Automated Architecture Search for Brain-inspired Hyperdimensional Computing

Junhuan Yang[1] **Yi Sheng**[2] **Sizhe Zhang**[3] **Ruixuan Wang**[3] **Kenneth Foreman**[4]
Mikell Paige[4] **Dayane Reis**[5] **Xun Jiao**[3] **Weiwen Jiang**[2] **Lei Yang**[1]

[1]Department of Electrical and Computer Engineering, University of New Mexico
[2]Department of Electrical and Computer Engineering, George Mason University
[3]Department of Electrical and Computer Engineering, Villanova University
[4]Department of Chemistry and Biochemistry, George Mason University
[5]Department of Computer Science and Engineering, University of Notre Dame

Abstract This paper represents the first effort to develop an automated architecture search framework for hyperdimensional computing (HDC), a type of brain-inspired neural network. The framework, named AutoHDC, fills the gap in the optimization of HDC architecture design for given applications, which is currently carried out in an application-specific ad-hoc manner. Automated exploration will not only push HDC to more general applications, but also significantly diminish the heavy labor in architecture optimization for high performance and efficiency. To enable automated exploration, we present a thorough study to formulate the HDC architecture search space. On top of this, we apply reinforcement learning to automatically explore the HDC architectures. AutoHDC is evaluated in case studies on drug screening tasks in drug discovery. On the ClinTox dataset, AutoHDC can identify an architecture that outperforms the state-of-the-art deep learning approach with 4.77% higher ROC-AUC scores on average, and 2.26% higher scores against the manually designed HDC.

1 Introduction

Recently, brain-inspired hyperdimensional computing (HDC) has demonstrated its superiority in different machine learning tasks in terms of robustness, scalability, and high energy efficiency (Ge and Parhi (2020), Kanerva (2009), Thomas et al. (2021)). The main concept of HDC is to formulate a space with a set of high-dimension orthogonal vectors, so machine learning (ML) tasks are performed in such a space (Karunaratne et al. (2020)). Taking the classification task as an example, each training/testing sample is represented by a hypervector (HV) in the space. By applying the vector addition of training HVs in the same class, we obtain the HV to represent a class (called class HV) (Joshi et al. (2016)). In the inference phase, a given testing sample is encoded to an HV and the model searches for the class with the highest similarity to complete the task (Thomas et al. (2021)).

Similar to the neural architecture in deep neural networks (DNNs) (Yosinski et al. (2014)), HDC is also based on a set of architecture hyperparameters, called HDC architecture. Known from DNN applications, the customization of neural architectures can provide high performance and efficiency. HDC architecture also needs to be customized according to the target applications in

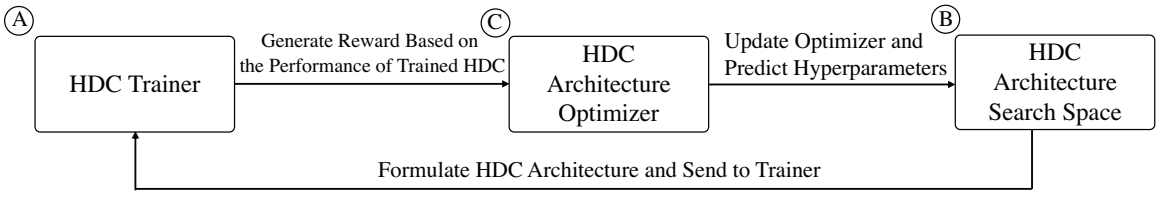

Figure 1: An overview of AutoHDC framework

different domains (Osipov et al. (2021), Neubert et al. (2019)). However, current HDC architectures are manually designed, which brings a high design cost, and in turn greatly limits the generality of HDC (Khan et al. (2021); Duan and Xu (2021)).

To enable the customization of HDC for different applications, the automation of HDC architecture will be the key. In the customization of DNNs, the automated ML (AutoML), in particular neural architecture search (NAS), has achieved great success (Zoph and Le (2016), Elsken et al. (2019)). It seems straightforward to apply the NAS to explore the HDC architecture. However, the fundamentally different computing schemes between HDC and DNN bring new challenges. More specifically, it is unclear what is the search space of the HDC architecture.

