Andisheh Amrollahi

Spectral Bias in Supervised Learning

Diss. ETH No. 29670

DISS. ETH NO. 29670

SPECTRAL BIAS IN SUPERVISED LEARNING

A dissertation submitted to attain the degree of

DOCTOR OF SCIENCES
(Dr. sc. ETH Zurich)

presented by

ANDISHEH AMROLLAHI
M. sc. ETH Zurich

born on 31 March 1994

accepted on the recommendation of

Prof. Dr. Andreas Krause, examiner
Prof. Dr. Markus Püschel , co-examiner
Prof. Dr. Guy Van den Broeck , co-examiner

2023

DOI: 10.3929/ethz-b-000658791

ANDISHEH AMROLLAHI

SPECTRAL BIAS IN SUPERVISED LEARNING

To my parents,

ABSTRACT

We build upon advances of the past few decades in supervised learning, focusing on two successful model classes: ensembles of decision trees and neural networks. These models exhibit a property called "spectral bias," where their learned functions can be represented using compact Fourier (Walsh-Hadamard) representations. This property forms the foundation of the thesis.

For neural networks, we know that despite their capacity to learn complex functions, the algorithms used for training, such as stochastic gradient descent, tend to result in simpler learned functions. The notion of simplicity in this context is examined through the Walsh-Hadamard transform. Neural networks are found to learn lower-degree Fourier frequencies, which correspond to functions with factorized forms. (Ensembles of) Decision trees of depth d are exactly represented as sparse functions with degrees up to d.

The thesis comprises three chapters:

- "A scalable Walsh-Hadamard regularizer to overcome the low-degree spectral bias of neural networks" addresses the trade-off between simplicity and complexity in neural networks. The low-degree spectral bias can hinder learning complex functions, leading to sub-optimal generalization. A scalable regularization approach is proposed to mitigate this bias, improving generalization on various datasets.

- "Amortized SHAP values via function approximation" presents a method to compute SHAP values, an interpretable AI feature-attribution method. Compact Fourier representations of neural networks and decision tree ensembles are used to efficiently compute SHAP values, resulting in substantial speedups compared to existing methods.

- "Efficiently Learning Fourier Sparse Set Functions" introduces an algorithm for efficiently computing the Fourier transform of black-box functions with sparsity and low-degree assumptions. This algorithm outperforms existing sparse Walsh-Hadamard transforms by utilizing low-degree characteristics and novel hashing schemes. An extension of this work further improves query efficiency.

Overall, the thesis sets the stage for the exploration of spectral bias in supervised learning models and provides three different chapters that tackle different aspects of this phenomenon, presenting novel approaches to address spectral bias in neural networks, compute SHAP values, and efficiently compute Fourier transforms of sparse functions.

ZUSAMMENFASSUNG

Wir bauen auf den Fortschritten der letzten Jahrzehnte im Bereich des überwachten Lernens auf und konzentrieren uns auf zwei erfolgreiche Modellklassen: Ensembles von Entscheidungsbäumen und neuronale Netze. Diese Modelle weisen eine Eigenschaft auf, die als "spektrale Verzerrung" bezeichnet wird, d. h. ihre gelernten Funktionen lassen sich durch kompakte Fourier-Repräsentationen (Walsh-Hadamard) darstellen. Diese Eigenschaft bildet die Grundlage für diese Arbeit. Von neuronalen Netzen wissen wir, dass trotz ihrer Fähigkeit, komplexe Funktionen zu erlernen, die zum Training verwendeten Algorithmen, wie z. B. der stochastische Gradientenabstieg, tendenziell zu einfacheren erlernten Funktionen führen. Der Begriff der Einfachheit wird in diesem Zusammenhang anhand der Walsh-Hadamard-Transformation untersucht. Es ist bekannt, dass neuronale Netze Fourier-Frequenzen niedrigeren Grades lernen, die Funktionen mit faktorisierten Formen entsprechen. (Ensembles von) Entscheidungsbäumen der Tiefe d werden exakt als spärliche Funktionen mit Graden bis zu d dargestellt. Die Arbeit besteht aus drei Kapiteln:

- "Low degree spectral bias of neural networks" befasst sich mit dem Kompromiss zwischen Einfachheit und Komplexität in neuronalen Netzen. Die spektrale Verzerrung niedrigen Grades kann das Lernen komplexer Funktionen behindern, was zu suboptimaler Generalisierung führt. Es wird ein skalierbarer Regularisierungsansatz vorgeschlagen, um diese Verzerrung abzuschwächen und die Generalisierung auf verschiedenen Datensätzen zu verbessern.

- "Utilizing the spectral bias to compute SHAP values" stellt eine Methode zur Berechnung von SHAP-Werten vor, eine interpretierbare KI-Merkmalszuweisungsmethode. Kompakte Fourier-Darstellungen von neuronalen Netzen und Entscheidungsbaum-Ensembles werden zur effizienten Berechnung von SHAP-Werten verwendet, was zu einer erheblichen Beschleunigung der Berechnungen im Vergleich zu bestehenden Methoden führt.

- "Computing sparse and/or low-degree transforms" stellt einen Algorithmus zur effizienten Berechnung der Fourier-Transformation von Black-Box-Funktionen mit Sparsity- und Low-Degree-Annahmen

vor. Dieser Algorithmus übertrifft bestehende dünn besetzte Walsh-Hadamard-Transformationen, indem er Merkmale niedrigen Grades und neuartige Hashing-Schemata nutzt. Eine Erweiterung dieser Arbeit verbessert die Abfrageeffizienz weiter.

Zusammenfassend legt die Arbeit den Grundstein für die Erforschung der spektralen Verzerrung in überwachten Lernmodellen und gibt einen Überblick über die drei Arbeiten, die sich mit verschiedenen Aspekten dieses Phänomens befassen und neue Ansätze zur Berechnung von SHAP-Werten, zur Bewältigung der spektralen Verzerrung in neuronalen Netzen und zur effizienten Berechnung von Fourier-Transformationen dünnbesetzte Funktionen vorstellen.

ACKNOWLEDGEMENTS

I would like to thank my professor Andreas Krause for his supervision, support, trust, and help throughout these years.

I would like to thank my collaborator Chris Wendler for all of his contributions both in terms of technical content and being a good friend.

I would like to thank my close friends and colleagues Bhavya, Lenart, Scott, Parnian, Max, and Pragnya for making the workplace that much more friendly and fun.

I would like to thank my girlfriend Lena for her emotional support throughout my whole PhD.

I would like to thank Rita, the group secretary for all her help during my PhD.

Finally, I would like to thank my family and especially my parents, without whom I would not even be here.

CONTENTS

NOTATION

FREQUENTLY USED SYMBOLS

$g : \{0,1\}^n \to \mathbb{R}$	Pseduo-boolean function of dimensionality n
$\mathbb{F}$ or $\{0,1\}$	Unique field containing two elements {0,1}
f	frequency $f \in \{0,1\}^n$
$\deg(f)$	degree of frequency i.e. number of its non-zero elements
d	degree of a pseudo-boolean function
k	sparsity of a pseudo-boolean function
Ψ_f	Fourier basis function corresponding to frequency f $\Psi_f(x) = (-1)^{\langle f,x \rangle}$

1

INTRODUCTION

There have been great advances in the realm of supervised learning in the past three decades. Most of these efforts have focused on providing algorithms that produce more accurate models for a variety of different (supervised) tasks. In this thesis, we study two classes of remarkably successful models that work well on tabular data: (ensembles of) decision trees and neural networks. The functions both these models represent after training, have a property commonly referred to as a *spectral bias*. Namely, we can *exactly represent* (in the case of ensembles of decision trees) or *efficiently approximate* (in the case of neural networks) these models using compact *Fourier* a.k.a Walsh-Hadamard representations. This fundamental fact is the basis of this thesis. Before discussing our contributions, we make clear what we mean by the spectral bias of decision tree models and neural networks.

Spectral bias of ensembles of neural networks:

We know that deep fully connected networks trained through (stochastic) gradient descent represent functions that are "simple". This, on the surface, may seem in contrast to classical work on neural networks showing that deep fully connected neural networks can approximate arbitrary (complex) functions, more commonly known as the universal approximation theorem [1, 2]. However as made formal by numerous works [3–7], even though deep networks can learn arbitrary complex functions, the algorithm used to train them, namely (stochastic) gradient descent, gives rise to a learned function that is "simple". This notion of simplicity is not agreed upon and works such as [8–11] each introduces a different notion of "simplicity".

One way to quantify this simplicity is through the Fourier (spectral) domain. In *discrete* domains, with tabular datasets, where the input to the neural network is a high dimensional zero-one vector Valle-Perez, Camargo & Louis [10] and Yang & Salman [12] provide spectral bias results for the function learned by the neural network. By viewing a fully connected neural network as a function that maps zero-one vectors to real values, one can expand this function in terms of the Fourier –a.k.a Walsh-Hadamard – basis functions. Through analysis of the NTK gram matrix on the Boolean cube, Yang & Salman [12] theoretically show that roughly speaking, neural networks tend to learn lower-degree Fourier frequencies.

We clarify what the low-degree frequencies signify. The Walsh-Hadamard basis functions have a natural ordering in terms of their complexity called their *degree*. The degree specifies how many features each basis function is dependent upon. For example, the zero-degree basis function is the constant function and the degree-one basis functions are functions that depend on exactly one feature. Therefore, when the neural network learns lower-degree frequencies, it means that the function it represents admits a simple factorized form. Namely, if the function is of low-degree d, then it can be written as a summation of (Fourier basis) functions, where each function depends on at most d variables.

Spectral bias of ensembles of decision trees:

We know that the function a decision tree of depth d represents is sparse, $k = O(4^d)$-sparse, that is it contains at most $k = O(4^d)$ non-zero Fourier (Walsh-Hadamard) coefficients in its support [13–15]. A decision tree of depth d contains frequencies of degree at most d. Extending this to ensembles of decision trees, if the ensemble consists of T different trees then its Fourier transform is $k = O(T4^d)$-sparse and contains frequencies with degrees less or equal to its maximum depth.

In this thesis, I present three chapters. Here, I provide an overview of what type of question each one of them addresses.

1.1 LOW-DEGREE SPECTRAL BIAS OF NEURAL NETWORKS

The simplicity/spectral bias of neural networks is a double-edged sword. On one hand, bias towards simpler functions can help the neural network generalize better (on the test set) since it stops overfitting. On the other hand, it can stop the network from learning a "complex enough" function and hurt its generalization. This is essentially the bias-variance trade-off in Machine Learning.

In this work, we show that the second case can indeed be true on some datasets. That is the low-degree spectral bias of the neural network guides it towards a function that is not complex enough. We empirically evaluate the spectral bias of neural networks through various extensive experiments and consolidate previous theoretical findings that neural networks tend to learn lower degree frequencies. To remedy the spectral bias, we propose a new scalable functional regularization scheme that aids the neural network in learning higher degree frequencies. We extensively evaluate our regularizer on synthetic datasets to gain insights into its behavior. Finally, we show

significantly improved generalization on four different datasets compared to standard neural networks and other relevant baselines.

1.2 UTILIZING THE SPECTRAL BIAS TO COMPUTE SHAP VALUES

This chapter provides an application of what we can do using the compact Fourier representations of supervised learning models such as a neural network or an ensemble of decision trees. Namely, we use the representation to compute SHAP values. SHAP values – a.k.a. SHapley Additive exPlanations – are a popular local feature-attribution method widely used in interpretable and explainable AI. We propose a two-stage approach for estimating SHAP values. As mentioned in the introduction we can extract compact Fourier representations of both neural networks and ensembles of decision trees. Given the representation, we use the Fourier representation to exactly compute SHAP values. The second step is computationally very cheap because firstly, the representation is compact and secondly, we prove that there exists a closed-form expression for SHAP values for the Fourier basis functions. Thirdly, the expression we derive effectively *linearizes* the computation into a summation and is amenable to parallelization on multiple cores or a GPU. Since the function approximation (first step) is only done once, it allows us to produce Shapley values in an amortized way. This makes our algorithm orders of magnitudes faster than previous methods such as KernelShap where each explained instance requires solving an expensive optimization problem. We show speedups of 10-10000x compared to relevant baseline methods for equal levels of accuracy.

1.3 COMPUTING THE SPARSE AND/OR LOW-DEGREE SPECTRUM

Given access to the structure of an ensemble of trees model (white-box access) one can compute its Fourier transform recursively. However, we can not say the same about a neural network. If the input to the neural network (the number of features) is n-dimensional, to compute its Walsh-Hadamard (Fourier) transform one would need to evaluate the network on all 2^n many inputs.

In this work, we provide a new sparse Walsh-Hadamard transform algorithm to compute the Fourier transform of any given black box function given only query access to the function. More precisely, let $h : \{0,1\}^n \to \mathbb{R}$ be a black-box function, query access means that we can pick an arbitrary x and query the value $h(x)$. In this work, we provide algorithms that exploit

both the *sparsity* and *low-degree* assumptions on the black-box to efficiently compute the Fourier transform. The algorithm is efficient in terms of both query complexity and computational complexity.

The contributions of this algorithm are:

- It is the first sparse Walsh Hadamard transform algorithm to exploit low-degreeness of the underlying function to achieve better computational and query complexity.
- It introduces new hashing schemes that allow it to overcome restrictive randomness assumptions on the support that all previous sparse Walsh-Hadamard transforms usually have [16–18]

In unpublished work, this work was later extended by making its frequency recovery primitive even more query-efficient while trading off computational complexity. The key insight here was that one can view the frequency recovery primitive of our proposed algorithm as a linear error-correcting code problem. We include the unpublished theoretical and experimental results in Sections 5.5 and 5.6.

1.4 OVERVIEW OF PUBLICATIONS

In what follows is a list of my publications in chronological order with a short explanation.

- Amrollahi, A. *et al. Efficiently Learning Fourier Sparse Set Functions* in *Advances in Neural Information Processing Systems* **32** (Curran Associates, Inc., 2019)

In this paper, I presented an algorithm for efficiently extracting Fourier transforms from sparse and low-degree set functions, including neural networks with zero-one inputs. It situates itself within the broader context of learning sparse, low-degree set functions. The key contribution is an efficient algorithm for learning functions with low-degree Fourier supports, offering significant sample complexity and runtime advantages, and applicable to sparse graphs, decision trees, and more, including a robust version with strong approximation guarantees. Chapter 6 of the thesis is based on ideas from this paper.

- Wendler, C. *et al. Learning set functions that are sparse in non-orthogonal Fourier bases* in *Proceedings of the AAAI Conference on Artificial Intelligence* **35** (2021), 10283

This work introduces a new family of algorithms for learning Fourier-sparse set functions in discrete machine learning applications for **non-**

orthogonal Fourier transforms. This paper is essentialy an extension of the previous paper to non-orthogonal bases. Examples of functions that are dense in the orthogonal Fourier basis but sparse in an non-orthogonal basis are coverage functions or auction preference functions [20]. The algorithms are demonstrated to be query-efficient and can be utilized real-world scenarios like recommender systems and auctions.

- Valentin, R. *et al.* Instance-wise algorithm configuration with graph neural networks. *arXiv preprint arXiv:2202.04910* (2022)

Our submission for the ML4CO NeurIPS 2021 competition involved a graph neural network trained on a dataset of SCIP solver performances, significantly improving MILP solving efficiency and earning us third place globally and first in the student category.

- Gorji, A., Amrollahi, A. & Krause, A. *A scalable Walsh-Hadamard regularizer to overcome the low-degree spectral bias of neural networks* in *The 39th Conference on Uncertainty in Artificial Intelligence* (2023)

This paper explores the spectral bias in neural networks with discrete inputs, revealing a tendency to favor lower-degree frequencies in Fourier transforms and how it negatively affects generalization. To counter this, a new scalable regularization scheme is proposed, enhancing the network's ability to learn higher-degree frequencies and improve generalization, as evidenced by extensive evaluations of synthetic and real-world datasets. Chapter 3 of the thesis is based on ideas from this paper.

