NORTHWESTERN
UNIVERSITY

Computer Science Department

Technical Report
Number: NU-CS-2024-07

March, 2024

Modern Hopfield Model for Deep Tabular Learning

Chenwei Xu

Abstract

In this work, we study tabular learning from modern Hopfield model perspectives. Specifically, we use a generalized sparse modern Hopfield model for learning tabular data representation and prediction. In this work, the **BiSHop** (**Bi**-Directional **S**parse **Hop**field Model) is introduced as an innovative framework for end-to-end tabular learning, tackling the two challenges in deep tabular learning: non-rotationally invariant data structures and feature sparsity. Inspired by the newly established connection between associative memory and attention mechanisms, BiSHop adopts a dual-component strategy. It sequentially processes data column-wise and row-wise through Bi-Directional learning modules, each equipped with generalized sparse Hopfield layers. These layers extend the traditional Hopfield model by introducing learnable sparsity. Methodologically, BiSHop enables multi-scale representation learning, effectively capturing intra-feature and inter-feature interactions with adaptive sparsity at various scales. Empirical validation on diverse real-world datasets shows that BiSHop exceeds the performance of current state-of-the-art methods with fewer hyperparameter optimization (HPO) runs, marking a significant advancement in deep tabular learning.

Keywords

Tabular Learning, Tabular Data, Deep Learning, Modern Hopfield Model, Modern Hopfield Network

NORTHWESTERN UNIVERSITY

Modern Hopfield Model for Deep Tabular Learning

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

MASTER OF SCIENCE

Field of Computer Science

By

Chenwei Xu

EVANSTON, ILLINOIS

March 2024

ABSTRACT

In this work, we study tabular learning from modern Hopfield model perspectives. Specifically, we use a generalized sparse modern Hopfield model for learning tabular data representation and prediction. In this work, the **BiSHop** (**Bi**-Directional **S**parse **Hop**field Model) is introduced as an innovative framework for end-to-end tabular learning, tackling the two challenges in deep tabular learning: non-rotationally invariant data structures and feature sparsity. Inspired by the newly established connection between associative memory and attention mechanisms, **BiSHop** adopts a dual-component strategy. It sequentially processes data column-wise and row-wise through Bi-Directional learning modules, each equipped with generalized sparse Hopfield layers. These layers extend the traditional Hopfield model by introducing learnable sparsity. Methodologically, **BiSHop** enables multi-scale representation learning, effectively capturing intra-feature and inter-feature interactions with adaptive sparsity at various scales. Empirical validation on diverse real-world datasets shows that **BiSHop** exceeds the performance of current state-of-the-art methods with fewer hyperparameter optimization (HPO) runs, marking a significant advancement in deep tabular learning.

ACKNOWLEDGEMENTS

First and foremost, I extend my profound gratitude to my advisor, Professor Han Liu, whose guidance, encouragement, inspiration, and steadfast support have been indispensable. Over the past two years, his dedication to research excellence, his courage in navigating uncharted territories, his discerning approach to selecting research problems, and his vast knowledge of statistical machine learning have profoundly influenced my academic journey. Engaging in discussions with him has been incredibly enriching, covering a wide spectrum from mathematical methods to strategic thinking, from the art of technical writing to public speaking, and from advanced coding techniques to professional conduct. His thoughtful and considerate feedback has significantly shaped my life choices and philosophy. I am immensely fortunate to have had him as my mentor.

I am also thankful to Professor David Demeter for his invaluable contributions as a member of my committee. His insights laid the groundwork for my understanding of language models. My gratitude extends to Professor Chris Riesbeck, the director of the master's program, and to all the faculty and staff within the MSCS program at Northwestern. Their unwavering support and guidance have been crucial in navigating my studies.

I owe a great debt of thanks to my project collaborators, Jerry Yao-Chieh Hu, Haozheng Luo, Weijian Li, Tim Lau, Yu-Chao Huang, Dennis Wu, Guo Ye, Mattson Thieme, Alex Reneau, Zhenyu Pan, Donglin Yang, and Ammar Gilani. Their collaboration was essential in completing this thesis, generously dedicating their time to help me refine my topic. Through this partnership, I learned not only about topic development but also about conducting meaningful research. Their expertise in various areas, including academic writing, theoretical development, and programming, has significantly bolstered my research skills.

Being a member of the MAGICS Lab has been an extraordinary honor. I express my special thanks once again to Professor Han Liu and Jerry Yao-Chieh Hu and Haozheng Luo and Tim Lau for their extensive support in my academic and personal life. The knowledge and skills I have acquired from them reach far beyond the academic realm and will undoubtedly enrich my future endeavors. I am also grateful for the opportunity to collaborate with remarkable lab members like Dennis Wu, Guo Ye, Qinjie Lin, Mattson Thieme, Alex Reneau, Shang Wu, Weimin Wu, Yufeng Zou, Zhenyu Pan, Zhihan Zhou, Donglin Yang, Yibo Wen, Chiao-Wei Hsu, and all other outstanding team members. Their deep insights into machine learning and their willingness to help have been tremendously motivating and beneficial to my work. Interacting and collaborating with them has been an incredibly productive and unforgettable experience, significantly contributing to my research publications. Special thanks are also due to Holly Jia for her diligent efforts in managing our computational resources, without which much of my work would have been unfeasible.

I would also like to extend my heartfelt gratitude to the READS collaborators at Fermilab: Kyle Hazelwood, Aakaash Narayanan, Jason St. John, and Vladimir Nagaslaev. While their contributions are not featured in this thesis, their partnership in the READS collaboration significantly enriches my master's research. This collaboration offers me a unique chance to employ machine learning techniques in addressing challenges within the field of particle physics.

Additionally, I wish to express my appreciation to Shifeng Li. As my closest friend in machine learning, collaborating with him on various side projects has greatly expanded my knowledge base.

Last but not least, I must acknowledge the unwavering support of my parents. Their unlimited support has been the bedrock of my academic pursuit, allowing me to dedicate myself fully to my studies without any concern.

It's inevitable that there are people I've inadvertently forgotten from this list, yet I am equally appreciative of their contributions. I extend my thanks to all such persons collectively and anonymously.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

Recent advancements in the development of deep learning architectures for tabular data have been notable, as evidenced by several key publications [1, 2, 3, 4]. These developments are largely fueled by the inadequacies of tree-based methods, which, despite their effectiveness in tabular learning, fail to seamlessly integrate with deep learning frameworks. Thus, the exploration into deep learning for tabular data is essential not only for performance improvement but also for bridging the current methodological divide. Nonetheless, findings from a recent benchmark study on tabular data [5] indicate that tree-based methods continue to outperform their deep learning counterparts, highlighting two principal obstacles for deep learning in tabular contexts:

(C1) **Non-Rotationally Invariant Data Structure:** The inherent non-rotationally invariant nature of tabular data diminishes the efficacy of deep learning techniques that rely on rotational invariance.

(C2) **Feature Sparsity:** Tabular datasets are often more sparse compared to the conventional datasets utilized in deep learning, presenting difficulties for deep learning models in extracting useful information from features that lack informativeness.

Drawing inspiration from the hierarchical and interconnected structure of the human brain [6, 7], we introduce the Bi-Directional Sparse Hopfield Model, a novel deep learning framework based

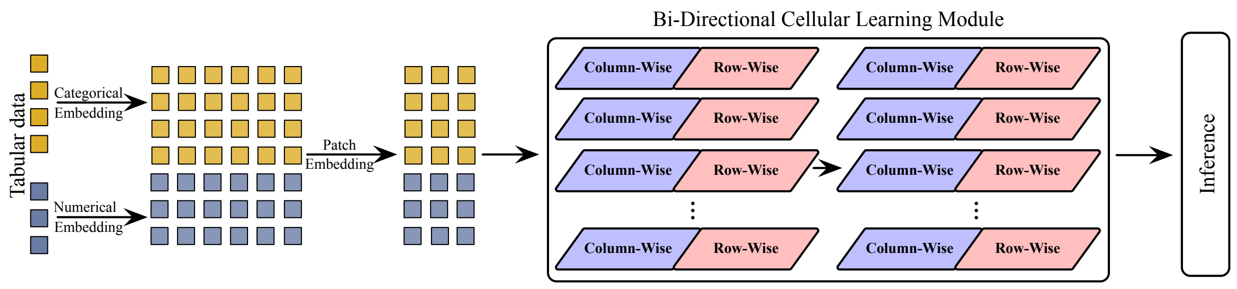

Figure 1.1: High-Level Visualization of BiSHop's Pipeline.

on the Hopfield network, specifically designed for tabular data.

