BiSHop: Bi-Directional Cellular Learning for Tabular Data with Generalized Sparse Modern Hopfield Model

Chenwei Xu [*1] **Yu-Chao Huang** [*2] **Jerry Yao-Chieh Hu** [*1] **Weijian Li** [1] **Ammar Gilani** [1] **Hsi-Sheng Goan** [2 3 4] **Han Liu** [1 5]

Abstract

We introduce the **B**i-Directional **S**parse **Hop**field Network (**BiSHop**), a novel end-to-end framework for tabular learning. BiSHop handles the two major challenges of deep tabular learning: non-rotationally invariant data structure and feature sparsity in tabular data. Our key motivation comes from the recently established connection between associative memory and attention mechanisms. Consequently, BiSHop uses a dual-component approach, sequentially processing data both column-wise and row-wise through two interconnected directional learning modules. Computationally, these modules house layers of generalized sparse modern Hopfield layers, a sparse extension of the modern Hopfield model with learnable sparsity. Methodologically, BiSHop facilitates multi-scale representation learning, capturing both intra-feature and inter-feature interactions, with adaptive sparsity at each scale. Empirically, through experiments on diverse real-world datasets, BiSHop surpasses current SOTA methods with significantly fewer HPO runs, marking it a robust solution for deep tabular learning. The code is available on GitHub; future updates are on arXiv.

[*]Equal contribution [1]Department of Computer Science, Northwestern University, Evanston, IL, USA [2]Department of Physics and Center for Theoretical Physics, National Taiwan University, Taipei, Taiwan [3]Center for Quantum Science and Engineering, National Taiwan University, Taipei, Taiwan [4]Physics Division, National Center for Theoretical Sciences, Taipei, Taiwan [5]Department of Statistics and Data Science, Northwestern University, Evanston, IL, USA. Correspondence to: Chenwei Xu <cxu@northwestern.edu>, Yu-Chao Huang <r11222015@ntu.edu.tw>, Jerry Yao-Chieh Hu <jhu@u.northwestern.edu>, Weijian Li <weijianli@u.northwestern.edu>, Ammar Gilani <ammargilani2024@u.northwestern.edu>, Hsi-Sheng Goan <goan@phys.ntu.edu.tw>, Han Liu <hanliu@northwestern.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

1. Introduction

The field of developing deep learning architectures for tabular data has recently experienced rapid advancements (Somepalli et al., 2022; Gorishniy et al., 2021; Arik and Pfister, 2021; Huang et al., 2020). The primary driving force behind this trend is to overcome the limitations of the current dominant methods for tabular data: tree-based methods. Although tree-based methods excel in tabular learning, they cannot integrate with deep learning architectures. Therefore, pursuing deep tabular learning is not just a matter of enhancing performance but is also crucial to bridging the existing gap. Recent research has proposed new methodologies in deep tabular learning. However, a recent tabular benchmark study (Grinsztajn et al., 2022) reveals that tree-based methods still surpass deep learning models. This is due to two main challenges in deep tabular learning, as highlighted by Grinsztajn et al. (2022, Section 5.3 & 5.4):

(C1) **Non-Rotationally Invariant Data Structure:** The non-rotationally invariant structure of tabular data weakens the effectiveness of deep learning models that have rotationally invariant learning procedures.

(C2) **Feature Sparsity:** Tabular datasets are generally sparser than typical datasets used in deep learning, which makes it challenging for deep learning models to learn from uninformative features.

Inspired by the hierarchical and interconnected nature of the human brain (Presigny and Fallani, 2022; Krotov, 2021), we introduce the **Bi**-Directional **S**parse **Hop**field Network (**BiSHop**), a Hopfield-based deep learning framework tailored for tabular data. To address the non-rotationally invariant data structure of tabular data (C1), our model employs a dual-component design, named the Bi-directional Sparse Hopfield Module (BiSHopModule). This design mirrors the human brain's memory mechanisms, where different regions work collaboratively to form and retrieve associative memories. Our model uses bi-directional learning through two separate Hopfield models, focusing on column-wise and row-wise patterns separately. This approach incorporates the tabular data's inherent structure as an inductive bias.

For tackling the features sparsity in tabular data (C2), we utilize the generalized sparse modern Hopfield model (Wu

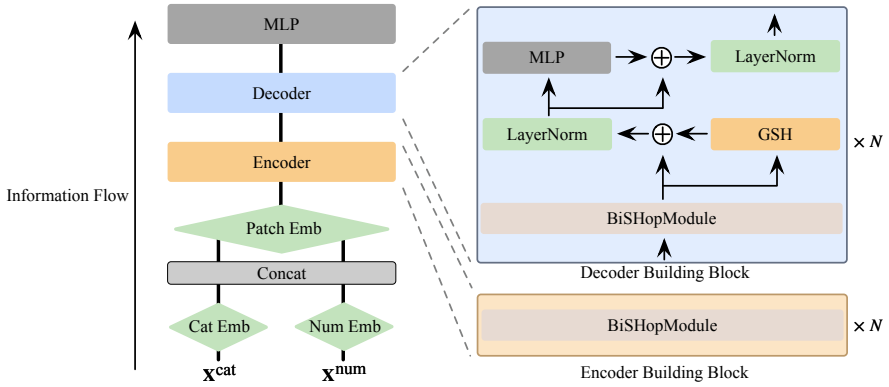

Figure 1. High-Level Visualization of BiSHop's Pipeline.

et al., 2024b). The generalized sparse modern Hopfield model is an extension to the sparse modern Hopfiled model (Hu et al., 2023) and modern Hopfiled model (Ramsauer et al., 2021) with the learnable sparsity. It offers robust representation learning and seamlessly integrates with existing deep learning architectures, ensuring focus on crucial information. Furthermore, inspired by brain's multi-level organization of associative memory, we stack multiple layers of the generalized sparse modern Hopfield model within BiSHopModule. As a result, each layer learns representations at unique scales, adjusting its sparsity accordingly, adding (C2) as another inductive bias to the model.

At its core, BiSHop facilitates multi-scale representation learning (Shi et al., 2022), capturing both intra-feature and inter-feature dynamics while adjusting sparsity for each scale. The model identifies representations across various scales in all directions, whether column-wise or row-wise. These refined representations are subsequently concatenated for downstream inference, ensuring a holistic bi-directional learning approach tailored for tabular data.

Contributions. Our contributions are twofold:

- Methodologically, we propose BiSHop, a novel deep-learning model for tabular data. BiSHop integrates two inductive biases (C1, C2) using the BiSHopModule and a hierarchical learning structure. The BiSHopModule utilizes the generalized sparse modern Hopfield model (Wu et al., 2024b) for tabular feature learning, enabling multi-scale sparsity learning with superior noise-robustness. We also present a hierarchical two-joint design to handle the intrinsic structure of tabular data with learnable sparsity and multi-scale cellular learning. Additionally, we adopt tabular embedding (Huang et al., 2020; Gorishniy et al., 2021; 2022) to enhance representation learning for both numerical and categorical features.
- Experimentally, we conduct comprehensive experiments on diverse real-world datasets as well as a tabular benchmark (Grinsztajn et al., 2022). This encompasses a total of 18 classification tasks and 11 regression tasks. We compare BiSHop with both SOTA tree-based and deep learning methods. Our results show that BiSHop outperforms baselines across most of tested datasets, including both regression and classification tasks.

Related Works. Appendix B includes the discussions of related works. **Notations.** We denote vectors by lowercase bold letters, and matrices by upper bold letters For vectors $\mathbf{a}, \mathbf{b}$, we define their inner product as $\langle \mathbf{a}, \mathbf{b} \rangle = \mathbf{a}^\top \mathbf{b}$. We use the shorthand $[I]$ to represent the index set $\{1, \cdots, I\}$ with I being a positive integer. For matrices, we denote the spectral norm as $\|\cdot\|$, which aligns with the l_2-norm for vectors. We denote the memory patterns by $\boldsymbol{\xi} \in \mathbb{R}^d$ and the query pattern by $\mathbf{x} \in \mathbb{R}^d$, and $\boldsymbol{\Xi} := [\boldsymbol{\xi}_1, \cdots, \boldsymbol{\xi}_M] \in \mathbb{R}^{d \times M}$ as shorthand for memory patterns $\{\boldsymbol{\xi}_\mu\}_{\mu \in [M]}$.

2. Background: Dense and Generalized Sparse Modern Hopfield Model

This section provides a concise overview of the modern Hopfield model (Ramsauer et al., 2021) and the generalized sparse modern Hopfield model (Wu et al., 2024b). Wu et al. (2024b) presents an extension to (Hu et al., 2023; Ramsauer et al., 2021), utilizing the Tsallis α-entropy (Tsallis, 1988)

2.1. (Dense) Modern Hopfield Models

Let $\mathbf{x} \in \mathbb{R}^d$ be the query pattern and $\boldsymbol{\Xi} = [\boldsymbol{\xi}_1, \cdots, \boldsymbol{\xi}_M] \in \mathbb{R}^{d \times M}$ the memory patterns. The aim of Hopfield models (Hopfield, 1982; 1984; Krotov and Hopfield, 2016; Demircigil et al., 2017; Krotov and Hopfield, 2021) is to store these memory patterns $\boldsymbol{\Xi}$ and retrieve a specific memory $\boldsymbol{\xi}_\mu$ when given a query $\mathbf{x}$. These models comprise two primary components: an *energy function* $E(\mathbf{x})$ that encodes memories into its local minima, and a *retrieval dynamics* $\mathcal{T}(\mathbf{x})$ that fetches a memory by iteratively minimizing $E(\mathbf{x})$ starting with a query.

Ramsauer et al. (2021) propose the (dense/vanilla) modern Hopfield model with a specific set of E and $\mathcal{T}$, and integrate it into deep learning architectures via its connection with attention mechanism, offering enhanced performance, and theoretically guaranteed exponential memory capacity. Specifically, they introduce a Hopfield energy function:

$$E(\mathbf{x}) = -\operatorname{lse}(\beta, \boldsymbol{\Xi}^\mathsf{T}\mathbf{x}) + \frac{1}{2}\langle \mathbf{x}, \mathbf{x}\rangle, \tag{2.1}$$

and the corresponding memory retrieval dynamics:

$$\mathcal{T}_{\text{Dense}}(\mathbf{x}) = \boldsymbol{\Xi} \cdot \operatorname{Softmax}(\beta \boldsymbol{\Xi}^\mathsf{T}\mathbf{x}) = \mathbf{x}^{\text{new}}.$$

The function $\operatorname{lse}(\beta, \mathbf{z}) \coloneqq \log\left(\sum_{\mu=1}^{M} \exp\{\beta z_\mu\}\right)/\beta$ is the log-sum-exponential for any given vector $\mathbf{z} \in \mathbb{R}^M$ and $\beta > 0$. Surprisingly, their findings reveal:

- The $\mathcal{T}_{\text{Dense}}$ dynamics converge to memories provably and retrieve patterns accurately in just one step.
- The modern Hopfield model from (2.1) possesses an exponential memory capacity in pattern size d.
- Notably, the one-step approximation of $\mathcal{T}_{\text{Dense}}$ mirrors the attention mechanism in transformers, leading to a novel architecture design: the Hopfield layers.

2.2. Generalized Sparse Modern Hopfield Model

This section follows (Wu et al., 2024b, Section 3). For self-containedness, we also summarize the useful theoretical results of (Wu et al., 2024b) in Appendix C.

Associative Memory Model. Let $\mathbf{z}, \mathbf{p} \in \mathbb{R}^M$, and $\Delta^M \coloneqq \{\mathbf{p} \in \mathbb{R}_+^M \mid \sum_\mu^M p_\mu = 1\}$ be the $(M-1)$-dimensional unit simplex. Wu et al. (2024b) introduce the generalized sparse Hopfield energy as a new associative memory model

$$E(\mathbf{x}) = -\Psi^\star\left(\beta \boldsymbol{\Xi}^\mathsf{T}\mathbf{x}\right) + \frac{1}{2}\langle \mathbf{x}, \mathbf{x}\rangle, \tag{2.2}$$

where $\Psi^\star(\mathbf{z}) \coloneqq \int \mathrm{d}\mathbf{z}\, \alpha\text{-EntMax}(\mathbf{z})$, and $\alpha\text{-EntMax}(\cdot)$ is defined as follows.

Definition 2.1 ((Peters et al., 2019))**.** The variational form of α-EntMax is defined as

$$\alpha\text{-EntMax}(\mathbf{z}) \coloneqq \underset{\mathbf{p}\in\Delta^M}{\operatorname{ArgMax}}[\mathbf{p}^\mathsf{T}\mathbf{z} - \Psi^\alpha(\mathbf{p})], \tag{2.3}$$

where $\Psi^\alpha(\cdot)$ is the Tsallis entropic regularizer

$$\Psi^\alpha(\mathbf{p}) \coloneqq \begin{cases} \frac{1}{\alpha(\alpha-1)}\sum_{\mu=1}^{M}\left(p_\mu - p_\mu^\alpha\right), & \alpha \neq 1, \\ \sum_{\mu=1}^{M} p_\mu \ln p_\mu, & \alpha = 1, \end{cases} \quad \text{for } \alpha \geq 1.$$

The corresponding memory retrieval dynamics is given as

Lemma 2.1 (Retrieval Dynamics, Lemma 3.2 of (Wu et al., 2024b))**.** Given t as the iteration number, the generalized sparse modern Hopfield model exhibits a retrieval dynamic

$$\mathcal{T}(\mathbf{x}_t) = \boldsymbol{\Xi}\, \alpha\text{-EntMax}(\beta \boldsymbol{\Xi}^\mathsf{T}\mathbf{x}_t) = \mathbf{x}_{t+1}, \tag{2.4}$$

which ensures a monotonic decrease of the energy (2.2).

This model also enjoys nice memory retrieval properties:

Lemma 2.2 (Convergence of Retrieval Dynamics $\mathcal{T}$, Lemma 3.3 of (Wu et al., 2024b))**.** Given the energy function E and retrieval dynamics $\mathcal{T}$ defined in (2.2) and (2.3), respectively. For any sequence $\{\mathbf{x}_t\}_{t=0}^\infty$ generated by the iteration $\mathbf{x}_{t'+1} = \mathcal{T}(\mathbf{x}_{t'})$, all limit points of this sequence are stationary points of E.