- Amrollahi, A., Gorji, A. & Krause, A. *Amortized SHAP values via function approximation* in (2023)

In this chapter, we leverage the low-degree and sparsity properties of neural networks and tree ensembles, for the efficient computation of SHAP values, crucial for interpretable AI. We propose a novel two-stage approach for estimating SHAP values in a model-agnostic, black-box setting, building on insights from the earlier paper on the spectral biases of neural networks and the nature of tree ensembles. Chapter 4 is based on ideas from this paper.

2

BACKGROUND

In this chapter, we first review Fourier a.k.a Walsh-Hadamard transforms, and notions of *degree* and *sparsity* in the Walsh-Hadamard domain [24]. Next, we review the notion of simplicity biases in neural networks and discuss why they are spectrally biased toward low-degree and sparse function functions. Finally, we do the same for ensembles of decision tree functions.

2.1 WALSH HADAMARD TRANSFORMS

Let $g : \{0,1\}^n \to \mathbb{R}$ be a function mapping Boolean zero-one vectors to the real numbers, also known as a "pseudo-Boolean" function. The family of 2^n functions $\{\Psi_f : \{0,1\}^n \to \mathbb{R} \,|\, f \in \{0,1\}^n\}$ defined below consists of the Fourier basis functions. This family forms a basis over the vector space of all pseudo-Boolean functions:

$$\Psi_f(x) = \frac{1}{\sqrt{2^n}}(-1)^{\langle f,x\rangle}, f, x \in \{0,1\}^n$$

where $\langle f, x\rangle = \sum_i f_i x_i$. Here, $f \in \{0,1\}^n$ is called the *frequency* of the basis function. For any frequency $f \in \{0,1\}^n$ we denote its *degree* by $\deg(f)$ which is defined as the number of non-zero elements. For example, $f_1 = [0,0,0,0,0]$ and $f_2 = [0,0,1,0,1]$ have degrees $\deg(f_1) = 0$ and $\deg(f_2) = 2$, respectively. One can think of the degree as a measure of the complexity of basis functions $\Psi_f(x)$. For example, $\Psi_0(x)$ is constant, and $\Psi_{e_i}(x)$ where e_i is a standard basis vector ($\deg(e_i) = 1$) only depends on feature i of the input. It is equal to $+1$ when feature i is zero and equal to -1 when feature i is one. More generally, a degree d basis function depends on exactly d input dimensions (features). These basis functions are orthonormal: $\sum_{x\in\{0,1\}^n} \Psi_f(x)\Psi_{f'}(x) = \begin{cases} 0 & f \neq f' \\ 1 & f = f' \end{cases}, \quad f, f' \in \{0,1\}^n$. Therefore, they form a basis for the vector space of all pseudo-Boolean functions $h : \{0,1\}^n \to \mathbb{R}$. Since the Fourier basis functions form a basis for the vector space of all

pseudo-Boolean functions, any function $g : \{0,1\}^n \to \mathbb{R}$ can be written as a unique linear combination of these basis functions:

$$g(x) = \frac{1}{\sqrt{2^n}} \sum_{f \in \{0,1\}^n} \widehat{g}(f)(-1)^{\langle f,x \rangle}$$

The (unique) coefficients $\widehat{g}(f)$ are called the "Fourier coefficients" or "Fourier amplitudes" and are computed as

$$\widehat{g}(f) = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} g(x)(-1)^{\langle f,x \rangle}$$

. The *Fourier spectrum* of g is the vector consisting of all of its 2^n Fourier coefficients, which we denote by the bold symbol $\widehat{\mathbf{g}} \in \mathbb{R}^{2^n}$.

We define the support of g to be $\text{supp}(g) = \{f \in \{0,1\}^n | \widehat{g}(f) \neq 0\}$. We say that a function g is *k-sparse* if at most k of the 2^n Fourier coefficients $\widehat{g}(f)$ are non-zero, i.e., $\|\widehat{\mathbf{g}}\|_0 = |\text{supp}(g)| \leq k$. We say a function is *degree d* when the frequencies $f \in \{0,1\}^n$ corresponding to non-zero Fourier coefficients are of degree less or equal to d i.e. $\forall f \in \text{supp}(g)$ it holds that $\deg(f) \leq d$.

By definition of the Fourier basis, a k-sparse degree d function can be written as a summation of k (Fourier basis) functions, each one depending on at most d input variables. The converse is also true:

Proposition 2.1.1. *Assume $g : \{0,1\}^n \to \mathbb{R}$ can be decomposed as follows: $g(x) = \sum_{i=1}^{p} h_i(x_{S_i}), S_i \subseteq [n]$. That is, each function $g_i : \{0,1\}^{|S_i|} \to \mathbb{R}$ depends on at most $|S_i|$ variables. Then, g is $k = O(\sum_{i=1}^{p} 2^{|S_i|})$-sparse and of degree $d = \max(|S_1|, \ldots, |S_p|)$. (Proof in Appendix A.0.1)*

The sparsity k and degree d capture a notion of *complexity* for the underlying function. Intuitively speaking, the sparsity factor k puts a limit on the number of functions in the decomposition, and the degree d puts a limit on the order of interactions among the input variables.

One can see from Proposition 2.1.1, that modular functions, i.e., functions that can be written as a sum of functions each depending on exactly one variable, are $k = O(n)$-sparse and of degree $d = 1$. A slightly more "complex" function capturing second-order interactions among the input variables, i.e., a function that can be written as a sum where each term depends on at most two variables is going to be $k = O(n^2)$-sparse and of degree $d = 2$. The following proposition generalizes this result.

Proposition 2.1.2. *Let, $g : \{0,1\}^n \to \mathbb{R}$ be pseudo-Boolean function and let $d \in \mathbb{N}$ be some constant (w.r.t. n). If g is of degree d, then, it is $k = O(n^d)$-sparse. (Proof in Appendix A.0.1)*

This proposition formally shows that limiting the order of interactions among the input variables implies an upper bound on the sparsity.

2.2 SPECTRAL BIAS THEORY OF NEURAL NETWORKS

The function that a ReLU neural network represents at initialization can be seen as a sample from a Gaussian Process [25] $N(0, K)$ in the infinite width limit [5, 26] (randomness is over the initialization of the weights and biases). The kernel K of the GP is called the "Conjugate Kernel" [4, 26] or the "nn-GP kernel" [4]. Let the kernel Gram matrix $\mathcal{K}$ be formed by evaluating the kernel on the Boolean cube i.e. $\{0,1\}^n$ and let $\mathcal{K}$ have the following spectral decomposition: $\mathcal{K} = \sum_{i=1}^{2^n} \lambda_i u_i u_i^\top$, where we assume that the eigenvalues $\lambda_1 \geq \cdots \geq \lambda_{2^n}$ are in decreasing order.

Let $u_f, f \in \{0,1\}^n$ be obtained by evaluating the Fourier basis function Ψ_f at the 2^n possible inputs on $\{0,1\}^n$. Yang & Salman [12] show that u_f is a eigenvector for $\mathcal{K}$. Moreover, they show (weak) spectral bias results in terms of the degree of f. Namely, the eigenvalues corresponding to higher degrees have smaller values [1]. The result is *weak* as they do not provide a *rate* as to which the eigenvalues decrease with increasing degrees.

Given the kernel, K, and viewing a randomly initialized neural network function evaluated on the Boolean $\{0,1\}^n$ as a sample from a GP one can see the following: This sample, roughly speaking, looks like a linear combination of the eigenvectors with the largest eigenvalues. This is because a sample of the GP can be obtained as $\sum_{i=1}^{2^n} \lambda_i \boldsymbol{w_i} u_i, \boldsymbol{w_i} \sim \mathcal{N}(0,1)$. Combining this with the spectral bias results implies that neural networks are low-degree functions when randomly initialized.

Going beyond neural networks at initialization, numerous studies have investigated the behavior of fully connected neural networks trained through (stochastic) gradient descent. These works show that in infinite-width neural networks weights after training via (stochastic) gradient descent do not end up too far from the initialization [7, 27–30], referred to as "lazy training" by

1 To be more precise, they show that the eigenvalues corresponding to even and odd degree frequencies form decreasing sequences. That is, even and odd degrees are considered separately.

Chizat, Oyallon & Bach [27]. Lee *et al.* [5, 6] show that training the last layer of a randomly initialized neural network via full batch gradient descent for an infinite amount of time corresponds to GP posterior inference with the kernel K. Lee *et al.* [6] and Jacot, Gabriel & Hongler [7] proved that when training *all* the layers of a neural network (not just the final layer), the evolution can be described by a kernel called the "Neural Tangent Kernel" and the trained network yields the mean prediction of GP $N(0, K_{NTK})$ [12] after an infinite amount of time. Lee *et al.* [6] empirically showed the results carry over to the finite-width setting through extensive experiments. Yang & Salman [31] again showed that the $u_f \in \mathbb{R}^{2^d}$ defined above are eigenvectors of the NTK Gram matrix and spectral bias holds. We [22] validated these theoretical findings through extensive experiments in finite-width neural networks by showing that neural networks have "less tendency" to learn high-degree frequencies.

The aforementioned literature shows that neural networks can be approximated by low-degree functions. By Proposition 2.1.2, they can be approximated by sparse functions for a large enough sparsity factor k. Our experiments in Section 3.5 provide further evidence that neural networks trained on real-world datasets are approximated well with sparse (and therefore compact) Fourier representations.

2.3 SPECTRAL BIAS OF ENSEMBLES OF DECISION TREES

In our context, a decision tree is a rooted binary tree, where each non-leaf node corresponds to one of n binary (zero-one) features, and each leaf node has a real number assigned to it. We denote the function that a decision tree represents by $t : \{0,1\}^n \to \mathbb{R}$. Let $i \in [n]$ denote the feature corresponding to the root, and let $t_{\text{left}} : \{0,1\}^{n-1} \to \mathbb{R}$ and $t_{\text{right}} : \{0,1\}^{n-1} \to \mathbb{R}$ be the left and right sub-trees, respectively. Then the tree can be represented as:

$$t(x) = \frac{1 + (-1)^{\langle e_i, x\rangle}}{2} t_{\text{left}}(x) + \frac{1 - (-1)^{\langle e_i, x\rangle}}{2} t_{\text{right}}(x) \tag{2.1}$$

Hereby, $t_{\text{left}} : \{0,1\}^{n-1} \to \mathbb{R}$ and $t_{\text{right}} : \{0,1\}^{n-1} \to \mathbb{R}$ are the left and right sub-trees respectively.

Thus, the Fourier transform of a decision tree can be computed recursively [13, 32]. The degree of a decision tree function of depth d is d, and if $|\text{supp}(t_{\text{left}})| = k_{\text{left}}$ and $|\text{supp}(t_{\text{right}})| = k_{\text{right}}$, then $|\text{supp}(t)| \leq 2(k_{\text{left}} + k_{\text{right}})$. As a result, a decision tree function is k-sparse, where $k = O(4^d)$, although in some cases, when the decision tree is not balanced

or cancellations occur, the Fourier transform can be sparser, i.e., admit a lower k, than the above upper bound on the sparsity suggests. This often occurs, e.g., in the case of classification where labels (leaf values) are zero and one. Nevertheless, in all cases $k = O(4^d)$, which is polynomial in the size of the tree, since a tree of depth d contains at most 2^d nodes.

Due to the linearity of the Fourier transform, the Fourier transform of an *ensemble of trees*, such as those produced by the random forest, cat-boost [33], and XGBoost [34] algorithms/libraries, can be computed by taking the average of the Fourier transform of each tree. If the random forest model has T trees, then its Fourier transform is $k = O(T4^d)$-sparse and of degree d equal to its maximum depth of the constituent trees.

3

LOW-DEGREE SPECTRAL BIAS OF NEURAL NETWORKS

As mentioned before, this thesis is focused on exploring spectral biases (in the Walsh-Hadamard sense) of two classes of important supervised machine learning models: neural networks and ensembles of trees. While the spectral bias of ensembles of trees is a fairly well-known fact [13, 14], the spectral bias of functions a neural network represents is a relatively new concept and has not been explored as much. This chapter exactly addresses this.

We empirically explore the spectral bias of fully connected neural networks. More precisely, we focus on the case of neural networks with discrete (zero-one) inputs through the lens of their Fourier (Walsh-Hadamard) transforms, where the notion of bias can be captured through the *degree* of the Fourier coefficients.

As mentioned earlier, the spectral bias may seem contradictory to the fact that deep and wide enough neural nets can learn arbitrary functions [35]. The key point here is that, since these models are trained through gradient descent, they exhibit this bias.

In this chapter we do the following:

- We empirically show that neural networks tend to learn lower-degree frequencies.
- We show how this spectral bias towards simpler features can in fact *hurt* the neural network's generalization on real-world datasets.
- To remedy this we propose a new scalable functional regularization scheme that aids the neural network to learn higher degree frequencies. Our regularizer also helps avoid erroneous identification of low-degree frequencies, which further improves generalization.
- We extensively evaluate our regularizer on synthetic datasets to gain insights into its behavior.
- We show significantly improved generalization on four different datasets compared to standard neural networks and other relevant baselines.

Point 1 of the above, namely our empirical results that show neural networks learn low-degree functions, consolidates previous theoretical results [31] with very extensive experiments. They also lay the foundation of our next chapter which is one example application of how one can utilize the spectral bias for the down-stream task related to interpretability.

3.1 RELATED WORK AND DETAILS OF OUR CONTRIBUTIONS

Classical work on neural networks shows that deep fully connected neural networks have the capacity to approximate arbitrary functions [1, 2]. However, in practice, neural networks trained through (stochastic) gradient descent have a "simplicity" bias. This notion of simplicity is not agreed upon and works such as [8–11] each introduce a different notion of "simplicity".

As mentioned in the background Chapter 2, the simplicity bias can also be studied by considering the function the neural net represents (function space view) and modeling it as Gaussian processes (GP)[25]. Lee *et al.* [5] and Daniely, Frostig & Singer [26] show that a wide, randomly initialized, neural network in function space is a sample from a GP with a kernel called the "Conjugate Kernel" [36]. Moreover, the evolution of gradient descent on a randomly initialized neural network can be described through the "Neural Tangent Kernel" Lee *et al.* [6] and Jacot, Gabriel & Hongler [7]. These works open up the road for analyzing the simplicity bias of neural nets in terms of a *spectral* bias in Fourier space. Rahaman *et al.* [37] show empirically that neural networks tend to learn sinusoids of lower frequencies earlier on in the training phase compared to those of higher frequencies. Through the GP perspective introduced by Lee *et al.* [6] and Jacot, Gabriel & Hongler [7], among others, Ronen *et al.* [38] and Basri *et al.* [39] were able to prove these empirical findings. These results focus on *continuous* domains and mainly emphasize the case where the input and output are both one-dimensional.

Here, we focus on *discrete* domains where the input is a high dimensional zero-one vector and we analyze the function learned by the neural network in terms of the amount of interactions among its inputs in a quantitative manner. Valle-Perez, Camargo & Louis [10] and Yang & Salman [12] provide spectral bias results for this set-

ting. By viewing a fully connected neural network as a function that maps zero-one vectors to real values, one can expand this function in terms of the Fourier –a.k.a Walsh-Hadamard – basis functions. The Walsh-Hadamard basis functions have a natural ordering in terms of their complexity called their *degree*. The degree specifies how many features each basis function is dependent upon. For example, the zero-degree basis function is the constant function and the degree-one basis functions are functions that depend on exactly one feature. Through analysis of the NTK gram matrix on the Boolean cube, Yang & Salman [12] theoretically show that, roughly speaking, neural networks learn the lower degree basis functions earlier in training.

This tendency to prioritize simpler functions in neural networks has been suggested as a cardinal reason for their remarkable generalization ability despite their over-parameterized nature [8, 11, 40, 41]. However, much less attention has been given to the case where the simplicity bias can *hurt* generalization [42, 43]. Tancik *et al.* [42] show how transforming the features with random Fourier features embedding helps the neural network overcome its spectral bias and achieve better performance in a variety of tasks. They were able to explain, in a unified way, many empirical findings in computer vision research such as sinusoidal positional embeddings through the lens of overcoming the spectral bias. In the same spirit as these works, we show that the spectral bias towards low-degree functions can hurt generalization and how to remedy this through our proposed regularizer.

