 0.8s*
NysHD	**98 ± 0.3%**	97 ± 0.2%	**89 ± 3.4%**	72 ± 1.7%
	19 ± 0.7s	*34 ± 0.0s*	*1 ± 0.0s*	*21 ± 0.3s*

GIN and GIUNet by 6% for graph classification. Although graph convolutional network (GCN) still yield the best accuracy for some datasets, our method is on average 52% faster than GCN. On string datasets, our method with the gappy kernel achieves better accuracy on 2 out of 3 biosequence datasets. For SMS and Splice dataset, the LM finetune method achieves better classification accuracy, but their training time is 6 to 83 times longer than our method.

Overall, the strength of NysHD becomes most apparent when dealing with data with complex structures or attributes that traditional HDC encoding methods do not exploit.

Overhead & Scalability

This work allows future HDC works to exploit the power of kernel methods while still conforming to the general formalism and benefits of HDC. We recognize that the improvements in NysHD also comes with additional computation costs in the form of kernel evaluation. How to minimize such cost for HDC applications (especially from a hardware perspective) are non-trivial problems that need further investigations. The main scalability challenge is to obtain a set of landmarks that is as small as possible, while still providing a good approximation to the true kernel matrix. There is a large body of work on more sophisticated sampling schemes for the Nystrom method that could help make our method scalable to very large datasets (Kumar, Mohri, and Talwalkar 2012; Musco and Musco 2017). We would be interested to study these in future work.

Conclusion

The success of HDC based learning methods is contingent upon identifying an encoding function that preserves a suitable notion of similarity (kernel) for the task at hand. In this paper, we leverage the connection between the kernel method and HDC through the lens of Nyström method for kernel estimation. Particularly, we propose NysHD, a new HDC encoding method that constructs encoding functions using suitable kernel functions for specific tasks. As a result, NysHD is able to achieve substantial improvements (11% on graph datasets,17% on string datasets) when compared with previous HDC encoding method. There are many situations in which methods from the kernel literature outperform existing HDC based solutions, therefore, our approach can be expected to lead to performance improvements in many HDC applications as our experiments demonstrate.

References

1992. Molecular Biology (Splice-junction Gene Sequences). UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5M888.

Almeida, T.; and Hidalgo, J. 2012. SMS Spam Collection. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5CC84.

Amouzad, A.; Dehghanian, Z.; Saravani, S.; Amirmazlaghani, M.; and Roshanfekr, B. 2024. Graph isomorphism U-Net. *Expert Systems with Applications*, 236: 121280.

Asgarinejad, F.; Thomas, A.; and Rosing, T. 2020. Detection of epileptic seizures from surface eeg using hyperdimensional computing. In *EMBC*, 536–540. IEEE.

Borgwardt, K. M.; Ong, C. S.; Schönauer, S.; Vishwanathan, S.; Smola, A. J.; and Kriegel, H.-P. 2005. Protein function prediction via graph kernels. *Bioinformatics*, 21.

Brin, S.; and Page, L. 1998. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7): 107–117.

Charikar, M. S. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, 380–388.

Chen, T.; Bian, S.; and Sun, Y. 2019. Are powerful graph neural nets necessary? a dissection on graph classification. *arXiv preprint arXiv:1905.04579*.

Debnath, A. K.; Lopez de Compadre, R. L.; Debnath, G.; Shusterman, A. J.; and Hansch, C. 1991. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2): 786–797.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dobson, P. D.; and Doig, A. J. 2003. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4): 771–783.

Drineas, P.; Mahoney, M. W.; and Cristianini, N. 2005. On the Nyström Method for Approximating a Gram Matrix for Improved Kernel-Based Learning. *JMLR*, 6(12).

Dutta, A.; Gupta, S.; Khaleghi, B.; Chandrasekaran, R.; Xu, W.; and Rosing, T. 2022. Hdnn-pim: Efficient in memory design of hyperdimensional computing with feature extraction. In *GLSVLSI*, 281–286.

Gudmundsson, S.; Runarsson, T. P.; and Sigurdsson, S. 2008. Support vector machines and dynamic time warping for time series. In *2008 IEEE International Joint Conference on Neural Networks*, 2772–2776. IEEE.

Harley, R. R., C.; and Noordewier, M. 1990. Molecular Biology (Promoter Gene Sequences). UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5S01D.