Lemma 2.2 ensures the (asymptotically) exact memory retrieval of this model ((2.2) and (2.4)), Thus, it serves as a well-defined associative memory model.

In essence, Wu et al. (2024b) present this sparse extension of the modern Hopfield model through a construction of both E and $\mathcal{T}$ by convex conjugating the Tsallis entropic regularizers. This model not only adheres to the conditions for a well-defined modern Hopfield model, but also equips greater robustness (Corollary C.1.2) and retrieval speed (Theorem C.1 and Corollary C.1.1) than the modern Hopfield model (Ramsauer et al., 2021), see Appendix C.2 for details. In Figure 4, we also provide proof-of-concept experimental validations on tabular datasets for Theorem C.1, Corollary C.1.1 and Corollary C.1.2.

Generalized Sparse Modern Hopfield (GSH) Layers for Deep Learning. Importantly, the generalized sparse modern Hopfield model serves as a valuable component in deep learning due to its connection to the transformer attention akin to its cousins. Next, we review such connections and the Generalized Sparse Modern Hopfield (GSH) layers.

Following (Wu et al., 2024b; Hu et al., 2023; Ramsauer et al., 2021), $\mathbf{X}$ and $\boldsymbol{\Xi}$ are defined in the associative space, embedded from the raw query $\mathbf{R}$ and memory patterns $\mathbf{Y}$, respectively, using $\mathbf{X}^\top = \mathbf{R}\mathbf{W}_Q \coloneqq \mathbf{Q}$ and $\boldsymbol{\Xi}^\top = \mathbf{Y}\mathbf{W}_K \coloneqq \mathbf{K}$ with matrices $\mathbf{W}_Q$ and $\mathbf{W}_K$. By transposing $\mathcal{T}$ from (2.4) and applying $\mathbf{W}_V$ such that $\mathbf{V} \coloneqq \mathbf{K}\mathbf{W}_V$, we obtain:

$$\mathbf{Z} \coloneqq \mathbf{Q}^{\text{new}}\mathbf{W}_V = \alpha\text{-EntMax}(\beta \mathbf{Q}\mathbf{K}^\top)\mathbf{V}, \tag{2.5}$$

introducing an attention mechanism with the α-EntMax activation function. Substituting $\mathbf{R}$ and $\mathbf{Y}$ back in, the Generalized Sparse Modern Hopfield (`GSH`) layer is formulated as:

$$\texttt{GSH}(\mathbf{R}, \mathbf{Y}) = \alpha\text{-EntMax}(\beta \mathbf{R}\mathbf{W}_Q\mathbf{W}_K^\top\mathbf{Y}^\top)\mathbf{Y}\mathbf{W}_K\mathbf{W}_V.$$

This allows the seamless integration of the generalized sparse modern Hopfield model into deep learning architectures. Concretely, the `GSH` layer takes matrices $\mathbf{R}$, $\mathbf{Y}$ as inputs, with the weight matrices $\mathbf{W}_Q$, $\mathbf{W}_K$, $\mathbf{W}_V$. Depending on its configuration, it offers several functionalities:

1. **Memory Retrieval:** In this learning-free setting, weight matrices $\mathbf{W}_K$, $\mathbf{W}_Q$, and $\mathbf{W}_V$ are set as identity matrices. Here, $\mathbf{R}$ represents the query input, and $\mathbf{Y}$ denotes the stored memory patterns for retrieval.
2. `GSH`**:** This configuration takes $\mathbf{R}$ and $\mathbf{Y}$ as inputs. Intending to substitute the attention mechanism, the weight matrices $\mathbf{W}_K$, $\mathbf{W}_Q$, and $\mathbf{W}_V$ are rendered learnable. Furthermore, $\mathbf{R}$, $\mathbf{Y}$, and $\mathbf{Y}$ serve as the sources for query, key, and value respectively. To achieve a self-

attention-like mechanism, $\mathbf{R}$ is set equal to $\mathbf{Y}$.

3. `GSHPooling`**:** With inputs $\mathbf{Q}$ and $\mathbf{Y}$, this layer uses $\mathbf{Q}$ as a static **prototype pattern**, while $\mathbf{Y}$ contains patterns over which pooling is desired. Given that the query pattern is replaced by the static prototype pattern $\mathbf{Q}$, the only learnable weight matrices are $\mathbf{W}_K$ and $\mathbf{W}_V$.
4. `GSHLayer`**:** The `GSHLayer` takes the query $\mathbf{R}$ as its single input. It includes learnable weight matrices $\mathbf{W}_K$ and $\mathbf{W}_V$, which function as our stored patterns and their corresponding projections. This design ensures that our key and value are decoupled from the input. In practice, we set $\mathbf{W}_Q$ and $\mathbf{Y}$ as identity matrices.

In this work, we utilize `GSH` and `GSHPooling` layers[1] .

3. Methodology

As in Figure 1, BiSHop uses three distinct parts to integrate two pivotal inductive biases in tabular data: non-rotationally invariant data structures (C1) and sparse information in features (C2) (Grinsztajn et al., 2022, Section 5.3 & 5.4):

- A joint **Tabular Embedding** layer is designed to process categorical and numerical data separately.
- The **Bi-Directional Sparse Hopfield Module (BiSHopModule)** leverages the generalized sparse modern Hopfield model. This module incorporates the non-rotationally invariant bias through two interconnected `GSH` blocks for row-wise and column-wise learning.
- **Stacked BiSHopModules** for hierarchical learning, addressing sparse features. Each layer in the stack module captures information at different scales, allowing for scale-specific sparsity.

We provide a detailed breakdown of each part as follows.

3.1. Tabular Embedding

Tabular embedding consists of three parts: **categorical embedding** $\mathbf{E}^{\text{cat}}$, **numerical embedding** $\mathbf{E}^{\text{num}}$, and **patch embedding** $\mathbf{E}^{\text{patch}}$. The categorical embedding not only learns the representations within individual categorical features but also capture the inter-relation among all categorical features. The numerical embedding represents each numerical feature with a one-hot-like representation and thus benefits neural network learning numerical features. The patch embedding captures localized feature information by aggregating across feature dimensions, at the same time reducing computation overhead. Starting from this section, we denote $\mathbf{x} \in \mathbb{R}^N$ any given tabular data point with N features. We suppose each $\mathbf{x}$ has N^{num} numerical feature $\mathbf{x}^{\text{num}}$ and N^{cat} categorical feature $\mathbf{x}^{\text{cat}}$, where $\mathbf{x} = (\mathbf{x}^{\text{num}}, \mathbf{x}^{\text{cat}})$. The categorical embedding $\mathbf{E}^{\text{cat}}$ and numerical embedding $\mathbf{E}^{\text{num}}$ transforms $\mathbf{x}^{\text{cat}}$ and $\mathbf{x}^{\text{num}}$ to a embedding dimension G, separately. The patch embedding $\mathbf{E}^{\text{patch}}$ then reduces G to the patch embedding dimension P.

Categorical Embedding. For categorical embedding $\mathbf{E}^{\text{cat}}$, we use learnable column embedding proposed by Huang et al. (2020). For a tabular data point $\mathbf{x} = (\mathbf{x}^{\text{num}}, \mathbf{x}^{\text{cat}})$, the column embedding only acts on the categorical features $\mathbf{x}^{\text{cat}}$, denoted as $\mathbf{E}^{\text{cat}}(\mathbf{x}^{\text{cat}})$. It comprises a shared embedding $\mathbf{E}^{\text{shared}}(\mathbf{x}^{\text{cat}})$ for all categorical features and N^{cat} individual embeddings for each categorical feature $\{x_i^{\text{cat}}\}_{i\in[N^{\text{cat}}]}$, where $[N^{\text{cat}}] = \{1, \cdots, N^{\text{cat}}\}$. We denote the shared embedding dimension as G^{shared} and the individual embedding dimension as G^{ind}, where $G = G^{\text{shared}} + G^{\text{ind}}$. The shared embedding $\mathbf{E}^{\text{shared}}(\mathbf{x}^{\text{cat}}) \in \mathbb{R}^{N^{\text{cat}}\times G^{\text{shared}}}$ represents each categorical feature differently. The individual embedding $\mathbf{E}^{\text{ind}} = \{\mathbf{E}_1^{\text{ind}}, \cdots, \mathbf{E}_{N^{\text{cat}}}^{\text{ind}}\}$ represents each category within one categorical feature differently. Each individual embedding $\mathbf{E}_i^{\text{ind}}(\cdot) \in \mathbb{R}^{1\times G^{\text{ind}}}$ is a scalar-to-vector map acting on each categorical feature $\{x_i^{\text{cat}}\}_{i\in[N^{\text{cat}}]}$. To obtain the final categorical embedding, we first concatenate all individual embeddings row-wise: $\mathbf{E}^{\text{ind}}(\mathbf{x}^{\text{cat}}) := \texttt{Concat}([\mathbf{E}_1^{\text{ind}}(x_1^{\text{cat}}), \ldots, \mathbf{E}_{N^{\text{cat}}}^{\text{ind}}(x_{N^{\text{cat}}}^{\text{cat}})], \texttt{axis} = 0) \in \mathbb{R}^{N^{\text{cat}}\times G^{\text{ind}}}$. Then, we concatenate the shared embedding with all individual embeddings column-wise: $\mathbf{E}^{\text{cat}}(\mathbf{x}^{\text{cat}}) := \texttt{Concat}([\mathbf{E}^{\text{shared}}(\mathbf{x}^{\text{cat}}), \mathbf{E}^{\text{ind}}(\mathbf{x}^{\text{cat}})], \texttt{axis} = 1) \in \mathbb{R}^{N^{\text{cat}}\times G}$. $\mathbf{E}^{\text{ind}}$ represents the unique category in each feature and $\mathbf{E}^{\text{shared}}$ represents the unique feature. $\mathbf{E}^{\text{cat}}$ enables our model to capture both the relationship between each feature and each category, with the flexibility to train shared and individual components separately.

Numerical Embedding. We employ the numerical embedding method as described in Gorishniy et al. (2021; 2022). The numerical embedding $\mathbf{E}^{\text{num}}$ only acts on the numerical features $\mathbf{x}^{\text{num}}$, as in $\mathbf{E}^{\text{num}}(\mathbf{x}^{\text{num}}) \in \mathbb{R}^{N^{\text{num}}\times G}$. Given a numerical feature $\{x_i^{\text{num}}\}_{i\in[N^{\text{num}}]}$, the embedding process begins by determining G quantiles. To start, we determine G quantiles for each numerical feature. Quantiles represent each numerical data distribution by dividing it into equal parts. For a numerical feature $\{x_j^{\text{num}}\}_{j\in N^{\text{num}}}$, we first sort all its values in the training data, $\mathbf{x}_j^{\text{num}}$, in ascending order. Then, we split the sorted data into G equal parts, where each part contains an equal fraction of the total data points. We define the boundaries of these parts as $b_{j,0}, \cdots, b_{j,G}$, where $b_{j,0}$ is the smallest value in $\mathbf{x}_j^{\text{num}}$. We express the embedding for a specific value x_j as a G-dimensional vector, $\mathbf{E}_j^{\text{num}}(x_j) = (e_{j,1}, \cdots, e_{j,G}) \in \mathbb{R}^G$. We compute the value of each $e_{j,g}$, where $1 \le g \le G$ according to the following function:

$$e_{j,g} := \begin{cases} 0, & \text{if } x_j < b_{j,g-1} \text{ and } g > 1, \\ 1, & \text{if } x_j \ge b_{j,g} \text{ and } g < G, \\ \frac{x_j - b_{j,g-1}}{b_{j,g} - b_{j,g-1}}, & \text{otherwise.} \end{cases}$$

[1] https://github.com/MAGICS-LAB/STanHop

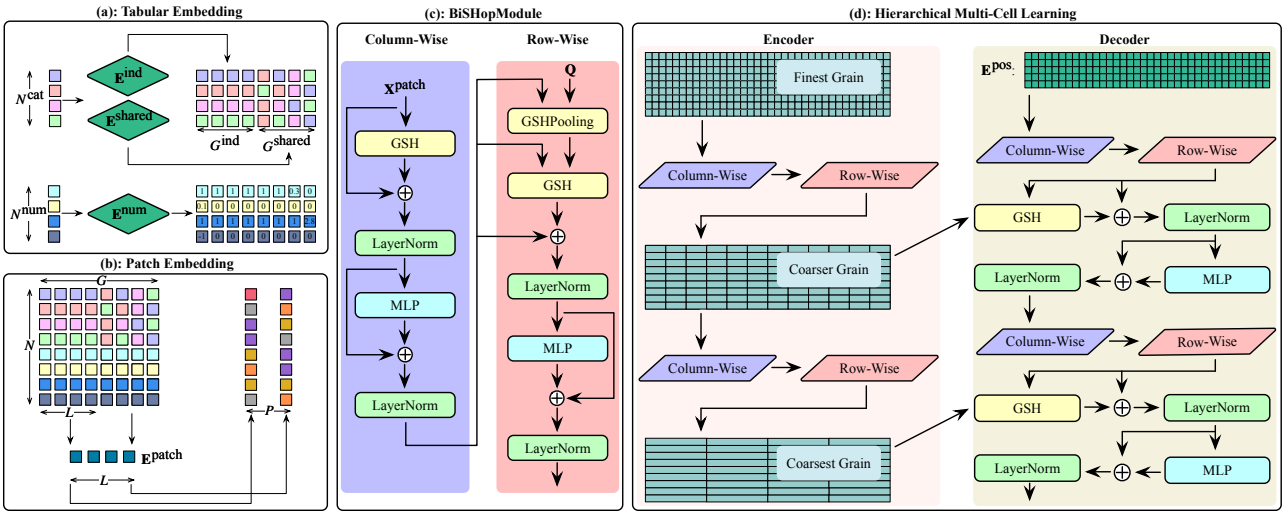

Figure 2. **BiSHop. (a) Tabular Embedding:** For a given input feature $\mathbf{x} = (\mathbf{x}^{\text{cat}}, \mathbf{x}^{\text{num}}) \in \mathbb{R}^{N=N^{\text{cat}}+N^{\text{num}}}$, the tabular embedding produces embeddings denoted as $\mathbf{E}^{\text{emb}}(\mathbf{x}) \in \mathbb{R}^{N\times G}$. **(b) Patch Embedding:** Using the combined numerical and categorical embeddings $\mathbf{E}^{\text{emb}}(\mathbf{x}) \in \mathbb{R}^{N\times G}$, the patch embedding gathers embedding information, subsequently reducing dimensionality from G to $P = \lceil G/L \rceil$ for all N features using a stride length of L. **(c) BiSHopModule:** The Bi-Directional Sparse Hopfield Module (BiSHopModule) leverages the generalized sparse modern Hopfield model. It integrates the tabular structure's inductive bias (C1) by deploying interconnected row-wise and column-wise `GSH` layers. **(d) Hierarchical Cellular Learning Module:** Employing a stacked encoder-decoder structure, we facilitate hierarchical cellular learning where both the encoder and decoder consist of the BiSHopModule across H layers. This arrangement enables BiSHop to derive refined representations from both directions across multiple scales. These representations are then concatenated for downstream inference, ensuring a holistic bi-directional cellular learning specially tailored for tabular data.