To the best of our knowledge, this is the first work to conduct an automated architecture search for HDC. We proposed a holistic framework, namely **AutoHDC**, to carry out the exploration, as shown in Figure 1. Through a thorough analysis of all operations in HDC, AutoHDC first constructs the fundamental search space in the design of the HDC architecture. On top of the search space, given an application, a reinforcement learning based search optimization is devised to automatically explore the best HDC architecture. Case studies on a series of drug discovery applications are conducted. Results show that our proposed AutoHDC can outperform DNNs and the existing manually designed HDC architectures.

2 Automated Architecture Search for Hyperdimensional Computing Framework

Figure 1 shows the overview of the AutoHDC framework, which comprises three components: Ⓐ an HDC Trainer to evaluate the identified HDC architecture; Ⓑ the HDC Architecture Search Space to formulate the searchable hyperparameters; Ⓒ an HDC Architecture Optimizer to control the search process. The optimization is iteratively conducted. In the following of this section, we introduce each component in details using a drug discovery classification task.

2.1 HDC Trainer

We first present the details of the trainer, as shown in Figure 2. It provides insights into what hyperparameters are searchable in HDC computing. Specifically, the trainer is composed of 4 phases: input, encoding, training, and output.

In "Phase 1", AutoHDC represents the molecules with Simplified Molecular-Input Line-Entry System (SMILES), which is used to describe chemicals' three-dimensional structure as a string.

In "Phase 2", there are further 4 steps to convert the input to an HV as shown in Figure 2. At step ①, each SMILES string is broken down into sub-strings of length N, and this method is called *N-gram* (Cavnar et al. (1994), Kondrak (2005)). As the example shown in the figure, the length $N = 3$, indicating that each gram has 3 characters, say "[N+" being the first, and "N+]" being the second. At step ②, a unique *base HV* will be randomly generated for each character. For example, the notation "[" corresponds to $HV_{[}$. In the next step ③, we use different rotation shifts to describe the appearance order of a character. For example, let HV_N be the HV to represent character N. In sub-unit "[N+" and "N+]", N is at different positions. The basic idea is that the same char (i.e., N in

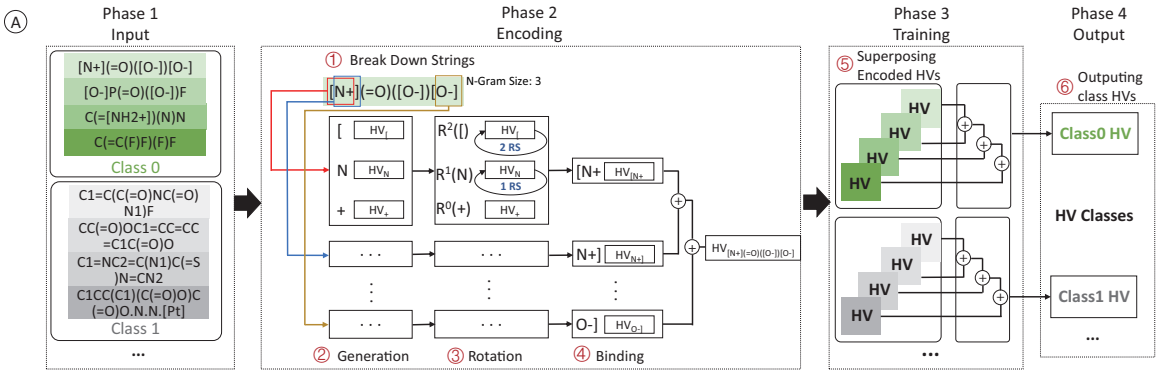

Figure 2: AutoHDC: 4-Phase Trainer

Ⓑ

Step	Properties or Operations	Options	Step	Properties or Operations	Options
①	N-Gram Size	1,2,3,4,5,6	③	Rotation Shift (RS)	0,1,2,3,4,5,6,7
②	Base HV Dimension	1k, 2k, ..., 20k	④	Element-wise Operation	*,XOR,OR,AND
②	Base HV Sparsity	10%,20%,...,90%	⑤	Encoded HV Type	binary, bipolar, int8, int16, 32int, int64
②	Base HV Type	binary, bipolar	⑥	Class HV Type	binary, bipolar, int8, int16, 32int, int64