In more recent lines of work, regularization schemes have been proposed to directly impose priors on the function the neural network represents [44–46]. This is in contrast to other methods such as dropout, batch normalization, or other methods that regularize the weight space. In this work, we also regularize neural networks in function space by imposing sparsity constraints on their Walsh-Hadamard transform. Closest to ours is the work of Aghazadeh *et al.* [47]. Inspired by studies showing that biological landscapes are sparse and contain high-degree frequencies [48–52], they propose a functional regularizer to enforce sparsity in the Fourier domain and report improvements in generalization scores.

Our contributions:

– We analyze the spectral behavior of a simple MLP during training through extensive experiments. We show that the standard (unregularized) network not only is unable to learn (more complex) high-

degree frequencies but it also starts learning erroneous low-degree frequencies and hence overfitting on this part of the spectrum.

- We propose a novel regularizer – HashWH (Hashed Walsh Hadamard) – to remedy the aforementioned phenomenon. The regularizer acts as a "sparsifier" on the Fourier (Walsh-Hadamard) basis. In the most extreme cases, it reduces to simply imposing an L_1-norm on the Fourier transform of the neural network. Since computing the exact Fourier transform of the neural net is intractable, our regularizer hashes the Fourier coefficients to buckets and imposes an L1 norm on the buckets. By controlling the number of hash buckets, it offers a smooth trade-off between computational complexity and the quality of regularization.
- We empirically show that HashWH aids the neural network in avoiding erroneous low-degree frequencies and also learning relevant high-degree frequencies. The regularizer guides the training procedure to allocate more energy to the high-frequency part of the spectrum when needed and allocate less energy to the lower frequencies when they are not present in the dataset.
- We show on real-world datasets that, contrary to popular belief of simplicity biases for neural networks, fitting a low degree function does not imply better generalization. Rather, what is more important, is keeping the *higher amplitude* coefficients regardless of their degree. We use our regularizer on four real-world datasets and provide state of the art results in terms of R^2 score compared to standard neural networks and other baseline ML models, especially for the low-data regime.

3.2 NOTATION

The *Fourier spectrum* of a pseudo-boolean function g (as defined in Chapter 2) is the vector consisting of all of its 2^n Fourier coefficients, which we denote by the bold symbol $\widehat{\mathbf{g}} \in \mathbb{R}^{2^n}$. Assume $\mathbf{X} \in \{0,1\}^{2^n \times n}$ to be the matrix of an enumeration over all possible n-dimensional binary sequences ($\{0,1\}^n$), and $\mathbf{g}(\mathbf{X}) \in \mathbb{R}^{2^n}$ to be the vector of g evaluated on the rows of $\mathbf{X}$. We can compute the Fourier spectrum using the Walsh-Hadamard transform as $\widehat{\mathbf{g}} = \frac{1}{\sqrt{2^n}}\mathbf{H}_n\mathbf{g}(\mathbf{X})$, where $\mathbf{H}_n \in \{\pm 1\}^{2^n \times 2^n}$ is the orthogonal Hadamard matrix (see Appendix B.1).

3.3 LOW-DEGREE SPECTRAL BIAS

In this section, we conduct experiments on synthetically generated datasets to show neural networks' spectral bias and their preference toward learning lower-degree functions over higher-degree ones. Firstly, we show that the neural network is not able to pick up the high-degree frequency components. Secondly, it can learn erroneous lower-degree frequency components. To address these issues, in Section 3.4, we introduce our regularization scheme called HASHWH (Hashed Walsh Hadamard) and demonstrate how it can remedy both problems.

3.3.1 *Fourier spectrum evolution*

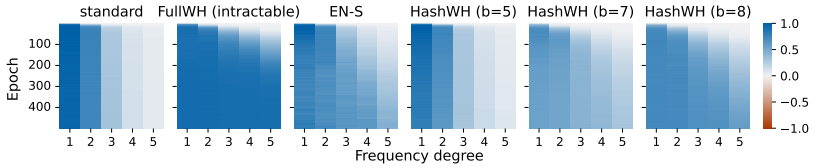
We analyze the evolution of the function learned by neural networks during training. We train a neural network on a dataset arising from a synthetically generated sparse function with a low-dimensional input domain. Since the input is low-dimensional it allows us to calculate the Fourier spectrum of the network (exactly) at the end of each epoch.

Setup. Let $g^* : \{0,1\}^{10} \to \mathbb{R}$ be a synthetic function with five frequencies in its support with degrees 1 to 5 i.e.

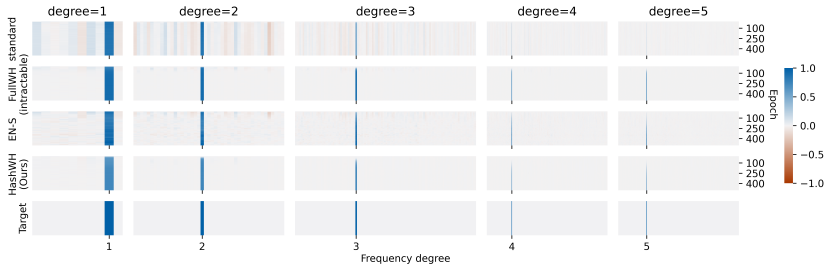
$$\text{supp}(g^*) = \{f_1, f_2, f_3, f_4, f_5\}, \deg(f_i) = i$$

, all having equal Fourier amplitudes of $\widehat{g^*}(f_i) = 1$. Each f_i is sampled uniformly at random from all possible frequencies of degree i. The training set is formed by drawing uniform samples from the Boolean cube $x \sim \mathcal{U}_{\{0,1\}^{10}}$ and evaluating $g^*(x)$.

We draw five such target functions g^* (with random support frequencies). For each draw of the target function, we create five different datasets all with 200 training points, and sampled uniformly from the input domain but with different random seeds. We then train a standard five-layer fully connected neural network using five different random seeds for the randomness in the training procedure (such as initialization weights and SGD). We aggregate the results over the 125 experiments by averaging. We experiment with the same setting with three other training set sizes. Results with training set size other than 200 and further setup details are reported in Appendices B.6.1 and B.4, respectively.



(a) Target support



(b) Whole Fourier spectrum

FIGURE 3.1: Evolution of the Fourier spectrum during training. STANDARD is the unregularized neural network. FULLWH imposes L_1-norm regularization on the exact Fourier spectrum and is intractable. EN-S alternates between computing a sparse Fourier approximation (computationally very expensive) and regularization. HASHWH (ours) imposes L_1 regularization on the hashed spectrum. Figure (a) is limited to the target support. The standard neural network is unable to learn higher degree frequencies. Our regularizer fixes this. Figure (b) is on the whole spectrum. The standard neural network picks up erroneous low-degree frequencies while not being able to learn the higher-degree frequencies. Our regularizer fixes both problems.

Results. We first inspect the evolution of the learned Fourier spectrum over different epochs and limited to the target support ($\text{supp}(g^*)$). Figure 3.1a shows the learned amplitudes for frequencies in the target support at each training epoch. Aligned with the literature on simplicity bias [10, 12], we observe that neural networks learn the low-degree frequencies earlier in the epochs. Moreover, we can see in the left-most figure in Figure 3.1a that despite eventually learning low-degree frequencies, the standard network is unable to learn high-degree frequencies.

Next, we expand the investigation to the whole Fourier spectrum instead of just focusing on the support frequencies. The first row of Figure 3.1b shows the evolution of the Fourier spectrum during training and compares it to the spectrum of the target function on the bottom row. We average the spectrum linked to one of the five target synthetic functions (over the randomness of the dataset sampling and training procedure) and report the other four in Appendix B.6.1. We observe that in addition to the network not being able to learn the high-degree frequencies, the standard network is prone to learning incorrect low-degree frequencies as well.

3.4 OVERCOMING THE SPECTRAL BIAS VIA REGULARIZATION

Now, we introduce our regularization scheme HASHWH (Hashed Walsh-Hadamard). Our regularizer is essentially a "sparsifier" in the Fourier domain. That is, it guides the neural network to have a sparse Fourier spectrum. We empirically show later how sparsifying the Fourier spectrum can both stop the network from learning erroneous low-degree frequencies and aid it in learning the higher-degree ones, hence remedying the two aforementioned problems.

Assume $\mathcal{L}_{net}$ is the loss function that a standard neural network minimizes, e.g., the MSE loss in the above case. We modify it by adding a regularization term $\lambda\mathcal{L}_{sparsity}$. Hence the total loss is given by: $\mathcal{L} = \mathcal{L}_{net} + \lambda\mathcal{L}_{sparsity}$.

The most intuitive choice is $\mathcal{L}_{sparsity} = \|\widehat{\mathbf{g_N}}\|_0$, where $\widehat{\mathbf{g_N}}$ is the Fourier spectrum of the neural network function $g_N : \{0,1\}^n \to \mathbb{R}$. Since the L_0-penalty's derivative is zero almost everywhere, one can use its tightest convex relaxation, the L_1-norm, which is also sparsity-inducing, as a surrogate loss. Aghazadeh *et al.* [47] use this idea and name it as Epistatic-Net or "EN" regularization: $\mathcal{L}_{EN} := \mathcal{L}_{net} + \lambda\|\widehat{\mathbf{g_N}}\|_1$. In this work, we call this regularization FULLWH (Full Walsh Hadamard transform).

FULLWH requires the evaluation of the network output on all 2^n possible inputs at each iteration of back-prop. Therefore, the computational complexity grows *exponentially* with the number of dimensions n, making it computationally intractable for $n > 20$ in all settings of practical importance.

Aghazadeh *et al.* [47] also suggest a more scalable version of FULLWH, called "EN-S", which roughly speaking, alternates between computing the sparse *approximate* Fourier transform of the network at the end of each epoch and doing normal back-prop, as opposed to the exact computation of the exact Fourier spectrum when back-propagating the gradients. In our experiments, we show EN-S can be computationally expensive because the sparse Fourier approximation primitive can be time-consuming. For a comprehensive comparison see Appendix B.2.3. Later, we show that empirically, it is also less effective in overcoming the spectral bias as measured by achievable final generalization error.

3.4.1 HASHWH

We avoid the exponentially complex burden of computing the exact Fourier spectrum of the network by employing a hashing technique to approximate the regularization term $\lambda\|\widehat{\mathbf{g_N}}\|_1$. Let $g : \{0,1\}^n \to \mathbb{R}$ be a pseudo-boolean function. We define the lower dimensional function $u_{\mathbf{\alpha}} : \{0,1\}^b \to \mathbb{R}$, where $b \ll n$, by sub-sampling g on its domain: $u_{\mathbf{\alpha}}(\tilde{x}) \triangleq \sqrt{\frac{2^n}{2^b}}\, g(\mathbf{\alpha}\tilde{x}),\ \tilde{x} \in \{0,1\}^b$ where $\mathbf{\alpha} \in \{0,1\}^{n\times b}$ is some matrix which we call the *hashing matrix*. The matrix-vector multiplication $\sigma\tilde{x}$is taken modulo 2. u_σ is defined by sub-sampling g on all the points lying on the (at most) b-dimensional subspace spanned by the columns of the hashing matrix σ. The special property of sub-sampling the input space from this subspace is in the arising Fourier transform of u_σ which we will explain next.

The Fourier transform of $u_{\mathbf{\alpha}}$ can be derived as (see Appendix B.2.1):

$$\widehat{u}_{\mathbf{\alpha}}(\tilde{f}) = \sum_{f\in\{0,1\}^n:\ \mathbf{\alpha}^\top f=\tilde{f}} \widehat{g}(f),\ \tilde{f} \in \{0,1\}^b \tag{3.1}$$

One can view $\widehat{u}_{\mathbf{\alpha}}(\tilde{f})$ as a "bucket" containing the sum of all Fourier coefficients $\widehat{g}(\tilde{f})$ that are "hashed" (mapped) into it by the linear hashing function $h(f) = \sigma^\top f$. There are 2^b such buckets and each bucket contains frequencies lying in the kernel (null space) of the hashing map plus some shift.

In practice, we let $\mathbf{\alpha} \sim \mathcal{U}_{\{0,1\}^{n\times b}}$ be a uniformly sampled hash matrix that is re-sampled after each iteration of back-prop. Let $\mathbf{X}_b \in \{0,1\}^{2^b\times b}$ be a matrix containing as rows the enumeration over all points on the

Boolean cube $\{0,1\}^b$. Our regularization term approximates (3.4) and is given by:

$$\mathcal{L}_{\text{HASHWH}} \triangleq \mathcal{L}_{net} + \lambda \|\mathbf{H}_b \mathbf{g}_{\mathbf{N}}(\mathbf{X}_b \boldsymbol{\text{œ}}^T)\|_1 = \mathcal{L}_{net} + \lambda \|\widehat{\mathbf{u}_{\boldsymbol{\text{œ}}}}\|_1$$

That is, instead of imposing the L_1-norm directly on the whole spectrum, this procedure imposes the norm on the "bucketed" (or partitioned) spectrum where each bucket (partition) contains sums of coefficients mapped to it. The larger b is the more partitions we have and the finer-grained the sparsity-inducing procedure is. Therefore, the quality of the approximation can be controlled by the choice of b. Larger b allows for a finer-grained regularization but, of course, comes at a higher computational cost because a Walsh-Hadamard transform is computed for a higher dimensional sub-sampled function u. Note that $b = n$ corresponds to hashing to 2^n buckets. As long as the hashing matrix is invertible, this precisely is the case of FULLWH regularization.

The problem with the above procedure arises when, for example, two "important" frequencies f_1 and f_2 are hashed into the same bucket, i.e., $\boldsymbol{\text{œ}}^\top f_1 = \boldsymbol{\text{œ}}^\top f_2$, an event which we call a "collision". This can be problematic when the absolute values $|\widehat{g}(f_1)|$ and $|\widehat{g}(f_2)|$ are large (hence they are important frequencies) but their sum can cancel out due to differing signs. In this case, the hashing procedure can zero out the sum of these coefficients. We can reduce the probability of a collision by increasing the number of buckets, i.e., increasing b [53].

In Appendix B.2.2 we show that the expected number of collisions C is given by: $\mathbb{E}[C] = \frac{(k-1)^2}{2^b}$ which decreases linearly with the number of buckets 2^b. Furthermore, we can upper bound the probability p that a given important frequency f_i collides with any other of the $k-1$ important frequencies in one round of hashing. Since we are independently sampling a new hashing matrix $\boldsymbol{\text{œ}}$ at each round of back-prop, the number of collisions of a given frequency over the different rounds has a binomial distribution. In Appendix B.2.2 we show that picking $b \geq \log_2(\frac{k-1}{\epsilon}), \epsilon > 0$ guarantees that collision of a given frequency happens approx. an ϵ-fraction of the T rounds, and not much more.

Fourier spectrum evolution of different regularization methods. We analyze the effect of regularizing the network with various Fourier sparsity regularizers in the setting of the previous section. Our regularizers of interest are FULLWH, EN-S with $m = 5$ (2^m is the number of buckets their sparse Fourier approximation algorithm hashes into), and HASHWH with $b \in \{5, 7, 8\}$.

Returning to Figure 3.1a, we see that despite the inability of the standard neural network in picking up the *high-degree* frequencies, all sparsity-inducing regularization methods display the capacity for learning them. FullWH is capable of perfectly learning the entire target support. It can also be seen that increasing the size of the hashing matrix in HashWH (ours) boosts the learning of high-degree frequencies. Furthermore, Figure 3.1b shows that in addition to the better performance of the sparsity-inducing methods in learning the target support, they are also better at filtering out non-relevant *low-degree* frequencies.

We define a notion of approximation error which is basically the normalized energy of the error in the learned Fourier spectrum on an arbitrary subset of frequencies.