For the final embedding, we have $\mathbf{E}^{\text{num}}(\mathbf{x}^{\text{num}}) \in \mathbb{R}^{N^{\text{num}}\times G}$. We denote this numerical embedding as piece-wise linear embedding. This technique normalizes the scale of numerical features and captures the quantile information for each data point within the numerical feature. It enhances the representation of numerical feature in deep learning. Concatenating $\mathbf{E}^{\text{num}}(\mathbf{x}^{\text{num}})$ with $\mathbf{E}^{\text{cat}}(\mathbf{x}^{\text{cat}})$ row-wise, we obtain: $\mathbf{E}^{\text{emb}}(\mathbf{x}) := \texttt{Concat}\,(\mathbf{E}^{\text{num}}(\mathbf{x}^{\text{num}}), \mathbf{E}^{\text{cat}}(\mathbf{x}^{\text{cat}}), \texttt{axis} = 0)$, where $\mathbf{E}^{\text{emb}}(\mathbf{x}) \in \mathbb{R}^{N\times G}$. Namely, we call each point $\mathbf{E}^{\text{emb}}_{n,g}(\mathbf{x})$ as a single cell. The categorical and numerical embedding is in Figure 2 (a).

Patch Embedding. Motivated by (Nie et al., 2023; Zhang and Yan, 2023; Qin et al., 2022), we adopt patch embedding (shown in Figure 2 (b)) to enhance the awareness of both local and non-local patterns, capturing intricate details often missed at the single-cell level. Specifically, we divide embeddings into patches that aggregate multiple cells. To simplify the computation process, we transpose the numerical and categorical embedding dimensions. For convenience, we denote the previous embedding outcomes as $\mathbf{X}^{\text{emb}} := (\mathbf{E}^{\text{emb}}(\mathbf{x}))^{\top} \in \mathbb{R}^{G\times N}$. The patch embedding $\mathbf{E}^{\text{patch}}$ reduces the embedding dimension G by a stride factor L, leading to a new and smaller patched embedding dimension $P := \lceil G/L \rceil$, where $\lceil\cdot\rceil$ is the ceiling function. Furthermore, we introduce a new embedding dimension D^{model} to represent each patch's hidden states. The patched embedding is $\mathbf{E}^{\text{patch}}(\mathbf{X}^{\text{emb}}) \in \mathbb{R}^{P\times N\times D^{\text{model}}}$. For future computation, we flip the patch dimension and feature dimension, resulting final output of patch embedding $\mathbf{X}^{\text{patch}} := (\mathbf{E}^{\text{patch}}(\mathbf{X}^{\text{emb}}))^{\top} \in \mathbb{R}^{N\times P\times D^{\text{model}}}$. This patch embedding method enhances our model's ability to interpret and integrate detailed local and broader contextual information from the data, crucial for in-depth analysis in deep learning scenarios. For the $\mathbf{X}^{\text{patch}}$, we denote it as having N rows (features) and P columns (embeddings).

3.2. Bi-Directional Sparse Hopfield Module

By drawing parallels with the intricate interplay of different parts in the brain (Presigny and Fallani, 2022), we present the core design of the BiSHop framework, the Bi-Directional Sparse Hopfield Module (BiSHopModule), as visualized in Figure 2 (c). The BiSHopModule incorporates the generalized sparse modern Hopfield model and integrates the inductive bias of tabular structure (C1) through a unique structure of stacked row-wise and column-wise `GSH` blocks. Specifically, the row-wise `GSH` focuses on capturing the embedding details for individual features, whereas the column-wise `GSH` aggregates information across all features. We denote $\mathbf{X}^{\text{patch}}_{n,p}$, where $n \in [N]$ and $p \in [P]$, as the element in the n-th row (feature) and p-th column (embedding).

Column-Wise Block. The column-wise `GSH` block (purple block on the LHS of Figure 2 (c)) captures hidden information across the embedding dimension P for each feature. The process begins by passing the patch embeddings of the n-th row of $\mathbf{X}^{\text{patch}}$, $\mathbf{X}^{\text{patch}}_{n,:}$, where $n \in [N]$, to the `GSH` layer for self-attention. This process is then followed by adding the original patch embeddings (similar to the residual con-

nection in the standard transformer). Next, we pass the output above through one `LayerNorm` layer, a Multi-Layer Perception (MLP) layer, and another `LayerNorm` to obtain the final output of the column-wise block, $\mathbf{X}^{\text{col}}$:

$$\widehat{\mathbf{X}}^{\text{patch}}_{n,:} := \texttt{LayerNorm}\left(\mathbf{X}^{\text{patch}}_{n,:} + \texttt{GSH}(\mathbf{X}^{\text{patch}}_{n,:}, \mathbf{X}^{\text{patch}}_{n,:})\right),$$
$$\mathbf{X}^{\text{col}} := \texttt{LayerNorm}\left(\widehat{\mathbf{X}}^{\text{patch}} + \texttt{MLP}(\widehat{\mathbf{X}}^{\text{patch}})\right),$$

This sequence of operations ensures the effective transformation of the embeddings, facilitating the extraction of meaningful information from the feature space (Shi et al., 2021; 2024).

Row-Wise Block. The row-wise `GSH` block (pink block on the RHS of 2 (c)) captures information across the feature dimension N. For each feature, we apply both `GSHPooling` and `GSH` layers to its embedding dimensions. Specifically, we use C learnable pooling vectors in each feature dimension to aggregate information across all embedding dimensions, forming a pooling matrix $\mathbf{Q} \in \mathbb{R}^{C\times P\times D^{\text{model}}}$. We represent the pooling at p-th embedded dimension as the p-the columns of $\mathbf{Q}$, $\mathbf{Q}_{:,p}$, where $p \in [P]$. The process begins by pooling the row-wise output $\mathbf{X}^{\text{row}}_{:,p}$ using $\mathbf{Q}_{:,p}$ in the `GSHPooling` step. Next, we combine this pooled output with the row-wise output again, and add the row-wise output to the result. Following this, we pass the output through a `LayerNorm` layer, an MLP layer, and another `LayerNorm` layer. This sequence of operations yields the final output of the row-wise block:

$$\widehat{\mathbf{Q}}_{:,p} := \texttt{GSHPooling}(\mathbf{Q}_{:,p}, \mathbf{X}^{\text{col}}_{:,p}),$$
$$\widehat{\mathbf{X}}^{\text{row}}_{:,p} := \texttt{GSH}(\mathbf{X}^{\text{col}}_{:,p}, \widehat{\mathbf{Q}}_{:,p}),$$
$$\overline{\mathbf{X}}^{\text{row}} := \texttt{LayerNorm}(\widehat{\mathbf{X}}^{\text{row}} + \mathbf{X}^{\text{col}}),$$
$$\mathbf{X}^{\text{row}} := \texttt{LayerNorm}(\overline{\mathbf{X}}^{\text{row}} + \texttt{MLP}(\overline{\mathbf{X}}^{\text{row}})),$$

This $\mathbf{Q}$ pooling matrix design aggregates information from all patch embedding dimensions, and by setting $C \ll N$, it significantly reduces computational complexity.

Together with the row-wise block, we summarize the entire BiSHopModule as a function:

$$\texttt{BiSHopModule}(\cdot)\colon \mathbb{R}^{P\times N} \to \mathbb{R}^{P\times N},$$

where the input is $\mathbf{X}^{\text{patch}}$ and the output is $\mathbf{X}^{\text{row}}$.

3.3. Stacked BiSHopModules for Multi-Scale Learning with Scale-Specific Sparsity

Motivated by the human brain's multi-level organization of associative memory (Presigny and Fallani, 2022; Krotov, 2021), we utilize a hierarchical structure to learn multi-scale information similar to (Zhang and Yan, 2023; Zhou et al., 2021). This is illustrated in Figure 2 (d). This structure consists of two main components: the encoder and the decoder, both of which incorporate H layers of BiSHopModules. Specifically, the encoder captures coarser-grained information across different scales, while the decoder makes forecasts based on the information encoded by the encoder.

Encoder. The encoder (pink block on LHS of Figure 2 (d)) encodes data at multiple levels of granularity. To accomplish this multi-level encoding, we use H stacked BiSHopModules. These modules help in processing and understanding the data from different perspectives. We also employ a learnable merging matrix (Liu et al., 2021) to aggregate r adjacent patches of $\mathbf{X}^{\text{patch}}$. We denote the merging matrix at layer $h \in [H]$ as $\mathbf{E}^{\text{merge}}_h \in \mathbb{R}^{r\times 1}$, which refines its input embeddings to be coarser at each level. We refer to h-th level encoder output as $\mathbf{X}^{\text{enc},h}$ and input as $\mathbf{X}^{\text{enc},h-1}$. Concretely, at level h, we use $\mathbf{E}^{\text{merge}}_h$ to aggregate r adjacent embedding vectors from $\mathbf{X}^{\text{enc},h-1}$, producing a coarser embedding $\widehat{\mathbf{X}}^{\text{enc},h-1}$. We then pass $\widehat{\mathbf{X}}^{\text{enc},h-1}$ through the BiSHopModules, resulting in the output encoded embedding, denoted as $\mathbf{X}^{\text{enc},h}$. It is worth noting that $\mathbf{X}^{\text{enc},0} = \mathbf{X}^{\text{patch}}$. This granularity-decreasing process is iteratively applied across all layers in $1 \le h \le H$. We summarize the merging procedure at level h as:

$$\widehat{\mathbf{X}}^{\text{enc},h}_{n,p} := \mathbf{E}^{\text{merge}}_h \left(\mathbf{X}^{\text{enc},h}_{n,r\times p}, \ldots, \mathbf{X}^{\text{enc},h}_{n,r\times(p+1)}\right), 0 \le p \le \frac{P}{r^h},$$

for $0 \le h \le H-1$, and then

$$\mathbf{X}^{\text{enc},h} := \texttt{BiSHopModule}(\widehat{\mathbf{X}}^{\text{enc},h-1}), \quad \text{for } 1 \le h \le H.$$

Decoder. The decoder (yellow block on RHS of Figure 2 (d)) captures information from each level of encoded data. To accomplish this, we utilize H stacked BiSHopModules and employ a positional embedding matrix $\mathbf{E}^{\text{pos}} \in \mathbb{R}^{P\times S}$ to extract encoded information for prediction, where S represents the number of extracted feature used for future forecast. Specifically, at the first level, we use the learnable matrix $\mathbf{E}^{\text{pos}}$ to decode S different representations through BiSHopModules, obtaining $\mathbf{X}^{\text{pos},0}$. We then pass $\mathbf{X}^{\text{pos},0}$ through `GSH` with the corresponding encoded data, followed by the addition to the encoded data at the h-th level $\mathbf{X}^{\text{enc},h}$. Next, we process the output through one `LayerNorm` layer, one `MLP` layer, and another `LayerNorm` layer, as follows:

$$\mathbf{X}^{\text{pos},h} := \begin{cases} \texttt{BiSHopModule}(\mathbf{E}^{\text{pos}}), & h = 0, \\ \texttt{BiSHopModule}(\mathbf{X}^{\text{dec},h-1}), & 1 \le h \le H. \end{cases}$$
$$\widehat{\mathbf{X}}^{\text{dec},h} := \texttt{GSH}(\mathbf{X}^{\text{pos},h}, \mathbf{X}^{\text{enc, h}}), 1 \le h \le H,$$
$$\overline{\mathbf{X}}^{\text{dec},h} := \texttt{LayerNorm}(\widehat{\mathbf{X}}^{\text{dec},h} + \mathbf{X}^{\text{pos},h}),$$
$$\mathbf{X}^{\text{dec},h} := \texttt{LayerNorm}(\overline{\mathbf{X}}^{\text{dec},h} + \texttt{MLP}(\overline{\mathbf{X}}^{\text{dec},h})).$$

For the final prediction, we flatten $\mathbf{X}^{\text{dec},H}$ and pass it to a new MLP predictor.

Learnable Sparsity at Each Scale. Drawing inspiration from the dynamic sparsity observed in the human brain (Stokes et al., 2013; Leutgeb et al., 2005; Willshaw et al.,

1969), the parameter α for each GSH layer is a learnable parameter by design (Wu et al., 2024b; Correia et al., 2019), which allows BiSHopModule to adapt to different sparsity for different resolutions. Namely, the learned representations at each scale are equipped with scale-specific sparsity.

4. Experimental Studies

In this section, we compare BiSHop with SOTA tabular learning methods, following the tabular learning benchmark paper (Grinsztajn et al., 2022). We summarize our experimental results in Table 1 and Table 2.

4.1. Experimental Setting

Our experiment consists of two parts: firstly, we benchmark commonly used datasets in the literature; secondly, we follow the tabular benchmark (Grinsztajn et al., 2022), applying it to a broader range of datasets on both classification and regression tasks.

Datasets I. In the first experimental setting, we evaluate BiSHop on 9 common classification datasets used in previous works (Grinsztajn et al., 2022; Somepalli et al., 2022; Gorishniy et al., 2021; Huang et al., 2020). These datasets vary in characteristics: some are well-balanced, and others show highly skewed class distributions; We set the train/validation/test proportion of each dataset as 70/10/20%. Please see Appendix D.1 for datasets' details.

