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

Quanling Zhao, Anthony Hitchcock 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 from the cognitive science literature for solving information processing tasks using data represented as high-dimensional random 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 effectiveness of HDC in machine learning largely depends on how raw data is mapped 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.

Code — https://github.com/QuanlingZhao/NysHD

Introduction

Biological brains "compute" using data representations that are intrinsically fault-tolerant, suitable for highly-parallel circuitry, and reveal complex structures in an environment that are easy to learn (Hertz 2018). Motivated by these desirable qualities, hyperdimensional computing (HDC) builds on theories of representation from cognitive science (Kanerva 2009; Plate 1995) to develop novel hardware and algorithms for information processing tasks. 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 algorithms 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 (Hernández-Cano et al. 2021; 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 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 $\phi(x)$ residing in a high-dimensional inner product space $\mathcal{H}$ (Kleyko et al. 2023). HDC uses addition and multiplication (called 'bundling" and "binding" in the HDC literature), 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 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 prob-

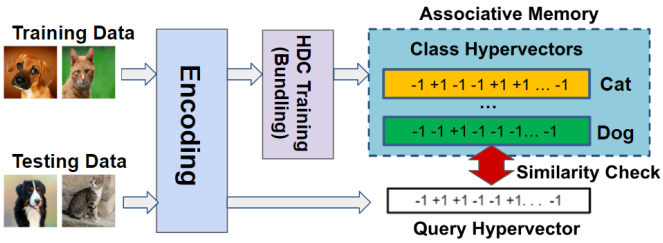

Figure 1: Overview of HDC training and inference for classification tasks.

lem of designing good similarity functions, which are also 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 kernel function into an equivalent encodings.
- We analyze the kernel-preserving properties of NysHD formally and show that the inner product between encoded samples preserves normalized kernel values.
- We perform an empirical evaluation against existing 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. Specifically, class j is represented as $\theta_j = \sum_{i=1}^{n} \alpha_{ij} \phi(x_i)$ where $\alpha_{ij} = 1(y_i = j)$. In practice, θ_j is sometimes quantized to reduce precision in some fashion, which is beneficial in some hardware settings. The label of a query x is predicted by via $\hat{y} = \arg\max_{j \in \{1,...,c\}} \phi(x) \cdot \theta_j$ where the operands are sometimes normalized if appropriate. In either case, this procedure can be interpreted as associating each class to a linear scoring function in $\mathcal{H}$ and performing inference by picking the class of highest score - a common paradigm in machine learning. Fine-tuning the class vectors θ_j is common, often achieved by running the Perceptron algorithm (Rosenblatt 1958), referred to as "retraining" in HDC literature.

A number of HDC encoding methods have been proposed to encode different types of data. For example, string or text data can be encoded 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 latin 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, we represent S as $a_1 \circ \rho(a_2) \circ \rho^2(a_3) \circ \cdots \circ \rho^{N-1}(a_N)$ where $\circ$ is element-wise 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 sum 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).

Existing 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). Much 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 function $K(x, x') = \psi(x) \cdot \psi(x')$, 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 $K(x, x') = (1+x \cdot x')^p$. 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 body of theoretical and applied literature on kernel methods that has developed kernel functions 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 provide a concrete example of how the literature on kernel methods can offer insights for the HDC community, we first consider the encoding of time-series data in HDC. Similar to the previously discussed N-gram encoding, the permutation operation is applied to encode the temporal information of time-series data (Joshi, Halseth, and Kanerva 2017; Asgarinejad, Thomas, and Rosing 2020). However, such encoding schemes can fail if the events in two time-series do not align exactly. Since each time step is associated with a unique permutation during encoding, even a small shift in events between time-series can cause existing HDC encoding methods to map the two time-series to nearly orthogonal vectors. In practice, however, if two time-series reflect the same underlying activity or nature, one would expect their similarity to be preserved after encoding, even if some event misalignment exists. On this issue, the literature on kernel methods suggests a solution: the dynamic-time-warping kernel (Gudmundsson, Runarsson, and Sigurdsson 2008), which can handle time-series sequences with misalignment or time-stretching/compression.