To address the challenge posed by the non-rotationally invariant structure of tabular data (C1), our framework incorporates a bi-directional cellular learning module (`BiSHopModule`). This design mirrors the human brain's memory mechanisms, where different regions work collaboratively to form and retrieve associative memories. This module employs a dual Hopfield model approach, focusing separately on column-wise and row-wise data patterns. This method leverages the tabular data's structure as an inductive bias, akin to the collaborative memory formation and retrieval processes in different brain regions.

For tackling the features sparsity in tabular data (C2), we draw inspiration from the human brain, which dynamically adjusts neural activity based on the importance and relevance of information [8, 9, 10]. With this perspective, we utilize the generalized sparse Hopfield model [11]. The generalized sparse Hopfield model [11] is an extension of both the sparse Hopfield model [12] and the modern Hopfield model [13], with learnable sparsity. This model excels in robust representation learning and integrates smoothly with existing deep learning frameworks, emphasizing essential information. Furthermore, by stacking multiple layers of the generalized sparse Hopfield model within the `BiSHopModule`, our design mimics the brain's multi-tier associative memory system, enabling each layer to capture representations at different scales and adjust its sparsity, thus adding another layer of inductive bias (C2) to our model.

At its foundation, BiSHop excels in multi-scale representation learning, adept at capturing both intra-feature and inter-feature dynamics, while fine-tuning sparsity levels at each scale, mirroring the hierarchical structure observed in human cognition. By identifying representations in both column-wise and row-wise manners across various scales, and then concatenating these multi-scale representations for downstream tasks, BiSHop offers a comprehensive solution for tabular data, tailored to its unique challenges.

Our contributions are twofold:

1. Methodologically, we introduce BiSHop, an innovative deep-learning framework designed for tabular learnining. BiSHop incorporates two critical inductive biases (C1, C2) into its ar-

chitecture, utilizing the BiSHopModule and a hierarchical learning approach for tabular data processing. The BiSHopModule leverages the generalized sparse Hopfield model [11] for feature learning in tabular datasets, facilitating multi-scale learning of sparsity and providing enhanced resilience against noise. Our framework further employs a hierarchical two-joint design tailored to the unique aspects of tabular data, supporting learnable sparsity and multi-scale cellular learning. To augment the learning of representations for both numerical and categorical features, we integrate advanced tabular embedding techniques [4, 2, 14].

2. Experimentally, we conduct extensive evaluations using various real-world datasets alongside a renowned tabular benchmark [5], covering 17 classification and 11 regression tasks. We benchmark BiSHop against state-of-the-art (SOTA) models from both tree-based and deep learning domains. Our findings demonstrate that BiSHop achieves superior performance across a majority of the datasets evaluated, excelling in both regression and classification challenges.

1.1 Related Works

Advancements in Machine Learning for Tabular Data Tabular data is a common data types across various domains such as time series prediction, fraud detection, physics, and recommendation systems. The current state-of-the-art machine learning models on tabular data are tree-based models such as the family of gradient boosting decision trees (GBDT) models [15, 16, 17]. Lately, as deep learning has done well in understanding language and recognizing images, there's been many attempts to use these deep learning methods, like Multi-layer Perceptron (MLP) [18], Convolutional Neural Network (CNN) [19], and Transformer [4, 20, 1], for table data too. There's also work on making tree models that can learn and adjust, adding more power to the GBDT models [3, 21, 22]. However, these deep learning methods haven't yet beaten the tree models in working with table data [23, 5]. Recent studies, like TabR [24], show a slight edge over GBDT in some data sets. For small data sets, TabPFN [25] uses a special network and does better than tree models, but it needs a lot of memory and time to train. T2G-FORMER [26] doesn't beat XGBoost but does better than other deep learning models by learning feature relations. TANGOS [27] narrows the performance gap with tree models by using special training techniques. Still, no deep learning model has consistently outperformed tree models for table data yet.

Modern Hopfield Models and Attention Mechanisms The classic Hopfield models [28, 29, 30] serve as a key example of how the human brain can store and retrieve memories. There's a resurgence of interest in these models because of new insights into how memories can be stored more efficiently [30, 31], advancements in model architecture [32, 33, 34, 13], and a better understanding of their basis in biology [35, 36]. Modern Hopfield models [13] connect closely with deep learning's attention mechanisms, offering improved performance and the potential for substantially larger memory storage. These models are now seen as a natural evolution of focus mechanisms, paving the way for new machine learning architectures. They're being applied in a range of fields, from immunology [37] to time series forecasting [11, 38], reinforcement learning [39], and large language models [34] . This research aims to further refine these models, particularly towards more resource-efficient designs, highlighting the potential for Hopfield-based approaches and inspiration from natural systems.

Sparse Attention Attention mechanisms in transformers have shown remarkable success across domains like large lanugae models [40, 41], time series prediction [42, 43], and biomedical science [44, 45, 46]. This structure presents computational challenges, especially for longer sequences, given its $\mathcal{O}(n^2)$ complexity for an input sequence of length n. To address this, research has been exploring ways to make attention mechanisms more efficient without losing effectiveness. For a comprehensive review, readers may refer to [47]. Generally, these efforts can be grouped into:

1. **Structured-Sparsity Attentions** [48, 49, 50]: This approach uses Structured patterns to limit the parts of the data each segment needs to consider. Typically, each sequence token attends to a predetermined subset of tokens instead of the entire sequence.

2. **Dynamic Sparsity via Normalization Maps** [51, 52, 30]: Unlike structured patterns, this method adjusts sparsity based on the relevance of data segments, potentially improving effi-

ciency while maintaining or enhancing clarity and scalability. The dynamic sparsity does not retain a space complexity of $\mathcal{O}(n^2)$.

Our work mainly aligns with adaptive patterns, utilizing the generalized sparse Hopfield model [11] to introduce efficiency into the model through alternatives to traditional focus mechanisms.

1.2 Thesis Organizations

The rest of this thesis is structured as follows: Chapter 2 delves into the backdrop of the contemporary Hopfield model, encompassing its theoretical examination. Chapter 3 unveils **BiSHop**, our innovative strategy for leveraging the modern Hopfield model within tabular learning. Chapter 4 presents extensive experimental findings on real-world datasets, demonstrating the effectiveness and precision of our approach. Finally, Chapter 5 concludes the paper by summarizing the entire study and providing a comparison with existing work in the field of tabular learning.

CHAPTER 2

BACKGROUND: GENERALIZED SPARSE MODERN HOPFIELD MODELS

This section offers a brief summary of the modern Hopfield model [13] and the generalized sparse modern Hopfield model [11] and its extension, the generalized sparse modern Hopfield model [11]. The work by [11] extends the ideas found in [12, 13] by incorporating the concept of Tsallis α-entropy [53].

2.1 Notations

We denote vectors by lowercase bold letters, and matrices by upper case bold letters For vectors $\mathbf{a}$, $\mathbf{b}$, we define their inner product as $\langle \mathbf{a}, \mathbf{b} \rangle = \mathbf{a}^{\mathsf{T}}\mathbf{b}$. 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 stored memory patterns $\{\boldsymbol{\xi}_\mu\}_{\mu \in [M]}$.