Metric 3.4.1 (Spectral Approximation Error (SAE))**.** *Let $g_N : \{0,1\}^n \to \mathbb{R}$ be an approximation of the target function $g^* : \{0,1\}^n \to \mathbb{R}$. Consider a subset of frequencies $S \subseteq \{0,1\}^n$, and assume $\widehat{\mathbf{g_N}}_S$ and $\widehat{\mathbf{g}^*}_S$ to be the vector of Fourier coefficients of frequencies in S, for g_N and g^* respectively. As a measure of the distance between g_N and g on the subset of frequencies S, we define Spectral Approximation Error as:* $SAE = \frac{\|\widehat{\mathbf{g_N}}_S - \widehat{\mathbf{g}^*}_S\|_2^2}{\|\widehat{\mathbf{g}^*}_S\|_2^2}$

Figure 3.2 shows the SAE of the trained network using different regularization methods over epochs, for both when S is target support as well as when $S = \{0,1\}^n$ (whole Fourier spectrum). The standard network displays a significantly higher (worse) SAE on the whole Fourier spectrum compared to the target support, while Walsh-Hadamard regularizers exhibit consistent performance across both. This shows the importance of enforcing the neural network to have zero Fourier coefficients on the non-target frequencies. Moreover, we can see HashWH (ours) leads to a reduction in SAE that can be smoothly controlled by the size of its hashing matrix.

To gain more insight, we split the frequencies into subsets S consisting of frequencies with the same degree. We visualize the evolution of SAE and also the Fourier energy of the network defined as $\|\widehat{\mathbf{g_N}}_S\|_2^2$ in Figure 3.3. Firstly, the energy of high-degree frequencies is essentially zero for the standard neural network when compared to the low-degree frequencies, which further substantiates the claim that standard neural network training does not learn any high-degree frequencies. We can see that our HashWH regularization scheme helps the neural network learn higher degree frequencies as there is more

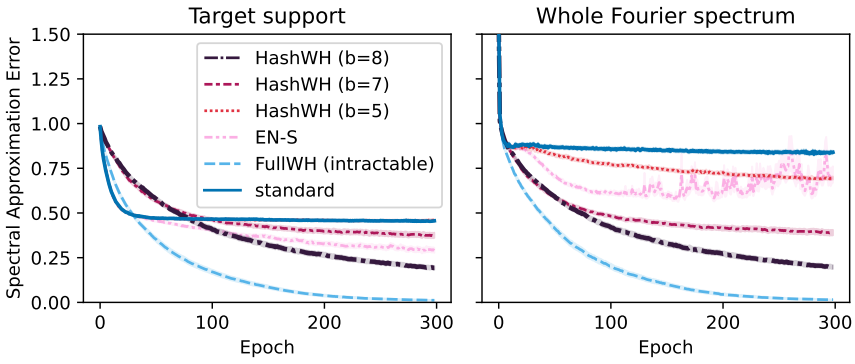


FIGURE 3.2: Evolution of the spectral approximation error (SAE) during training. The left plot limits the error to the target support, while the right one considers the whole Fourier spectrum. For the standard neural network, the SAE is considerably worse on the full spectrum which shows the importance of eliminating the erroneous frequencies that are not in the support of the target function. We also see the graceful scaling of SAE of HASHWH (ours) with the hashing matrix size.

energy in the high degree components. Secondly, looking at the lower degrees 2 and 3 we can see that the standard neural network reduces the SAE up to some point but then starts overfitting. Looking at the energy plot one can attribute the overfitting to picking up irrelevant degree 2 and 3 frequencies. We see that the regularization scheme helps prevent the neural net from overfitting on the low-degree frequencies and their SAE reduces roughly monotonously. We observe that HASHWH (ours) with a big enough hashing matrix size exhibits the best performance among tractable methods in terms of SAE on all degrees. Finally, we can see HASHWH is distributing the energy to where it should be for this dataset: less in the low-degree and more in the high-degree frequencies.

Finally, it is worth noting that our regularizer makes the neural network behave more like a *decision tree*. It is well known that ensembles of decision tree models have a sparse and low-degree Fourier transform [14]. Namely, let $g : \{0,1\}^n \to \mathbb{R}$ be a function that can be represented as an ensemble of T trees each of depth at most d. Then g is $k = O(T \cdot 4^d)$-sparse and of degree at most d. Importantly, their spectrum is *exactly sparse* and unlike standard neural networks, which seem to

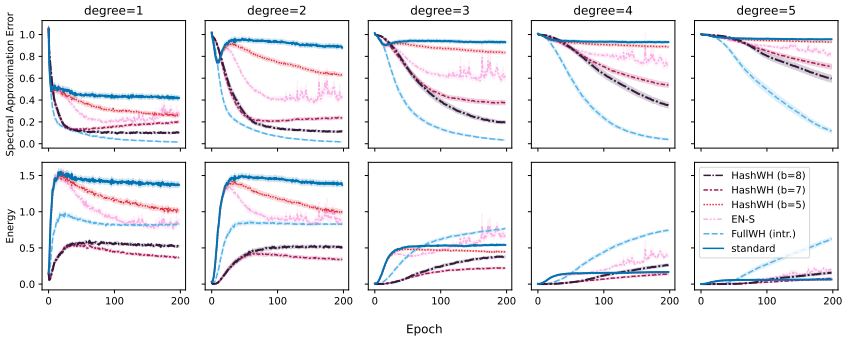


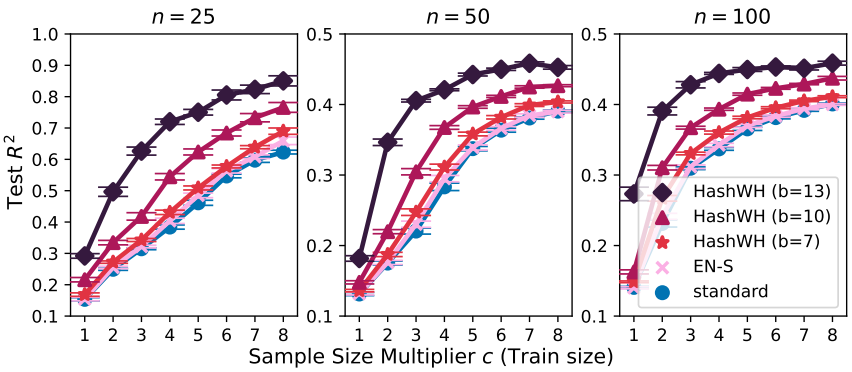
FIGURE 3.3: Evolution of the Spectral Approximation Error (SAE) and energy of the network during training, split by frequency degree. Firstly, in a standard neural network, the energy of high-degree frequencies is essentially zero compared to low-degree frequencies. Secondly, for low degrees (2 and 3) the energy continues to increase while the SAE exhibits overfitting behavior. This implies the neural network starts learning erroneous low-degree frequencies after some epochs. Our regularizer prevents overfitting in lower degrees and enforces higher energy on higher-degree frequencies. Regularized networks show lower energies for lower degrees and higher energy for higher degrees when compared to the standard neural network.

"fill up" the spectrum on the low-degree end, i.e., learn irrelevant low-degree coefficients, decision trees avoid this. Decision trees are well-known to be effective on discrete/tabular data [54], and our regularizer prunes the spectrum of the neural network so it behaves similarly.

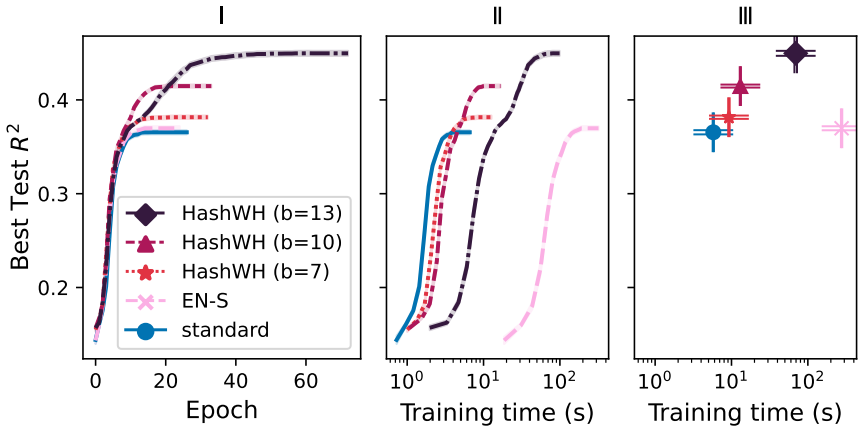
3.5 EXPERIMENTS

In this section, we first evaluate our regularization method on higher dimensional input spaces (larger n) on synthetically generated datasets. In this setting, FULLWH is not applicable due to its exponential runtime in n. In addition, we allow varying training set sizes to showcase the efficacy of the regularizer in improving generalization at varying levels in terms of the number of training points in the dataset and especially in the low-data sample regime. Next, we move on to four real-world datasets. We first show the efficacy of our proposed regularizer HASHWH on real-world datasets in terms of achieving

better generalization errors, especially in the low-data sample regimes. Finally, using an ablation study, we experimentally confirm that the low-degree bias does not result in lower generalization error.



(a) Generalization comparison



(b) Runtime comparison

FIGURE 3.4: (a) Generalization performance on learning a synthetic function $g^* : \{0,1\}^n \to \mathbb{R}$ with train set size: $c \cdot 25n$ (b) Best achievable test R^2 (I) at end of each epoch (II) up to a certain time (seconds). (III) Shows the early stopped R^2 score vs. time (seconds). We provide significant improvements across all training sizes over EN-S and standard neural networks, while also showing an order of magnitude speed-up compared to EN-S.

3.5.1 *Synthetic data*

Setup. Again, we consider a synthetic pseudo-boolean target function $g^* : \{0,1\}^n \to \mathbb{R}$, which has 25 frequencies in its support $|\text{supp}(g^*)| = 25$, with the degree of maximum five, i.e., $\forall f \in \text{supp}(g^*) : \deg(f) \leq 5$. To draw a g^*, we sample each of its support frequencies f_i by first uniformly sampling its degree $d \sim \mathcal{U}_{\{1,2,3,4,5\}}$, based on which we then sample $f_i \sim \{f \in \{0,1\}^n | \deg(f) = d\}$ and its corresponding amplitude uniformly $\widehat{g^*}(f_i) \sim \mathcal{U}_{[-1,1]}$.

We draw g^* as above for different input dimensions $n \in \{25, 50, 100\}$. We pick points uniformly at random from the input domain $\{0,1\}^n$ and evaluate g^* to generate datasets of various sizes: we generate five independently sampled datasets of size $c \cdot 25n$, for different multipliers $c \in \{1, .., 8\}$ (40 datasets for each g^*). We train a 5-layer fully-connected neural network on each dataset using five different random seeds to account for the randomness in the training procedure. Therefore, for each g^* and dataset size, we train and average over 25 models to capture variance arising from the dataset generation, and also the training procedure.

Results. Figure 3.4a shows the generalization performance of different methods in terms of their R^2 score on a hold-out dataset (details of dataset splits in Appendix B.4) for different dataset sizes. Our regularization method, HashWH, outperforms the standard network and EN-S in all possible combinations of input dimension, and dataset size. Here, EN-S does not show any significant improvements over the standard neural network, while HashWH (ours) improves generalization by a large margin. Moreover, its performance is tunable via the hashing matrix size b.

To stress the computational scalability of HashWH (ours), Figure 3.4b shows the achievable R^2-score by the number of training epochs and training time for different methods, when $n = 50$ and $c = 5$ (see Appendix B.6.2 for other settings). The trade-off between the training time and generalization can be directly controlled with the choice of the hashing size b. More importantly, comparing HashWH with EN-S, we see that for any given R^2 we have runtimes that are orders of magnitude smaller. This is primarily due to the very time-consuming approximation of the Fourier transform of the network at each epoch in EN-S.

3.5.2 *Real data*

Next, we assess the performance of our regularization method on four different real-world datasets of varying nature and dimensionality. For baselines, we include not only standard neural networks and EN-S regularization, but also other popular machine learning methods that work well on discrete data, such as ensembles of trees. Three of our datasets are related to protein landscapes [52, 55, 56] which are identical to the ones used by the proposers of EN-S [47], and one is a GPU-tuning [57] dataset. See Appendix B.3 for dataset details.

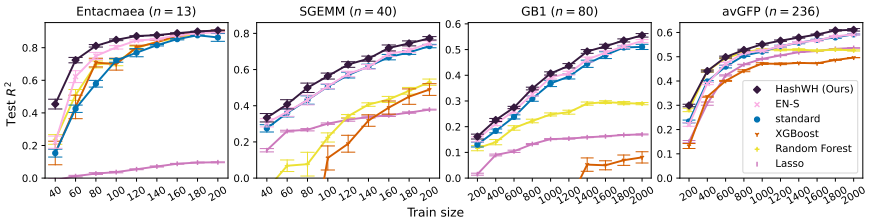
Results. Figure 3.5a displays the generalization performance of different models in learning the four datasets mentioned, using training sets of small sizes. For each given dataset size we randomly sample the original dataset with five different random seeds to account for the randomness of the dataset sub-sampling. Next, we fit five models with different random seeds to account for the randomness of the training procedure. One standard deviation error bars and averages are plotted accordingly over the 25 runs. It can be seen that our regularization method significantly outperforms the standard neural network as well as popular baseline methods on nearly all datasets and dataset sizes. The margin, however, is somewhat smaller than on the synthetic experiments in some cases. This may be partially explained by the distribution of energy in a real dataset (Figure 3.5c), compared to the uniform distribution of energy over different degrees in our synthetic setting.

To highlight the importance of higher degree frequencies, we compute the exact Fourier spectrum of the Entacmaea dataset (which is possible, since all possible input combinations are evaluated in the dataset). Figure 3.5c shows the energy of 100 frequencies with the highest amplitude (out of 8192 total frequencies) categorized into varying degrees. This shows that the energy of the higher degree frequencies 3 and 4 is comparable to frequencies of degree 1. However, as we showed in the previous section, the standard neural network may not be able to pick up the higher degree frequencies due to its simplicity bias (while also learning erroneous low-degree frequencies).

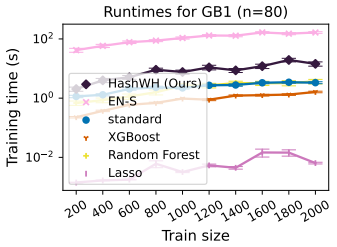
We also study the relationship between the low-degree spectral bias and generalization in Figure 3.5d. The study is conducted on the two datasets "Entacmaea" and "SGEMM". We first fit a sparse Fourier function to our training data (see Appendix B.5). We then start deleting coefficients once according to their degree (highest to lowest and

ties are broken randomly) and in another setting according to their amplitude (lowest to highest). To assess generalization, we evaluate the R^2 of the resulting function on a hold-out (test) dataset. This study shows that among functions of equal complexity (in terms of size of support), functions that keep the higher amplitude frequencies as opposed to ones that keep the low-degree ones exhibit better generalization. This might seem evident according to Parseval's identity, which states that time energy and Fourier energy of a function are equal. However, because the dataset distribution is not necessarily uniform, there is no reason for this to hold in practice. Furthermore, it shows the importance of our regularization scheme: deviating from low-degree functions and instead aiding the neural network to learn higher amplitude coefficients *regardless* of the degree.

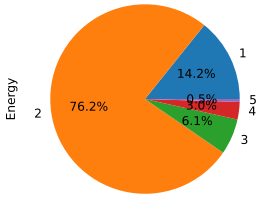
Conclusion We showed through extensive experiments how neural networks tend to not learn high-degree frequencies and overfit in the low-degree part of the spectrum. We proposed a computationally efficient regularizer that aids the network in not overfitting in the low-degree frequencies and also picking up the high-degree frequencies. Finally, we exhibited significant improvements in terms of R^2 score on four real-world datasets compared to various popular models in the low-data regime.



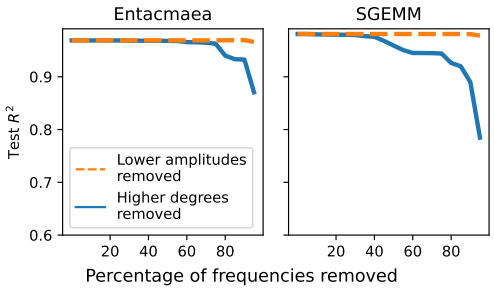
(a) Performance on learning real datasets



(b) Runtimes for GB1



(c) Energy distribution over degrees in Entacmaea



(d) Ablation studies

FIGURE 3.5: (a) Generalization performance of standard and regularized neural networks and benchmark ML models on four real datasets. (b) Training times of different models on the GB1 dataset (c) Results of an ablation study on the potential effect of simplicity bias in the generalization error. This figure shows picking higher amplitude coefficients results in better generalization compared to picking the lower degree terms (d) Distribution of the energy over degree-based sets of frequencies in Entacmaea's top 100 Fourier coefficients. This shows high-degree components constitute a non-negligible portion of the energy of the function.