Harris, Z. S. 1954. Distributional structure. *Word*, 10(2-3): 146–162.

Hernández-Cano, A.; Matsumoto, N.; Ping, E.; and Imani, M. 2021. Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system. In *DATE*, 56–61. IEEE.

Hertz, J. A. 2018. *Introduction to the theory of neural computation*. Crc Press.

Hofmann, T.; Schölkopf, B.; and Smola, A. J. 2008. Kernel methods in machine learning.

Imani, M.; Morris, J.; Messerly, J.; Shu, H.; Deng, Y.; and Rosing, T. 2019. Bric: Locality-based encoding for energy-efficient brain-inspired hyperdimensional computing. In *DAC*, 1–6.

Imani, M.; Nassar, T.; Rahimi, A.; and Rosing, T. 2018. Hdna: Energy-efficient dna sequencing using hyperdimensional computing. In *BHI*, 271–274. IEEE.

Imani, M.; Pampana, S.; Gupta, S.; Zhou, M.; Kim, Y.; and Rosing, T. 2020. Dual: Acceleration of clustering algorithms using digital-based processing in-memory. In *MICRO*, 356–371. IEEE.

Joshi, A.; Halseth, J. T.; and Kanerva, P. 2017. Language geometry using random indexing. In *Quantum Interaction: 10th International Conference, QI 2016, San Francisco, CA, USA, July 20-22, 2016, Revised Selected Papers 10*, 265–274. Springer.

Kanerva, P. 2009. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive computation*, 1: 139–159.

Kang, J.; Khaleghi, B.; Kim, Y.; and Rosing, T. 2022a. Xcelhd: An efficient gpu-powered hyperdimensional computing with parallelized training. In *ASP-DAC*, 220–225. IEEE.

Kang, J.; Khaleghi, B.; Rosing, T.; and Kim, Y. 2022b. Openhd: A gpu-powered framework for hyperdimensional computing. *IEEE Transactions on Computers*, 71(11): 2753–2765.

Khaleghi, B.; Kang, J.; Xu, H.; Morris, J.; and Rosing, T. 2022. GENERIC: highly efficient learning engine on edge using hyperdimensional computing. In *DAC*, 1117–1122.

Kleyko, D.; Rachkovskij, D.; Osipov, E.; and Rahimi, A. 2023. A survey on hyperdimensional computing aka vector symbolic architectures, part ii: Applications, cognitive models, and challenges. *ACM Computing Surveys*, 55(9): 1–52.

Kumar, S.; Mohri, M.; and Talwalkar, A. 2012. Sampling methods for the Nyström method. *JMLR*, 13(1): 981–1006.

Leslie, C.; Eskin, E.; and Noble, W. S. 2001. The spectrum kernel: A string kernel for SVM protein classification. In *Biocomputing 2002*, 564–575. World Scientific.

Leslie, C.; Kuang, R.; and Bennett, K. 2004. Fast string kernels using inexact matching for protein sequences. *JMLR*, 5(9).

Meanti, G.; Carratino, L.; Rosasco, L.; and Rudi, A. 2020. Kernel methods through the roof: handling billions of points efficiently. *NeurIPS*, 33: 14410–14422.

Menon, A.; Sun, D.; Sabouri, S.; Lee, K.; Aristio, M.; Liew, H.; and Rabaey, J. M. 2022. A highly energy-efficient hyperdimensional computing processor for biosignal classification. *TBCAS*.

Miranda, V.; and d'Aliberti, O. 2022. Hyperdimensional computing encoding schemes for improved image classification. In *HST*, 1–9. IEEE.

Morris, C.; Kriege, N. M.; Bause, F.; Kersting, K.; Mutzel, P.; and Neumann, M. 2020. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*.

Morris, J.; Ergun, K.; Khaleghi, B.; Imani, M.; Aksanli, B.; and Rosing, T. 2021. Hydrea: Towards more robust and efficient machine learning systems with hyperdimensional computing. In *DATE*, 723–728. IEEE.

Musco, C.; and Musco, C. 2017. Recursive sampling for the nystrom method. *NeurIPS*, 30.

Neumann, M.; Garnett, R.; Bauckhage, C.; and Kersting, K. 2016. Propagation kernels: efficient graph kernels from propagated information. *Machine learning*, 102: 209–245.