Figure 3: AutoHDC Search Space: hyperparameters in HDC and their possible options

the above example) at the different positions should have similarities but should not be exactly the same. So, we will perform 1 rotation step for HV_N in "[N+" while 2 rotation steps for "N+]". Let i be the step for each rotation, it is relative to the $N - gram\ size$. And the rotation shift (RS) may not be stable. Specifically, the $i - th$ character in the string should rotate $(N - i) * RS$ elements. At step ④, after rotation, an element-wise operation will be carried out, which is called binding (Khan et al. (2021)). Binding is used to associate two HVs. Traditional HDC usually uses element-wise production to produce the resultant HV. Here, we may have other operations (e.g, XOR) to bind two HVs. After binding, the mathematical operation accumulates all the HVs resulting from the element-wise operation in one sub-unit (e.g., "[N+"). We also call this process "Superpose".

In "Phase 3", we get the HVs after encoding, also called encoded HVs which represent SMILES strings (e.g., "[N+](=O)([O-])[O-]"). At step ⑤, we specify the datatype of the encoded HVs to build up the model. The "Superpose" in "Phase 3" accumulates each encoded HV in the same class.

In "Phase 4", the results of "Superpose" in "Phase 3" formulate the class HVs. At step ⑥, similar with step ⑤, we need to specify the datatype of class HVs to build up the model, which will be stored in the associative memory.

2.2 HDC Architecture Search Space

We formulate the search space, as shown in Figure 3, according to the detailed analysis of steps (① to ⑥) in the previous subsection. Throughout the entire process of HDC trainer, there are 8 kinds of HV properties or operations during the 6 steps.

At step ①, the size of N-$Gram$ can be varied, that is, N is a hyperparameter. In the example, we set the search range of N from 1 to 6. At step ②, the process of base HV has 3 properties. First, the dimension of the base HV is a hyperparameter . Note that, once the base HV dimension is selected, the dimension of all of the HVs in this model will be consistent. In the example (Figure 3), the dimension ranges from 1000 to 20000 with step 1000. Second, the sparsity (i.e., the number of 1 in HV) is a hyperparameter. In traditional HDC, the value is fixed at 50%(Ma and Jiao (2021)). However, the fixed sparsity may not produce the best outcome. In AutoHDC, we provide the flexibility on sparsity, ranging from 10% to 90% with step 10%. Last, the datatype of the base HV is a hyperparameter, which can be in binary, consisting of "0" and "1", or bipolar format, consisting of "-1" and "1". At step ③, every HV rotates $(N - i) \times RS$ elements, where i is the appearance order in the substring and RS is a hyperparameter, representing the rotation shift. In the example, RS is set in a range from 0 to 7, where 0 means all HVs do not need to be rotated. At step ④, the type of element-wise operation is a hyperparameter. It is used to bind two HVs. In the example, we set 4 choices: 1) element-wise multiplication, 2) element-wise XOR, 3) element-wise AND, 4) element-wise OR. At step ⑤, we can specify the different datatype of encoded HVs. Unlike the base HV, the encoded HV datatype has more options (binary, bipolar, int8, int16, int32, and int64). Last, at step ⑥, we get the class HV. Similar to the encoded HVs, we can specify the class HV datatype as binary, bipolar, int8, int16, int32, and int64.