4

UTILIZING THE SPECTRAL BIAS TO COMPUTE SHAP VALUES

In the previous Chapter 3 we saw how the function represented by a fully connected neural network, after being trained by gradient descent, is low-degree and therefore also sparse. We also saw in the Background Chapter 2 how ensembles of trees are also both sparse and low-degree.

As opposed to the previous chapter where we "fixed" this bias for neural networks to allow for better generalization, here we use it to our advantage. We discuss an application of how we can utilize the bias for a very important downstream task of computing SHAP values – a.k.a. SHapley Additive exPlanations.

SHAP values are a popular local feature-attribution method widely used in interpretable and explainable AI. Efficiently computing them is challenging, especially in the model-agnostic (black-box) setting, where one only has query access to the model. This subsumes neural networks as a special case as well.

Building on the previous two chapters we propose a two-stage approach for estimating SHAP values for the black-box setting:

- Our algorithm's first step harnesses results from the previous two chapters that showed that many real-world black-box predictors have a spectral bias that allows us to either *exactly represent* (in the case of ensembles of decision trees), or *efficiently approximate* them (in the case of neural networks) using a compact Fourier representation. Exploiting this insight, given query access to a black-box function, we utilize a sparse Fourier approximation algorithm to efficiently extract its compact Fourier approximation.
- In the second step, we use the Fourier representation to *exactly* compute SHAP values. The second step is computationally very cheap because firstly, the representation is compact and secondly, we prove that there exists a closed-form expression for SHAP values for the Fourier basis functions.

- The expression we derive effectively "linearizes" the computation into a simple summation and is amenable to parallelization on multiple cores or a GPU.
- Since the function approximation (first step) is only done once, it allows us to produce Shapley values in an *amortized* way. This makes our algorithm orders of magnitudes faster than previous methods such as KernelShap where each explained instance requires solving an expensive optimization problem.
- We show speedups of 10-10000x compared to relevant baseline methods for equal levels of accuracy.

4.1 RELATED WORK AND DETAILS OF OUR CONTRIBUTIONS

Interpretability of machine learning models is paramount, especially in high-stakes applications in areas such as medicine, fraud detection, or credit scoring. This is crucial to the extent that in Europe, the General Data Protection Regulation (GDPR) mandates the *legal* right to an explanation of algorithmic decisions [58]. Say we are given a predictor/model $h : \mathcal{X}^n \to \mathbb{R}$ which maps an input (data) instance $x^* \in \mathcal{X}^n$ to a prediction $h(x^*)$. *Instance-wise* a.k.a. *local* feature attribution methods assign "importances" to each of the features $x_i^* \in \mathcal{X}$ of the instance x^* which quantify how influential that feature was in the model predicting the value $h(x^*)$.

A widely used method for deriving attributions (importances) is the notion of *SHapley Additive exPlanations*, commonly referred to simply as SHAP values. Originally, the notion of Shapley values was introduced in the seminal work of Shapley [59] in the context of cooperative game theory. The Shapley value is a mathematically well-founded and "fair" way of distributing a reward among all the members of a group playing a cooperative game and it is computed based on the rewards that would be received for all possible coalitions. The Shapley value is the unique way of distributing the reward that satisfies several reasonable mathematical properties that capture a notion of fairness [59]. In the context of machine learning and statistics, the players become features, the reward is the prediction of the predictor h and the SHAP value is the "contribution" or "influence" of that feature on the prediction. Shapley values are widely used due to their mathematical soundness and desirable properties [60–66] [67].

Despite their prevalence, the computation of SHAP values is challenging, since it involves an exponential sum (considering the importance of a feature in the context of all possible "coalitions" of other features). Therefore, approximating them and speeding up the computation has received attention in a variety of settings. SHAP value computation can easily dominate the computation time of industry-level machine learning solutions on datasets with millions or more entries [68]. Yang [68] point out that industrial applications sometimes require hundreds of millions of samples to be explained. Examples include feed ranking, ads targeting, and subscription propensity models. In these modeling pipelines, spending tens of hours in model interpretation becomes a significant bottleneck [68] and one usually needs to resort to multiple cores and parallel computing.

Significant work has gone into speeding up the computation of SHAP values for a variety of settings. In the *white-box* setting, full access to the model is assumed. For the white-box tree-model setting, Yang [68] and Bifet, Read, Xu, *et al.* [69] provide theoretical and practical computational speedups to the classic TreeShap [66]. For the case of neural network white-box setting DeepLift Shrikumar, Greenside & Kundaje [70] provides a way of approximating SHAP values by assuming we have access to the activation of the neurons. FastShap Jethani *et al.* [71], introduces a method for estimating Shapley values in a single forward pass using an end-to-end learned explainer model for the task of classification (not regression).

As opposed to the white-box setting, in the *model-agnostic* a.k.a *black-box* setting, we only have *query access* to the model. Here, our only means of access to the predictor is that we can pick an arbitrary $x \in \mathcal{X}^n$ and query the predictor for its value $h(x)$. The usual approach here is to approximate the exponential sum of the SHAP value computations using stochastic sampling [65, 72, 73]. In this setting, Covert & Lee [72] and Mitchell *et al.* [73] provide sampling methods that require fewer queries to the black-box compared to vanilla KernelShap [65] for equal approximation accuracy. Our algorithm FourierShap falls into the query-access black-box setting.

However, we take a different approach. We are guided by the key insight that many models used in practice have a "spectral bias". Valle-Perez, Camargo & Louis [10] and Yang & Salman [31] provably and experimentally show that fully connected neural networks with binary (zero-one) inputs learn low-degree – and therefore sparse –

functions in a basis called the *Walsh-Hadamard a.k.a Fourier* basis. It is well known that the Walsh-Hadamard transform (WHT) of an ensemble of T trees of depth d is also of degree at most d and moreover, $k = O(T4^d)$-sparse [13, 32].

Our contributions: Guided by the aforementioned insights, we provide an algorithm to compute SHAP values in the model-agnostic a.k.a black-box setting, using the Fourier representation of the model. We first approximate the black-box function by taking its sparse Fourier transform. We theoretically justify, and show through extensive experiments, that for many real-world models such as fully connected neural networks and (ensembles of) trees this representation is accurate. Subsequently, we prove that SHAP values for a single Fourier basis function admit a closed-form expression not involving an exponential summation. Therefore, using the Fourier representation we overcome the exponential sum and can utilize compute power effectively to compute SHAP values. Furthermore, the closed-form expression we derive effectively "linearizes" the computation into a simple summation and is amenable to parallelization on multiple cores or a GPU. The Fourier approximation step is only done *once*, therefore FourierShap amortizes the cost of computing explanations for many inputs. Subsequently, SHAP computations using the Fourier approximation are extremely fast compared to other black-box approximation methods such as KernelShap and other variations of it which all involve a computationally expensive optimization. We show speedups of 10-10000x compared to KernelShap [65] and LinRegShap [72]. We also show a 10-100x speedup over DeepLift [70], a white-box algorithm for neural networks, even though we only assume query access (black-box setting).

4.2 BACKGROUND

This section reviews the notion of SHAP values, and sparse and low-degree Fourier transforms.

4.2.1 *Shapley values*

In game theory, a *cooperative game* is a function $v : 2^n \to \mathbb{R}$ that maps a subset (coalition) $S \subseteq N$ of a group of players $N = \{1, \ldots, n\}$ to their total reward (when they cooperate). If all the players cooperate they win a total reward of value $v(N)$ and the main question is how they would distribute this reward among themselves. Shapley [59] resolved

this problem by a deriving a *unique* value based on "fairness axioms" proposed in his seminal work [59]. The *Shapley value* of player $i \in N$ is:

$$\phi_i(v) = \frac{1}{n} \sum_{S \subseteq N \setminus \{i\}} \frac{v(S \cup \{i\}) - v(S)}{\binom{n-1}{|S|}} \tag{4.1}$$

Intuitively, one can view the term $v(S \cup \{i\}) - v(S)$ as the marginal contribution of player i when they are added to the coalition S. This marginal value is weighted according to the number of permutations the leading $|S|$ players and trailing $n - |S| - 1$ players can form.

In the machine learning context, we have a predictor $h : \mathcal{X}^n \to \mathbb{R}$ mapping an n-dimensional feature vector to a value. In this context, the players become features $x_i \in \mathcal{X}$ and the reward is the prediction of the predictor h and the Shapley value is the "contribution" or "influence" of the i'th feature on the prediction. We define $v(S)$ accordingly to capture this notion [65]:

$$v(S) = \mathbb{E}_{\boldsymbol{x}_{N \setminus S} \sim p(\boldsymbol{x}_{N \setminus S}))}[h(x^*_S, \boldsymbol{x}_{N \setminus S})],$$

where $x^* \in \mathcal{X}^n$ is the instance we are explaining. This definition implicitly captures the way we handle the missing features (feature not present in the coalition): we integrate the missing features concerning the marginal distribution $p(\boldsymbol{x}_{N \setminus S}))$. In practice, the marginalization is performed with an empirical distribution by taking a subset of the training data as *background dataset*.

The choice of which distribution to average the missing features from has been discussed thoroughly in the relevant literature. As mentioned before, in this work, we focus on the SHAP values as defined in KernelShap introduced by Lundberg & Lee [65] and Lundberg *et al.* [66] also know as "Interventional" [74, 75] or "Baseline" [76] SHAP, where the missing features are integrated out from the *marginal distribution* $p(\boldsymbol{x}_{N \setminus S})$, as opposed to the *conditional distribution* $p(\boldsymbol{x}_{N \setminus S} | \boldsymbol{x}_S)$. In the next section we provide a comprehensive overview of the literature discussing these two notions and their conceptual differences.

4.2.2 *Shapley values in the context of Machine learning*

In the context of ML, many works have derived a different notion of Shapley value depending on what they mean by data distribution,

deleted features, etc. We refer the reader to the survey by Janzing, Minorics & Bloebaum [74] and Sundararajan & Najmi [76] for a comprehensive overview. In this work we focus on the notion of SHAP introduced by Lundberg & Lee [65] and Lundberg *et al.* [66] also known as "Interventional" [74, 75] or "Baseline" SHAP [76] where the missing features are integrated out from the marginal distribution as opposed to the conditional distribution see Section 4.2.1. As pointed out by Janzing, Minorics & Bloebaum [74] there are two main ways to define the SHAP value "interventional" and "observational" SHAP. These are referred to by Sundararajan & Najmi [76] as "baseline" and "conditional" SHAP respectively.

As pointed out by Janzing, Minorics & Bloebaum [74] the difference between these definitions can be better viewed with the lens of causality [77]. Roughly speaking "observational" SHAP tells us about how influential a feature is to the prediction of the predictor if it goes from the state of being unobserved to observed. "Interventional" SHAP is *causal* and tells us how influential a feature is if we were to reach in (through a process called an intervention) and change that feature in order to change the prediction. Put into the context of credit scores and loan approvals, "observational" SHAP will provide us with important features which are "observed" by the predictor and hence are influential in predicting if a particular loan request will be rejected or approved. Interventional SHAP would tell us which feature we could change or "intervene" in order to change the outcome of the loan request.

The original (ML) SHAP paper [65] proposes "observational" SHAP as the correct notion of SHAP. Van den Broeck *et al.* [75] and Arenas *et al.* [78] provide intractability results for observational SHAP in a variety of simple distributional assumptions on the data and simple predictors f. This has lead to many attempts to *approximate* observational SHAP values [66, 67, 72, 76, 79]. It is interesting to note that the version of "Kernel"-SHAP in Lundberg & Lee [65] is also an approximation for observational SHAP values that ends up coinciding precisely with interventional SHAP values, which explains a lot of the confusion in the community. Janzing, Minorics & Bloebaum [74] boldly claims that researchers should stop the pursuit of approximations to "observation" SHAP values as it lacks certain properties, for example, sensitivity i.e. the SHAP value of a feature can be non-zero while the predictor f has no dependence on that feature. This phenomenon happens because when features are correlated, the presence

of a feature can provide information about other features that the predictor does depend on. This does *not* happen in interventional SHAP. Finally, Chen *et al.* [80] argues that both SHAP definitions are worthy of pursuit. They emphasize that the interventional framework provides explanations that are more "true to the data", and the observational approach's explanations are more "true to the model".

4.2.3 KernelShap

Since we will be using the well-known KernelShap [65] and its variant LinRegShap [72] as a baseline we briefly review their method here. Lundberg & Lee [65] propose the "least squares characterization" of SHAP values. They prove that SHAP values are the solution to the following minimization problem:

$$\beta_0^*, \dots, \beta_n^* \triangleq \arg\min_{\beta_0,\dots,\beta_n} \sum_{0<|S|<n} \frac{n-1}{\binom{n}{|S|}|S|(n-|S|)} \left(\beta_0 + \sum_{i\in S}\beta_i - v(S)\right)$$
$$s.t.: \quad \beta_0 = v(\{\}), \beta_0 + \sum_{i=1}^{n}\beta_i = v(N)$$

Then $\phi_i(v) = \beta_i^*$.

However, the above optimization still involves an exponential sum. Therefore, Lundberg & Lee [65] propose to sample subsets S uniformly at random. Covert & Lee [72] and Mitchell *et al.* [73] provide better ways of sampling and solving the optimization to get approximations with lower variances and biases. Nevertheless, all these methods require solving a least squares minimization subject to constraints for each explained instance x^* and, therefore, are computationally expensive.

4.2.4 *Efficient SHAP values in the context of coalitional games*

Finally, we would like to note that the connection between SHAP values and Walsh-Hadamard transforms was initially made by Wendler [81] in the context of a coalitional game. More concretely let $v : 2^n \to \mathbb{R}$ as in Section 4.2.1. Moreover, assume this function is k-sparse in the Fourier basis. Then, Wendler [81] shows how one can compute the

SHAP values as defined in Equation (4.1), with a summation over k summands. Wendler [81] overcomes the exponential sum in (4.1) by expressing the function v in the k-sparse Fourier basis. However, no formal proof is provided as to why the exponential summation admits a closed form. Moreover, the results are not extended to the case of the SHAP values in the ML context as described in Section 4.2.1.

Our algorithm is inspired by his idea. We provide combinatorial proofs for why a summation similar to this work admits a closed form. Moreover, we explain this in the context of Interventional SHAP values in Machine learning to make the results comparable with other relevant baselines such as KernelSHAP.

4.2.5 *Many real-world black-box predictors have sparse Fourier transforms*

In the Background Chapter 2, we discussed the sparsity of the Fourier transforms of ensembles of trees and why neural networks can be approximated by a sparse Fourier representation because of their spectral bias. This shows both these classes of functions can be compactly represented in the Fourier basis. The results here will become useful in the next section, where we present our main contribution on how we can leverage this compact representation, to precisely compute SHAP values cheaply.

4.3 COMPUTING SHAP VALUES WITH FOURIER REPRESENTATIONS OF FUNCTIONS

In the previous section, we saw that many real-world models trained on tabular/discrete data can be exactly represented (in the case of ensembles of decision trees), or efficiently approximated (in the case of neural networks) using a compact (sparse) Fourier representation. We saw neural networks have a tendency to learn approximately low-degree, and hence by Proposition 2.1.2 sparse, functions. This has been attributed in numerous works to the reason why they generalize well and do not overfit despite their over-parameterized nature [8, 10, 11, 31, 40, 82, 83]. We also saw that (ensembles) of decision trees, by nature, have sparse Fourier representations [13, 32]. More generally, as made formal in Proposition 2.1.1 and the remarks after, any "simple" function that can be written as a summation of a "few" functions each

depending on a "few" of the input variables is sparse and low-degree in the Fourier domain.