2.2 (Dense) Modern Hopfield Models

Given a query pattern $\mathbf{x} \in \mathbb{R}^d$ and a set of memory patterns $\boldsymbol{\Xi} = [\boldsymbol{\xi}_1, \cdots, \boldsymbol{\xi}_M] \in \mathbb{R}^{d \times M}$, Hopfield models [29, 28, 30, 31, 36] are designed to encode these memory vectors and identify a particular memory $\boldsymbol{\xi}_\mu$ upon receiving a query $\mathbf{x}$. These models embed the memories within the energy landscape $E(\mathbf{x})$ of a physical model (for example, the Ising model discussed in [29]; refer to fig. 2.1 for an illustration), where each memory $\boldsymbol{\xi}_\mu$ represents a distinct local minimum. Upon receiving a query $\mathbf{x}$, the system initiates a process of energy-minimizing dynamics $\mathcal{T}$ starting from the query point, navigating through the energy landscape to locate the closest local minimum, thereby efficiently retrieving the memory that most closely resembles the query.

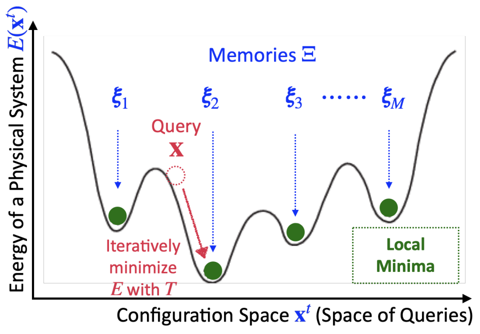
Energy of a Physical System $E(\mathbf{x}^t)$
Memories Ξ
ξ_1
ξ_2
ξ_3 ······ ξ_M
Query
$\mathbf{x}$
Iteratively minimize E with $\mathcal{T}$
Local Minima
Configuration Space $\mathbf{x}^t$ (Space of Queries)

Figure 2.1: Visualizing Hopfield Models.

The models are founded on two primary elements: the *energy function* $E(\mathbf{x})$, which maps memories to local minima, and the *retrieval dynamics* $\mathcal{T}(\mathbf{x})$, tasked with identifying a memory by reducing $E(\mathbf{x})$ from an initial query.

To formulate the energy function $E(\mathbf{x})$, memories are encoded [30] through the overlap method: $E(\mathbf{x}) = F(\Xi^{\mathsf{T}}\mathbf{x})$, where $F : \mathbb{R}^M \to \mathbb{R}$ is a function chosen to be continuous, ensuring that memories are positioned at the stationary points of the energy landscape. The selection of F dictates the specific variant of Hopfield model [30, 31, 13, 36]. Identifying appropriate retrieval dynamics $\mathcal{T}$ for a particular $E(\mathbf{x})$ is complex, as $\mathcal{T}$ must consistently lower $E(\mathbf{x})$ and ensure its fixed points coincide with the stationary points of $E(\mathbf{x})$, facilitating precise memory recall. For successful memory retrieval, $\mathcal{T}$ is required to:

(T1) Monotonically reduce $E(\mathbf{x})$ when applied iteratively.

(T2) Ensure its fixed points coincide with the stationary points of $E(\mathbf{x})$ for precise retrieval.

The proposed (dense/vanilla) modern Hopfield model [13] introduces a unique set of E and $\mathcal{T}$, integrating it seamlessly into deep learning frameworks through its linkage with the attention mechanism. This integration not only boosts performance but also ensures a theoretically guaranteed exponential memory capacity. Specifically, the model features a newly defined Hopfield energy function:

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

and the corresponding memory retrieval dynamics

$$\mathcal{T}_{\text{Dense}}(\mathbf{x}) = \boldsymbol{\Xi} \cdot \operatorname{Softmax}(\beta \boldsymbol{\Xi}^{\mathsf{T}} \mathbf{x}) = \mathbf{x}^{\text{new}}. \tag{2.2}$$

The function $\operatorname{lse}(\beta, \mathbf{z}) := \frac{1}{\beta} \log \left(\sum_{\mu=1}^{M} \exp(\beta z_\mu) \right)$ represents the log-sum-exponential operation for any given vector $\mathbf{z} \in \mathbb{R}^M$ with a positive scalar $\beta > 0$. This discovery uncovers:

- The retrieval dynamics $\mathcal{T}_{\text{Dense}}$ are proven to converge to stored memories and can accurately recall patterns in a single iteration.
- The contemporary Hopfield network, as described in equation (2.1), is characterized by its exponential capacity for memory storage relative to the pattern dimension d.
- Remarkably, the one-step execution of $\mathcal{T}_{\text{Dense}}$ aligns closely with the attention mechanism found in transformers, paving the way for the innovative introduction of Hopfield layers into network architecture designs.

2.3 Generalized Sparse Modern Hopfield Model

This section delivers a succinct summary of the generalized sparse modern Hopfield model [11], an extension of the modern Hopfield model [13] that incorporates Tsallis α-entropy [53], offering a sparser framework. This model also expands upon the sparse modern Hopfield model [12], thereby presenting a broader generalization. Subsequently, we elucidate the link between the memory retrieval dynamics within the generalized sparse Hopfield model and the attention mechanism. This connection fosters the development of the Generalized Sparse Hopfield (`GSH`) layer, a novel component for enhancing deep learning architectures.

Associative Memory Model. Let $\mathbf{z}, \mathbf{p} \in \mathbb{R}^M$, and $\Delta^M := \{\mathbf{p} \in \mathbb{R}_+^M \mid \sum_{\mu}^{M} p_\mu = 1\}$ be the $(M-1)$-dimensional unit simplex. [11] introduce the generalized sparse Hopfield energy

$$E(\mathbf{x}) = -\Psi^\star \left(\beta \boldsymbol{\Xi}^{\mathsf{T}} \mathbf{x} \right) + \frac{1}{2} \langle \mathbf{x}, \mathbf{x} \rangle , \tag{2.3}$$

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 1 ([51]) *The variational form of* α-EntMax *is defined as*

$$\alpha\text{-EntMax}(\mathbf{z}) \coloneqq \mathop{\mathrm{ArgMax}}_{\mathbf{p}\in\Delta^M}[\mathbf{p}^\mathsf{T}\mathbf{z} - \Psi^\alpha(\mathbf{p})], \tag{2.4}$$

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

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

The corresponding memory retrieval dynamics is given as

Lemma 1 (Retrieval Dynamics, Lemma 3.2 of [11]) *Given t as the iteration number, the generalized sparse Hopfield model exhibits a retrieval dynamic*

$$\mathcal{T}(\mathbf{x}_t) = \alpha\text{-EntMax}(\beta\boldsymbol{\Xi}^\mathsf{T}\mathbf{x}_t) = \mathbf{x}_{t+1}, \tag{2.5}$$

which ensures a monotonic decrease of the energy (2.3).

This model also enjoys nice memory retrieval properties:

Lemma 2 (Convergence of Retrieval Dynamics $\mathcal{T}$, Lemma 3.3 of [11]) *Given the energy function E and retrieval dynamics $\mathcal{T}$ defined in* (2.3) *and* (2.4)*, respectively. For any sequence $\{\mathbf{x}_t\}_{t=0}^\infty$ generated by iteration $\mathbf{x}_{t'+1} = \mathcal{T}(\mathbf{x}_{t'})$, all limit points of this sequence are stationary points of E.*

lemma 2 ensures the (asymptotically) exact memory retrieval of this model ((2.3) and (2.5)), Thus, it serves as a well-defined associative memory model.

Fundamentally, the work in [11] introduces a sparse variant of the modern Hopfield model by developing the energy function E and retrieval dynamics $\mathcal{T}$ via the convex conjugate of Tsallis entropic regularizers. This approach not only satisfies the criteria for a properly constructed modern

Hopfield model but also endows the system with enhanced robustness (corollary 2) and improved retrieval speeds (theorem 1 and corollary 1) in comparison to the conventional modern Hopfield model cited in [13]. Detailed theoretical underpinnings are discussed in section 2.5. Additionally, fig. 4.2 showcases experimental validations on tabular datasets, substantiating the assertions of increased sparsity (theorem 1), accelerated convergence (corollary 1), and noise resilience (corollary 2).