2.3 HDC Architecture Optimizer

Several optimizers can be used in the framework, like reinforcement learning and metaheuristic. The optimizer chooses the options from the search space for HDC trainer, and updates according to the reward from the trainer. Here in our tasks, the drug discovery datasets are much more likely to be imbalanced, so it is not effective to use accuracy as a metric to evaluate the performance of

Table 1: (Average) ROC-AUC scores of various approaches on ClinTox, BBBP, SIDER and BACE datasets

Models	ClinTox	BBBP	SIDER	BACE
Smi2Vec-BiGRU (Lin et al. (2020))	0.978	0.946	0.607	0.854
Smi2Vec-LSTM (Quan et al. (2018))	-	0.876	0.530	0.814
D-MPNN (Yang et al. (2019); Swanson (2019))	0.898	0.932	0.655	-
MoleHD(Ma and Jiao (2021))	0.982	0.916	0.568	0.662
AutoHDC	**0.995**	**0.959**	**0.661**	**0.872**

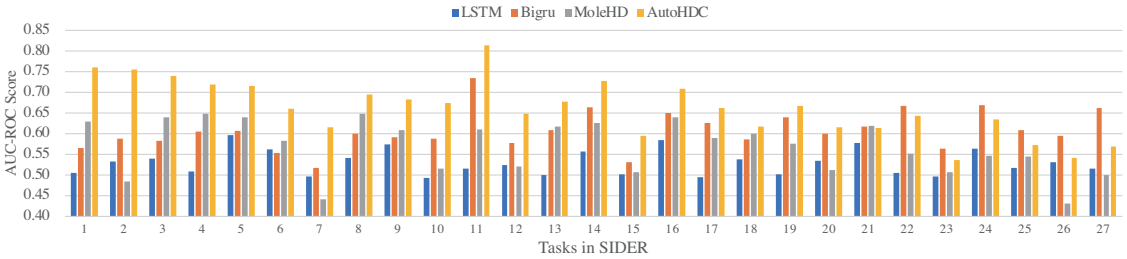

Figure 4: ROC-AUC scores of every task in SIDER.

models. Instead, receiver operating characteristics (ROC) curves and ROC Area-under-curve (AUC) scores are commonly used for binary and imbalanced datasets. Moreover, it is also suggested by widely used benchmark datasets along with the majority of literature (Wu et al. (2018); Ramsundar et al. (2019)). So, in this specific case, we use the ROC-AUR score as the reward.

3 Experiments

We employ 4 drug discovery datasets, including ClinTox (Gayvert et al. (2016)), BBBP (Martins et al. (2012)) , SIDER (Kuhn et al. (2016)), and BACE (Subramanian et al. (2016)), for performance evaluation. The stratified random splitting method is used to separate the datasets into 80%, 10%, and 10% to build the training, validation, and test sets, respectively. For each dataset, the framework will run 500 episodes to search for the best models. In each episode, each candidate HDC architecture will run 20 iterations and thus will be evaluated 20 times with the randomly generated base HVs. We use the average score of 20 iterations to avoid the good result produced by a specific seed. We evaluate the performance of the AutoHDC by comparing it with a set of state-of-the-art methods, including D-MPNN (Yang et al. (2019); Swanson (2019)), MoleHD (Ma and Jiao (2021)), Smi2Vec-LSTM (Quan et al. (2018)) and Smi2Vec-BiGRU(Lin et al. (2020)).

Table 1 shows the ROC-AUC results of the test set ("-" means the data is unavailable). Results show that our proposed AutoHDC framework can identify the model with the highest score for all 4 datasets. The model identified by AutoHDC can achieve a ROC-AUC score of 0.995 on ClinTox (average score for 2 tasks), which has improved the score at most 10.8%. On BBBP, the model searched by AutoHDC achieves a 0.959 score and improves the score by at least 1.37% and up to 9.47% compared with competitors. As well on SIDER (average score for 27 tasks), AutoHDC can achieve a score of 0.661. On the BACE dataset, the model found by AutoHDC with a score of 0.872 (at least 2.10% and up to 31.7% improvement), dominates other models.