We propose the following method to approximate SHAP values for black-box functions. In the first step, given query access to a black-box function, we utilize a sparse Fourier approximation algorithm such as [84–86] to efficiently extract its sparse and hence compactly represented Fourier approximation. See Appendix C.1.1 for a more detailed explanation. Next, in our second step presented here, we use the Fourier representation to exactly compute SHAP values.

Let $h : \{0,1\}^n \to \mathbb{R}$ be some predictor which we assume to be k-sparse with n binary input features. Let $N = \{1, \ldots, n\}$ be the set of features and let $x^* \in \{0,1\}^n$ be the instance we are explaining. As outlined in Equation (4.1), SHAP values are obtained from the following equation:

$$\phi_i^h = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(n-|S|-1)!}{M!}(v_h(S \cup \{i\}) - v_h(S)), \quad \forall i \in [n]$$

where $v_h(S)$ is the average prediction when one only knows x_S^*. The average is taken over the (background) dataset $\mathcal{D} = \{(x,y)^i\}$ [65]. More precisely:

$$v_h(S) \triangleq \frac{1}{|\mathcal{D}|} \sum_{(x,y) \in \mathcal{D}} h(x_S^*, x_{N \setminus S})$$

The above equations show the SHAP values are linear with respect to the prediction function h. Therefore we proceed by computing the Shapley values for a single Fourier basis function $\Psi_f(x) = (-1)^{\langle f, x \rangle}, f \in \{0,1\}^n$.

$$\begin{aligned} \phi_i^{\Psi_f} = \frac{1}{|\mathcal{D}|} \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(n-|S|-1)!}{n!} \cdot \\ \sum_{(x,y) \in \mathcal{D}} \left((-1)^{\langle f, x^*_{S \cup \{i\}} \oplus x_{N \setminus S \cup \{i\}} \rangle} - (-1)^{\langle f, x^*_S \oplus x_{N \setminus S} \rangle} \right) \end{aligned} \tag{4.2}$$

The $\oplus$ operator concatenates two vectors along the same axis.

This expression still has an exponential (in n) sum, since we are summing over all subsets S. As a main contribution, we find a closed-form analytic expression for the inner summation using a combinatorial argument. This results in the following key Lemma:

Lemma 4.3.1. *Let $\Psi_f(x) = (-1)^{\langle f,x \rangle}$ be the Fourier basis function for some $f \in \{0,1\}^n$. Then the SHAP value of the Fourier basis function Ψ_f with respect to the background dataset $\mathcal{D}$ is given as:*

$$\phi_i^{\Psi_f} = \frac{2f_i}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} \mathbb{1}_{x_i \neq x_i^*} (-1)^{\langle f,x \rangle} \frac{(|A|+1) \mod 2}{|A|+1} \tag{4.3}$$

where $A \triangleq \{j \in N | x_j \neq x_j^*, j \neq i, f_j = 1\}$. (Proof in Appendix C.2.1)

Finally, by the linearity of SHAP values w.r.t. the explained function h, and by utilizing Lemma 4.3.1 we arrive at the final expression for SHAP values of h. We present this closed-form expression alongside its computational complexity in our main Theorem:

Theorem 4.3.2. *Let $h : \{0,1\}^n \to \mathbb{R}$ be a k-sparse pseudo-boolean function with Fourier frequencies $f^1, \ldots, f^k \in supp(h)$ and amplitudes $\widehat{h}(f), \forall f \in supp(f)$. Let $\mathcal{D}$ be a background dataset of size $|\mathcal{D}|$. Then, Equation (4.4) provides a precise expression for the SHAP value vector $\phi^h = (\phi_1^h, \ldots, \phi_n^h)$. One can compute this vector with $\Theta(n \cdot |\mathcal{D}| \cdot k)$ flops (floating point operations).*

$$\phi_i^h = \frac{2}{|\mathcal{D}|} \sum_{f\in supp(h)} \widehat{h}(f) \cdot f_i \sum_{(x,y)\in\mathcal{D}} \mathbb{1}_{x_i \neq x_i^*} (-1)^{\langle f,x \rangle} \frac{(|A|+1) \mod 2}{|A|+1} \tag{4.4}$$

where A is the same as in Lemma 4.3.1. (Proof in Appendix C.2.2)

Theorem 4.3.2 gives us a computationally efficient way to go from a Fourier representation of a function h to SHAP values. The SHAP values from this equation are *exact*, i.e., as long as the Fourier representation is exact, the SHAP values are also precise values. This is in contrast to KernelSHAP, where the SHAP values are approximated using stochastic sampling and one needs to check for convergence to make sure the approximation is accurate. The approximation in our method is constrained to the first step: computing the (approximate) sparse Fourier representation of the black-box.

Most importantly, the sum in Equation 4.4 is *tractable*. This is because we overcome the exponential sum involved in Equation (4.1) by analytically computing the sum, with a combinatorial argument, for a single Fourier basis function. The theorem shows the number of flops that are required to compute SHAP values in Theorem 4.3.2 to be asymptotically equal to $\Theta(n \cdot |\mathcal{D}| \cdot k)$. We note that the $|\mathcal{D}|$ and k factors in the asymptotic computational complexity arise from the two summations present in Equation 4.4. Through this expression, we are

able to "linearize" the computation of SHAP values to a summation over the Fourier coefficients and background dataset. This allows us to maximally utilize the parallelization on multiple cores and/or GPUs to speed up the computation significantly. Therefore, in the presence of multiple cores or a GPU, we can get a speedup equal to the level of parallelization, as each core or worker can compute one part of this summation.

We implement our algorithm called FourierShap using JAX [87], which allows for fast vectorized operations on GPUs. Each term inside the summations of Equation (4.4) can be implemented with simple vector operations. Furthermore, summations over the k different frequencies in the support of h and also background data points can both be efficiently implemented and parallelized using this library using its vmap operator. We perform all upcoming experiments on a single GPU. Nevertheless, we believe faster computation can also be achieved by crafting dedicated code designed to efficiently compute Equation 4.4 on GPU.

Finally, we note that the function approximation (first step) is only done once, i.e., we compute the sparse Fourier approximation of the black-box only once. This is typically the most expensive part of the computation. For any new query to be explained, we resort to an efficient implementation of Equation (4.4). As our experiments will show, this yields orders of magnitudes faster computation than previous methods such as KernelShap where, as mentioned before in Section 4.2, each explained instance requires solving an expensive optimization problem.

4.4 EXPERIMENTS

We assess the performance of our algorithm, FourierSHAP, on four different real-world datasets of varying nature and dimensionality. Three of our datasets are related to protein fitness landscapes [52, 55, 56] and are referred to as "Entacmaea", "GB1", and "avGFP" respectively. The fourth dataset is a GPU-tuning [57] dataset referred to as "SGEMM". The features of these datasets are binary (zero-one) and/or categorical with standard one-hot encodings. See Appendix C.3 for dataset details.

For the Entacmaea and SGEMM datasets, we train fully connected neural networks with 3 hidden layers containing 300 neurons each.

For GB1 we train ensembles of trees models of varying depths using the random forest algorithm and for avGFP we train again, ensembles of trees models of varying depths using the cat-boost algorithm/library[33].

Black-box setting. The first step of the FourierSHAP algorithm is computing a sparse Fourier approximation of the black-box model. We use a GPU implementation of a sparse Walsh-Hadamard Transform (sparse WHT) a.k.a Fourier transform algorithm [85] for each of the four trained models. The algorithm accepts a sparsity parameter k which is the sparsity of the computed Fourier representation. Higher sparsity parameter k results in a better function approximation but the sparse-WHT runtime increase linearly in k as well. In Figure 4.1 we plot the accuracy of the Fourier function approximation as measured by the R^2-score for different values of k (which result in different runtimes). The R^2 score is computed over a dataset formed by randomly sampling the Boolean cube $\{0,1\}^n$.

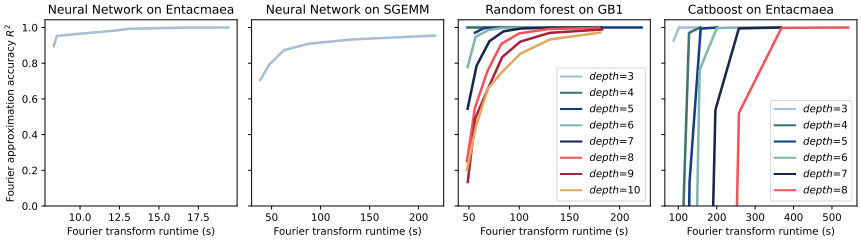


FIGURE 4.1: Step 1 of FourierShap: Accuracy of the Fourier transform (in terms of approximating the black-box function) vs. runtime of the sparse Fourier algorithm. The accuracy is evaluated by R^2 score and comparing the outputs of the black-box and the Fourier representation on a uniformly generated random dataset on the Boolean cube $\{0,1\}^n$. For a fixed level of accuracy, higher depth trees require a higher number of Fourier coefficients k therefore a higher runtime. For the case of trees, we eventually are able to reach a perfect approximation since the underlying function is truly sparse.

The second step of FourierSHAP utilizes the Fourier approximation to compute SHAP values using Equation (4.4). We implement this step on a GPU using the JAX [87] library. For each model to be explained, we choose four different values for the number of background samples and four different values for the number of query points

to be explained, resulting in a total of 16 runs of FourierSHAP for each model. Error bars capture these variations. We take the values produced by KernelSHAP to be the ground truth. Therefore, we compute the R^2 values of Shapley values computed by FourierSHAP (ours) method vs KernelSHAP as a measure of accuracy. For our method, a higher sparsity k for the Fourier representation results in a more accurate function approximation therefore higher R^2 values for the SHAP value quality. On the other hand, a higher k results in a slower runtime as Equation (4.4) is a sum over the k different frequencies. In Figure 4.2 we plot this trade-off.

We compare against the following baselines in Figure 4.2. The first is LinRegShap, a variance-reduced version of KernelSHAP [72]. We found that although this algorithm requires fewer queries from the black-box, it takes orders of magnitudes longer to run compared to ours. Secondly, for the neural network models, we also compare against a state-of-the-art *white-box* method – DeepLift [70]. This algorithm, requires access to the neural network's activations in all layers. In comparison, we achieve a 10-100x speedup while being both more accurate and only assuming query access to the neural net (true black-box setting).

White-box setting – ensemble of trees. While our main focus is on the black-box setting, FourierSHAP can also be utilized for the computation of SHAP values for (ensembles of) trees models in the white-box setting, where full access to the tree's structure is available. In this setting, the exact sparse Fourier representation of an ensemble of trees can be efficiently computed (the first step of FourierSHAP) using Equation 2.1. With the exact Fourier representation at hand, SHAP values can be *efficiently and exactly* computed using Equation (4.4), the second step of FourierSHAP.

We compute SHAP values for random forests fitted on the Entacmaea dataset. We compare the runtime of our algorithm, FourierSHAP, to TreeSHAP [66], which is the most commonly employed algorithm for the exact computation of SHAP values for tree-based models, and FastTreeSHAP [68], a fast implementation of TreeSHAP. To the best of our knowledge, these are the fastest available frameworks for computation of the "interventional" SHAP values. Table 4.1 illustrates the performance improvement achieved by our algorithm on the Entacmaea dataset. Note that the SHAP values computed by all methods are precise and identical to the values produced by TreeSHAP which

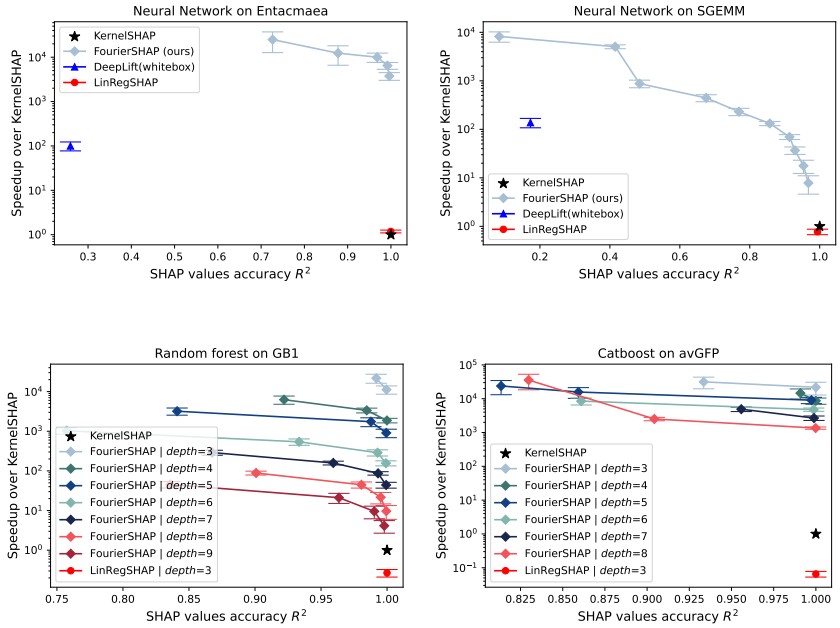


FIGURE 4.2: Speedup vs. Accuracy. Speedup of different algorithms is reported as a multiple compared to the runtime of KERNELSHAP. Accuracy is quantified by the R^2-score using KERNELSHAP SHAP values as ground truth. DEEPLIFT is a white-box algorithm for neural networks. LINREGSHAP is black-box algorithm and a variance-reduced version of KERNELSHAP. FOURIERSHAP is ours. We are 10-10000x faster than KERNELSHAP and LINREGSHAP on all dataset/model variations. More notably, we outperform DeepLift (a white-box algorithm) in the neural network model setting even though we assume only query access (black-box setting) to the neural network.

is to be expected since we are using the exact Fourier representation of random forests for FOURIERSHAP. While the speedup decreases for trees with larger depth, FOURIERSHAP offers a consistent improvement over the state-of-the-art with no loss of accuracy even in the white-box setting.

Maximum Depth	Model accuracy R^2	FourierSHAP Speedup	FastTreeSHAP Speedup
3	0.80	21.03 ± 0.55	0.96 ± 0.06
4	0.86	13.82 ± 0.24	1.01 ± 0.02
5	0.91	10.44 ± 0.22	1.04 ± 0.02
6	0.94	5.81 ± 0.14	1.03 ± 0.01
7	0.96	2.71 ± 0.04	1.03 ± 0.00
8	0.97	1.94 ± 0.02	1.02 ± 0.00

TABLE 4.1: Speedup of FourierSHAP and FastTreeSHAP over TreeSHAP on random forests of multiple depths in the *white-box* setting. We measure the speedup for five independent runs of each model and report the standard deviation as the error. We achieve a significant speedup over both in all settings.

CONCLUSIONS

We illustrated in theory and practice how many black-box functions can be represented or efficiently approximated by a compact Fourier representation. We proved that SHAP values of Fourier basis functions admit a closed form expression, and therefore, we can compute SHAP values efficiently using the compact Fourier representation. Moreover, this closed form expression is amenable to parallelization. These two factors helped us in gaining speedups of 10-10000x over baseline methods for the computation of SHAP values.

5

COMPUTING THE SPARSE AND/OR LOW-DEGREE SPECTRUM

We discussed in Chapter 3 how neural networks have a low-degree and therefore sparse Fourier transform. In Chapter 4 we discussed how one can utilize this compact representation to compute SHAP values faster. However, we did not discuss an algorithm to extract this sparse and low-degree Fourier representation. This chapter addresses this missing link.

In this chapter, we provide an algorithm that can utilize the sparsity and/or low degree to efficiently extract Fourier transforms. The algorithm works on any black-box function $g : \{0,1\}^n \to \mathbb{R}$ with query access. Therefore we frame this chapter in the more general framework of learning set functions that are sparse and low-degree. The reader can keep in mind that a neural network with zero-one inputs represents a set function that has these two desirable properties.

In the more general case, learning set functions arise in other applications ranging from sketching graphs to black-box optimization with discrete parameters. We consider the problem of efficiently learning set functions that are defined over a ground set of size n and that are sparse (say k-sparse) in the Fourier domain. This is a wide class, that includes graph and hypergraph cut functions, decision trees, neural networks, and more.