Datasets II. In the second experimental setting, we test BiSHop in the tabular benchmark (Grinsztajn et al., 2022). The datasets compiled by this benchmark consist of 4 OpenML suites:

- Categorical Classification (**CC**, suite_id: 334),
- Numerical Classification (**NC**, suite_id: 337),
- Categorical Regression (**CR**, suite_id: 335),
- Numerical Regression (**NR**, suite_id: 336).

Both **CC** and **CR** include datasets with numerical and categorical features, whereas **NC** and **NR** only contain numerical features. Due to limited computational resources, we randomly select one-third of the datasets from each suite for evaluation. We evaluate BiSHop on each suit with 3-6 different datasets and truncate to 10,000 training samples for larger datasets (corresponding to medium-size regimes in the benchmark). For these datasets, we allocate 70% of the data for the training set (7,000 samples). Of the remaining 30%, we allocate 30% for the validation set (900 samples), and the rest 70% for the test set (2,100 samples). All samples are randomly chosen from the original datasets and undergo the identical preprocessing steps of the previous benchmark (Grinsztajn et al., 2022).

Metrics. We use the AUC score for the 1st experimental setting, aligned with literature. For the 2nd experimental setting, we use accuracy for classification and the R^2 score for regression tasks, aligned with Grinsztajn et al. (2022).

Baselines I. In the first experimental setting, we select 5 deep learning and 3 tree-based baselines, including (i) DL-based method such as MLP, TabNet, TabTransformer, FT-Transformer (Gorishniy et al., 2021), SAINT (Somepalli et al., 2022), TabPNF (Hollmann et al., 2022), TANGOS (Jeffares et al., 2023), T2G-FORMER (Yan et al., 2023), and (ii) tree-based methods such as LightGBM, CatBoost, and XGBoost (Chen et al., 2015). For each dataset, we conduct up to 200 random searches on BiSHop to report the score of the best hyperparameter configuration. We stop HPOs when observing the best result. Baselines and benchmark datasets' results are quoted from competing papers when possible and reproduced otherwise. We report the reproduced results in Appendix D. Notably, we quote the best result from all baselines if multiple results are available.

Baselines II. In the second experimental setting, we reference baselines results[2] from the benchmark paper (Gorishniy et al., 2021), comprising 4 deep learning methods and 3 tree-based methods, including (i) DL-based method such as MLP, ResNet (He et al., 2016), FT-Transformer (Gorishniy et al., 2021), SAINT (Somepalli et al., 2022) and (ii) tree-based methods such as RandomForest, GradientBoostingTree (GBDT), and XGBoost (Chen et al., 2015). We select the best results of each method from the benchmark (Grinsztajn et al., 2022). Notably, these best results use 400 HPOs according to Grinsztajn et al. (2022).

Setup. BiSHop's default parameter settings are as follows: Embedding dimension $G = 32$; Stride factor $L = 8$; Number of pooling vectors $C = 10$; Number of BiSHopModules $H = 3$; Number of aggregations in the encoder $r = 4$; Number of representations decoded $S = 24$; Dropout = 0.2; Learning rate 5×10^{-5}. For numerical embedding, we only gather quantile information from the training data to process the embedding function. For hyperparameter tuning, we use the "sweep" feature of Weights and Biases (Biewald et al., 2020). Notably, due to computational constraints, we manually end the HPO once our method surpasses the best performance observed in the benchmarks. We report the search space for all hyperparameters in Table 7 and other training details in Appendix D.2. We conduct the optimization on training/validation sets and report the average test set scores over 3 iterations, using the best-performing configurations on the validation set. Implementation and training details are provided in the appendix.

Results. We summarize our results of the Baselines I in Table 1 and the results of the Baselines II in Table 2. In Table 1, BiSHop outperforms both tree-based and deep-learning-based methods by a significant margin in most

[2] https://github.com/LeoGrin/tabular-benchmark

Table 1. **BiSHop versus SOTA Tabular Learning Methods (Dataset I).** We evaluate BiSHop against predominant SOTA methods, including deep learning methods (MLP, TabNet, TabTransformer, FT-Transformer, SAINT, TabPFN, TANGOS, T2G-FORMER) and tree-based methods (LightGBM, CatBoost, XGBoost), across various datasets. We report the average AUC scores (in %) of 3 runs, with variances omitted as they are all $\leq 1.3\%$. Results quoted from (Liu et al., 2022; Somepalli et al., 2022; Borisov et al., 2022; Huang et al., 2020) are marked with *, *, †, and ‡, respectively. If multiple results are available across different benchmark papers, we quote the best one. When unavailable, we reproduce the baseline results independently. Hyperparameter optimization employs the "sweep" feature of Weights and Biases (Biewald et al., 2020), with 200 iterations of random search for each setting. Our results indicate that BiSHop outperforms both tree-based and deep-learning-based methods by a significant margin.

	Adult	Bank	Blastchar	Income	SeismicBump	Shrutime	Spambase	Qsar	Jannis
MLP	72.5‡	92.9‡	83.9‡	90.5‡	73.5‡	84.6‡	98.4‡	91.0‡	82.59
TabNet	90.49	91.76*	79.61*	90.72*	77.77	84.39	99.80	67.55*	87.81
TabTransformer	73.7‡	93.4‡	83.5‡	90.6‡	75.1‡	85.6‡	98.5‡	91.8‡	82.85
FT-Transformer	90.60	91.83	86.06	92.15	74.60	80.83	**100.00**	92.04	89.02
SAINT	91.6†	93.30*	84.67*	91.67*	76.6*	86.47*	98.54*	93.21*	85.52
TabPFN	88.48	88.17	84.03	88.59	75.32	83.30	100	93.31	78.34
TANGOS	90.23	88.98	85.74	90.44	73.52	84.32	100	90.83	83.59
T2G-FORMER	85.96$^\diamond$	**94.47**	85.40	92.35	82.58	86.42	100	94.86	73.68$^\diamond$
LightGBM	92.9†	93.39*	83.17*	92.57*	77.43	85.36*	**100.00**	92.97*	87.48
CatBoost	92.8†	90.47*	84.77*	90.80*	81.59	85.44*	**100.00**	93.05*	87.53
XGBoost	92.8†	92.96*	81.78*	92.31*	75.3*	83.59*	**100.00**	92.70*	86.72
BiSHop	**92.97**	93.95	**88.49**	**92.97**	**91.88**	**87.99**	**100.00**	**96.14**	**90.63**

datasets. In Table 2, BiSHop achieves optimal or near-optimal results with fewer than 10% numbers (on average) of HPO in a tabular benchmark (Grinsztajn et al., 2022).

4.2. Ablation Studies

We conduct the following sets of ablation studies on **Datasets I** align with Grinsztajn et al. (2022).

Changing Feature Sparsity. In Figure 3, we change feature sparsity on our datasets following Grinsztajn et al. (2022, Figure 4 & 5). Firstly, we compute the feature importance using Random Forest. Secondly, we remove features in both **increasing** (solid curves) and **decreasing** (dashed curves) order of feature importance. For each order, we report the average AUC score over all datasets at each percentage for BiSHop, XGBoost, and LighGBM. Our results indicate that BiSHop has the capacity to handle sparse features.

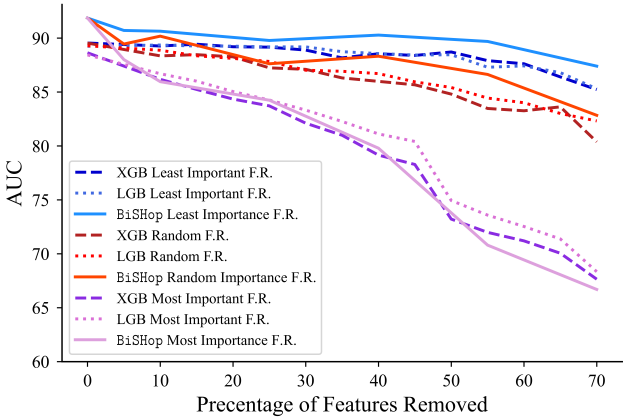

Figure 3. **Changing Feature Sparsity.** Following (Grinsztajn et al., 2022), we remove features in three ways: **randomly** (red), in **increasing** order of feature importance (purple), and in **decreasing** order of feature importance (blue), with feature importance determined by random forest. We report the average AUC score across all datasets for BiSHop, XGBoost, and LightGBM. The results highlight BiSHop's capability in handling sparse features.

Rotation Invariance. In Table 19, we conduct experiments by rotating the datasets and BiSHopModule's direction, both individual rotation and combined rotation:

(R1) Rotate the two Directions (Row- and Column-wise).

(R2) Rotate the Datasets.

(R3) Rotate the two Directions and the Datasets.

Our results indicate (i) BiSHop is robust against column-row switch in BiSHopModule, and (ii) BiSHop addresses the Non-Rotationally Invariant Data Structure challenge (C1).

We report the average AUC score over all datasets for each rotation scenario. First, we assess (R1), where the results show a marginal ($< 1\%$) performance drop across datasets. To further discuss the rotational invariance problem, we assess (R2) following the procedure outlined in Grinsztajn et al. (2022, Section 5.4). The results for (R2) do not indicate a significant drop in performance. Additionally, the results for (R3) validate the findings from both (R1) and (R2). Our results confirm that BiSHop addresses (C1).

Hierarchy of BiSHopModule. In Table 20, we assess the impacts of stacking different layers of BiSHopModule. We report the average AUC over **Datasets I** for different layers of BiSHopModule. Our results indicate that 4 layers of BiSHopModule marginally maximize the performance.

Component Analysis. In Table 17, we remove each com-

Table 2. **BiSHop versus SOTA Tabular Learning Methods (Dataset II).** Following the benchmark (Grinsztajn et al., 2022), we evaluate BiSHop against SOTA methods, including deep learning methods (MLP, ResNet, FT-Transformer, SAINT) and tree-based methods (GBDT, RandomForest, XGBoost), across various datasets. We randomly select a total of 19 datasets encompassing four different tasks: categorical classification (**CC**), numerical classification (**NC**), categorical regression (**CR**), and numerical regression (**NR**). **CC** and **CR** contain both categorical and numerical features, while **NC** and **NR** contain only numerical features. Baseline results are quoted from the benchmark paper (Grinsztajn et al., 2022). We report with the best Accuracy scores for **CC** and **NC**, and R^2 score for **CR** and **NR**, (both in %) obtained through HPO. We also report the number of HPOs used in BiSHop. Hyperparameter optimization of our method employs the "sweep" feature of Weights and Biases (Biewald et al., 2020). In the 19 different datasets, BiSHop delivers 11 optimal and 8 near-optimal results (within a 1.3% margin), using less than 10% (on average) of the number of HPOs used by the baselines.

	Dataset ID	BiSHop	# of HPOs	FT-Transformer	GBDT	MLP	RandomForest	ResNet	SAINT	XGBoost
CC	361282	**66.08**	16	65.63	65.76	65.32	65.53	65.23	65.52	65.70
	361283	**72.69**	1	71.90	72.09	71.41	72.13	71.4	71.9	72.08
	361286	**69.80**	10	68.97	68.62	69.06	68.49	69.00	68.87	68.20
CR	361093	**98.98**	23	98.06	98.34	98.07	98.25	98.04	97.77	98.42
	361094	99.98	64	99.99	**100**	99.99	**100**	99.97	99.98	**100**
	361099	94.12	64	94.09	94.26	93.71	93.69	93.71	93.75	**94.77**
	361104	99.94	70	99.97	**99.98**	**99.98**	**99.98**	99.96	99.9	**99.98**
	361288	57.96	93	57.48	55.75	58.03	55.79	**58.3**	57.09	55.75
NC	361055	**78.29**	4	77.73	77.52	77.41	76.35	77.53	77.41	75.91
	361062	**98.82**	15	98.50	98.16	94.70	98.24	95.22	98.21	98.35
	361065	**86.32**	2	86.09	85.79	85.6	86.55	86.3	86.04	86.19
	361273	**60.76**	9	60.57	60.53	60.50	60.49	60.54	60.59	60.67
	361278	**73.05**	2	72.67	72.35	72.4	72.1	72.41	72.37	72.16
NR	361073	**99.51**	8	**99.51**	99.0	97.31	98.67	96.19	**99.51**	99.15
	361074	87.96	34	91.83	85.07	91.81	83.3	91.56	**91.86**	90.76
	361077	82.4	53	73.28	**83.97**	83.72	83.72	71.85	70.1	83.66
	361079	**60.76**	19	53.09	57.45	48.62	50.16	51.77	46.79	55.42
	361081	98.67	13	99.69	99.65	99.52	99.31	99.67	99.38	**99.76**
	361280	56.98	96	57.48	54.87	**58.46**	55.27	57.81	56.84	55.49
Score	mean	**81.21**	-	80.34	80.48	80.3	79.84	79.81	79.68	80.65
Rank	mean	**2.79**	-	3.58	4.21	4.74	5.53	5.05	5.26	3.84
	min	**1**	-	1	1	1	1	1	1	1
	max	8	-	6	8	8	8	8	8	8
	med.	**1**	-	4	4	5	6	5	5	3

ponent one at a time. We report the implementation details in Appendix E.1. For each removal, we report averaged AUC scores over all datasets. Overall, each component contributes to varying degrees of performance improvement.

Comparison with the Dense Modern Hopfield Model. In Appendix E.2, we compare the performance of Sparse Hopfield Models, Dense Hopfield Models, and Attention Mechanisms. Our results that the generalized Sparse Hopfield Model outperforms the other two methods.

Convergence Analysis. In Appendix E.3, we compare the converging rate of Sparse and Dense Hopfield Models. Our results indicate that the generalized sparse modern Hopfield model converges faster than the Dense Model.

Appendix E includes all details of ablation experiments.

5. Conclusion

We address the gap highlighted by Grinsztajn et al. (2022) where deep learning methods trail behind tree-based methods. We present the Bi-Directional Sparse Hopfield Model (BiSHop) for deep tabular learning, inspired by the recent intersection of Hopfield models with attention mechanisms. Leveraging the generalized sparse Hopfield layers as its core component, BiSHop effectively handles the challenges of deep tabular learning, by incorporating two important inductive biases of tabular data (C1, C2).