Nunes, I.; Heddes, M.; Givargis, T.; Nicolau, A.; and Veidenbaum, A. 2022. GraphHD: Efficient graph classification using hyperdimensional computing. In *DATE*, 1485–1490. IEEE.

Paxon Frady, E.; Kleyko, D.; Kymn, C. J.; Olshausen, B. A.; and Sommer, F. T. 2021. Computing on Functions Using Randomized Vector Representations. *arXiv e-prints*, arXiv–2109.

Plate, T. A. 1995. Holographic reduced representations. *IEEE Transactions on Neural networks*, 6(3): 623–641.

Qiu, X.; Sun, T.; Xu, Y.; Shao, Y.; Dai, N.; and Huang, X. 2020. Pre-trained models for natural language processing: A survey. *Science China technological sciences*, 63(10): 1872–1897.

Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; and Liu, P. J. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 21(140): 1–67.

Rahimi, A.; Kanerva, P.; Benini, L.; and Rabaey, J. M. 2018. Efficient biosignal processing using hyperdimensional computing: Network templates for combined learning and classification of ExG signals. *Proceedings of the IEEE*, 107(1): 123–143.

Rahimi, A.; and Recht, B. 2007. Random features for large-scale kernel machines. *NeurIPS*, 20.

Riesen, K.; and Bunke, H. 2008. IAM graph database repository for graph based pattern recognition and machine learning. In *SSPR & SPR 2008, Orlando, USA, December 4-6, 2008. Proceedings*, 287–297. Springer.

Rives, A.; Meier, J.; Sercu, T.; Goyal, S.; Lin, Z.; Liu, J.; Guo, D.; Ott, M.; Zitnick, C. L.; Ma, J.; et al. 2021. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences*, 118(15): e2016239118.

Rosenblatt, F. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6): 386.

Salamat, S.; Imani, M.; Khaleghi, B.; and Rosing, T. 2019. F5-hd: Fast flexible fpga-based framework for refreshing hyperdimensional computing. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 53–62.

Selvaraj, M. K.; Thakur, A.; Kumar, M.; Pinnaka, A. K.; Suri, C. R.; Siddhardha, B.; and Elumalai, S. P. 2023. Ion-pumping microbial rhodopsin protein classification by machine learning approach. *BMC bioinformatics*, 24(1): 29.

Shawe-Taylor, J.; and Cristianini, N. 2004. *Kernel methods for pattern analysis*. Cambridge university press.

Shimodaira, H.; Noma, K.-i.; Nakai, M.; and Sagayama, S. 2001. Dynamic time-alignment kernel in support vector machine. *NeurIPS*, 14.

Smola, A. J.; and Schölkopf, B. 1998. *Learning with kernels*, volume 4. Citeseer.

Steinwart, I. 2001. On the influence of the kernel on the consistency of support vector machines. *JMLR*, 2(Nov).

Sutherland, J. J.; O'brien, L. A.; and Weaver, D. F. 2003. Spline-fitting with a genetic algorithm: A method for developing classification structure- activity relationships. *Journal of chemical information and computer sciences*, 43(6): 1906–1915.

Thomas, A.; Dasgupta, S.; and Rosing, T. 2021. A theoretical perspective on hyperdimensional computing. *JAIR*, 72: 215–249.

Wale, N.; Watson, I. A.; and Karypis, G. 2008. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14: 347–375.

Williams, C.; and Seeger, M. 2000. Using the Nyström method to speed up kernel machines. *NeurIPS*, 13.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.

Xu, W.; Kang, J.; and Rosing, T. 2023. FSL-HD: Accelerating Few-Shot Learning on ReRAM using Hyperdimensional Computing. In *DATE*, 1–6. IEEE.

Yu, T.; Zhang, Y.; Zhang, Z.; and Sa, C. D. 2022. Understanding Hyperdimensional Computing for Parallel Single-Pass Learning. In Oh, A. H.; Agarwal, A.; Belgrave, D.; and Cho, K., eds., *NeurIPS*.

Zhang, M.; Cui, Z.; Neumann, M.; and Chen, Y. 2018. An end-to-end deep learning architecture for graph classification. In *AAAI*, volume 32.

Zhang, T.; Morris, J.; Stewart, K.; Lui, H. W.; Khaleghi, B.; Thomas, A.; Goncalves-Marback, T.; Aksanli, B.; Neftci, E. O.; and Rosing, T. 2023. : Accelerating Event-based Workloads with HyperDimensional Computing and Spiking Neural Networks. *TCAD*.