For graphs, GraphHD (Nunes et al. 2022) proposes to encode graph topology induced by PageRank centrality metric (Brin and Page 1998). However, this process does not utilize crucial information such as node labels or node attributes (where each node in the graph is associated with a feature vector or a label). Such limitations have been addressed by kernel methods. For example, the propagation kernell (Neumann et al. 2016) works with graphs that include node labels or node attributes, therefore is capable of capturing a potentially richer notion of similarity.

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 HDC representations.

Related Work

The connection between HDC and kernel approximation is generally well known (Yu et al. 2022; Thomas, Dasgupta, and Rosing 2021; Paxon Frady et al. 2021; Voelker 2020), 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 (kernel function that depends only on the relative distance between inputs, e.g. the Gaussian kernel and Laplacian kernels). 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, which many useful kernels do not satisfy, such as kernels on graphs and strings. The Nyström method 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 the Nyström method for kernel approximation. We also demonstrate that the inner product between encoded samples, in expectation, 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 $\phi(x) \cdot \phi(x') \approx K(x, x') \quad \forall x, x' \in \mathcal{D}$, where $\mathcal{D}$ is some subset of $\mathcal{X}$. This property is useful for learning algorithms that are widely employed in HDC as it enables them to exploit more useful similarities captured by the kernel.

Nyström Method

Conventional realizations of kernel-based learning algorithms commonly require storing all pairwise evaluations of the kernel function in a large matrix G defined elementwise by $G_{ij} = K(x_i, x_j)$, which is 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 store the entire kernel matrix (Williams and Seeger 2000; Drineas, Mahoney, and Cristianini 2005; Kumar, Mohri, and Talwalkar 2012). In this way, the 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 the Nyström method is that the Nyström method can work with a larger class of kernels than RFF, many of which are useful for applications involving discrete structures such as string and graph.

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 $\mathcal{Z} = \{z_1, ..., z_s\}$, 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 $H_{\mathcal{Z}}$. There is a robust theoretical literature on the Nyström method, providing bounds on approximation error based on the number of selected landmarks and various sampling strategies (Kumar, Mohri, and Talwalkar 2012). Let Q and Λ be the eigenvectors and eigenvalues of $H_{\mathcal{Z}}$ then:

$$H_Z^+ = Q\Lambda^{-1}Q^T$$

This allows one to generate encodings explicitly via $\phi_{\text{nys}}(x_i) = \Lambda^{-\frac{1}{2}}Q^T C^{(i)}$, which approximates the kernel:

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

Where $x_i, x_j \in \mathcal{D}$ and $C^{(i)}$ denotes the i^{th} row of C in a column vector. While the encodings produced by Nyström method directly satisfy the desired kernel approximation property, they are, in general, of high-precision which is undesirable in some settings of interest in HDC (Khaleghi et al. 2022). 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 depends on the size and composition of $\mathcal{Z}$. As an initial step, in this paper we simply use uniform sampling without replacement (Williams and Seeger 2000; Kumar, Mohri, and Talwalkar 2012) 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 lead to better performance and warrant further exploration in future work.

Encoding Process

To achieve the aforementioned goals, we compose random hyperplane rounding (Charikar 2002) with the Nyström method to generate HDC embeddings that approximate a desired kernel. Alg. 1 describes the process for generating the Nyström embedding matrix, followed by data point encoding in Alg. 2.