Generalized Sparse Hopfield (GSH) Layers for Deep Learning. The generalized sparse Hopfield model plays a pivotal role in deep learning architectures, especially due to its parallels with the transformer attention mechanism, akin to its predecessor models. In the subsequent sections, we will dissect this connection in detail and delve into the operational intricacies of the Generalized Sparse Hopfield (GSH) layers. This examination will shed light on how GSH layers can be seamlessly incorporated into and augment transformer-based frameworks, offering a comprehensive insight into their significance and influence within contemporary deep-learning endeavors.

Following [11, 12, 13], $\mathbf{X}$ and $\boldsymbol{\Xi}$ are defined in the associative space, encoded 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 (2.5) and applying $\mathbf{W}_V$ such that $\mathbf{V} \coloneqq \mathbf{K}\mathbf{W}_V$, we obtain:

$$\mathbf{Z} \coloneqq \mathbf{Q}^{\text{new}}\mathbf{W}_V = \alpha\text{-EntMax}(\beta\mathbf{Q}\mathbf{K}^\top)\mathbf{V}, \tag{2.6}$$

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

$$\mathtt{GSH}(\mathbf{R}, \mathbf{Y}) = \alpha\text{-EntMax}(\beta\mathbf{R}\mathbf{W}_Q\mathbf{W}_K^\top\mathbf{Y}^\top)\mathbf{Y}\mathbf{W}_K\mathbf{W}_V. \tag{2.7}$$

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. Achieving a self-attention-like mechanism requires setting $\mathbf{R}$ 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` layer takes the query $\mathbf{R}$ as its single input. The layer equips with 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].

2.4 Definition of Memory Storage and Retrieval and Separation of Patterns

We adopt the formal definition of memory storage and retrieval from [13] for continuous patterns.

Definition 2 (Stored and Retrieved) *Assuming that every pattern $\boldsymbol{\xi}_\mu$ 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 $\mathcal{T}$, $\mathbf{x}^\star_\mu \in S_\mu$, 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.*

Then we introduce the definition of pattern separation for later convenience.

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

Definition 3 (Pattern Separation) *Let's consider a memory pattern $\boldsymbol{\xi}_\mu$ within a set of memory patterns Ξ.*

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:*

$$\Delta_\mu = \langle \boldsymbol{\xi}_\mu, \boldsymbol{\xi}_\mu \rangle - \underset{\nu, \nu \neq \mu}{\mathrm{Max}} \langle \boldsymbol{\xi}_\mu, \boldsymbol{\xi}_\nu \rangle . \tag{2.8}$$

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

$$\widetilde{\Delta}_\mu = \underset{\nu, \nu \neq \mu}{\mathrm{Min}} \left(\langle \mathbf{x}, \boldsymbol{\xi}_\mu \rangle - \langle \mathbf{x}, \boldsymbol{\xi}_\nu \rangle \right) . \tag{2.9}$$

2.5 Theoretical Results for Generalized Sparse Modern Hopfield Model

Theorem 1 (Retrieval Error, Theorem 3.1 of [11]) *Let $\mathcal{T}_{Dense}$ be the retrieval dynamics of the dense modern Hopfield model [13]. It holds $\|\mathcal{T}(\mathbf{x}) - \boldsymbol{\xi}_\mu\| \leq \|\mathcal{T}_{Dense}(\mathbf{x}) - \boldsymbol{\xi}_\mu\|$ for all μ.*

theorem 1 implies two computational advantages:

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

Corollary 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 $\|\mathcal{T}_{Dense}(\mathbf{x}) - \boldsymbol{\xi}_\mu\|$ (or $\|\mathcal{T}(\mathbf{x}) - \boldsymbol{\xi}_\mu\|$ with $2 \geq \alpha \geq 1$) is exponential.*

Remark 1 *corollary 1 does not imply computational efficiency. The proposed model's sparsity falls under the category of sparsity-inducing normalization maps [47, 51, 52, 30]. 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 2 *Nevertheless, corollary 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.*

CHAPTER 3
BI-DIRECTIONAL CELLULAR LEARNING FOR TABULAR DATA WITH GENERALIZED SPARSE HOPFIELD MODEL

As in fig. 2.1, BiSHop use three distinct parts to integrate two pivotal inductive biases in tabular data: non-rotationally invariant data structures (C1) and sparse information in features (C2) [**:**]

- A joint **Tabular Embedding** layer is designed to processing categorical and numerical data separately.
- The **Bi-Directional Sparse Hopfield module (BiSHopModule)** leverages the generalized sparse 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 learn 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

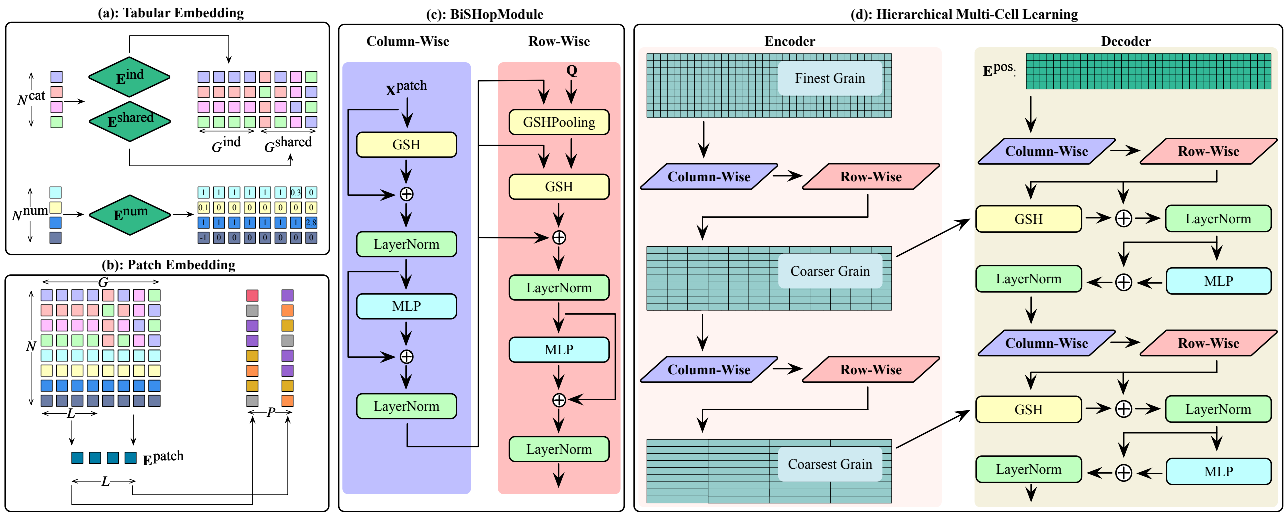

Figure 3.1: **BiSHop.** **(a) Tabular Embedding:** For a given input feature $\mathbf{x} = (\mathbf{x}^{\text{cat}}, \mathbf{x}^{\text{num}}) \in \mathbb{R}^{N=N^{\text{num}}+N^{\text{cat}}}$, the tabular embedding produces embeddings denoted as $\mathbf{E}_{\text{emb}} \in \mathbb{R}^{N\times G}$. **(b) Patch Embedding:** Using the combined numerical and categorical embeddings $\mathbf{E}^{\text{emb}} \in \mathbb{R}^{N\times G}$, 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 Hopfield model. It integrates the tabular structure's inductive bias (C1) by deploying interconnected row-wise and column-wise `GSH` blocks. **(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.