Zhao, Q.; Lee, K.; Liu, J.; Huzaifa, M.; Yu, X.; and Rosing, T. 2022. FedHD: federated learning with hyperdimensional computing. In *MobiCom*, 791–793.

Bridging the Gap between Hyperdimensional Computing and Kernel Methods via the Nyström Method - supplemental material

Anonymous submission

Omitted proof of equation 3

Equation 3 states that the the sign-thresholded random projection approximates a version of the angular kernel (Honeine and Richard 2010):

$$\mathbb{E}\left[\frac{1}{d}\text{sign}(Pv)\cdot\text{sign}(Pv')\right] = (1 - 2\cos^{-1}(v\cdot v')/\pi)$$

Proof: Recall that w_i is sampled from s dimensional unit sphere and v, v' are unit vectors. This allows us to use result from locality sensitive hashing regarding random hyperplane rounding (Goemans and Williamson 1995; Charikar 2002):

$$\Pr\left[\text{sign}(w_i^T\cdot v)\neq\text{sign}(w_i^T\cdot v')\right] = \frac{\theta(v,v')}{\pi} = \frac{cos^{-1}(v\cdot v')}{\pi}$$

$$\mathbb{E}\left[\text{sign}(w_i^T\cdot v)\cdot\text{sign}(w_i^T\cdot v')\right] = \begin{cases}1 & \text{if } (\text{sign}(w_i^T\cdot v)=\text{sign}(w_i^T\cdot v'))\\ -1 & \text{if } (\text{sign}(w_i^T\cdot v)\neq\text{sign}(w_i^T\cdot v'))\end{cases}$$

That is to say the angle between v, v' is proportional to the probability of a random hyperplane separating v and v'. The rest of the proof directly follows the linearity of expectation:

$$\begin{aligned}&\mathbb{E}\left[\frac{1}{d}\text{sign}(Pv)\cdot\text{sign}(Pv')\right]\\ &=\frac{1}{d}\mathbb{E}\left[\text{sign}(Pv)\cdot\text{sign}(Pv')\right]\\ &=\frac{1}{d}\mathbb{E}\left[\sum_{i\in\{1,2\ldots d\}}\text{sign}(w_i^T\cdot v)\cdot\text{sign}(w_i^T\cdot v')\right]\\ &=\frac{1}{d}\sum_{i\in\{1,2\ldots d\}}\mathbb{E}\left[\text{sign}(w_i^T\cdot v)\cdot\text{sign}(w_i^T\cdot v')\right]\\ &=\frac{1}{d}\sum_{i}^{d}(1-\frac{cos^{-1}(v\cdot v')}{\pi}-\frac{cos^{-1}(v\cdot v')}{\pi})\\ &=1-2\cos^{-1}(v\cdot v')/\pi\end{aligned}$$

Graph dataset details

Table 1 lists the graph information for each of the graph dataset used in our experiment. Note that since COX2, BZR and ENZYMES come with both node labels and node attributions, only attributions are used for propagation kernel computation.

Table 1: Statistics of graph datasets used for evaluation

dataset	Avg # nodes	Avg # edges
NCI1	29.87	32.30
ENZYMES	32.63	62.14
D&D	284.32	715.66
BZR	35.75	38.36
MUTAG	17.93	19.79
COX2	41.22	43.45
NCI109	29.68	32.13
Mutagenicity	30.32	30.77

Details of kernel function

Propagation kernel: The propagation kernel introduced by Neumann and Garnett (Neumann et al. 2016) is a similarity function over attributed (or labeled, in which case node labels are replaced with one-hot-vectors) graphs. Utilizing the idea of graph diffusion, the basic idea is to use attribute distributions after random walk to measure the similarity between graphs. After t iteration of information propagation on the graph, propagation kernel is defined by comparing all pairs of nodes in the two graphs:

$$K(G_i^t, G_j^t) = \sum_{u\in G_i^t}\sum_{v\in G_j^t} k(u,v)$$

Where u, v are nodes of G_i^t, G_j^t, superscrip t denotes graph after t^{th} iteration of information propagation and $k(\cdot)$ denotes node kernel. We refer readers to the original paper (Neumann et al. 2016) for the details regarding the node kernel and information propagation scheme.