Figure 4 shows the ROC-AUC scores of each task in the SIDER dataset (detailed results see Appendix A.3). We can see that, on 20 out of 27 tasks, the models identified by AutoHDC achieve the best scores. For the remaining 7 tasks (the last 7 tasks in the figure), AutoHDC can even identify the models with the second-best score for 6 tasks and the third-best score for 1 task, where the scores are close to the highest one. Overall, AutoHDC can obtain the leading scores, which show the competitiveness among other model candidates.

Combining all evaluation results, our proposed AutoHDC framework has been verified to be effective to find models with the best performance for drug discovery datasets compared with state-of-the-art approaches.

References

Cavnar, W. B., Trenkle, J. M., et al. (1994). N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval*, volume 161175. Citeseer.

Duan, S. and Xu, X. (2021). Hdcog: A lightweight hyperdimensional computing framework with feature extraction. In *2021 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pages 1–6. IEEE.

Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017.

Gayvert, K. M., Madhukar, N. S., and Elemento, O. (2016). A data-driven approach to predicting successes and failures of clinical trials. *Cell chemical biology*, 23(10):1294–1301.

Ge, L. and Parhi, K. K. (2020). Classification using hyperdimensional computing: A review. *IEEE Circuits and Systems Magazine*, 20(2):30–47.

Joshi, A., Halseth, J. T., and Kanerva, P. (2016). Language geometry using random indexing. In *International Symposium on Quantum Interaction*, pages 265–274. Springer.

Kanerva, P. (2009). Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive computation*, 1(2):139–159.

Karunaratne, G., Le Gallo, M., Cherubini, G., Benini, L., Rahimi, A., and Sebastian, A. (2020). In-memory hyperdimensional computing. *Nature Electronics*, 3(6):327–337.

Khan, A. A., Ollivier, S., Longofono, S., Hempel, G., Castrillon, J., and Jones, A. K. (2021). Brain-inspired cognition in next generation racetrack memories. *arXiv preprint arXiv:2111.02246*.

Kondrak, G. (2005). N-gram similarity and distance. In *International symposium on string processing and information retrieval*, pages 115–126. Springer.

Kuhn, M., Letunic, I., Jensen, L. J., and Bork, P. (2016). The sider database of drugs and side effects. *Nucleic acids research*, 44(D1):D1075–D1079.

Lin, X., Quan, Z., Wang, Z.-J., Huang, H., and Zeng, X. (2020). A novel molecular representation with bigru neural networks for learning atom. *Briefings in bioinformatics*, 21(6):2099–2111.

Ma, D. and Jiao, X. (2021). Molehd: Automated drug discovery using brain-inspired hyperdimensional computing. *arXiv e-prints*, pages arXiv–2106.

Martins, I. F., Teixeira, A. L., Pinheiro, L., and Falcao, A. O. (2012). A bayesian approach to in silico blood-brain barrier penetration modeling. *Journal of chemical information and modeling*, 52(6):1686–1697.

Neubert, P., Schubert, S., and Protzel, P. (2019). An introduction to hyperdimensional computing for robotics. *KI-Künstliche Intelligenz*, 33(4):319–330.

Osipov, E., Kahawala, S., Haputhanthri, D., Kempitiya, T., De Silva, D., Alahakoon, D., and Kleyko, D. (2021). Hyperseed: Unsupervised learning with vector symbolic architectures. *arXiv preprint arXiv:2110.08343*.

Quan, Z., Lin, X., Wang, Z.-J., Liu, Y., Wang, F., and Li, K. (2018). A system for learning atoms based on long short-term memory recurrent neural networks. In *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 728–733. IEEE.

Ramsundar, B., Eastman, P., Walters, P., and Pande, V. (2019). *Deep learning for the life sciences: applying deep learning to genomics, microscopy, drug discovery, and more*. O'Reilly Media.

Subramanian, G., Ramsundar, B., Pande, V., and Denny, R. A. (2016). Computational modeling of β-secretase 1 (bace-1) inhibitors using ligand based approaches. *Journal of chemical information and modeling*, 56(10):1936–1949.