$\mathbf{E}^{\text{num}}$ transforms $\mathbf{x}^{\text{cat}}$ and $\mathbf{x}^{\text{num}}$ to a embedding dimension G, seperately. 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 [4]. For a tabular data point $\mathbf{x} = (\mathbf{x}^{\text{num}}, \mathbf{x}^{\text{cat}})$, a column embedding only acts on the categorical features $\mathbf{x}^{\text{cat}}$, as in $\mathbf{E}^{\text{cat}}(\mathbf{x}^{\text{cat}})$. 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 features $\{x_i^{\text{cat}}\}_{i\in[N^{\text{cat}}]}$, 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 in 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 concat all individual embedding row-wise, i.e. $\mathbf{E}^{\text{ind}}(\mathbf{x}^{\text{cat}}) := \texttt{Concat}([\mathbf{E}_1^{\text{ind}}(x_1^{\text{cat}}), \ldots, \mathbf{E}_{N^{\text{cat}}}^{\text{ind}}(x_{N^{cat}}^{\text{cat}})], \texttt{axis} = 0) \in \mathbb{R}^{N^{\text{cat}}\times G^{\text{emb}}}$. Then, we concatenate the shared embedding with all individual embeddings column-wise, i.e., $\mathbf{E}^{\text{cat}}(\mathbf{x}^{\text{cat}}) := \texttt{Concat}([\mathbf{E}^{\text{shared}}(\mathbf{x}^{\text{cat}}), \mathbf{E}^{\text{ind}}(x^{\text{cat}})], \texttt{axis} = 1) \in \mathbb{R}^{N^{\text{cat}}\times G}$. $\mathbf{E}^{\text{ind}}$ 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 [2, 14]. 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:

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

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: $\mathbf{E}^{\text{emb}}(\mathbf{x}) := \texttt{Concat}\left(\mathbf{E}^{\text{num}}(\mathbf{x}^{\text{num}}), \mathbf{E}^{\text{cat}}(\mathbf{x}^{\text{cat}}), \texttt{axis} = 0\right)$, 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 fig. 3.1 (a).

Patch Embedding. Motivated by [54, 55], we adopt patched embedding (shown in fig. 3.1 (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 $\mathbf{X}^{\text{emb}} := (\mathbf{E}^{\text{emb}}(\mathbf{x}))^T \in \mathbb{R}^{G\times N}$. 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 ceil function. Furthermore, we introduce a new embedding dimension D^{model} to represent each patch's hidden states. The patched embedding as $\mathbf{E}^{\text{patch}}(\mathbf{X}^{\text{emb}}) \in \mathbb{R}^{P\times N\times D^{\text{model}}}$. For future computation, we flip the patch dimension and feature dimension, resulting final output of patch embedding $\mathbf{X}^{\text{patch}} := (\mathbf{E}^{\text{patch}}(\mathbf{X}^{\text{emb}}))^T \in \mathbb{R}^{N\times P\times D^{\text{model}}}$. 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 [6], we present the core design of the BiSHop framework, the Bi-Directional Sparse Hopfield Module (`BiSHopModule`), as visualized in fig. 3.1 (b). The BiSHopModule incorporates the generalized sparse Hopfield model and integrate 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 $\mathbf{X}^{\text{patch}}_{n,p}, n \in [N], p \in [P]$ as the element in n-th row (feature) and p-th column (embedding).

Column-Wise Block. The column-wise `GSH` block (purple block on the LHS of fig. 3.1 (c)) is responsible for capturing embedding hidden information across the embedding dimension P for each feature. The process begins by passing the patch embeddings of n-th row of $\mathbf{X}^{\text{patch}}$, $\mathbf{X}^{\text{patch}}_{n,:}, n \in [N]$, to the `GSH` layer, followed by the addition of the original patch embeddings (similar to the residual connection of the standard transformer). Next, we pass the output above through one `LayerNorm` layer, one Multi-Layer Perception (MLP) layer, and another `LayerNorm`, and obtain the final output of the column-wise block $\mathbf{X}^{\text{col}}$:

$$\widehat{\mathbf{X}}^{\text{patch}}_{n,:} := \texttt{LayerNorm}\left(\mathbf{X}^{\text{patch}}_{n,:} + \texttt{GSH}(\mathbf{X}^{\text{patch}}_{n,:})\right), \tag{3.1}$$

$$\mathbf{X}^{\text{col}} := \texttt{LayerNorm}\left(\widehat{\mathbf{X}}^{\text{patch}} + \texttt{MLP}(\widehat{\mathbf{X}}^{\text{patch}})\right), \tag{3.2}$$

This sequence of operations ensures the effective transformation of the embeddings, facilitating the extraction of meaningful information from the feature space.

Row-Wise Block. The row-wise `GSH` block (pink block on the RHS of 3.1 (c)) serves a vital function in capturing 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 combined this pooled output with the row-wise output again, and add the row-wise output to the result. Following this, we pass the output of the above approach through a `LayerNorm` layer, then through an MLP layer, and finally through another `LayerNorm` layer. This sequence of operations yields the final output of the row-wise block:

$$\widehat{\mathbf{Q}}_{:,p} := \texttt{GSHPooling}(\mathbf{Q}_{:,p}, \mathbf{X}^{\text{col}}_{:,p}), \tag{3.3}$$

$$\widehat{\mathbf{X}}^{\text{row}}_{:,p} := \texttt{GSH}(\mathbf{X}^{\text{col}}_{:,p}, \widehat{\mathbf{Q}}_{:,p}), \tag{3.4}$$

$$\bar{\mathbf{X}}^{\text{row}} := \texttt{LayerNorm}(\widehat{\mathbf{X}}^{\text{row}} + \mathbf{X}^{\text{col}}), \tag{3.5}$$

$$\mathbf{X}^{\text{row}} := \texttt{LayerNorm}(\bar{\mathbf{X}}^{\text{row}} + \texttt{MLP}(\bar{\mathbf{X}}^{\text{row}})), \tag{3.6}$$

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

$$\texttt{BiSHopModule}(\cdot)\colon \mathbb{R}^{P\times N} \to \mathbb{R}^{P\times N}, \tag{3.7}$$

where input is $\mathbf{X}^{\text{patch}}$ and 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 [6, 7], we utilize a hierarchical structure to learn multi-scale information similar to [54, 43]. This is illustrated in fig. 3.1 (d). This structure consists of two main components: the encoder and the decoder, both of which incorporate the H layer 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 fig. 3.1 (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 [56] 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 the level h, we use $\mathbf{E}_h^{\text{merge}}$ 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 then iteratively applied across all layers in $1 \leq h \leq H$. We summarize the merging procedure at level h as:

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

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

$$\mathbf{X}^{\text{enc},h} := \texttt{BiSHopModule}(\widehat{\mathbf{X}}^{\text{enc},h-1}), \quad \text{for } 1 \leq h \leq H. \tag{3.8}$$

Decoder. The decoder (yellow block on RHS of fig. 3.1 (d)) captures information from each level of encoded data. To accomplish this, we utilize H stacked BiSHopModuless 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 a 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 in

$$\mathbf{X}^{\text{pos},h} := \begin{cases} \texttt{BiSHopModules}(\mathbf{E}^{\text{pos}}), & h = 0, \\ \texttt{BiSHopModules}(\mathbf{X}^{\text{dec},h-1}), & 1 \leq h \leq H. \end{cases} \tag{3.9}$$

$$\widehat{\mathbf{X}}^{\text{dec},h} := \texttt{GSH}(\mathbf{X}^{\text{pos},h}, \mathbf{X}^{\text{enc, h}}), 1 \leq h \leq H, \tag{3.10}$$

$$\bar{\mathbf{X}}^{\text{dec},h} := \texttt{LayerNorm}(\widehat{\mathbf{X}}^{\text{dec},h} + \mathbf{X}^{\text{pos},h}), \tag{3.11}$$

$$\mathbf{X}^{\text{dec},h} := \texttt{LayerNorm}(\bar{\mathbf{X}}^{\text{dec},h} + \texttt{MLP}(\bar{\mathbf{X}}^{\text{dec},h})). \tag{3.12}$$

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

Learnable Sparsity at Each Scale. Drawing inspiration from the dynamic sparsity observed in the human brain [8, 9, 10], the parameter α for each GSH layer is a learnable parameter by design [11, 52], which allows BiSHopModule to adapt to different sparsity for different resolutions. Namely, the learned representations at each scale are equipped with scale-specific sparsity.