Gappy Kernel: The Gappy kernel (Leslie, Kuang, and Bennett 2004) is a similarity function commonly used in bioinformatics and natural language processing. Broadly speaking, the Gappy kernel is a modification of the well-known spectrum kernel (Leslie, Eskin, and Noble 2001).

The spectrum kernel measures the similarity between two sequences of symbols (i.e. DNA sequence, text). Based on

the concept of N-gram (k-mers), the spectrum kernel works by counting the number of appearances of each unique N-gram. Specifically:

$$K_{Spectrum}(x, y) = \sum_{a \in \text{unique k-mer}} \phi_a(x)\phi_a(y)$$

$$\phi_a(x) = \text{number of times } a \text{ occurs in } x$$

The intuition is that the kernel value between two sequences will be large if two sequences contain many of the similar sub-sequences. Conversely, if two sequences share no similar sub-sequences (dissimilar), their kernel value will be small.

Because it can make sense in certain settings to not distinguish between a N-gram with or without some number of gaps, the gappy kernel can be used to achieve this functionality.

Implementation detail

When evaluating different HDC encoding methods, we only change the encoding scheme while keeping the HDC learning algorithms fixed. We provide the details of HDC learning algorithms used in our experiments. The HDC learning algorithm is adapted from OnlineHD (Hernández-Cano et al. 2021). Following is the pseudo-code of the algorithm we used:

Algorithm 1: HDC Learning Algorithm

Require: Dataset $\mathcal{D}$, Encoderϕ, Class prototypes HV, Learning rate lr, number of epoch e
  for all epoch $\in \{1 \dots e\}$ **do**
    for all $(x, \text{label}) \in \mathcal{D}$ **do**
      enc $\leftarrow \phi(x)$
      predict $\leftarrow \arg\max_i \text{CosineSimilarity}(HV_i, \text{enc})$
      /*HV_i ***denotes prototype (hypervector) for i^{th} ***class*****/**
      if predict $\neq$ label **then**
        $HV_{label} \leftarrow HV_{label} + lr \times (1 - \text{CosineSimilarity}(\text{enc}, HV_{label})) \times \text{enc}$
        $HV_{predict} \leftarrow HV_{predict} - lr \times (1 - \text{CosineSimilarity}(\text{enc}, HV_{predict})) \times \text{enc}$
      end if
    end for
  end for
  return HV

Use the perceptron algorithm (Rosenblatt 1958) The above algorithm trains class prototype hypervectos iteratively until converge. Class prototypes are initialized to be vectors of zeros at the beginning of the training procedure. Learning rate are chosen from $lr \in \{0.1, 0.2 \dots 1\}$ and we use $e = 20$ for all experiments.

Kernel specific hyperparameters

To ensure reproducbility, we further provide relevant hyperparameters for kernel function used. Note that all hyperparameters were chosen empirically.

Table 2: Kernel specific hyperparameters for each dataset

dataset	Kernel	Parameter
NCI1	Propagation	$t = 5$
ENZYMES	Propagation	$t = 1$
D&D	Propagation	$t = 2$
BZR	Propagation	$t = 1$
MUTAG	Propagation	$t = 10$
COX2	Propagation	$t = 5$
NCI109	Propagation	$t = 5$
Mutagenicity	Propagation	$t = 3$
SMS	Gappy	$k = 4, g = 1$
Protein	Gappy	$k = 4, g = 1$
Promoter	Gappy	$k = 4, g = 1$
Splice	Gappy	$k = 4, g = 1$

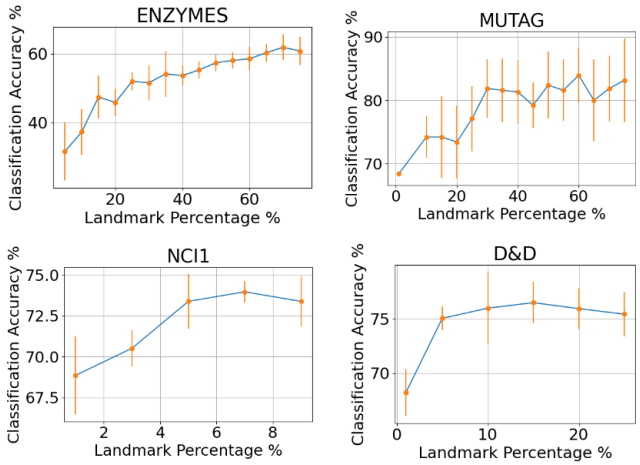

Figure 1: Effects of different number of landmarks

For the propagation kernel, $t \in \{1, 2 \dots 10\}$ denotes the maximum number of iterations of graph information propagation. For the gappy kernel, $k \in \{2, 4, 6\}$ determines the length of N-gram (K-mer) and $g \in \{1, 2\}$ denotes the number of gaps the kernel allows.