Our central contribution is the first algorithm that allows learning functions whose Fourier support only contains low degree (say degree $d = o(n)$) polynomials using $O(kd \log n)$ sample complexity and runtime $O(kn \log^2 k \log n \log d)$. This implies that sparse graphs with k edges can, for the first time, be learned from $O(k \log n)$ observations of cut values and in linear time in the number of vertices. Our algorithm can also efficiently learn (sums of) decision trees of small depth.

The algorithm exploits techniques from the sparse Fourier transform literature and is easily implementable. Lastly, we also develop an efficient robust version of our algorithm and prove ℓ_2/ℓ_2 approximation guarantees without any statistical assumptions on the noise.

5.1 RELATED WORK AND OUR DETAILS OF OUR CONTRIBUTIONS

Consider a set function $g : \{0,1\}^n \to \mathbb{R}$. Set functions that arise in applications often exhibit structure, which can be effectively captured in the Fourier (also called Walsh-Hadamard) basis. One commonly studied structure for set functions is *Fourier sparsity* [88]. A k-Fourier-sparse set function contains no more than k non-zero Fourier coefficients. A natural example for k-Fourier-sparse set functions are cut functions of graphs with k edges or evaluations of a decision tree of depth d [13, 20, 32, 89]. The cut function of a graph only contains polynomials of degree at most two in the Fourier basis and in the general case, the cut function of a hypergraph of degree d only contains polynomials of degree at most d in the Fourier basis [89]. Intuitively this means that these set functions can be written as sums of terms where each term depends on at most d elements in the ground set. Also, a decision tree of depth d only contains polynomials of degree at most d in the Fourier basis [32] [13]. Learning such functions has recently found applications in neural network hyper-parameter optimization [90]. Therefore, the family of Fourier sparse set functions whose Fourier support only contains low-degree terms is a natural and important class of functions to consider.

RELATED WORK One approach for learning Fourier sparse functions uses Compressive Sensing (CS) methods [89]. Suppose we know that the Fourier transform of our function $\widehat{g}$ is k-sparse i.e. $|\text{supp}(\widehat{g})| \leq k$, and $\text{supp}(\widehat{g}) \subseteq P$ for some known set P of size p. In [89] it is shown that recovery of $\widehat{g}$ is possible (with high probability) by observing the value of x on $O(k \log^4 p)$ subsets chosen independently and uniformly at random. They utilize results from [91, 92] which prove that picking $O(k \log^4 p)$ rows of the Walsh-Hadamard matrix independently and uniformly at random results in a matrix satisfying the RIP which is required for recovery. For the case of graphs $p = \binom{n}{2} = O(n^2)$ and one can essentially learn the underlying graph with $O(k \log^4 n)$ samples. In fact, this result can be further improved, and $O(k \log^2 k \log n)$ samples suffice [93]. Computationally, for the CS approach, one may use matching pursuit which takes $\Omega(kp)$ time and thus results in a runtime of $\Omega(kn^d)$ for k Fourier sparse functions of order d. This equals $\Omega(kn^2)$ for graphs, where $d = 2$. In [19, 89], proximal methods are used to optimize the Lagrangian form of

the ℓ_1 norm minimization problem. Optimization is performed on p variables which results in $\Omega(n^2)$ runtime for graphs and $\Omega(n^d)$ time for the general order d sparse recovery case. Hence, these algorithms scale *exponentially* with d and have *at least quadratic dependence on n* even in the simple case of learning graph cut functions.

There is another line of work on this problem in the sparse Fourier transform literature. [16] provides a non-robust version of the sparse Walsh Hadamard Transform (WHT). This algorithm makes restrictive assumptions on the signal, namely that the k non-zero Fourier coefficients are chosen uniformly at random from the Fourier domain. This is a strong assumption that does not hold for the case of cut functions or decision trees. This work is extended in [93] to a robust sparse WHT called SPRIGHT. In addition to the random uniform support assumption, [93] further presumes that the Fourier coefficients are finite-valued and the noise is Gaussian. Furthermore, all existing sparse WHT algorithms are unable to exploit low-degree Fourier structures.

OUR RESULTS We build on techniques from the sparse Fourier transform literature [88, 94, 95] and develop an algorithm to compute the Walsh-Hadamard transform (WHT) of a k-Fourier-sparse function whose Fourier support is constrained to low degree frequencies (low degree polynomials). For recovering frequencies with low degree, we utilize ideas that are related to compressive sensing over finite fields [96]. We show that if the frequencies present in the support of $\widehat{g}$ are of low-degree then there exists an algorithm that computes WHT in $O(kn \log^2 k \log n \log d)$ time using $O(kd \log n)$ samples. As opposed to [16], we avoid distributional assumptions on the support using hashing schemes. Our approach is the first one to achieve the sampling complexity of $O(kd \log n)$. Moreover, its running time scales *linearly* in n and there is no exponential dependence on d. For the important special case of graphs, where $d = 2$, our sampling complexity is near optimally $O(k \log n)$ and our runtime is $O(kn \log^2 k \log n)$ which is strictly better than CS methods which take at least quadratic time in n. This allows us to learn sparse graphs which have in the range of 800 vertices in ≈ 2 seconds whereas the previous methods [89] were constrained to the range of 100 for similar runtimes.

For the case where $\widehat{g}$ is not exactly k-sparse, we provide novel robust algorithms that recover the k dominant Fourier coefficients with provable ℓ_2/ℓ_2 approximation guarantees. We provide a robust algorithm

using appropriate hashing schemes and novel analysis. We further develop a robust recovery algorithm that uses $O(kd \log n \log(d \log n))$ samples and runs in time:

$$O\left(nk \log^3 k + nk \log^2 k \log n \log(d \log n) \log d\right).$$

COMPARISON TO OTHER RESULTS As discussed before, there are two primary methodologies address the problem at hand: Compressive Sensing and L_1 minimization and Sparse Fast Fourier Transform (FFT).

- **Compressive Sensing and L_1 minimization**: Given that the Fourier basis is orthogonal, it is feasible to construct a measurement matrix using Fourier basis functions as columns. By selecting specific rows from this matrix at random we preserve RIP properties, and therefore we can recover the sparse Fourier representation.
- **Sparse FFT**: The methodology underpinning Sparse FFT involves hashing Fourier coefficients into bins and subsequently retrieving those coefficients isolated within individual bins. Our study primarily adopts this approach.

The Sparse FFT method's salient advantage is its logarithmic runtime in the number of basis functions. In contrast, the L_1 minimization approach sees a linear runtime scaling. For instance, to capture order d interactions, $O(\binom{n}{d})$ basis functions would be required, which implies that the runtime would scale exponentially with d.

As for sampling (query) complexities, the gold standard is provided by Compressive Sensing methods. Sparse FFT, however, does not capitalize on the low-degree assumptions at all.

The Compressive Sensing method doesn't presuppose anything about the Fourier spectrum's support, whereas Sparse FFT operates under the assumption that Fourier frequencies are randomly and independently chosen.

Our methods retains the Sparse FFT algorithm's computational efficiency, which scales logarithmically in number of basis functions.

Our primary contribution lies in enhancing the sampling complexity, making it comparable to Compressive Sensing methods. This enhancement is realized through a novel frequency detection primitive, which

fully utilizes the low-degree assumption. This method incorporates concepts from Compressive Sensing over finite fields, thereby amalgamating the strengths of both techniques in terms of runtime and sampling complexities.

Furthermore, we've mitigated randomness assumptions by integrating novel hashing schemes.

	Compressive sensing	**Sparse WH**	**Ours**
Runtime	$\tilde{O}(kn^d)$	$\tilde{O}(kn^2)$	$\tilde{O}(kn)$
Sampling complexity	$\tilde{O}(kd)$	$\tilde{O}(kn)$	$\tilde{O}(kd)$
Assumptions	None	Randomness of support	None

5.2 PROBLEM STATEMENT

Here we define the problem of learning set functions. Consider a set function that maps subsets of a ground set $V \triangleq \{1, \ldots, n\} = [n]$ of size n to real numbers, $g : 2^V \to \mathbb{R}$. We assume oracle access to this function, that is, we can observe the function value $g(A)$ for any subset A that we desire. The goal is to learn the function, that is to be able to evaluate it for all subsets $B \subseteq V$. A problem that has received considerable interest is learning *cut functions* of sparse (in terms of edges) graphs [89]. Given a weighted undirected graph $G = (V, E, w)$, the cut function associated to G is defined as $g(A) = \sum_{s \in A, t \in V \setminus A} w(s, t)$, for every $A \subseteq V$.

Note that we can equivalently represent each subset $A \subseteq V$ by a vector $x \in \mathbb{F}_2^n$ which is the indicator of set A. Here $\mathbb{F}_2 = \{0, 1\}$ denotes the finite field with 2 elements. Hence the set function can be viewed as $g : \mathbb{F}_2^n \to \mathbb{R}$. We denote the Walsh-Hadamard transform of $g : \mathbb{F}_2^n \to \mathbb{R}$ by $\widehat{g} : \mathbb{F}_2^n \to \mathbb{R}$. It is defined as:

$$\widehat{g}(f) = \frac{1}{\sqrt{N}} \sum_{x \in \mathbb{F}_2^n} g(x) \cdot (-1)^{\langle f, x \rangle} \quad , f \in \mathbb{F}_2^n.$$

The inner product $\langle f, x \rangle$ throughout the thesis is performed modulo 2.

The Fourier transform of the graph cut function $\widehat{g}$ is the following,

$$\widehat{g}(f) = \begin{cases} \frac{1}{2}\sum_{s,t\in V} w(s,t) & \text{if } f = (0,\ldots,0) \\ -w(s,t)/2 & \text{if } f_s = f_t = 1 \text{ and } f_i = 0\ \forall i \neq s,t \\ 0 & \text{otherwise} \end{cases}.$$

It is clear that the Fourier support of the cut function for graph G contains only $|E|+1$ nonzero elements (and hence it is *sparse*). Furthermore, the nonzero Fourier coefficients correspond to frequencies with hamming weights at most 2.

One of the classes of set functions that we consider is that of **exactly low-degree Fourier sparse** functions. Under this model, we address the following problem:

Input: oracle access to $g : \mathbb{F}_2^n \to \mathbb{R}$
such that $\|\widehat{g}\|_0 \leq k$ and $|f| \leq d$ for all $f \in \text{support}(\widehat{g})$
Output: nonzero coefficients of $\widehat{g}$ and their corresponding frequencies (5.1)

where $|f|$ denotes the *Hamming weight* of f.

We also consider the **robust** version of problem (5.1) where we only have access to noisy measurements of the input set function. We make no assumption about the noise, which can be chosen adversarially. Equivalently one can think of a general set function whose spectrum is well approximated by a low-degree sparse function which we refer to as *head*. *Head* of $\widehat{g}$ is just the top k Fourier coefficients $\widehat{g}(f)$ such that the frequency has low Hamming weight $|f| \leq d$. We refer to the noise spectrum as *tail*.

Definition 5.2.1 (Head and Tail norm)**.** *For all integers n, d, and k we define the* head *of $\widehat{g} : \mathbb{F}_2^n \to \mathbb{R}$ as,*

$$\widehat{g}_{head} := arg \min_{\substack{y:\mathbb{F}_2^n\to\mathbb{R} \\ \|y\|_0\leq k \\ |j|\leq d \text{ for all } j\in supp(y)}} \|\widehat{g} - y\|_2.$$

The tail norm *of $\widehat{g}$ is defined as, $Err(\widehat{g}, k, d) := \|\widehat{g} - \widehat{g}_{head}\|_2^2$.*

Since the set function to be learned is only *approximately* in the low-degree Fourier sparse model, it makes sense to consider the *approxi-*

mate version of problem (5.1). We use the well known ℓ_2/ℓ_2 *approximation* to formally define the **robust** version of problem (5.1) as follows,

$$\begin{aligned} \textbf{Input:} \quad & \text{oracle access to } g : \mathbb{F}_2^n \to \mathbb{R} \\ \textbf{Output:} \quad & \text{function } \widehat{\chi} : \mathbb{F}_2^n \to \mathbb{R} \\ & \text{such that } \|\widehat{\chi} - \widehat{g}\|_2^2 \leq (1+\delta)\mathrm{Err}(\widehat{g}, k, d), \\ & |f| \leq d \text{ for all } f \in \mathrm{support}(\widehat{\chi}) \end{aligned} \tag{5.2}$$

Note that no assumptions are made about the function g and it can be any general set function.

5.3 ALGORITHM AND ANALYSIS

In this section, we present our algorithm and analysis. We use techniques from the sparse FFT literature [88, 94, 95]. Our main technical novelty is a new primitive for estimating a low-degree frequency, i.e., $|f| \leq d$, efficiently using an optimal number of samples $O(d \log n)$ given in Section 5.3.1. This primitive relies heavily on the fact that a low-degree frequency is constrained on a subset of size $\binom{n}{d}$ as opposed to the whole universe of size 2^n. We show that problem (5.1) can be solved quickly and using a few samples from the function x by proving the following theorem,

Theorem 5.3.1. *For any integers n, k, and d, the procedure* ExactSHT *solves problem* (5.1) *with probability* $9/10$. *Moreover, the runtime of this algorithm is* $O\left(kn \log^2 k \log n \log d\right)$ *and the sample complexity of this procedure is* $O\left(kd \log n\right)$.

5.3.1 *Low-degree frequency recovery*

In this section we provide a novel method for recovering a frequency $f \in \mathbb{F}_2^n$ with bounded Hamming weight $|f| \leq d$, from measurements $\langle m_i, f \rangle$ $i \in [s]$ for some $s = O(d \log n)$. The goal of this section is to design a measurement matrix $M \in \mathbb{F}_2^{s \times n}$ with small s, such that for any $f \in \mathbb{F}_2^n$ with $|f| \leq d$ the following system of constraints, with constant probability, has a unique solution $j = f$ and has an efficient solver,

$$j \in \mathbb{F}_2^n \text{ such that } \begin{cases} Mj = Mf \\ |j| \leq d \end{cases}.$$

To design an efficient solver for the above problem with optimal s, we first need an optimal algorithm for recovering frequencies with weight one $|f| \leq 1$. In this case, we can locate the index of the nonzero coordinate of f optimally via binary search using $O(\log n)$ measurements and runtime.

Definition 5.3.2 (Binary search vectors)**.** *For any integer n, the ensemble of vectors $\{v^l\}_{l=0}^{\lceil \log_2 n \rceil} \subseteq \mathbb{F}_2^n$ corresponding to binary search on n elements is defined as follows. Let $v^0 = \{1\}^n$ (the all ones vector). For every $l \in \{1, \cdots, \lceil \log_2 n \rceil\}$ and every $j \in [n]$, $v^l_j = \left\lfloor \frac{(j \mod 2^l)}{2^{l-1}} \right\rfloor$.*

Lemma 5.3.3. *There exists a set of measurements $\{m_i\}_{i=1}^s$ for $s = \lceil \log_2 n \rceil + 1$ together with an algorithm such that for every $f \in \mathbb{F}_2^n$ with $|f| \leq 1$ the algorithm can recover f from the measurements $\langle f, m_i \rangle$ in time $O(\log_2 n)$.*

To recover a frequency f with Hamming weight d, we hash the coordinates of f randomly into $O(d)$ buckets. In expectation, a constant fraction of nonzero elements of f get isolated in buckets, and hence the problem reduces to the weight one recovery. We know how to solve this using binary search as shown in Lemma 5.3.3 in time $O(\log n)$ and with sample complexity $O(\log n)$. We recover a constant fraction of the nonzero indices of f and then we subtract those from f and recurse on the residual. The pseudocode of the recovery procedure is presented in Algorithm 1.

Lemma 5.3.4. *For any integers n and d , any power of two integer $D \geq 128d$, and any frequency $f \in \mathbb{F}_2^n$ with $|f| \leq d$, the procedure* RecoverFrequency *given in Algorithm 1 outputs f with probability at least $7/8$, if we have access to the following,*

1. *For every $r = 0, 1, \cdots, \log_4 D$, a hash function $h_r : [n] \to [D/2^r]$ which is an instance from a pairwise independent hash family.*
2. *For every $l = 0, 1, \cdots, \lceil \log_2 n \rceil$ and every $r = 0, 1, \cdots, \log_4 D$, the measurements $\phi^l_r(i)$ that are equal to $\phi^l_r(i) = \sum_{j \in h_r^{-1}(i)} f_j \cdot v^l_j$ for every $i \in [D/2^r]$.*

Moreover, the runtime of this procedure is $O(D \log D \log n)$ and the number of measurements is $O(D \log n)$.