Comparing with Existing Works. Empirically, our model consistently surpasses SOTA tree-based and deep learning methods by 3% across common benchmark datasets. Moreover, our model achieves optimal or near-optimal results with only 16% number of HPOs, compared with methods in the tabular benchmark (Grinsztajn et al., 2022). We deem these results as closing the performance gap between DL-based and tree-based tabular learning methods, making BiSHop a promising solution for deep tabular learning.

Limitation. One notable limitation of our study is the non-utilization of the external memory capabilities inherent in modern Hopfield models. We see the integration of these capabilities, especially in memory-augmented large models, as a compelling direction for future research.

Impact Statement

Our work aim at addressing the long standing problem of tabular learning of DL-based model. We do not expect any negative social impact of our work.

Acknowledgments

JH would like to thank Dino Feng and Andrew Chen for enlightening discussions, the Red Maple Family for support, and Jiayi Wang for facilitating experimental deployments. CX would like to thank Yibo Wen for helpful comments. The authors would also like to thank the anonymous reviewers and program chairs for their constructive comments.

JH is partially supported by the Walter P. Murphy Fellowship. HL is partially supported by NIH R01LM1372201, NSF CAREER1841569, DOE DE-AC02-07CH11359, DOE LAB 20-2261 and a NSF TRIPODS1740735. H.-S.G. acknowledges support from the National Science and Technology Council, Taiwan under Grants No. NSTC 113-2119-M-002 -021, No. NSTC112-2119-M-002-014, No. NSTC 111-2119-M-002-007, and No. NSTC 111-2627-M-002-001, from the US Air Force Office of Scientific Research under Award Number FA2386-20-1-4052, and from the National Taiwan University under Grants No. NTU-CC-112L893404 and No. NTU-CC-113L891604. H.-S.G. is also grateful for the support from the "Center for Advanced Computing and Imaging in Biomedicine (NTU-113L900702)" through The Featured Areas Research Center Program within the framework of the Higher Education Sprout Project by the Ministry of Education (MOE), Taiwan, and the support from the Physics Division, National Center for Theoretical Sciences, Taiwan. This research was supported in part through the computational resources and staff contributions provided for the Quest high performance computing facility at Northwestern University which is jointly supported by the Office of the Provost, the Office for Research, and Northwestern University Information Technology. The content is solely the responsibility of the authors and does not necessarily represent the official views of the funding agencies.

References

Ami Abutbul, Gal Elidan, Liran Katzir, and Ran El-Yaniv. Dnf-net: A neural architecture for tabular data. *arXiv preprint arXiv:2006.06465*, 2020. URL https://arxiv.org/abs/2006.06465.

Sercan Ö Arik and Tomas Pfister. TabNet: Attentive interpretable tabular learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 6679–6687, 2021. URL https://arxiv.org/abs/1908.07442.

Andreas Auer, Martin Gauch, Daniel Klotz, and Sepp Hochreiter. Conformal prediction for time series with modern hopfield networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 36, 2024. URL https://arxiv.org/abs/2303.12783.

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020. URL https://arxiv.org/abs/2004.05150.

Alberto Bietti, Vivien Cabannes, Diane Bouchacourt, Herve Jegou, and Leon Bottou. Birth of a transformer: A memory viewpoint. *Advances in Neural Information Processing Systems (NeurIPS)*, 36, 2023. URL https://arxiv.org/abs/2306.00802.

Lukas Biewald et al. Experiment tracking with weights and biases. *Software available from wandb. com*, 2:233, 2020.

Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

Johannes Brandstetter. Blog post: Hopfield networks is all you need. https://ml-jku.github.io/hopfield-layers/, 2021. Accessed: April 4, 2023.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems (NeurIPS)*, 33: 1877–1901, 2020. URL https://arxiv.org/abs/2005.14165.

Thomas F Burns. Semantically-correlated memories in a dense associative model. In *Forty-first International Conference on Machine Learning (ICML)*, 2024. URL https://arxiv.org/abs/2404.07123.

Thomas F Burns and Tomoki Fukai. Simplicial hopfield networks. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=_QLsH8gatwx.

Ljubomir Buturović and Dejan Miljković. A novel method for classification of tabular data using convolutional neural networks. *BioRxiv*, pages 2020–05, 2020.

Vivien Cabannes, Elvis Dohmatob, and Alberto Bietti. Scaling laws for associative memories. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024a. URL https://openreview.net/forum?id=Tzh6xAJSll.

Vivien Cabannes, Berfin Simsek, and Alberto Bietti. Learning associative memories with gradient descent. *arXiv preprint arXiv:2402.18724*, 2024b. URL https://arxiv.org/abs/2402.18724.

Hamza Chaudhry, Jacob Zavatone-Veth, Dmitry Krotov, and Cengiz Pehlevan. Long sequence hopfield memory. *Advances in Neural Information Processing Systems (NeurIPS)*, 36, 2023. URL https://arxiv.org/abs/2306.04532.

Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, Rory Mitchell, Ignacio Cano, Tianyi Zhou, et al. Xgboost: extreme gradient boosting. *R package version 0.4-2*, 1(4):1–4, 2015.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019. URL https://arxiv.org/abs/1904.10509.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022. URL https://arxiv.org/abs/2204.02311.

Gonçalo M Correia, Vlad Niculae, and André FT Martins. Adaptively sparse transformers. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2174–2184, 2019. URL https://arxiv.org/abs/1909.00015.

Mete Demircigil, Judith Heusel, Matthias Löwe, Sven Upgang, and Franck Vermet. On a model of associative memory with huge storage capacity. *Journal of Statistical Physics*, 168:288–299, 2017. URL https://arxiv.org/abs/1702.01929.

Andreas Fürst, Elisabeth Rumetshofer, Johannes Lehner, Viet T Tran, Fei Tang, Hubert Ramsauer, David Kreil, Michael Kopp, Günter Klambauer, Angela Bitto, et al. Cloob: Modern hopfield networks with infoloob outperform clip. *Advances in neural information processing systems (NeurIPS)*, 35:20450–20468, 2022. URL https://arxiv.org/abs/2110.11316.

Josh Gardner, Zoran Popovic, and Ludwig Schmidt. Benchmarking distribution shift in tabular data with tableshift. *Advances in Neural Information Processing Systems (NeurIPS)*, 36, 2024. URL https://arxiv.org/abs/2312.07577.

Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:18932–18943, 2021. URL https://arxiv.org/abs/2106.11959.

Yury Gorishniy, Ivan Rubachev, and Artem Babenko. On embeddings for numerical features in tabular deep learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:24991–25004, 2022. URL https://arxiv.org/abs/2203.05556.

Yury Gorishniy, Ivan Rubachev, Nikolay Kartashev, Daniil Shlenskii, Akim Kotelnikov, and Artem Babenko. TabR: Unlocking the power of retrieval-augmented tabular deep learning, 2023. URL https://arxiv.org/abs/2307.14338.

Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? *Advances in Neural Information Processing Systems (NeurIPS)*, 35:507–520, 2022. URL https://arxiv.org/abs/2207.08815.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90. URL https://arxiv.org/abs/1512.03385.

Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. TabPFN: A transformer that solves small tabular classification problems in a second. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2022. URL https://arxiv.org/abs/2207.01848.

Benjamin Hoover, Yuchen Liang, Bao Pham, Rameswar Panda, Hendrik Strobelt, Duen Horng Chau, Mohammed J Zaki, and Dmitry Krotov. Energy transformer. In *Thirty-seventh Conference on Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2302.07253.

John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

John J Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the national academy of sciences*, 81(10):3088–3092, 1984.

Jerry Yao-Chieh Hu, Donglin Yang, Dennis Wu, Chenwei Xu, Bo-Yu Chen, and Han Liu. On sparse modern hopfield model. In *Thirty-seventh Conference on Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2309.12673.

Jerry Yao-Chieh Hu, Pei-Hsuan Chang, Robin Luo, Hong-Yu Chen, Weijian Li, Wei-Po Wang, and Han Liu. Outlier-efficient hopfield layers for large transformer-based models. In *Forty-first International Conference on Machine*

Learning (ICML), 2024a. URL https://arxiv.org/abs/2404.03828.

Jerry Yao-Chieh Hu, Bo-Yu Chen, Dennis Wu, Feng Ruan, and Han Liu. Nonparametric modern hopfield models. *arXiv preprint arXiv:2404.03900*, 2024b. URL https://arxiv.org/abs/2404.03900.

Jerry Yao-Chieh Hu, Thomas Lin, Zhao Song, and Han Liu. On computational limits of modern hopfield models: A fine-grained complexity analysis. In *Forty-first International Conference on Machine Learning (ICML)*, 2024c. URL https://arxiv.org/abs/2402.04520.

Xin Huang, Ashish Khetan, Milan Cvitkovic, and Zohar Karnin. TabTransformer: Tabular data modeling using contextual embeddings. *arXiv preprint arXiv:2012.06678*, 2020. URL https://arxiv.org/abs/2012.06678.

Alan Jeffares, Tennison Liu, Jonathan Crabbé, Fergus Imrie, and Mihaela van der Schaar. TANGOS: Regularizing tabular neural networks through gradient orthogonalization and specialization. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023. URL https://arxiv.org/abs/2303.05506.

Yanrong Ji, Zhihan Zhou, Han Liu, and Ramana V Davuluri. Dnabert: pre-trained bidirectional encoder representations from transformers model for dna-language in genome. *Bioinformatics*, 37(15):2112–2120, 2021.

Arlind Kadra, Marius Lindauer, Frank Hutter, and Josif Grabocka. Well-tuned simple nets excel on tabular datasets. *Advances in neural information processing systems (NeurIPS)*, 34:23928–23941, 2021. URL https://arxiv.org/abs/2106.11189.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems (NeurIPS)*, 30, 2017.

Leo Kozachkov, Ksenia V Kastanenka, and Dmitry Krotov. Building transformers from neurons and astrocytes. *bioRxiv*, pages 2022–10, 2022.

Dmitry Krotov. Hierarchical associative memory. *arXiv preprint arXiv:2107.06446*, 2021. URL https://arxiv.org/abs/2107.06446.

Dmitry Krotov. A new frontier for hopfield networks. *Nature Reviews Physics*, 5(7):366–367, Jul 2023. ISSN 2522-5820. doi: 10.1038/s42254-023-00595-y. URL https://doi.org/10.1038/s42254-023-00595-y.

Dmitry Krotov and John J Hopfield. Dense associative memory for pattern recognition. *Advances in neural information processing systems (NeurIPS)*, 29, 2016. URL https://arxiv.org/abs/1606.01164.

Dmitry Krotov and John J. Hopfield. Large associative memory problem in neurobiology and machine learning. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://arxiv.org/abs/2008.06996.

Jill K Leutgeb, Stefan Leutgeb, Alessandro Treves, Retsina Meyer, Carol A Barnes, Bruce L McNaughton, May-Britt Moser, and Edvard I Moser. Progressive transformation of hippocampal neuronal representations in "morphed" environments. *Neuron*, 48(2):345–358, 2005.

Guang Liu, Jie Yang, and Ledell Wu. PTab: Using the pre-trained language model for modeling tabular data. *arXiv preprint arXiv:2209.08060*, 2022. URL https://arxiv.org/abs/2209.08060.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021. URL https://arxiv.org/abs/2103.14030.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, et al. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning (ICML)*, pages 22137–22176. PMLR, 2023. URL https://arxiv.org/abs/2310.17157.

Duncan McElfresh, Sujay Khandagale, Jonathan Valverde, Vishak Prasad C, Ganesh Ramakrishnan, Micah Goldblum, and Colin White. When do neural nets outperform boosted trees on tabular data? *Advances in Neural Information Processing Systems (NeurIPS)*, 36, 2024. URL https://arxiv.org/abs/2305.02997.

Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://arxiv.org/abs/2211.14730.

Inkit Padhi, Yair Schiff, Igor Melnyk, Mattia Rigotti, Youssef Mroueh, Pierre Dognin, Jerret Ross, Ravi Nair, and Erik Altman. Tabular transformers for modeling multivariate time series. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3565–3569. IEEE, 2021. URL https://arxiv.org/abs/2011.01843.

Fabian Paischer, Thomas Adler, Vihang Patil, Angela Bitto-Nemling, Markus Holzleitner, Sebastian Lehner, Hamid Eghbal-Zadeh, and Sepp Hochreiter. History compression via language models in reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 17156–17185. PMLR, 2022. URL https://arxiv.org/abs/2205.12258.

Ben Peters, Vlad Niculae, and André F. T. Martins. Sparse sequence-to-sequence models. In Anna Korhonen, David Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1504–1519, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1146. URL https://arxiv.org/abs/1905.05702.

Sergei Popov, Stanislav Morozov, and Artem Babenko. Neural oblivious decision ensembles for deep learning on tabular data. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://arxiv.org/abs/1909.06312.

Charley Presigny and Fabrizio De Vico Fallani. Colloquium: Multiscale modeling of brain network organization. *Reviews of Modern Physics*, 94(3):031002, 2022.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. CatBoost: unbiased boosting with categorical features. *Advances in neural information processing systems (NeurIPS)*, 31, 2018. URL https://arxiv.org/abs/1706.09516.

Lianke Qin, Aravind Reddy, Zhao Song, Zhaozhuo Xu, and Danyang Zhuo. Adaptive and dynamic multi-resolution hashing for pairwise summations. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 115–120. IEEE, 2022. URL https://arxiv.org/abs/2212.11408.

Jiezhong Qiu, Hao Ma, Omer Levy, Wen-tau Yih, Sinong Wang, and Jie Tang. Blockwise self-attention for long document understanding. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2555–2565, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.232. URL https://arxiv.org/abs/1911.02972.

Hubert Ramsauer, Bernhard Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Lukas Gruber, Markus Holzleitner, Thomas Adler, David Kreil, Michael K Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. Hopfield networks is all you need. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://arxiv.org/abs/2008.02217.