Algorithm 1: Generate the Nyström embedding matrix

Require: kernel K over $\mathcal{X}$, dataset $\mathcal{D}$, number of landmarks $s > 0$, HDC dimension $d > 0$
$\mathcal{Z} \leftarrow$ sample s points from $\mathcal{D}$ without replacement
/*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*/
$P_{\text{rp}} = [w_1, w_1, \cdots, w_d]^T \in \mathbb{R}^{d\times s}$
/*w_i sampled from s dimensional unit sphere*/
$P_{\text{nys}} = P_{\text{rp}}\Lambda^{-\frac{1}{2}}Q^T$
return P_{nys} , $\mathcal{Z}$

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

Require: $x_i \in \mathcal{D}$, Nyström embedding matrix 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 $\sqrt{\frac{\pi}{2d}}\,\text{sign}\left(P_{\text{nys}}C^{(i)}\right)$

The rows of $P_{\text{rp}} \in \mathbb{R}^{d\times s}$ are sampled from the uniform distribution over the s-dimensional unit sphere. This enables 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}, the following result holds with randomness in the sampling of P_{rp} (see for instance (Charikar 2002)):

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

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

Theorem 1. *Let K be a positive-definite kernel and, using the notation of Algorithms 1 and 2, and recalling that $\phi_{nys}(x_i) = \Lambda^{-\frac{1}{2}}Q^T C^{(i)}$, define:*

$$\phi(x_i) = \sqrt{\frac{\pi}{2d}}\,sign(P_{rp}\phi_{nys}(x_i))$$

Then, for all $x_i, x_j \in \mathcal{D}$:

$$\mathbb{E}\left[\phi(x_i)\cdot\phi(x_j)\right] = \frac{\pi}{2} - \cos^{-1}\left(\frac{\hat{G}_{ij}}{\sqrt{\hat{G}_{ii}\hat{G}_{jj}}}\right) \tag{4}$$

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

Proof: Let $\bar{\phi}_{\text{nys}}(x_i)$ denote normalization: $\frac{\phi_{\text{nys}}(x_i)}{||\phi_{\text{nys}}(x_i)||}$. Noting that $\text{sign}(cv) = \text{sign}(v)$ for any v if $c \geq 0$:

$$\begin{aligned} &\phi(x_i) \cdot \phi(x_j) \\ &= \frac{\pi}{2d} \left(\text{sign}\left(P_{\text{rp}}\phi_{\text{nys}}\ (x_i)\right) \cdot \text{sign}\left(P_{\text{rp}}\phi_{\text{nys}}\left(x_j\right)\right)\right) \\ &= \frac{\pi}{2}\left(\frac{1}{d}\text{sign}\left(P_{\text{rp}}\bar{\phi}_{\text{nys}}(x_i)\right) \cdot \text{sign}\left(P_{\text{rp}}\bar{\phi}_{\text{nys}}(x_j)\right)\right) \end{aligned} \tag{5}$$

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

$$\begin{aligned} &\mathbb{E}\left[\phi(x_i) \cdot \phi(x_j)\right] \\ &= \frac{\pi}{2}\left(1 - \frac{2\cos^{-1}\left(\bar{\phi}_{\text{nys}}(x_i) \cdot \bar{\phi}_{\text{nys}}(x_j)\right)}{\pi}\right) \end{aligned} \tag{6}$$

Recalling the Nyström method in equation 2 and the fact that $||\phi_{\text{nys}}(x_i)|| = \sqrt{\phi_{\text{nys}}(x_i) \cdot \phi_{\text{nys}}(x_i)} = \sqrt{\hat{G}_{ii}}$:

$$\bar{\phi}_{\text{nys}}(x_i) \cdot \bar{\phi}_{\text{nys}}(x_j) = \frac{\hat{G}_{ij}}{\sqrt{\hat{G}_{ii}\hat{G}_{jj}}} \tag{7}$$

Finally:

$$\mathbb{E}\left[\phi(x_i) \cdot \phi(x_j)\right] = \frac{\pi}{2} - \cos^{-1}\left(\frac{\hat{G}_{ij}}{\sqrt{\hat{G}_{ii}\hat{G}_{jj}}}\right) \blacksquare \tag{8}$$