Sensitivity analysis on selected datasets

As discussed in the main body of this paper, for classification tasks using HDC, it is desirable that the inner products between embeddings capture some useful notion of similarity, or in other words, some kernel. The quality of embeddings generated by our method is determined by how well the inner products approximate the kernel value. Since the Nyström method itself is a sampling-based low-rank matrix approximation technique, the size of the landmark set s is crucial in ensuring a good classification result. However, while a large s is desirable for classification accuracy, it also increases the complexity of the encoding process. Choosing an appropriate value for s is important in balancing the trade-offs between encoding efficiency and accuracy.

In Figure. 1, we present the effects of using different numbers of landmarks on four datasets. As shown there, datasets

of different complexity require a different number of landmarks in order to achieve optimal accuracy.

Large model fine-tune information

Large models based on transformer architecture have been shown to be an effective way for sequence modeling (such as languages or signals), and they can generalize to unseen domains of problems via fine-tuning (Qiu et al. 2020; Raffel et al. 2020). Following those works, we apply this method to sting datasets as one of our baselines. Pre-trained models used in our experiments and training parameters are provided here:

Table 3: Large model fine-tune information

dataset	Pre-trained Model	Learning rate
Protein	esm2_t6_8M_UR50D	$1e-5$
Promoter	esm2_t6_8M_UR50D	$7e-5$
Splice	esm2_t6_8M_UR50D	$1e-5$
SMS	bert-base-cased	$1e-5$

Pre-trained weights were obtained from Hugging Face and one fully connected layer is attached at end for network fine-tuning. AdamW optimizer (Loshchilov and Hutter 2017) with cross entropy loss and 0.1 random dropout are used for fine-tuning.

Reproducibility

Randomness in experiments: Random seeds are used in experiments, and for each experiment, we report the mean and standard deviation of 10 trails. Note that since most of the dataset used in this paper does not have a train-test partition, we use a fix seed (42) for *train_test_split()* function from scikit-learn package to partition the data into 80% training set and 20% testing set.

Hardware Platform: Due to the extremely long training time on CPU for string datasets using large model fine-tune (LM Fine-tune) method, an Nvidia RTX 3050 GPU was used instead of CPU. All other experiments were conducted on an Intel i5-11400 CPU.

Code availability: We value the reproducibility of our work. Python code for replicating experimental results is submitted as a part of the supplemental material and we will make our code publicly available upon the acceptance of the paper.

References

Charikar, M. S. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, 380–388.

Goemans, M. X.; and Williamson, D. P. 1995. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *JACM*, 42(6): 1115–1145.

Hernández-Cano, A.; Matsumoto, N.; Ping, E.; and Imani, M. 2021. Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system. In *DATE*, 56–61. IEEE.

Honeine, P.; and Richard, C. 2010. The angular kernel in machine learning for hyperspectral data classification. In *2010 2nd Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing*, 1–4. IEEE.

Leslie, C.; Eskin, E.; and Noble, W. S. 2001. The spectrum kernel: A string kernel for SVM protein classification. In *Biocomputing 2002*, 564–575. World Scientific.

Leslie, C.; Kuang, R.; and Bennett, K. 2004. Fast string kernels using inexact matching for protein sequences. *JMLR*, 5(9).

Loshchilov, I.; and Hutter, F. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Neumann, M.; Garnett, R.; Bauckhage, C.; and Kersting, K. 2016. Propagation kernels: efficient graph kernels from propagated information. *Machine learning*, 102: 209–245.

Qiu, X.; Sun, T.; Xu, Y.; Shao, Y.; Dai, N.; and Huang, X. 2020. Pre-trained models for natural language processing: A survey. *Science China technological sciences*, 63(10): 1872–1897.

Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; and Liu, P. J. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 21(140): 1–67.

Rosenblatt, F. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6): 386.