hdlib: A Python library for designing Vector-Symbolic Architectures

Fabio Cumbo, Emanuel Weitschek, and Daniel Blankenberg

1 Genomic Medicine Institute, Lerner Research Institute, Cleveland Clinic, Cleveland, Ohio, United States of America
2 Department of Engineering, Uninettuno University, Rome, Italy
3 Department of Molecular Medicine, Cleveland Clinic Lerner College of Medicine, Case Western Reserve University, Cleveland, Ohio, United States of America

Corresponding author

DOI: 10.21105/joss.05704

Software
- Review
- Repository
- Archive

Editor: Daniel S. Katz

Reviewers:
- @mahfuz05062
- @anilbey

Submitted: 03 July 2023
Published: 11 September 2023

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

Summary

Vector-Symbolic Architectures (VSA, a.k.a. Hyperdimensional Computing) is an emerging computing paradigm that works by combining vectors in a high-dimensional space for representing and processing information (Kanerva, 2014, 2009). This approach has recently shown promise in various domains for dealing with different kind of computational problems, including artificial intelligence (Haputhanthri et al., 2022; Osipov et al., 2022), cognitive science (Gayler, 2004; Graben et al., 2022), robotics (Neubert et al., 2019), natural language processing (Quiroz-Mercado et al., 2020), bioinformatics (Chen & Imani, 2022; Cumbo et al., 2020; Kim et al., 2020; Poduval et al., 2021), medical informatics (Lagunes & Lee, 2018; Ni et al., 2022), cheminformatics (Jones et al., 2023; Ma et al., 2022), and internet of things (Simpkin et al., 2020) among other scientific disciplines (Schlegel et al., 2022).

Here we present hdlib, a Python library for designing Vector-Symbolic Architectures. Its code is available on GitHub at https://github.com/cumbof/hdlib and it is distributed under the MIT license as a Python package through PyPI (pip install hdlib) and Conda on the conda-forge channel (conda install -c conda-forge hdlib). GitHub releases are also available on Zenodo at https://doi.org/10.5281/zenodo.7996502. Documentation with examples of how to use the library is also available at https://github.com/cumbof/hdlib/wiki.

Statement of need

The need for a general framework for designing vector-symbolic architectures is driven by the increasing success of the hyperdimensional computing paradigm for addressing complex problems in different scientific domains.

The design of such architectures is usually a time consuming task which requires the tuning of multiple parameters that are dependent upon the input data. By providing a general framework, here called hdlib, researchers can focus on the creative aspects of the architecture design, rather than being burdened by low-level implementation details.

Despite the presence of a few existing libraries for building vector-symbolic architectures (Heddes et al., 2023; Kang et al., 2022; Simon et al., 2022), the development of hdlib was driven by the need to offer increased flexibility and a more intuitive interface to complex abstractions, thereby facilitating a wider adoption in the research community. It not only consolidates most of the features from the existing libraries but also introduces novel functionalities which are easily accessible through a set of abstractions and reusable components as described in the following section, enabling rapid prototyping and experimentation with various architectural configurations.
Library overview

**hdlib** provides a comprehensive set of modules summarized in Figure 1.

**hdlib.space**

The library provides the Space and Vector classes under **hdlib.space** (see Figure 1 point 1) for building the abstract representation of a hyperdimensional space which acts as a container for a multitude of vectors.

**Vector objects**

Vectors are characterized by (i) a name or ID, (ii) a dimensionality usually greater than or equal to 10,000 to guarantee the quasi-orthogonality of random vectors in the high-dimensional space, (iii) the actual vector, (iv) the type of vector which can be binary or bipolar (i.e., with a random distribution of 0s and 1s as values or -1s and 1s respectively), and (v) an optional list of tags used to group vectors with common features.

The Vector class also provides the following three arithmetic functions for manipulating and combining Vector objects:

- **bind**: (i) it is invertible, (ii) it distributes over bundling (see bundle), (iii) it preserves the distance, and (iv) the resulting vector is dissimilar to the input vectors;
- **bundle**: (i) the resulting vector is similar to the input vectors, (ii) the more vectors are involved in bundling, the harder it is to determine the component vectors, and (iii) if several copies of any vector are included in bundling, the resulting vector is closer to the dominant vector than to the other components;
- **permute**: (i) it is invertible, (ii) it distributes over bundling and any element-wise operation, (iii) it preserves the distance, and (iv) the resulting vector is dissimilar to the input vectors.

It also provides a **dist** function for computing the distance between two Vector objects in the hyperdimensional space according to a specific similarity or distance measure (i.e., cosine similarity, euclidean distance, and hamming distance).

---

Cumbo et al. (2023). **hdlib**: A Python library for designing Vector-Symbolic Architectures. *Journal of Open Source Software, 8*(89), 5704. 2

https://doi.org/10.21105/joss.05704.
The Space object

On the other hand, a Space object is also characterized by a dimensionality and the type of vectors it can host. It is worth noting that different types of vectors cannot co-exist in the same space.

It provides several class methods for inserting, removing, and retrieving Vector objects from the hyperdimensional space (insert, remove, and get respectively as shown in Figure 1 point 1). It also provides a find method that, given an input vector, allows searching for the closest vector in the space according to a specific similarity or distance measure.

hdlib.arithmetic

hdlib also provides the same set of arithmetic functions also accessible as Vector's class methods (i.e., bind, bundle, and permute; see Figure 1 point 2). However, while the result of calling these functions from a Vector object would be applied in place, invoking the same functions from the hdlib.arithmetic module would initialize new Vector objects.

hdlib.model

The library also implements a novel supervised learning method initially proposed within the chopin2 tool https://github.com/cumbof/chopin2 (Cumbo et al., 2020; Cumbo & Weitschek, 2020) for processing massive amounts of genomics data with commodity hardware which took inspiration from the hierarchical vector-symbolic architecture originally proposed in (Imani et al., 2018). Here we reimplemented the same procedure which makes use of the hyperdimensional space, vectors, and the set of arithmetic operations already described above. The classification model can be easily integrated into other Python routines by simply loading the hdlib.model module and initializing a Model class instance (see Figure 1 point 3) by specifying the vectors dimensionality and the number of level vectors (i.e., the actual size of vectors in space, which is usually 10,000, and the number of vectors used to encode data that strictly depends on the range of numerical data in the input dataset; see (Cumbo et al., 2020) for additional details).

The Model object

The process of encoding data as described in (Cumbo et al., 2020) is provided with the fit method, while the classification model is built and evaluated through the predict function. The Model class also provides the cross_val_predict method that internally invokes the predict function on a predefined number of training and test set combinations in order to cross-validate the classification model.

It also implements a Model class method auto_tune that must be called right after the initialization of the model object. It allows performing a parameter sweep analysis on size and levels to automatically establish the best vector dimensionality and the most suitable number of level vectors for a given dataset over specific numerical ranges (please have a look at the official documentation for additional details).

It also implements a stepwise regression class method stepwise_regression that provides a backward variable elimination and a forward variable selection technique for selecting relevant features in a dataset. As a result of calling this method, a dictionary with an importance score for each feature is returned as well as the best accuracy reached for each importance score (lower is better in the case of method="backward", higher is better in the case of method="forward").

To the best of our knowledge, this is the first attempt of implementing a feature selection algorithm according to the hyperdimensional computing paradigm.

Please note that a few examples involving the use of the hdlib features are outlined in the official Wiki at https://github.com/cumbof/hdlib/wiki under the section Examples.
References


and Bioengineering (BIBE), 201–206. https://doi.org/10.1109/BIBE.2018.00046