To make the relationship with kernel approximation more explicit, consider the first order Taylor expansion of $\cos^{-1}$. Since $||\bar{\phi}_{\text{nys}}(x_i)|| = ||\bar{\phi}_{\text{nys}}(x_j)|| = 1$, it follows that $-1 \leq \bar{\phi}_{\text{nys}}(x_i) \cdot \bar{\phi}_{\text{nys}}(x_j) \leq 1$, which means $\frac{\hat{G}_{ij}}{\sqrt{\hat{G}_{ii}\hat{G}_{jj}}}$ is within the domain of $\cos^{-1}$, So:

$$\begin{aligned} \mathbb{E}\left[\phi(x_i) \cdot \phi(x_j)\right] &\approx \frac{\pi}{2} - \left(\frac{\pi}{2} - \frac{\hat{G}_{ij}}{\sqrt{\hat{G}_{ii}\hat{G}_{jj}}}\right) \\ &= \frac{\hat{G}_{ij}}{\sqrt{\hat{G}_{ii}\hat{G}_{jj}}} \end{aligned} \tag{9}$$

Note that this is called normalized kernel (Ah-Pine 2010). As shown above, NysHD preserves the kernel in $\mathcal{H}$ (up to the first order approximation) 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 relies on the inner product in $\mathcal{H}$ for inference, as described in Section "Learning with HD" , the similarity metric captured by the kernel K is preserved, which benefits subsequent HDC learning algorithms.

Encoding Complexity

The proposed encoding method differs from existing HDC methods because generating embeddings using the Nyström method requires computing a partial kernel matrix. For this reason, similar to kernel machines, the efficiency of our algorithm depends heavily on the number of kernel computations, which is related to the size of landmarks set s.

In this section, we provide an asymptotic runtime analysis of the encoding algorithm. Assuming that evaluating the kernel function K takes $\mathcal{O}(m)$, that is to say the complexity of kernel function is linear in the dimension of the input (e.g., the 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 the 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 compares the N-gram composition 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. First, we validate 1 and then evaluate the practicality of our method in terms of accuracy and efficiency against both HDC and non-HDC baselines.

Datasets and Kernel Functions

The proposed encoding method is general-purpose and applicable to various data and tasks, provided a suitable kernel function is available. 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 benchmarking datasets, and 4 string datasets for bio-sequence and text classification. More information on datasets can be found in Table. 1.

The main advantage of our method is that it generates 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 kernel (Leslie, Kuang, and Bennett 2004) for string classification. The gappy kernel is a variant of the spectrum kernel (Leslie, Eskin, and Noble 2001) that allows a small amount of gap within the N-grams. The propagation kernel (Neumann et al. 2016) is used for graph classification for its ability to work with labeled or attributed graphs, as discussed in the 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 existing HDC encoding baselines within each domain: **Graph classification** using the encoding scheme from introduced in

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 (Towell, Noordewier, and Shavlik 1992)	2552	628	3	DNA sequence
	Promoter (Harley and Noordewier 1990)	84	22	2	DNA sequence

Table 1: Summary of tasks and datasets

GraphHD (Nunes et al. 2022) and **String classification** uses 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 Large protein model (**ESM-2-8m** (Rives et al. 2021)) for for bio-sequence datasets, and large language model (**BERT** (Devlin et al. 2018)) for natural language dataset.

For our method, we set the number of landmarks: 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 OnlineHD (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 models fine-tuning, which required an 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 classification accuracy of HDC-based models hinges on the capability of encodings to capture some salient notion of similarity via inner product. A straightforward way to verify our encodings preserve the kernel is to compare the spectral norm of the normalized kernel matrix (computed using the kernel function) with the pairwise inner products of the encodings. Using different percentages of the training samples as landmarks, the results for selected datasets (ENZYMES, NCI1, Protein, Promoter) are shown in Fig. 2. In line with Theorem 1, our method 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 here indicate that our method is effective in transferring similarity functions in kernel methods to HDC settings.