CHAPTER 4

EXPERIMENTAL EVALUATIONS

Table 4.1: **BiSHop versus SOTA Tabular Learning Methods (Dataset II).** Following the benchmark [5], 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 of 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 [5]. We report with the best Accuracy scores for **`CC`** and **`NC`**, and R^2 score for **`CR`** and **`NR`**, (both in %) by HPO. We also report the number of HPOs used in BiSHop. Hyperparameter optimization of our method employs the "sweep" feature of Weights and Biases [59]. In the 19 different datasets, BiSHop delivers 11 optimal and 8 near-optimal results (within 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

In this section, we compare BiSHop with SOTA tabular learning methods, following the tabular learning benchmark paper [5]. We summarize our experimental results in table 4.1 and fig. 4.1.

	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**

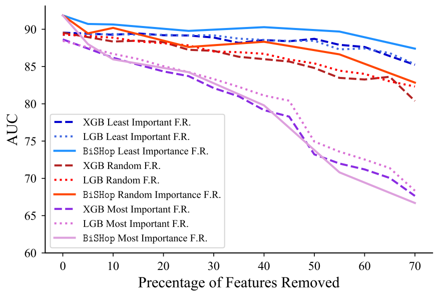

Figure 4.1: **(LHS:) BiSHop versus SOTA Tabular Learning Methods (Dataset I).** We evaluate BiSHop against predominant SOTA methods, including Deep Learning methods (MLP, TabNet, TabTransformer, FT-Transformer, SAINT, TabPFN, TANGOS, T2G-FORMER) and Tree-Based methods (LightGBM, CatBoost, XGBoost), across various datasets. We report the average AUC scores (in %) of 3 runs, with variances omitted as they are all $\leq 0.13\%$. Results quoted from [57, 1, 58, 4] 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 [59], 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. **(RHS:) Changing Feature Sparsity.** Following [5], we remove features in both **randomly** (red), **increasing** order of feature importance (purple), and **decreasing** (blue) order of feature importance (feature importance order obtained by random forest). We report the average AUC score across all datasets from BiSHop, XGBoost, and LightGBM. The results highlight BiSHop's capability in handling sparse features.

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 [5], 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 [5, 1, 2, 4, 1]. 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 section A.1 for datasets' details.

Datasets II. In the second experimental setting, we test BiSHop in the tabular benchmark [5]. The datasets compiled by this benchmark consist of 4 OpenML suites: (i) Categorical Classification (**`CC`**, suite_id: 334), (ii) Numerical Classification (**`NC`**, suite_id: 337), (iii) Categorical Regression (**`CR`**, suite_id: 335), and (iv) 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 dataset and perform identical preprocessing steps of the previous benchmark [5].

Metrics. We use the AUC score for the first experimental setting, aligned with literature. We use accuracy for classification task and R^2 score for regression task in the second experimental setting, aligned with [5].

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 [2], SAINT [1], TabPNF[25], TANGOS[27], T2G-FORMER [26] and (ii) tree-based methods such as LightGBM, CatBoost, and XGBoost [15]. 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 chapter A. 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[1] from the benchmark paper [2], comprising 4 deep learning methods and 3 tree-based methods, including (i)DL-based method such as (i) DL-based method such as MLP, ResNet [60], FT-Transformer [2], SAINT [1] and (ii) tree-based methods such as RandomForest, GradientBoostingTree (GBDT), and XGBoost [15]. We select the best results of each method from the benchmark [5]. Notably, these best results take 400 HPOs according to [5].

Setup. BiSHop's default parameter settings are as follows: Embedding dimension $G = 32$; Stride

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

factor L = 8; Number of pooling vector C = 10; Number of BiSHopModules H = 3; Number of aggregation in encoder r = 4; Number of representation decoded S = 24; Dropout = 0.2; Learning rate: 5×10^{-5}. For numerical embedding, we only gather quantile information from training data to process the embedding function. For hyperparameter tuning, we use the "sweep" feature of Weights and Biases [59]. Notably, due to the computational constraints, we manually end the HPO once our method surpass the best performance observed in the benchmarks. We report search space for all hyperparameters in table A.4 and other training details in section A.2. The optimization is conducted on training/validation sets, and we report the average test set scores over 3 iterations, using the best-performed configurations on the validation set. We show implementation and training details in the appendix.

Results. We summarize our results of the Baselines I in fig. 4.1 and the results of the Baselines II in table 4.1. In fig. 4.1, BiSHop outperforms both tree-based and deep-learning-based methods by a significant margin in most datasets. In table 4.1, BiSHop achieves optimal or near-optimal results with less 10% numbers (on average) of HPO in a tabular benchmark [5].

4.2 Ablation Studies

We conduct the following sets of ablation studies.

Changing Feature Sparsity. In fig. 4.1, we change feature sparsity on our datasets following [5, 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 from BiSHop, XGBoost, and LighGBM.

4.2.1 Component Analysis

We separately remove each component of BiSHop, and report the average AUC score of three runs using the default parameter for all datasets in table 4.2.

- Without Cat Emb: We remove both individual and shared embedding methods as described in the tabular embedding section, replacing them with one-hot encoding.

- Without Num Emb: We remove the `Piecewise Linear Encoding` method for numerical features, directly concatenating numerical features with the output of categorical embedding.
- Without Patch Embedding: We remove the patch embedding method by setting the stride factor L to 1.
- Without Decoder: We remove the decoder blocks 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 an MLP of the hidden state size of 256.

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 4.2: **Component Ablation**. We remove each component at one time and keep all other settings the same. 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
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

4.2.2 Comparison with the Dense Modern Hopfield Model

Using the default hyperparameters of BiSHop, we evaluate its performance using three distinct layers: (i) the `GSH` (generalized sparse Hopfield model), (ii) the `Hopfield` (dense modern Hopfield model [13]) and (iii) `Attn` (attention mechanism [61]). We report the average AUC score over 10 runs in table 4.3.

Table 4.3: **Comparing the Performance of Sparse versus Dense Hopfield Models and Attention Mechanism.** We contrast the performance of our generalized sparse 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 [13] and the `Attn` layer from [61]. We report the average AUC score (in %) over 10 runs, with variances omitted as they are all $\leq 0.08\%$. The results indicates the superior performance of our proposed generalized sparse 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

4.2.3 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 only plot the results of six datasets (Blastchar, Shrutime, Income, Bank, Qsar and Jannis). We use the same hyperparameter for each dataset for both `GSH` and `Hopfield`. We plot the results in Figure 4.2 with the mean of 30 runs. The result indicate that `GSH` converges faster and achieves an AUC score that is equal to or higher than `Hopfield`.

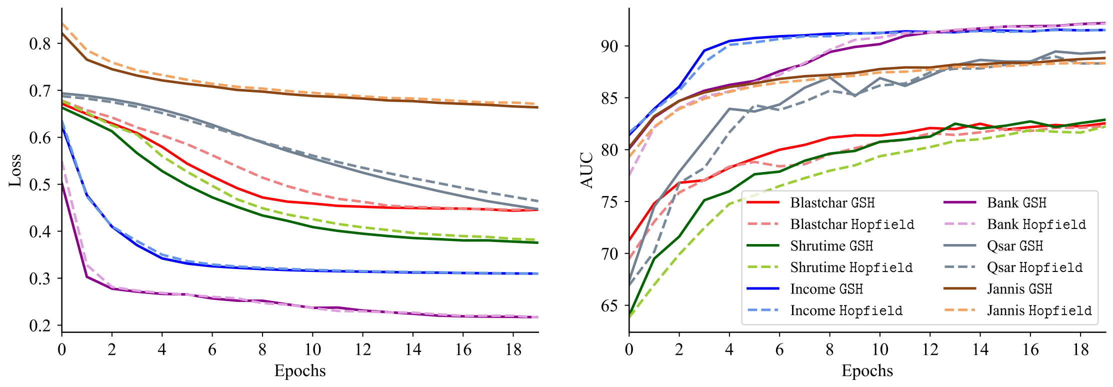

Figure 4.2: **Convergence Analysis.** We plot the validation loss and AUC score curves of generalized sparse (`GSH`) and dense (`Hopfield`) Hopfield models. The results indicate that the sparse Hopfield model (solid lines) converges faster and yields superior accuracy.