Philipp Seidl, Philipp Renz, Natalia Dyubankova, Paulo Neves, Jonas Verhoeven, Jorg K Wegner, Marwin Segler, Sepp Hochreiter, and Gunter Klambauer. Improving few- and zero-shot reaction template prediction using modern hopfield networks. *Journal of chemical information and modeling*, 62(9):2111–2120, 2022.

Zhenmei Shi, Junyi Wei, and Yingyu Liang. A theoretical analysis on feature learning in neural networks: Emergence from inputs and advantage over fixed features. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://arxiv.org/abs/2206.01717.

Zhenmei Shi, Jiefeng Chen, Kunyang Li, Jayaram Raghuram, Xi Wu, Yingyu Liang, and Somesh Jha. The trade-off between universality and label efficiency of representations from contrastive learning. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2022. URL https://arxiv.org/abs/2303.00106.

Zhenmei Shi, Junyi Wei, and Yingyu Liang. Provable guarantees for neural networks via gradient feature learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 36, 2024. URL https://arxiv.org/abs/2310.12408.

Gowthami Somepalli, Avi Schwarzschild, Micah Goldblum, C. Bayan Bruss, and Tom Goldstein. SAINT: Improved neural networks for tabular data via row attention and contrastive pre-training. In *NeurIPS 2022 First Table Representation Workshop*, 2022. URL https://arxiv.org/abs/2106.01342.

Mark G Stokes, Makoto Kusunoki, Natasha Sigala, Hamed Nili, David Gaffan, and John Duncan. Dynamic coding for cognitive control in prefrontal cortex. *Neuron*, 78(2):364–375, 2013.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Computing Surveys*, 55(6):1–28, 2022. URL https://arxiv.org/abs/2009.06732.

Constantino Tsallis. Possible generalization of boltzmann-gibbs statistics. *Journal of statistical physics*, 52:479–487, 1988.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems (NeurIPS)*, 30, 2017. URL https://arxiv.org/abs/1706.03762.

Michael Widrich, Bernhard Schäfl, Milena Pavlović, Hubert Ramsauer, Lukas Gruber, Markus Holzleitner, Johannes Brandstetter, Geir Kjetil Sandve, Victor Greiff, Sepp Hochreiter, et al. Modern hopfield networks and attention

for immune repertoire classification. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:18832–18845, 2020. URL https://arxiv.org/abs/2007.13505.

David J Willshaw, O Peter Buneman, and Hugh Christopher Longuet-Higgins. Non-holographic associative memory. *Nature*, 222(5197):960–962, 1969.

Dennis Wu, Jerry Yao-Chieh Hu, Teng-Yun Hsiao, and Han Liu. Uniform memory retrieval with larger capacity for modern hopfield models. In *Forty-first International Conference on Machine Learning (ICML)*, 2024a. URL https://arxiv.org/abs/2404.03827.

Dennis Wu, Jerry Yao-Chieh Hu, Weijian Li, Bo-Yu Chen, and Han Liu. STanhop: Sparse tandem hopfield model for memory-enhanced time series prediction. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024b. URL https://arxiv.org/abs/2312.17346.

Zhuoyan Xu, Zhenmei Shi, and Yingyu Liang. Do large language models have compositional ability? an investigation into limitations and scalability. In *ICLR 2024 Workshop on Mathematical and Empirical Understanding of Foundation Models*, 2024. URL https://openreview.net/forum?id=4XPeF0SbJs.

Jiahuan Yan, Jintai Chen, Yixuan Wu, Danny Z Chen, and Jian Wu. T2G-Former: organizing tabular features into relation graphs promotes heterogeneous feature interaction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 10720–10728, 2023. URL https://arxiv.org/abs/2211.16887.

Tzu-Hsien Yang, Sheng-Cian Shiue, Kuan-Yu Chen, Yan-Yuan Tseng, and Wei-Sheng Wu. Identifying pirna targets on mrnas in c. elegans using a deep multi-head attention network. *BMC bioinformatics*, 22(1):1–23, 2021.

Yunhao Zhang and Junchi Yan. Crossformer: Transformer utilizing cross-dimension dependency for multivariate time series forecasting. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2O: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems (NeurIPS)*, 36, 2023. URL https://arxiv.org/abs/2306.14048.

Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 11106–11115, 2021. URL https://arxiv.org/abs/2012.07436.

Tian Zhou, Ziqing Ma, Qingsong Wen, Liang Sun, Tao Yao, Wotao Yin, Rong Jin, et al. Film: Frequency improved legendre memory model for long-term time series forecasting. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:12677–12690, 2022. URL https://arxiv.org/abs/2205.08897.

Zhihan Zhou, Yanrong Ji, Weijian Li, Pratik Dutta, Ramana V Davuluri, and Han Liu. DNABERT-2: Efficient foundation model and benchmark for multi-species genomes. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2306.15006.

Supplementary Material

- **Appendix A: Table of Notations**
- **Appendix B: Related Works**
- **Appendix C: Supplementary Theoretical Backgrounds**
- **Appendix D: Experimental Details**
- **Appendix E: Additional Numerical Experiments**

A. Table of Notations

Table 3. Table of Notations.

Notation	Description
$\mathbf{a}, \mathbf{b}, \mathbf{c} \dots$	Vectors
$\mathbf{A}, \mathbf{B}, \mathbf{C} \dots$	Matrices
$\langle \mathbf{a}, \mathbf{b} \rangle$	Inner product of vectors $\mathbf{a}$ and $\mathbf{b}$, defined as $\mathbf{a}^\mathsf{T}\mathbf{b}$
$[I]$	Index set $\{1, \cdots, I\}$ for a positive integer I
$\lVert\cdot\rVert_2$	Spectral norm for matrices (aligned with l_2-norm for vectors)
$\boldsymbol{\xi} \in \mathbb{R}^d$	Memory patterns (keys)
$\mathbf{x} \in \mathbb{R}^d$	State/configuration/query pattern
$\boldsymbol{\Xi} \coloneqq [\boldsymbol{\xi}_1, \cdots, \boldsymbol{\xi}_M] \in \mathbb{R}^{d \times M}$	Shorthand for stored memory (key) patterns $\{\boldsymbol{\xi}_\mu\}_{\mu \in [M]}$
$n = \lVert\mathbf{x}\rVert$	Norm of the query pattern
$m = \mathrm{Max}_{\mu \in [M]} \lVert\boldsymbol{\xi}_\mu\rVert$	Maximum norm among the memory patterns
$\boldsymbol{\Xi}^\mathsf{T}\mathbf{x}$	M-dimensional overlap vector $(\langle \boldsymbol{\xi}_1, \mathbf{x} \rangle, \cdots, \langle \boldsymbol{\xi}_\mu, \mathbf{x} \rangle, \cdots, \langle \boldsymbol{\xi}_M, \mathbf{x} \rangle)$ in $\mathbb{R}^M$
$\left[\boldsymbol{\Xi}^\mathsf{T}\mathbf{x}\right]_\kappa$	The κ-th element of $\boldsymbol{\Xi}^\mathsf{T}\mathbf{x}$
κ	The number of non-zero element of $\mathrm{Sparsemax}$
n	Norm of $\mathbf{x}$, denoted as $n \coloneqq \lVert\mathbf{x}\rVert$
m	Largest norm of memory patterns, denoted as $m \coloneqq \mathrm{Max}_{\mu \in [M]} \lVert\boldsymbol{\xi}_\mu\rVert$
R	The minimal Euclidean distance across all possible pairs of memory patterns, $R \coloneqq \frac{1}{2} \mathrm{Min}_{\mu, \nu \in [M]} \lVert\boldsymbol{\xi}_\mu - \boldsymbol{\xi}_\nu\rVert$
S_μ	The sphere centered at the memory pattern $\boldsymbol{\xi}_\mu$ with finite radius R
$\mathbf{x}_\mu^\star$	The fixed point of $\mathcal{T}$ covered by S_μ, i.e. $\mathbf{x}_\mu^\star \in S_\mu$
Δ_μ	The separation of a memory pattern $\boldsymbol{\xi}_\mu$ from all other memory patterns $\boldsymbol{\Xi}$
$\widetilde{\Delta}_\mu$	The separation of $\boldsymbol{\xi}_\mu$ at a given $\mathbf{x}$ from all memory patterns $\boldsymbol{\Xi}$
$\mathbf{E}(\cdot)$	Embeddings
$\mathbf{x} \in \mathbb{R}^N$	Single tabular data point with N features (starting from Section 3).
$\mathtt{Concat}([\mathbf{A}, \mathbf{B}], \mathtt{axis} = 0)$	Concatenations of $\mathbf{A}, \mathbf{B}$ through first dimension ($\mathtt{axis} = 1$ for concatenation through second dimension)
$\mathbf{X}$	The internal embedding matrix of $\mathbf{x}$.
$\lceil\cdot\rceil$	Ceiling function
$\mathbf{X}_{i,j}$	The element of i-th rows and j-th columns in $\mathbf{X}$

B. Related Works

Machine Learning for Tabular Data. Tabular data is a common data type across various domains such as time series prediction, fraud detection, physics, and recommendation systems. The state-of-the-art machine learning models for tabular data are tree-based models, particularly the family of gradient boosting decision trees (GBDT) (Chen et al., 2015; Prokhorenkova et al., 2018; Ke et al., 2017). In recent years, as deep learning model architectures have thrived in the natural language processing (NLP) and computer vision (CV) domains, many attempts have been made to adapt and apply these successful architectures, such as Multi-layer Perceptron (MLP) (Kadra et al., 2021), Convolutional Neural Networks (CNN) (Buturović and Miljković, 2020), and Transformers (Huang et al., 2020; Padhi et al., 2021; Somepalli et al., 2022), to tabular data. Additionally, another line of work in deep learning involves creating differentiable tree-based models to enhance the capabilities of existing GBDT models (Arik and Pfister, 2021; Abutbul et al., 2020; Popov et al., 2020). However, unlike their dominance in NLP and CV, these deep learning models have struggled to surpass the performance of GBDTs on tabular data (Borisov et al., 2022; Grinsztajn et al., 2022). Recent work, such as TabR (Gorishniy et al., 2023), shows some marginal advantages over GBDTs on certain datasets For small datasets, TabPFN (Hollmann et al., 2022), utilizing Prior-Data Fitted Networks, performs better than tree-based methods. However, the memory and runtime usage scale quadratically with the training inputs. T2G-FORMER (Yan et al., 2023) fails to surpass XGBoost but performs better than other deep learning methods by learning feature relations. TANGOS (Jeffares et al., 2023) narrows the gap between deep learning models and tree-based models by applying specific regularization techniques during neural network training. To date, no deep learning model for tabular data has uniformly outperformed tree-based models

Modern Hopfield Models and Attention Mechanisms. Classical Hopfield models (Hopfield, 1984; 1982; Krotov and Hopfield, 2016) are quintessential representations of the human brain's associative memory, primarily functioning to store and retrieve specific memory patterns. Recently, there has been a resurgence of interest in associative memory models within the machine learning field (Burns, 2024; Burns and Fukai, 2023; Bietti et al., 2023; Cabannes et al., 2024b;a; Krotov, 2023; 2021; Krotov and Hopfield, 2021; Ramsauer et al., 2021), attributed to advances in understanding memory storage capacities (Wu et al., 2024a; Chaudhry et al., 2023; Demircigil et al., 2017; Krotov and Hopfield, 2016), innovative architectures (Hoover et al., 2023; Seidl et al., 2022; Fürst et al., 2022; Ramsauer et al., 2021), and their biologically plausibility (Kozachkov et al., 2022; Krotov and Hopfield, 2021). Notably, modern Hopfield models (Hu et al., 2024b;a; Wu et al., 2024a;b; Hu et al., 2023; Ramsauer et al., 2021)[3] demonstrate not only a strong connection to transformer attention mechanisms in deep learning but also superior performance and a theoretically guaranteed exponential memory capacity Their applicability spans diverse areas such as large language models (Hu et al., 2024a), immunology (Widrich et al., 2020), time series forecasting (Wu et al., 2024b; Auer et al., 2024), reinforcement learning (Paischer et al., 2022), and vision models (Fürst et al., 2022). In this context, this work emphasizes refining this line of research towards sparser models. We posit that this effort is crucial in guiding future research towards Hopfield-driven design paradigms and bio-inspired computing systems.

Sparse Attention. The attention mechanisms of transformers have demonstrated unparalleled performance in many domains, such as large language models (Xu et al., 2024; Liu et al., 2023; Zhang et al., 2023; Chowdhery et al., 2022; Brown et al., 2020), time series prediction (Zhou et al., 2022; 2021), and biomedical science (Zhou et al., 2024; Yang et al., 2021; Ji et al., 2021). However, the standard transformer architecture relies heavily on a dense quadratic attention score matrix. This structure presents computational challenges, especially for longer sequences, given its $\mathcal{O}(n^2)$ complexity for an input sequence of length n. In response to this challenge, a wealth of research has introduced sparse variants of attention mechanisms and transformers, aiming to strike a balance between computational efficiency and model expressiveness. For a comprehensive review, readers may refer to (Tay et al., 2022). Generally, these sparse attention/transformer methodologies fall into two categories:

1. **Structured-Sparsity Attentions** (Beltagy et al., 2020; Qiu et al., 2020; Child et al., 2019): These methods utilize structured, predetermined patterns in the attention matrix. Typically, each sequence token attends to a predetermined subset of tokens instead of the entire sequence.
2. **Dynamic Sparsity via Normalization Maps** (Peters et al., 2019; Correia et al., 2019; Krotov and Hopfield, 2016): In contrast to structured sparsity, these methods dynamically determine sparsity, centering on the most relevant input elements. Despite potentially retaining a space complexity of $\mathcal{O}(n^2)$, they dynamically tailor sparsity patterns to the data, bolstering scalability and clarity.

[3] For an in-depth tutorial, see (Brandstetter, 2021).

This work aligns more closely with the second category, as we employ the generalized sparse modern Hopfield model (Wu et al., 2024b) by substituting the softmax function in the modern Hopfield models with *sparsity-inducing alternatives*. A theoretical analysis of the efficiency of modern Hopfield models can be found in (Hu et al., 2024c).

C. Supplementary Theoretical Backgrounds

To highlight the computational benefits of the generalized sparse modern Hopfield model, we quote relevant results from (Wu et al., 2024b) here.