Accuracy and Efficiency Results

The accuracy results for graph and string datasets are summarized in Table 2 and Table 3. In comparison with the GraphHD, NysHD achieves, on average, 11% better accuracy on graph datasets. The improvement is especially significant on the ENZYMES dataset, which shows a 38% accuracy improvement. This is because the previous HDC encoding method on graph (Nunes et al. 2022) could not utilize node attributes in attributed graphs, whereas our method can. On string datasets, our method again consistently achieves better accuracy compared to N-gram based HDC encoding, with an average improvement of 17%. Efficiency-wise, our method is comparable to previous HDC encoding methods, although it is 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 deep neural network (DNN) models for more complex classification tasks. We hope this work will help reduce that gap. To that end, we also include several state-of-the-art DNN approaches on both graph and string classification 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%, 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 bio-sequence datasets. For SMS and Splice dataset, the LM fine-tune 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 existing HDC encoding methods do not exploit.

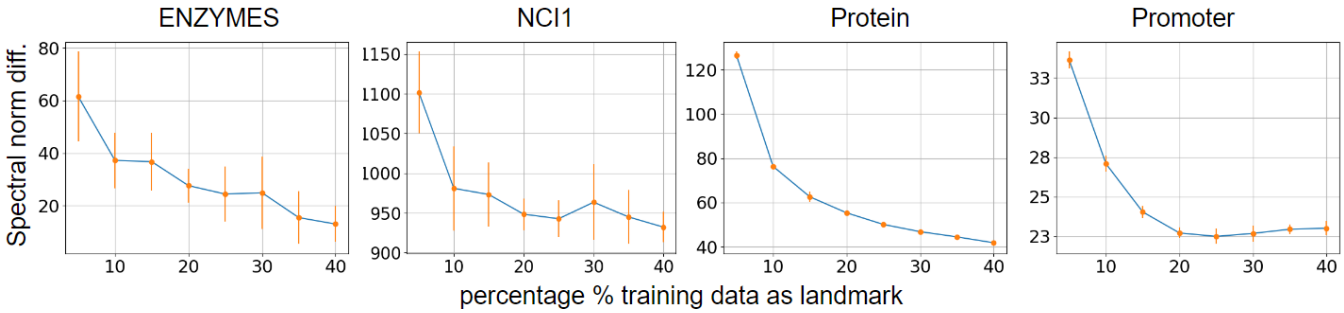

Figure 2: Numerical difference of spectral norm between normalized kernel matrices compute directly from kernel function and approximated kernel matrices with our encoding.

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

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*

Table 3: Accuracy and training time on string datasets

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 come with additional computation costs in the form of kernel evaluation. How to minimize such cost for HDC applications 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 Nyström method that could help make our methods scalable to larger datasets (Kumar, Mohri, and Talwalkar 2012; Musco and Musco 2017). We would be interested in studying 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, compared with previous HDC encoding methods, NysHD achieves substantial improvements - on average, 11% accuracy improvement on graph datasets and 17% on string datasets. 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.

Acknowledgements

This work was supported in part by National Science Foundation under Grants #2003279, #1826967, #2100237, #2112167, #1911095, #2112665, and in part by SRC under task #3021.001. This work was also supported in part by PRISM and CoCoSys, centers in JUMP 2.0, an SRC program sponsored by DARPA.

References

Ah-Pine, J. 2010. Normalized kernels as similarity indices. In *PAKDD 2010, Hyderabad, India, June 21-24, 2010. Proceedings. Part II 14*, 362–373. Springer.

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).

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–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.

Towell, G.; Noordewier, M.; and Shavlik, J. 1992. Primate splice-junction gene sequences (DNA) with associated imperfect domain theory.

Voelker, A. R. 2020. A short letter on the dot product between rotated fourier transforms. *arXiv preprint arXiv:2007.13462*.

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.