Swanson, K. (2019). *Message passing neural networks for molecular property prediction*. PhD thesis, Massachusetts Institute of Technology.

Thomas, A., Dasgupta, S., and Rosing, T. (2021). Theoretical foundations of hyperdimensional computing. *Journal of Artificial Intelligence Research*, 72:215–249.

Wu, Z., Ramsundar, B., Feinberg, E. N., Gomes, J., Geniesse, C., Pappu, A. S., Leswing, K., and Pande, V. (2018). Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530.

Yang, K., Swanson, K., Jin, W., Coley, C., Eiden, P., Gao, H., Guzman-Perez, A., Hopper, T., Kelley, B., Mathea, M., et al. (2019). Are learned molecular representations ready for prime time? *arXiv preprint arXiv:1904.01561*.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27.

Zoph, B. and Le, Q. V. (2016). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.

A Appendix

A.1 AutoHDC Training, Retraining and Interence

The input data has been split into training, validation, and testing data. The training data is used to train the HDC model in "Phase 3" and "Phase 3", and the validation data is used to validate the model and perform rewards to update the controller, while the test data is used to test the performance of the model. We use an example to illustrate the HDC process and give the details of training, retraining and inference in this section.

A.1.1 Training. The randomly generated base HVs which use a seed are stored in the item memory. The item memory contains *base HVs* ($\overrightarrow{B}$) with the same number of characters ever shown in the input. We denote the base HVs as $\mathbf{B} = \{\overrightarrow{B_1}, \overrightarrow{B_2}, ..., \overrightarrow{B_i}, ..., \overrightarrow{B_m}\}$ where the $\overrightarrow{B_i}$ represents the base HV with index i and m represents the number of input unique characters. Thus, at steps ① and ②, every string can be encoded into a set of HVs.

Specifically, at step ④ (binding), there are 4 choices to bind 2 HVs in AutoHDC: 1) element-wise multiplication (donated as $*$), 2) element-wise XOR (donated as XOR), 3) element-wise AND (donated as AND), 4) element-wise OR (donated as OR). In these operations, we denote the first HV as $\overrightarrow{H_P} = \{\overrightarrow{H_{p(1)}}, \overrightarrow{H_{p(2)}}, ..., \overrightarrow{H_{p(i)}}, ..., \overrightarrow{H_{p(d)}}\}$ and the second HV as $\overrightarrow{H_Q} = \{\overrightarrow{H_{q(1)}}, \overrightarrow{H_{q(2)}}, ..., \overrightarrow{H_{q(i)}}, ..., \overrightarrow{H_{q(d)}}\}$. Equation (1)-(4) show the 4 element-wise operations for 2 HVs. Here, we re-define the element-wise operation results for bipolar data. Table 2 shows the results of the element-wise operations for binary and bipolar data by using our method.

After encoding, at step ⑤, we refer to these hypervectors for unit strings (e.g., "[N+](=O)([O-])[O-]") as *encoded HVs*, denoted as $\overrightarrow{E}$. Training is the process of establishing the original associative memory which includes specific HV for each class relative to the training data. We call the class HVs in associative memory *class HVs*, denoted as $\mathbf{C} = \{\overrightarrow{C_1}, \overrightarrow{C_2}, ..., \overrightarrow{C_j}, ..., \overrightarrow{C_c}\}$, where j and c represent the class label. Each training encoded HV $\overrightarrow{E_k}$ is added to the corresponding class HV $\overrightarrow{R_j}$ according to its label.

A.1.2 Retraining. The original associative memory after training is sufficient for some scenarios, however, retraining can help to further improve HDC performance. Retraining is a process to fine-tune the class HVs using the encoded HVs $\overrightarrow{E}$. The original associative memory is used to predict the corresponding label of one encoded HV $\overrightarrow{E_k}$. If the prediction is not correct, the relative class HV does not show or does not contain the correct information. Thus, as formulated in Equation (5, 6), the class HV with the prediction label ($\overrightarrow{C_p}$) should subtract the $\overrightarrow{E_k}$, and the class HV ($\overrightarrow{C_t}$) with the true label should be added with $\overrightarrow{E_k}$.