C.1. Definition of Memory Storage and Retrieval and Separation of Patterns

We adopt the formal definition of memory storage and retrieval from (Ramsauer et al., 2021) for continuous patterns.

Definition C.1 (Stored and Retrieved)**.** Assuming that every pattern $\boldsymbol{\xi}_\mu$ is surrounded by a sphere S_μ with finite radius $R := \frac{1}{2}\operatorname{Min}_{\mu,\nu\in[M]} \|\boldsymbol{\xi}_\mu - \boldsymbol{\xi}_\nu\|$, we say $\boldsymbol{\xi}_\mu$ is *stored* if there exists a generalized fixed point of $\mathcal{T}$, $\mathbf{x}_\mu^\star \in S_\mu$, to which all limit points $\mathbf{x} \in S_\mu$ converge to, and $S_\mu \cap S_\nu = \emptyset$ for $\mu \neq \nu$. We say $\boldsymbol{\xi}_\mu$ is *ϵ-retrieved* by $\mathcal{T}$ with $\mathbf{x}$ for an error.

We then introduce the definition of pattern separation for later convenience.

Definition C.2 (Pattern Separation)**.** Let's consider a memory pattern $\boldsymbol{\xi}_\mu$ within a set of memory patterns $\boldsymbol{\Xi}$.

1. The separation metric Δ_μ for $\boldsymbol{\xi}_\mu$ with respect to other memory patterns is the difference between its self-inner product and the maximum inner product with any other pattern

$$\Delta_\mu = \langle \boldsymbol{\xi}_\mu, \boldsymbol{\xi}_\mu \rangle - \operatorname*{Max}_{\nu,\nu\neq\mu} \langle \boldsymbol{\xi}_\mu, \boldsymbol{\xi}_\nu \rangle . \tag{C.1}$$

2. Given a specific pattern $\mathbf{x}$, the relative separation metric $\widetilde{\Delta}_\mu$ for $\boldsymbol{\xi}_\mu$ with respect to other patterns in $\boldsymbol{\Xi}$ is defined as:

$$\widetilde{\Delta}_\mu = \operatorname*{Min}_{\nu,\nu\neq\mu} \left(\langle \mathbf{x}, \boldsymbol{\xi}_\mu \rangle - \langle \mathbf{x}, \boldsymbol{\xi}_\nu \rangle \right) . \tag{C.2}$$

C.2. Supplementary Theoretical Results for Generalized Sparse Modern Hopfield Model

Theorem C.1 (Retrieval Error, Theorem 3.1 of (Wu et al., 2024b))**.** Let $\mathcal{T}_{\text{Dense}}$ be the retrieval dynamics of the dense modern Hopfield model (Ramsauer et al., 2021). It holds $\|\mathcal{T}(\mathbf{x}) - \boldsymbol{\xi}_\mu\| \leq \|\mathcal{T}_{\text{Dense}}(\mathbf{x}) - \boldsymbol{\xi}_\mu\|$ for all μ.

Theorem C.1 implies two computational advantages:

Corollary C.1.1 (Faster Convergence)**.** Computationally, Theorem C.1 suggests that $\mathcal{T}$ converges to fixed points using fewer iterations than $\mathcal{T}_{\text{dense}}$ for the same error tolerance. This means that $\mathcal{T}$ retrieves stored memory patterns more quickly and efficiently than its dense counterpart.

Corollary C.1.2 (Noise-Robustness)**.** In cases of noisy patterns with noise $\boldsymbol{\eta}$, i.e. $\widetilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\eta}$ (noise in query) or $\widetilde{\boldsymbol{\xi}}_\mu = \boldsymbol{\xi}_\mu + \boldsymbol{\eta}$ (noise in memory), the impact of noise $\boldsymbol{\eta}$ on the sparse retrieval error $\|\mathcal{T}(\mathbf{x}) - \boldsymbol{\xi}_\mu\|$ is linear for $\alpha \geq 2$, while its effect on the dense retrieval error $\|\mathcal{T}_{\text{Dense}}(\mathbf{x}) - \boldsymbol{\xi}_\mu\|$ (or $\|\mathcal{T}(\mathbf{x}) - \boldsymbol{\xi}_\mu\|$ with $2 \geq \alpha \geq 1$) is exponential.

Remark C.1. Corollary C.1.1 does not imply computational efficiency. The proposed model's sparsity falls under the category of *sparsity-inducing normalization maps* (Tay et al., 2022; Peters et al., 2019; Correia et al., 2019; Krotov and Hopfield, 2016). This means that, during the forward pass, the space complexity remains at $\mathcal{O}(n^2)$, on par with the dense modern Hopfield model.

Remark C.2. Nevertheless, Corollary C.1.1 suggests a specific type of "efficiency" related to faster memory retrieval compared to the dense Hopfield model. In essence, a retrieval dynamic with a smaller error converges faster to the fixed points (stored memories), thereby enhancing efficiency.

D. Experimental Details

Computational Hardware. All experiments are conducted on the platforms equipped with NVIDIA GEFORCE RTX 2080 Ti, Tesla A100 SXM GPU, and INTEL XEON SILVER 4214 @ 2.20GHz.

D.1. Additional Details on Datasets

We describe all the datasets used in our experiments in Table 4 and provide the download links to each dataset in Table 6.

Table 4. **Details of Datasets.** We summarize the statistics of 9 datasets we have used in Baseline I, 8 of which involve binary classification and 1 of which involve multi-class classification (4 classes).

	Adult	Bank	Blastchar	Income	SeismicBump	Shrutime	Spambase	Qsar	Jannis
# Numerical	6	7	3	6	14	6	58	41	54
# Categorical	8	9	16	8	4	4	0	0	0
# Train	34190	31648	4923	34189	1809	7001	3221	738	58613
# Validation	9769	9042	1407	9768	517	2000	920	211	16747
# Test	4884	4522	703	4885	258	1000	461	106	8373
# Task type	Bi-Class	Bi-Class	Bi-Class	Bi-Class	Bi-Class	Bi-Class	Bi-Class	Bi-Class	Multi-Class

Table 5. **Details of Datasets.** We summarize the statistics of 19 datasets covering four suite: categorical classification (**CC**), numerical classification (**NC**), categorical regression (**CR**), and numerical regression (**NR**).

	Dataset ID	Dataset Name	# of Categorical	# of Numerical
CC	361282	albert	11	21
	361283	default-of-credit-card-clients	2	20
	361286	compas-two-years	9	3
CR	361093	analcatdata_supreme	5	3
	361094	visualizing_soil	1	4
	361099	Bike_Sharing_Demand	5	7
	361104	SGEMM_GPU_kernel_performance	6	4
	361288	abalone	1	8
NC	361055	credit	0	10
	361062	pol	0	26
	361065	MagicTelescope	0	10
	361273	Diabetes130US	0	7
	361278	heloc	0	22
NR	361073	pol	0	27
	361074	elevators	0	17
	361077	Ailerons	0	34
	361079	house_16H	0	17
	361081	Brazilian_houses	0	9
	361280	abalone	0	8

The links to the four OpenML suites from (Grinsztajn et al., 2022) are **CC**: [4], **NC**[5], **CR**[6], **NR**[7]

[4] https://www.openml.org/search?type=benchmark&sort=date&study_type=task&id=300
[5] https://www.openml.org/search?type=benchmark&study_type=task&sort=tasks_included&id=298
[6] https://www.openml.org/search?type=benchmark&study_type=task&sort=tasks_included&id=299
[7] https://www.openml.org/search?type=benchmark&study_type=task&sort=tasks_included&id=297

Table 6. Dataset Sources.

Dataset	URL
Adult	http://automl.chalearn.org/data
Bank	https://archive.ics.uci.edu/ml/datasets/bank+marketing
Blastchar	https://www.kaggle.com/blastchar/telco-customer-churn
Income	https://www.kaggle.com/lodetomasi1995/income-classification
SeismicBumps	https://archive.ics.uci.edu/ml/datasets/seismic-bumps
Shrutime	https://www.kaggle.com/shrutimechlearn/churn-modelling
Spambase	https://archive.ics.uci.edu/ml/datasets/Spambase
Qsar	https://archive.ics.uci.edu/dataset/254/qsar+biodegradation
Jannis	http://automl.chalearn.org/data

D.2. Baselines

We evaluate BiSHop by comparing it to SOTA tabular learning methods, specifically selecting top performers in recent studies (Grinsztajn et al., 2022; Somepalli et al., 2022; Gorishniy et al., 2021).

- **LightGBM** (Ke et al., 2017).
- **CatBoost** (Prokhorenkova et al., 2018).
- **XGBoost** (Chen et al., 2015).
- **MLP** (Somepalli et al., 2022).
- **TabNet** (Arik and Pfister, 2021).
- **TabTransformer** (Huang et al., 2020).
- **FT-Transformer** (Gorishniy et al., 2021).
- **SAINT** (Somepalli et al., 2022).
- **TabPFN** (Hollmann et al., 2022). We implement TabPFN using 32 data permutations for ensemble, as in the original paper setting and truncate the training set to 1024 instances.
- **T2G-FORMER** (Yan et al., 2023). We implement T2G-FORMER by applying quantile transformation from the Scikit-learn library to Baseline I datsets, aligning with the default setting in. The hyperparameter space is at Table 14.
- **TANGOS** (Jeffares et al., 2023). We adapted the official TANGOS source code to include the datasets from Baseline I alongside the original datasets. The hyperparameter space is at Table 15.

Selection of Benchmark. We select Grinsztajn et al. (2022) as our benchmark for several reasons. Unlike other benchmarks that focus solely on tasks such as classification (Gardner et al., 2024), this benchmark encompasses both regression and classification tasks. Additionally, it provides results from 400 hyperparameter optimization (HPO) trials, ensuring a thorough hyperparameter search for each model. In contrast, some methods, such as (McElfresh et al., 2024), restrict HPO to 10 hours on a specific GPU, which can be insufficient for deep-learning-based methods like BiSHop that require more training time compared to tree-based methods. Moreover, comparing models under the same time constraints on different GPUs is inherently unfair.

D.3. Implementation Details

Data Prepossessing. We initially transform the categorical features into discrete values (e.g., 0, 1, 2, 3) and retain the raw numerical features without any processing. For tree-based method baselines, we employ the built-in categorical embedding method for categorical features. For deep learning baselines, we further encode the categorical features using one-hot encoding.

Evaluation. For each model's hyperparameter configuration, we run 3 experiments using the best configuration and report the average AUC score on the test set.

D.4. Training Details

Table 7. BiSHop Hyperparameter Space.

Parameter	Distribution	Default
Number of representation decoded	[2, 4, 8, 16, 24, 32, 48, 64, 128, 256, 320]	24
Stride factor	[1, 2, 4, 6, 8, 12, 16, 24]	8
Embedding dimension	[16, 24, 32, 48, 64, 128, 256, 320]	32
Number of aggregation in encoder	[2, 3, 4, 5, 6, 7, 8]	4
Number of pooling vector	[5, 10, 15]	10
Dimension of hidden layers (D^{model})	[64, 128, 256, 512, 1024]	512
Dimension of feedforward network (in MLP)	[128, 256, 512, 1024]	256
Number of multi-head attention	[2, 4, 6, 8, 10, 12]	4
Number of Encoder	[2, 3, 4, 5]	2
Number of Decoder	[0, 1]	2
Learning rate	LogUniform[(1e-6, 1e-4)	5e-5
ReduceLROnPlateau	factor=0.1, eps=1e-6	factor=0.1, eps=1e-6

Learning Rate Scheduler. We use `ReduceLROnPlateau` to fine-tune the learning rate to improve convergence and model training progress.

Optimizer. We use the Adam optimizer to minimize cross-entropy. The coefficients of the Adam optimizer, betas, are set to (0.9, 0.999).

Patience. We continue training until there are $\texttt{Patience} = 20$ consecutive epochs where the validation loss doesn't decrease, or we reach 200 epochs. Finally, we evaluate our model on the test set with the last checkpoint.

HPO. We report the number of HPO for each dataset from baseline I in Table 9. We report hyperparameter configurations for CatBoost in Table 10, LightGBM in Table 11, TabNet in Table 12, XGBoost in Table 13, T2G-Former in Table 14, Tangos in Table 15. We follow the same procedure of HPOs for Tangos and T2G-Former in Yan et al. (2023) and Jeffares et al. (2023), including the number of trials. For other methods, we follow the same settings as BiSHop.

Hyperparameter Importance Analysis. During random hyperparameter search, we observe that learning rate is the most important hyperparameter (see Table 8). We use WandB "sweep" features (Biewald et al., 2020) to calculate the importance of each hyperparameter. Our findings agree with (Grinsztajn et al., 2022) suggesting that learning rate is the most important hyperparameter for both neural network and gradient-boosted trees.

Table 8. **Hyperparameter Importance Scores.** The importance is calculate from features importance in RandomForest, averaging across all datasets. This results highlight learning rate is the most crucial hyperparameter.

Hyperparameter	RF Importance
Learning rate	0.17
Dropout	0.10
Number of heads	0.08
Number of aggregation	0.06
Dimension of hidden layers	0.06
Dimension of feed-forward network	0.13
Number of pooling factor	0.05
Number of encoder layer	0.05
Number of representation decoded	0.10

Table 9. Number of HPO in Baseline I.

Dataset	# of HPO
Adult	36
Bank	26
Blastchar	52
Income	174
SeismicBumps	200
Shrutime	16
Spambase	1
Qsar	67
Jannis	137

Table 10. Hyperparameter Configurations for CatBoost.

Parameter	Distribution	Default
Depth	UniformInt[3,10]	6
L2 regularization coefficient	UniformInt[1,10]	3
Bagging temperature	Uniform[0,1]	1
Leaf estimation iterations	UniformInt[1,10]	None
Learning rate	LogUniform[1e-5, 1]	0.03

Table 11. Hyperparameter Configurations for LightGBM.

Parameter	Distribution	Default
Number of estimators	[50, 75, 100, 125, 150]	100
Number of leavs	UniformInt[10, 50]	31
Subsample	UniformInt[0, 1]	1
Colsample	UniformInt[0, 1]	1
Learning rate	LogUniform[1e-1,1e-3]	None

Table 12. Hyperparameter Configurations for TabNet.