CHAPTER 5
CONCLUSION AND FUTURE WORK

We address the gap highlighted by [5] 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 hardness of deep tabular learning, with the inclusion of two important inductive biases of tabular data (C1, C2).

5.1 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 within 16% number of HPOs, compared with methods in the tabular benchmark [5]. 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.

REFERENCES

[1] G. Somepalli, M. Goldblum, A. Schwarzschild, C. B. Bruss, and T. Goldstein, "Saint: Improved neural networks for tabular data via row attention and contrastive pre-training," *arXiv preprint arXiv:2106.01342*, 2021.

[2] Y. Gorishniy, I. Rubachev, V. Khrulkov, and A. Babenko, "Revisiting deep learning models for tabular data," *Advances in Neural Information Processing Systems*, vol. 34, pp. 18 932–18 943, 2021.

[3] S. Ö. Arik and T. Pfister, "Tabnet: Attentive interpretable tabular learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, 2021, pp. 6679–6687.

[4] X. Huang, A. Khetan, M. Cvitkovic, and Z. Karnin, "Tabtransformer: Tabular data modeling using contextual embeddings," *arXiv preprint arXiv:2012.06678*, 2020.

[5] L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why do tree-based models still outperform deep learning on typical tabular data?" *Advances in Neural Information Processing Systems*, vol. 35, pp. 507–520, 2022.

[6] C. Presigny and F. D. V. Fallani, "Colloquium: Multiscale modeling of brain network organization," *Reviews of Modern Physics*, vol. 94, no. 3, p. 031 002, 2022.

[7] D. Krotov, "Hierarchical associative memory," *arXiv preprint arXiv:2107.06446*, 2021.

[8] M. G. Stokes, M. Kusunoki, N. Sigala, H. Nili, D. Gaffan, and J. Duncan, "Dynamic coding for cognitive control in prefrontal cortex," *Neuron*, vol. 78, no. 2, pp. 364–375, 2013.

[9] J. K. Leutgeb *et al.*, "Progressive transformation of hippocampal neuronal representations in "morphed" environments," *Neuron*, vol. 48, no. 2, pp. 345–358, 2005.

[10] D. J. Willshaw, O. P. Buneman, and H. C. Longuet-Higgins, "Non-holographic associative memory," *Nature*, vol. 222, no. 5197, pp. 960–962, 1969.

[11] D. Wu, J. Y.-C. Hu, W. Li, B.-Y. Chen, and H. Liu, "Stanhop: Sparse tandem hopfield model for memory-enhanced time series prediction," *arXiv preprint arXiv:2312.17346*, 2023.

[12] J. Y.-C. Hu, D. Yang, D. Wu, C. Xu, B.-Y. Chen, and H. Liu, "On sparse modern hopfield model," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[13] H. Ramsauer *et al.*, "Hopfield networks is all you need," *arXiv preprint arXiv:2008.02217*, 2020.

[14] Y. Gorishniy, I. Rubachev, and A. Babenko, “On embeddings for numerical features in tabular deep learning,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 991–25 004, 2022.

[15] T. Chen *et al.*, “Xgboost: Extreme gradient boosting,” *R package version 0.4-2*, vol. 1, no. 4, pp. 1–4, 2015.

[16] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, “Catboost: Unbiased boosting with categorical features,” *Advances in neural information processing systems*, vol. 31, 2018.

[17] G. Ke *et al.*, “Lightgbm: A highly efficient gradient boosting decision tree,” *Advances in neural information processing systems*, vol. 30, 2017.

[18] A. Kadra, M. Lindauer, F. Hutter, and J. Grabocka, “Well-tuned simple nets excel on tabular datasets,” *Advances in neural information processing systems*, vol. 34, pp. 23 928–23 941, 2021.

[19] L. Buturović and D. Miljković, “A novel method for classification of tabular data using convolutional neural networks,” *BioRxiv*, pp. 2020–05, 2020.

[20] I. Padhi *et al.*, “Tabular transformers for modeling multivariate time series,” in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2021, pp. 3565–3569.

[21] A. Abutbul, G. Elidan, L. Katzir, and R. El-Yaniv, “Dnf-net: A neural architecture for tabular data,” *arXiv preprint arXiv:2006.06465*, 2020.

[22] S. Popov, S. Morozov, and A. Babenko, “Neural oblivious decision ensembles for deep learning on tabular data,” *arXiv preprint arXiv:1909.06312*, 2019.

[23] V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci, “Deep neural networks and tabular data: A survey,” *CoRR*, vol. abs/2110.01889, 2021. arXiv: 2110.01889.

[24] Y. Gorishniy, I. Rubachev, N. Kartashev, D. Shlenskii, A. Kotelnikov, and A. Babenko, *Tabr: Unlocking the power of retrieval-augmented tabular deep learning*, 2023. arXiv: 2307.14338 [cs.LG].

[25] N. Hollmann, S. Müller, K. Eggensperger, and F. Hutter, *Tabpfn: A transformer that solves small tabular classification problems in a second*, 2023. arXiv: 2207.01848 [cs.LG].

[26] J. Yan, J. Chen, Y. Wu, D. Z. Chen, and J. Wu, “T2g-former: Organizing tabular features into relation graphs promotes heterogeneous feature interaction,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 2023, pp. 10 720–10 728.

[27] A. Jeffares, T. Liu, J. Crabbé, F. Imrie, and M. van der Schaar, "Tangos: Regularizing tabular neural networks through gradient orthogonalization and specialization," *arXiv preprint arXiv:2303.05506*, 2023.

[28] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons.," *Proceedings of the national academy of sciences*, vol. 81, no. 10, pp. 3088–3092, 1984.

[29] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities.," *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[30] D. Krotov and J. J. Hopfield, "Dense associative memory for pattern recognition," *Advances in neural information processing systems*, vol. 29, 2016.

[31] M. Demircigil, J. Heusel, M. Löwe, S. Upgang, and F. Vermet, "On a model of associative memory with huge storage capacity," *Journal of Statistical Physics*, vol. 168, pp. 288–299, 2017.

[32] B. Hoover *et al.*, "Energy transformer," *arXiv preprint arXiv:2302.07253*, 2023.

[33] P. Seidl *et al.*, "Improving few-and zero-shot reaction template prediction using modern hopfield networks," *Journal of chemical information and modeling*, vol. 62, no. 9, pp. 2111–2120, 2022.

[34] A. Fürst *et al.*, "Cloob: Modern hopfield networks with infoloob outperform clip," *Advances in neural information processing systems*, vol. 35, pp. 20 450–20 468, 2022.

[35] L. Kozachkov, K. V. Kastanenka, and D. Krotov, "Building transformers from neurons and astrocytes," *bioRxiv*, pp. 2022–10, 2022.

[36] D. Krotov and J. Hopfield, "Large associative memory problem in neurobiology and machine learning," *arXiv preprint arXiv:2008.06996*, 2020.

[37] M. Widrich *et al.*, "Modern hopfield networks and attention for immune repertoire classification," *Advances in Neural Information Processing Systems*, vol. 33, pp. 18 832–18 845, 2020.

[38] A. Auer, M. Gauch, D. Klotz, and S. Hochreiter, "Conformal prediction for time series with modern hopfield networks," *arXiv preprint arXiv:2303.12783*, 2023.

[39] F. Paischer *et al.*, "History compression via language models in reinforcement learning," in *International Conference on Machine Learning*, PMLR, 2022, pp. 17 156–17 185.

[40] A. Chowdhery *et al.*, “Palm: Scaling language modeling with pathways,” *arXiv preprint arXiv:2204.02311*, 2022.