A.1.3 Inference. The validation or testing data is used to evaluate the performance of the model after training and retraining. The process of encoding these data to encoded HVs $\overrightarrow{E}$ is the same as the

$$\overrightarrow{H_P} * \overrightarrow{H_Q} = \{\overrightarrow{H_{p(1)}} * \overrightarrow{H_{q(1)}}, \overrightarrow{H_{p(2)}} * \overrightarrow{H_{q(2)}}, ..., \overrightarrow{H_{p(i)}} * \overrightarrow{H_{q(i)}}, ..., \overrightarrow{H_{p(d)}} * \overrightarrow{H_{q(d)}}\} \quad (1)$$

$$\overrightarrow{H_P}\ XOR\ \overrightarrow{H_Q} = \{\overrightarrow{H_{p(1)}}\ XOR\ \overrightarrow{H_{q(1)}}, \overrightarrow{H_{p(2)}}\ XOR\ \overrightarrow{H_{q(2)}}, ..., \overrightarrow{H_{p(i)}}\ XOR\ \overrightarrow{H_{q(i)}}, ..., \overrightarrow{H_{p(d)}}\ XOR\ \overrightarrow{H_{q(d)}}\} \quad (2)$$

$$\overrightarrow{H_P}\ AND\ \overrightarrow{H_Q} = \{\overrightarrow{H_{p(1)}}\ AND\ \overrightarrow{H_{q(1)}}, \overrightarrow{H_{p(2)}}\ AND\ \overrightarrow{H_{q(2)}}, ..., \overrightarrow{H_{p(i)}}\ AND\ \overrightarrow{H_{q(i)}}, ..., \overrightarrow{H_{p(d)}}\ AND\ \overrightarrow{H_{q(d)}}\} \quad (3)$$

$$\overrightarrow{H_P}\ OR\ \overrightarrow{H_Q} = \{\overrightarrow{H_{p(1)}}\ OR\ \overrightarrow{H_{q(1)}}, \overrightarrow{H_{p(2)}}\ OR\ \overrightarrow{H_{q(2)}}, ..., \overrightarrow{H_{p(i)}}\ OR\ \overrightarrow{H_{q(i)}}, ..., \overrightarrow{H_{p(d)}}\ OR\ \overrightarrow{H_{q(d)}}\} \quad (4)$$

$$\overrightarrow{C_p} = \overrightarrow{C_p} - \overrightarrow{E_k} \tag{5}$$

$$\overrightarrow{C_t} = \overrightarrow{C_t} + \overrightarrow{E_k} \tag{6}$$

Table 2: Truth table for binary and bipolar data when doing element-wise operation

Mult.	1	0	**XOR**	1	0	**AND**	1	0	**OR**	1	0
1	1	0	1	0	1	1	1	0	1	1	1
0	0	0	0	1	0	0	0	0	0	1	0
Mult.	1	-1	**XOR**	1	-1	**AND**	1	-1	**OR**	1	-1
1	1	-1	1	-1	1	1	1	-1	1	1	1
-1	-1	1	-1	1	-1	-1	-1	-1	-1	1	-1

process of encoding training data. The encoded HVs which are used to do inference are also called query HVs ($\overrightarrow{Q}$). The hamming similarity or cosine similarity is then calculated between a query HV $\overrightarrow{Q_l}$ and each class HV $\overrightarrow{C_j}$ in associative memory. The result of the largest similarity,(e.g., resulting from $\overrightarrow{Q_l}$ and $\overrightarrow{C_j}$), shows that $\overrightarrow{C_j}$ is the most similar class HV of the query HV $\overrightarrow{Q_l}$. Thus, the model predicts the query HV $\overrightarrow{Q_l}$ is most likely to have the same label as that of the class hypervector $\overrightarrow{C_j}$. The process of getting the largest similarity between a query HV $\overrightarrow{Q_l}$ and each class HV $\overrightarrow{C_j}$ is also called similarity search.