Parameter	Distribution	Default
n_d	UniformInt[8,64]	8
n_a	UniformInt[8,64]	8
n_steps	UniformInt[3,10]	3
Gamma	Uniform[1.0,2.0]	1.3
n_independent	UniformInt[1,5]	2
Learning rate	LogUniform[1e-3, 1e-1]	None
Lambda sparse	LogUniform[1e-4, 1e-1]	1e-3
Mask type	entmax	sparsemax

Table 13. Hyperparameter Configurations for XGBoost.

Parameter	Distribution	Default
Max depth	UniformInt[3,10]	6
Minimum child weight	LogUniform[1e-4,1e2]	1
Subsample	Uniform[0.5,1.0]	1
Learning rate	LogUniform[1e-3,1e0]	None
Colsample bylevel	Uniform[0.5,1.0]	1
Colsample bytree	Uniform[0.5,1.0]	1
Gamma	LogUniform[1e-3,1e2]	0
Alpha	LogUniform[1e-1,1e2]	0

Table 14. Hyperparameter Configurations for T2G-FORMER.

Parameter	Distribution	Default
# layers	UniformInt[1,3]	None
Feature embedding size	UniformInt[64,512]	None
Residual Dropout	Const(0.0)	None
Attention Dropout	Uniform[0, 0.5]	None
FNN Dropout	Uniform[0, 0.5]	None
Learning rate (main backbone)	LogUniform[3e-5, 3e-4]	None
Learning rate (column embedding)	LogUniform[5e-3, 5e-2]	None
Weight decay	LogUniform[1e-6, 1e-3]	None

Table 15. Hyperparameter Configurations for TANGOS.

Parameter	Distribution	Default
λ_1	LogUniform[0.001,10]	None
λ_2	LogUniform[0.0001,1]	None

E. Additional Numerical Experiments

E.1. Component Analysis

We separately remove each component of BiSHop to assess its impact on performance. We use the default hyperparameters as specified in Table 7 for the remaining components. We report the average AUC score of three runs using the default parameters for all datasets in Table 17.

- Without Cat Emb: We remove both individual and shared embedding methods as described in the tabular embedding section, replacing them with PyTorch's embedding layers (`torch.nn.Embedding`) while keeping the embedding dimension unchanged.
- Without Num Emb: We remove the `Piecewise Linear Encoding` method for numerical features, directly concatenating numerical features with the output of categorical embedding as detailed in Section 3.1.
- Without Patch Embedding: We remove the patch embedding method by setting the stride factor L to 1.
- Without Decoder: We remove the decoder blocks in BiSHop and pass the encoded data directly to MLP predictor.
- Without BiSHopModule: We replace the column-wise block and row-wise block in the BiSHop module with a MLP of hidden size 512.

The results demonstrate that each component contributes to varying degrees to the BiSHop model, with numerical embedding, decoder blocks, and the BiSHopModule being the most significant contributors.

Table 17. **Component Ablation**. In the ablation study, we remove one component at a time. By evaluating different crucial components in BiSHop, we prove that each component contributes to various degrees of model performance. In particular, numerical embedding, decoder blocks, and the BiSHopModule contribute the most.

Data	BiSHop	w/o Cat Emb	w/o Num Emb	w/o Patch Emb	w/o Decoder	w/o BiSHopModule
Adult	**91.74**	90.91	89.40	91.32	88.18	91.28
Bank	**92.73**	90.88	77.21	91.14	91.93	91.98
Blastchar	88.49	87.92	**88.81**	86.75	84.28	85.38
Income	**92.43**	91.01	90.38	91.56	91.44	91.36
SeismicBumps	**91.42**	90.03	87.85	89.33	80.75	79.34
Shrutime	**87.38**	86.49	81.75	81.32	86.26	85.41
Spambase	**100**	100	100	100	100	100
Qsar	92.85	91.15	**94.69**	91.50	93.04	91.65
Jannis	**89.66**	87.95	87.50	87.62	86.58	86.10
Average	**91.86**	90.82	88.62	90.06	89.16	89.17

E.2. Comparison with the Dense Modern Hopfield Model

Using the default hyperparameters of BiSHop, we evaluate its performance with three distinct layers: (i) `GSH` (generalized sparse modern Hopfield model), (ii) `Hopfield` (dense modern Hopfield model (Ramsauer et al., 2021)), and (iii) `Attn` (attention mechanism (Vaswani et al., 2017)). We report the average AUC score over 10 runs in Table 18.

Table 18. **Comparing the Performance of Sparse versus Dense Modern Hopfield Models and Attention Mechanism.** We contrast the performance of our generalized sparse modern Hopfield model with that of the dense modern Hopfield model and the attention mechanism. We achieve this by substituting the `GSH` layer with the `Hopfield` layer from (Ramsauer et al., 2021) and the `Attn` layer from (Vaswani et al., 2017). We report the average AUC score (in %) over 10 runs, with variances omitted as they are all $\leq 1.1\%$. The results indicates the superior performance of our proposed generalized sparse modern Hopfield model across datasets.

AUC (%)	Adult	Bank	Blastchar	Income	SeismicBump	Shrutime	Spambase	Qsar	Jannis	**Mean AUC**
`GSH`	**91.74**	**92.73**	**88.49**	**92.43**	**91.42**	**87.38**	**100**	**92.85**	**89.66**	**91.86**
`Hopfield`	91.72	92.60	85.31	91.65	78.63	86.81	100	91.27	85.04	89.23
`Attn`	91.44	92.46	83.14	91.46	78.42	83.04	100	89.88	88.28	88.68

E.3. Convergence Analysis

We calculate the validation loss and AUC score using the same default parameters and compare them with the dense modern Hopfield model. For ease of presentation, we plot the results of six datasets (Blastchar, Shrutime, Income, Bank, Qsar, and Jannis). We use the same hyperparameters for each dataset for both `GSH` and `Hopfield`. The results, averaged over 30 runs, are shown in Figure 4. The results indicate that `GSH` converges faster and achieves an AUC score that is equal to or higher than `Hopfield`.

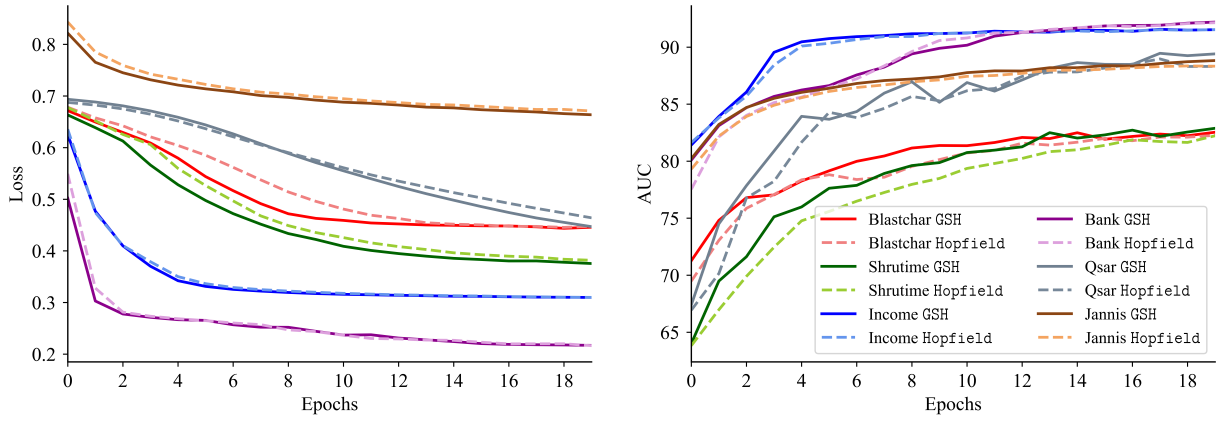

Figure 4. **Convergence Analysis.** We plot the validation loss and AUC score curves of the generalized sparse Hopfield model (`GSH`) and the dense Hopfield model (`Hopfield`). The results, as shown by the solid lines for `GSH`, indicate that the sparse Hopfield model converges faster and yields superior accuracy compared to the dense Hopfield model.

E.4. Rotation Experiments

In Table 19, we conduct the following experiments on rotating the datasets and the BiSHopModule's direction, both individually and in combination:

(R1) **Rotate the 2 Directions (Row-wise and Column-wise).** To validate the effectiveness of the bi-directional design in BiSHop, we conduct experiments by rotating these directions and reporting the performance and average results. The results indicate that the direction of BiSHop is vital for performance.

(R2) **Rotate the Datasets.** Following the experimental setup in (Grinsztajn et al., 2022), we randomly rotate datasets using a randomly generated special orthogonal matrix. The results indicate that BiSHop is robust against data rotation.

(R3) **Rotate the 2 Directions and the Datasets.** To further validate our findings, we apply both (R1) and (R2). The results show a drop in performance across nearly every dataset and align with our findings in (R1) and (R2).

The average AUC score across all datasets is reported for each type of rotation.

Table 19. **Comparing the Performance of Default BiSHop with Various Configurations on BiSHop Module and Datasets.** We apply the following configurations to BiSHopModule and datasets to validate BiSHop's ability to tackle (C1): rotate the 2 directions (R1), rotate the datasets (R2), and combined column-wise, row-wise, and rotate the 2 directions and the datasets (R3).

Method/Dataset	Adult	Bank	Blastchar	Income	SeismicBumps	Shrutime	Spambase	Qsar	Jannis	Average
BiSHop	91.74	92.73	88.49	92.43	91.42	87.38	100	92.85	89.66	**91.86**
(R1)	91.52	92.04	88.38	91.69	89.81	85.83	100	93.65	86.55	91.05
(R2)	91.67	92.21	88.51	91.41	92.74	87.68	100	93.08	85.03	91.37
(R3)	91.44	92.07	85.68	91.62	89.92	85.85	100	94.18	87.1	90.88

E.5. Hierarchy of BiSHopModule

In Table 20, we assess the impact of stacking different layers of BiSHopModule. We report the average AUC over all datasets for different layers of BiSHopModule.

Details. We progressively increase the layers within BiSHopModule from 1 to 8 in the Encoder and Decoder layers, keeping other parameters at their default settings. This approach allows us to examine how adding more layers affects the model's performance.

Results. Table 20 summarizes the performance in AUC, averaged over all datasets for various layers. The results suggest that 4 layers are the optimal setting to maximize performance.

Table 20. **Performance Comparison with Stacking Various Layers of BiSHopModule.** We vary different layers of BiSHopModule in the encoder-decoder structure. The results suggest 4 layers of BiSHopModule may maximize the model performance.

Layer/Dataset	Adult	Bank	Blstchar	Income	SeismicBumps	Shrutime	Spambase	Qsar	Jannis	Average
1 layer	91.52	92.32	88.55	91.58	92.20	86.33	100	94.03	84.39	91.21
2 layers	91.56	92.21	88.71	91.66	90.83	87.5	100	93.77	82.84	91.00
3 layers	91.65	92.38	88.47	91.50	93.11	87.34	100	93.08	85.09	91.40
4 layers	91.58	92.28	88.54	91.47	92.98	87.24	100	93.52	85.40	**91.45**
5 layers	91.57	92.17	88.55	91.47	90.12	87.69	100	93.37	84.93	91.10
6 layers	91.65	92.26	88.48	91.46	92.47	85.05	100	91.14	85.11	90.85
7 layers	91.54	92.16	87.88	91.47	93.04	87.26	100	92.05	84.69	91.12
8 layers	91.56	91.96	88.09	91.54	93.04	82.98	100	93.88	84.71	90.86

E.6. Component Analysis for Regression Datasets

We conduct ablation studies on 3 regression datasets to further validate the importance of various components in BiSHop. We randomly select 3 datasets from 'CR' in 4.1 'Datasets II' from our paper. For all experiments, we follow the same procedure as detailed in E.1. We use R2 for evaluation.

Details. We remove different components from BiSHop for the randomly selected regression tasks from 'CR' in 4.1 'Datasets II' since 'CR' includes both categorical and numerical features. This ablation study is presented alongside the classification task analysis in E.1 to validate the importance of various component design philosophies in BiSHop.

Results. We summarize the R2 score in Table 21. Different components in BiSHop contribute to various degrees of performance enhancement. Notably, BiSHopModule and CatEmb have the most significant impact on regression tasks.

Table 21. **Component Ablation**. We remove each component at one time and keep all other settings the same for regression datasets. For the experimental results, we prove that each component contributes to the model performance.

Data	BiSHop	w/o Cat Emb	w/o Num Emb	w/o Patch Emb	w/o Decoder	w/o BiSHopModule
361094	99.72	99.75	99.83	99.32	99.79	97.81
361288	54.91	54.09	54.67	50.7	53.93	51.41
361292	52.91	39.73	51.3	52.76	51.56	34.14
Average	69.18	64.52	68.60	67.59	68.43	61.12

F. Computational Time

Computational Complexity. We summarize the computational complexity for each function used in BiSHop in Table 22. Here we use the same notation as introduced in the main paper: N^{cat} represents the number of categorical features; N^{num} represents the number of numerical features; $N = N^{\text{num}} + N^{\text{cat}}$ represents the total number of features; G represents the embedding dimension; P represents the patch embedding dimension; D^{model} represents the hidden dimension; $\text{len}(Q)$ represents the size of the query pattern; C represents the number of pooling vectors; and $\text{len}(Y)$ represents the size of the memory pattern.

Table 22. Computational Complexity.

Function Name	Time Complexity
Categorical Embedding	$\mathcal{O}(G \times N^{\text{cat}})$
Numerical Embedding	$\mathcal{O}(G \times N^{\text{cat}})$
Patch Embedding	$\mathcal{O}(N \times P \times D^{\text{model}})$
GSH	$\mathcal{O}(\text{len}(Y) \times \text{len}(Q) \times D^{\text{model}})$
GSHPooling	$\mathcal{O}(\text{len}(Q) \times C \times D^{\text{model}} \times P^2)$
Merging	$\mathcal{O}(N \times P \times (D^{\text{model}})^2)$

Computationally Time. For each dataset and hyperparameter configuration, the average training time for BiSHop varies from 30 minutes to 2 hours. Based on different hyperparameter settings, the number of our model parameters varies from 10^7 to 10^8.