[41] T. Brown *et al.*, “Language models are few-shot learners,” *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[42] T. Zhou *et al.*, “Film: Frequency improved legendre memory model for long-term time series forecasting,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 12 677–12 690, 2022.

[43] H. Zhou *et al.*, “Informer: Beyond efficient transformer for long sequence time-series forecasting,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, 2021, pp. 11 106–11 115.

[44] Z. Zhou, Y. Ji, W. Li, P. Dutta, R. Davuluri, and H. Liu, “Dnabert-2: Efficient foundation model and benchmark for multi-species genome,” *arXiv preprint arXiv:2306.15006*, 2023.

[45] T.-H. Yang, S.-C. Shiue, K.-Y. Chen, Y.-Y. Tseng, and W.-S. Wu, “Identifying pirna targets on mrnas in c. elegans using a deep multi-head attention network,” *BMC bioinformatics*, vol. 22, no. 1, pp. 1–23, 2021.

[46] Y. Ji, Z. Zhou, H. Liu, and R. V. Davuluri, “Dnabert: Pre-trained bidirectional encoder representations from transformers model for dna-language in genome,” *Bioinformatics*, vol. 37, no. 15, pp. 2112–2120, 2021.

[47] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, “Efficient transformers: A survey,” *ACM Computing Surveys*, vol. 55, no. 6, pp. 1–28, 2022.

[48] I. Beltagy, M. E. Peters, and A. Cohan, “Longformer: The long-document transformer,” *arXiv preprint arXiv:2004.05150*, 2020.

[49] J. Qiu, H. Ma, O. Levy, S. W.-t. Yih, S. Wang, and J. Tang, “Blockwise self-attention for long document understanding,” *arXiv preprint arXiv:1911.02972*, 2019.

[50] R. Child, S. Gray, A. Radford, and I. Sutskever, “Generating long sequences with sparse transformers,” *arXiv preprint arXiv:1904.10509*, 2019.

[51] B. Peters, V. Niculae, and A. F. Martins, “Sparse sequence-to-sequence models,” *arXiv preprint arXiv:1905.05702*, 2019.

[52] G. M. Correia, V. Niculae, and A. F. Martins, “Adaptively sparse transformers,” *arXiv preprint arXiv:1909.00015*, 2019.

[53] C. Tsallis, “Possible generalization of boltzmann-gibbs statistics,” *Journal of statistical physics*, vol. 52, pp. 479–487, 1988.

[54] Y. Zhang and J. Yan, “Crossformer: Transformer utilizing cross-dimension dependency for multivariate time series forecasting,” in *The Eleventh International Conference on Learning Representations*, 2023.

[55] Y. Nie, N. H. Nguyen, P. Sinthong, and J. Kalagnanam, “A time series is worth 64 words: Long-term forecasting with transformers,” *arXiv preprint arXiv:2211.14730*, 2022.

[56] Z. Liu *et al.*, “Swin transformer: Hierarchical vision transformer using shifted windows,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 10 012–10 022.

[57] G. Liu, J. Yang, and L. Wu, “Ptab: Using the pre-trained language model for modeling tabular data,” *arXiv preprint arXiv:2209.08060*, 2022.

[58] V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci, “Deep neural networks and tabular data: A survey,” *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[59] L. Biewald *et al.*, “Experiment tracking with weights and biases,” *Software available from wandb. com*, vol. 2, p. 233, 2020.

[60] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. arXiv: 1512.03385 [cs.CV].

[61] A. Vaswani *et al.*, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.

[62] J. Gardner, Z. Popovic, and L. Schmidt, “Benchmarking distribution shift in tabular data with tableshift,” *arXiv preprint arXiv:2312.07577*, 2023.

[63] D. McElfresh *et al.*, *When do neural nets outperform boosted trees on tabular data?* 2023. arXiv: 2305.02997 [cs.LG].

APPENDIX A
EXPERIMENTAL DETAILS

Computational Hardware. All experiments are conducted on the platform with NVIDIA GEFORCE RTX 2080 Ti, A100 GPUs, and INTEL XEON SILVER 4214 @ 2.20GHz.

A.1 Additional Details on Datasets

We describe all the datasets used in our experiments in table A.1, as well as the download links to each dataset in table A.3.

Table A.1: **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

The links to the four OpenML suites from [5] are **CC**: [1], **NC**[2], **CR**[3], **NR**[4]

[1]https://www.openml.org/search?type=benchmark&sort=date&study_type=task&id=300

[2]https://www.openml.org/search?type=benchmark&study_type=task&sort=tasks_included&id=298

[3]https://www.openml.org/search?type=benchmark&study_type=task&sort=tasks_included&id=299

[4]https://www.openml.org/search?type=benchmark&study_type=task&sort=tasks_included&id=297

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

Table A.3: 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`

A.2 Baselines

We evaluate BiSHop by comparing it to state-of-the-art (SOTA) tabular learning methods, specifically choosing top performers in recent studies [5, 1, 2].

- **LightGBM** [17]
- **CatBoost** [16]
- **XGBoost** [15]
- **MLP** [1]
- **TabNet** [3]
- **TabTransformer** [4]
- **FT-Transformer** [2]
- **SAINT** [1]
- **TabPFN** [25]. We implement TabPFN using 32 data permutations for ensemble same as the original paper setting and truncate the training set to 1024 instances.
- **T2G-FORMER** [26]. 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 A.10.
- **TANGOS** [27] We adapted the official TANGOS source code to include the datasets from Baseline I alongside the original datasets. The hyperparameter space is at table A.11.

Selection of Benchmark. We select [5] as our benchmark for several reasons. Unlike other benchmarks that focus solely on tasks such as classification [62], this benchmark encompasses both regression and classification tasks. This benchmark provides results from 400 hyperparameter optimization (HPO) trials, ensuring each model's hyperparameter search is sufficient. In contrast, some methods, such as [63], restrict HPO to 10 hours on a specific GPU. As a deep-learning-based method, BiSHop requires more training time compared to tree-based methods. Moreover, the comparison under the same time constraints on different GPUs is unfair.

A.3 Implementation Details

Data Prepossessing. We label encoded the categorical features, and keep the raw numerical features for further encoding.

Categorical Features. For tree based method, we employ the build in categorical embedding method. For MLP we use one-hot encoding.

Numerical Features. We implement Piece-wise Linear Encoding from [2, 14] which change the original scalar values of numerical features to a one-hot-like encoding.

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

A.4 Training Details

Table A.4: BiSHop hyperparameter space

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

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

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

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

HPO. We report the number of hpo for each dataset from baseline I in table A.5. We report hyperparameter configurations for CatBoost in table A.6, LightGBM in table A.7, TabNet in table A.8, XGBoost in table A.9, T2G-Former in table A.10, Tangos in table A.11. We follow the same procedure of HPOs for Tangos and T2G-Former in [26] and [27], including the number of trials. For other methods, we follow the same settings as BiSHop.

Table A.5: Dataset Sources

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

Table A.6: 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 A.7: 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 A.8: 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 A.9: 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 A.10: 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 A.11: Hyperparameter configurations for TANGOS.

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

Table A.12: BiSHop Hyperparameter Search Space

Data	BiSHop	w/o Cat Emb	w/o Num Emb	w/o Patch Emb	w/o Decoder	w/o GSH
Adult	91.50	91.54	89.40	91.32	0.306	0.306
Bank	92.23	92.90	77.21	91.14	0.346	0.306
Blastchar	88.49	88.05	88.81	86.75	0.434	0.306
Income	91.47	91.38	90.38	91.56	0.462	0.306
SeismicBumps	91.42	91.72	87.85	89.33	0.524	0.306
Shrutime	87.38	87.16	81.75	81.32	0.524	0.306
Spambase	100	100	100	100	0.524	0.306
Qsar	92.85	91.54	94.69	91.50	0.524	0.306
Jannis	88.27	87.95	87.50	0.610	0.524	0.306
Average	91.51	91.36	87.50	0.610	0.524	0.306