A.2 Details of Datasets

The 4 employed drug discovery datasets include ClinTox (Gayvert et al. (2016)), BBBP (Martins et al. (2012)), SIDER (Kuhn et al. (2016)) and BACE (Subramanian et al. (2016)) (2 binary classification tasks on ClinTox, 1 binary classification task on BBBP and BACE, 27 binary classification task on SIDER).

- ClinTox (Gayvert et al. (2016)): The dataset includes two classification tasks for 1491 drug compounds with known binary chemical structures: (1) clinical trial toxicity and (2) FDA approval status. In our experiment, we concentrate on the task of clinical trial toxicity.
- BBBP (Martins et al. (2012)): The dataset contains 2052 drug compounds with the corresponding binary label (positive or negative) of permeability to the blood-brain barrier.
- SIDER(Kuhn et al. (2016)): The dataset contains 1428 marketed drugs corresponding to adverse drug reactions (ADR) in 27 individual tasks per MedDRA classifications with the positive (active) or negative (inactive) label of classifying the relationship between the drug compound and the ADR disorders of system organs.
- BACE(Subramanian et al. (2016)): The dataset provides quantitative (IC50) and qualitative (binary label) binding results for a set of inhibitors of human β-secretase 1 (BACE-1). The dataset contains 1522 compounds with binary labels, built as a classification task.

A.3 Details of Experiment results on SIDER

Table 3: ROC-AUC scores of every task in SIDER

Task Name	Smi2Vec-LSTM	Smi2Vec-BiGRU	MoleHD	AutoHDC
Product issues	0.5048	0.5662	0.6296	**0.7602**
Endocrine disorders	0.5323	0.5873	0.4842	**0.7543**
Neoplasms benign, malignant and unspecified (incl cysts and polyps)	0.5396	0.5818	0.6400	**0.7393**
Eye disorders	0.5087	0.6044	0.6484	**0.7186**
Reproductive system and breast disorders	0.5956	0.6061	0.6392	**0.7152**
Musculoskeletal and connective tissue disorders	0.5620	0.5533	0.5834	**0.6593**
Pregnancy, puerperium and perinatal conditions	0.4961	0.5164	0.4422	**0.6145**
Blood and lymphatic system disorders	0.5408	0.6000	0.6480	**0.6950**
Cardiac disorders	0.5734	0.5918	0.6077	**0.6822**
General disorders and administration site conditions	0.4929	0.5884	0.5147	**0.6741**
Nervous system disorders	0.5147	0.7350	0.6104	**0.8135**
Immune system disorders	0.5248	0.5770	0.5211	**0.6482**
Congenital, familial and genetic disorders	0.5000	0.6084	0.6168	**0.6767**
Gastrointestinal disorders	0.5564	0.6629	0.6255	**0.7278**
Vascular disorders	0.5011	0.5303	0.5066	**0.5949**
Hepatobiliary disorders	0.5843	0.6504	0.6402	**0.7088**
Respiratory, thoracic and mediastinal disorders	0.4954	0.6250	0.5901	**0.6611**
Psychiatric disorders	0.5378	0.5861	0.6007	**0.6166**
Ear and labyrinth disorders	0.5012	0.6395	0.5754	**0.6673**
Metabolism and nutrition disorders	0.5345	0.5998	0.5114	**0.6159**
Renal and urinary disorders	0.5767	0.6173	**0.6196**	0.6137
Investigations	0.5045	**0.6674**	0.5522	0.6430
Surgical and medical procedures	0.4960	**0.5642**	0.5071	0.5359
Skin and subcutaneous tissue disorders	0.5642	**0.6683**	0.5463	0.6350
Social circumstances	0.5170	**0.6089**	0.5452	0.5716
Injury, poisoning and procedural complications	0.5315	**0.5943**	0.4306	0.5406
Infections and infestations	0.5148	**0.6621**	0.5002	0.5696
Average Score	0.5297	0.6071	0.5680	**0.6612**