Proof. The proof is by induction on the iteration number $r = 0, 1, \cdots, T$. We denote by $\mathcal{E}_r$ the event $|f - \tilde{f}^{(r)}| \leq \frac{d}{4^r}$, that is the sparsity goes down by a factor of 4 in every iteration up to r^{th} iteration. The inductive hypothesis is $\Pr[\mathcal{E}_{r+1} | \mathcal{E}_r] \geq 1 - \frac{1}{16 \cdot 2^r}$.

Algorithm 1 RecoverFrequency

input: power of two integer D, hash functions $h_r : [n] \to [D/2^r]$ for every $r \in \{0, 1, \cdots, \log_4 D\}$, measurement vectors $\phi_r^l \in \mathbb{F}_2^{D/2^r}$ for every $l = 0, 1, \cdots \lceil \log_2 n \rceil$ and every $r = 0, 1, \cdots, \log_4 D$.
output: recovered frequency $\tilde{f}$.

$\{v^l\}_{l=0}^{\lceil \log_2 n \rceil} \leftarrow$ binary search vectors on n elements (Definition 5.3.2), $T \leftarrow \log_4 D$, $\tilde{f}^{(0)} \leftarrow \{0\}^n$
for $r = 0$ to T **do**
  $w \leftarrow \{0\}^n$.
  for $i = 1$ to $D/2^r$ **do**
    if $\phi_r^0(i) - \sum_{j \in h_r^{-1}(i)} \tilde{f}_j^{(r)} \cdot v_j^0 = 1$ **then**
      $index \leftarrow \{0\}^{\lceil \log_2 n \rceil}$, a $\lceil \log_2 n \rceil$ bits pointer.
      for $l = 1$ to $\lceil \log_2 n \rceil$ **do**
        if $\phi_r^l(i) - \sum_{j \in h_r^{-1}(i)} \tilde{f}_j^{(r)} \cdot v_j^l = 1$ **then**
          $[index]_l \leftarrow 1$, set l^{th} bit of $index$ to 1.
        end if
      end for
      $w(index) \leftarrow 1$, set the coordinate of w positioned at $index$ to 1.
    end if
  end for
  $\tilde{f}^{(r+1)} \leftarrow \tilde{f}^{(r)} + w$.
end for
return $\tilde{f}^{(T+1)}$.

Conditioning on $\mathcal{E}_r$ we have that $|f - \tilde{f}^{(r)}| \leq \frac{d}{4^r}$. For every $i \in [D/2^r]$ and every $l \in \{0, 1, \cdots, \lceil \log_2 n \rceil\}$ it follows from the definition of ϕ_r^l that,

$$\phi_r^l(i) - \sum_{j \in h_r^{-1}(i)} \tilde{f}_j^{(r)} \cdot v_j^l = \sum_{j \in h_r^{-1}(i)} \left(f_j - \tilde{f}_j^{(r)}\right) \cdot v_j^l.$$

Let us denote by S the support of vector $f - \tilde{f}^{(r)}$, namely let $S = \text{supp}\left(f - \tilde{f}^{(r)}\right)$.

From the pairwise independence of the hash function h_r the following holds for every $a \in S$,

$$\Pr[h_r(a) \in h_r(S \setminus \{a\})] \leq 2^r \cdot \frac{|S|}{D} \leq 2^r \cdot \frac{1}{128 \cdot 4^r} \leq \frac{1}{128 \cdot 2^r}.$$

This shows that for every $a \in S$, with probability $1 - \frac{1}{128 \cdot 2^r}$, the bucket $h_r(a)$ contains no other element of S. Because the vector $f - \tilde{f}^{(r)}$ restricted to the elements in bucket $h_r^{-1}(h_r(a))$ has Hamming weight one, for every $a \in S$,

$$\Pr\left[\left|\left(f - \tilde{f}^{(r-1)}\right)_{h_r^{-1}(h_r(a))}\right| = 1\right] \geq 1 - \frac{1}{128 \cdot 2^r}.$$

If the above condition holds, then it is possible to find the index of the nonzero element via binary search as in Lemma 5.3.3. The for loop in line 7 of Algorithm 1 implements this. Therefore with probability $1 - \frac{1}{16 \cdot 2^r}$ by Markov's inequality a $1 - 1/8$ fraction of the support elements, S, gets recovered correctly and at most $1/8$ fraction of elements remain unrecovered and possibly result in false positive. Since the algorithm recovers at most one element per bucket, the total number of falsely recovered indices is no more than the number of non-isolated buckets which is at most $1/8 \cdot |S|$. Therefore with probability $1 - \frac{1}{16 \cdot 2^r}$, the residual at the end of rth iteration has sparsity $1/8 \cdot |S| + 1/8 \cdot |S| = 1/4 \cdot |S|$, i.e. $\left|f - \tilde{f}^{(r+1)}\right| \leq \frac{|S|}{4} \leq \frac{d}{4^{r+1}}$. This proves the inductive step.

It follows from the event $\mathcal{E}_T$ for $T = \log_4 D$ that $\tilde{f}^{(T)} = f$, where $\tilde{f}^{(T)}$ is the output of Algorithm 1. The inductive hypothesis along with union bound implies that $\Pr\left[\bar{\mathcal{E}}_T\right] \leq \sum_{r=1}^{T} \Pr\left[\bar{\mathcal{E}}_r | \mathcal{E}_{r-1}\right] + \Pr\left[\bar{\mathcal{E}}_0\right] \leq \sum_{r=0}^{T} \frac{1}{16 \cdot 2^r} \leq 1/8$.

RUNTIME: The algorithm has three nested loops and the total number of repetitions of all loops together is $O(D \log n)$. The recovered

frequency $\tilde{f}^{(r)}$ always has at most $O(D)$ nonzero entries therefore the time to calculate $\sum_{j \in h_r^{-1}(i)} \tilde{f}_j^{(r-1)} \cdot v_j^l$ for a fixed r and a fixed l and all $i \in [D/2^r]$ is $O(D)$. Therefore the total runtime is $O(D \log D \log n)$.

NUMBER OF MEASUREMENTS: The number of measurements is the total size of the measurement vectors ϕ_r^l which is $O(D \log n)$. □

5.3.2 *Signal reduction*

We now develop the main tool for estimating the frequencies of a sparse signal, namely the HASH2BINS primitive. If we hash the frequencies of a k-sparse signal into $O(k)$ buckets, we expect most buckets to contain at most one of the elements of the support of our signal. The next definition shows how we compute the hashing of a signal in the time domain.

Definition 5.3.5. *For every $n, b \in \mathbb{N}$, every $a \in \mathbb{F}_2^n$, and every $\sigma \in \mathbb{F}_2^{n \times b}$ and every $x : \mathbb{F}_2^n \to \mathbb{R}$, we define the hashing of $\widehat{g}$ as $u_\sigma^a : \mathbb{F}_2^b \to \mathbb{R}$, where $u_\sigma^a(t) = \sqrt{\frac{2^n}{2^b}} \cdot x_{\sigma t + a}$, for every $t \in \mathbb{F}_2^b$.*

We denote by $B \triangleq 2^b$ the number of buckets of the hash function. In the next claim, we show that the Fourier transform of u_σ^a corresponds to hashing $\widehat{g}$ into B buckets.

Claim 5.3.6. *For every $j \in \mathbb{F}_2^b$, $\widehat{u}_\sigma^a(j) = \sum_{f \in \mathbb{F}_2^n : \sigma^\top f = j} \widehat{g}_f \cdot (-1)^{\langle a, f \rangle}$.*

Let $h(f) \triangleq \sigma^\top f$. For every $j \in \mathbb{F}_2^b$, $\widehat{u}_\sigma^a$ is the sum of $\widehat{g}_f \cdot (-1)^{\langle a, f \rangle}$ for all frequencies $f \in \mathbb{F}_2^n$ such that $h(f) = j$, hence $h(f)$ can be thought of as the bucket that f is hashed into. If the matrix σ is chosen uniformly at random then the hash function $h(\cdot)$ is pairwise independent.

Claim 5.3.7. *For any $n, b \in \mathbb{N}$, if the hash function $h : \mathbb{F}_2^n \to \mathbb{F}_2^b$ is defined as $h(\cdot) = \sigma^\top(\cdot)$, where $\sigma \in \mathbb{F}_2^{n \times b}$ is a random matrix whose entries are distributed independently and uniformly at random on $\mathbb{F}_2$, then for any $f \neq f' \in \mathbb{F}_2^n$ it holds that $\Pr[h(f) = h(f')] = \frac{1}{B}$, where the probability is over picking $n \cdot b$ random bits of σ.*

The HASH2BINS primitive computes the Fourier coefficients of the residue signal that are hashed to each of the buckets. We denote by $\widehat{\chi}$ the estimate of $\widehat{g}$ in each iteration. As we will see in Section 5.3.3, the recovery algorithm is iterative in the sense that we iterate over $\widehat{g} - \widehat{\chi}$

Algorithm 2 HASH2BINS

input: signal $x \in \mathbb{R}^{2^n}$, signal $\widehat{\chi} \in \mathbb{R}^{2^n}$, integer b, binary matrix $\sigma \in \mathbb{F}_2^{n \times b}$, shift vector $a \in \mathbb{F}_2^n$.
output: hashed signal $\widehat{u}_\sigma^a$.

1: Compute $\widehat{u}_\sigma^a = \mathsf{FHT}\left(\sqrt{\frac{2^n}{2^b}} \cdot x_{\sigma(\cdot)+a}\right)$. ▷ FHT is the fast Hadamard transform algorithm
2: $\widehat{u}_\sigma^a(j) \leftarrow \widehat{u}_\sigma^a(j) - \sum_{f \in \mathbb{F}_2^n : \sigma^\top f = j} \widehat{\chi}_f \cdot (-1)^{\langle a, f \rangle}$ for every $j \in \mathbb{F}_2^b$.
3: **return** $\widehat{u}_\sigma^a$.

(the residue) whose sparsity is guaranteed to decrease by a constant factor in each step.

Claim 5.3.8. *For any signal $x, \widehat{\chi} : \mathbb{F}_2^n \to \mathbb{R}$, integer b, matrix $\sigma \in \mathbb{F}_2^{n \times b}$, and vector $a \in \mathbb{F}_2^n$ the procedure* HASH2BINS$(x, \widehat{\chi}, b, \sigma, a)$ *given in Algorithm 2 computes the following using $O(B)$ samples from x in time $O(Bn \log B + \|\widehat{\chi}\|_0 \cdot n \log B)$*

$$\widehat{u}_\sigma^a(j) = \sum_{f \in \mathbb{F}_2^n : \sigma^\top f = j} \widehat{(x - \chi)}_f \cdot (-1)^{\langle a, f \rangle}.$$

5.3.3 *Exact Fourier recovery*

In this section, we present our algorithm for solving the exact low-degree Fourier sparse problem defined in (5.1) and prove Theorem 5.3.1. Let $S \triangleq \mathrm{supp}(\widehat{g})$. Problem (5.1) assumes that $|S| \leq k$ and also for every $f \in S$, $|f| \leq d$. The recovery algorithm hashes the frequencies into $B = 2^b$ buckets using Algorithm 2. Every frequency in the support $f \in S$ is recoverable, with constant probability, if no other frequency from the support collides with it in the hashed signal. The collision event is formally defined below,

Definition 5.3.9 (Collision)**.** *For any frequency $f \in \mathbb{F}_2^n$ and every sparse signal $\widehat{g}$ with support $S = supp(\widehat{g})$, the collision event $E_{coll}(f)$ corresponding to the hash function $h(f) = \sigma^\top f$ holds iff $h(f) \in h(S \setminus \{f\})$.*

Claim 5.3.10 (Probability of collision)**.** *For every $f \in \mathbb{F}_2^n$, if the hash function $h : \mathbb{F}_2^n \to \mathbb{F}_2^b$ is defined as $h(\cdot) = \sigma^\top(\cdot)$, where $\sigma \in \mathbb{F}_2^{n \times b}$ is a random matrix whose entries are distributed independently and uniformly at*

random on $\mathbb{F}_2$ *then* $\Pr[E_{coll}(f)] \leq \frac{k}{B}$ *(see Definition 5.3.9). The probability is over the randomness of matrix* σ*.*

If the hash function $h(\cdot) = \sigma^\top(\cdot)$ is such that the collision event $E_{coll}(f)$ does not occur for a frequency f, then it follows from Claim 5.3.6 and Definition 5.3.9 that for every $a \in \mathbb{F}_2^n$,

$$\widehat{u}_\sigma^a(h(f)) = \widehat{g}(f) \cdot (-1)^{\langle a,f\rangle}.$$

Therefore, under this condition, the problem reduces to d-sparse recovery. If $a = \{0\}^n$ then, $\widehat{u}_\sigma^a(h(f)) = \widehat{g}(f)$. Hence for any $m \in \mathbb{F}_2^n$, one can learn the inner product $\langle m, f\rangle$ by comparing the sign of $\widehat{u}_\sigma^m(h(f)) = \widehat{g}(f) \cdot (-1)^{\langle m,f\rangle}$ and $\widehat{u}_\sigma^a(h(f))$. If the signs are the same then $(-1)^{\langle m,f\rangle} = 1$ meaning that $\langle m, f\rangle = 0$ and if the signs are different then $\langle m, f\rangle = 1$. In Section 5.3.1 we gave an algorithm for learning a low-degree frequency $|f| \leq d$ from measurements of the form $\langle m, f\rangle$. So putting these together gives the inner subroutine for our sparse fast Hadamard transform, which performs one round of hashing, presented in Algorithm 3.

Lemma 5.3.11. *For all integers* b *and* d*, every signals* $x, \widehat{\chi} \in \mathbb{R}^{2^n}$ *such that* $|\xi| \leq d$ *for every* $\xi \in supp(\widehat{x - \chi})$*, and any parameter* $p > 0$*, Algorithm 3 outputs a signal* $\widehat{\chi}' \in \mathbb{R}^{2^n}$ *such that* $|supp(\widehat{\chi}')| \leq |supp(\widehat{x - \chi})|$ *and also for every frequency* $f \in supp(\widehat{x - \chi})$*, if the collision event* $E_{coll}(f)$ *does not happen then,*

$$\Pr\left[\widehat{\chi}'_f = (\widehat{x - \chi})_f\right] \geq 1 - p.$$

Moreover the sample complexity of this procedure is $O(Bd \log n \log \frac{1}{p})$ *and also its time complexity is*

$$O\left(B \log B(n + d \log n \log \frac{1}{p}) + nB \log n \log d \log \frac{1}{p} + \|\widehat{\chi}\|_0 \cdot n \right.$$
$$\left. (\log B + \log n \log d \log \frac{1}{p})\right.$$

Lemma 5.3.12. *For any parameter* $p > 0$*, all integers* k*,* d*, and* $b \geq \log_2(k/p)$*, every signal* $x, \widehat{\chi} \in \mathbb{R}^{2^n}$ *such that* $\|\widehat{x - \chi}\|_0 \leq k$ *and* $|\xi| \leq d$ *for every* $\xi \in supp(\widehat{x - \chi})$*, the output of* SHTINNER$(x, \widehat{\chi}, p, b, d)$*,* $\widehat{\chi}'$ *satisfies the following with probability at least* $1 - 32p$*,*

$$\|\widehat{g} - \widehat{\chi} - \widehat{\chi}'\|_0 \leq k/8.$$

Our sparse Hadamard transform algorithm iteratively calls the primitive SHTInner to reduce the sparsity of the residual signal by a constant factor in every iteration. Hence, it terminates in $O(\log k)$ iterations. See Algorithm 4.

Proof. Theorem 5.3.1 The proof is by induction. We denote by $\mathcal{E}_r$ the event corresponding to $\|\widehat{g} - w^{(r)}\|_0 \leq \frac{k}{8^r}$. The inductive hypothesis is $\Pr[\mathcal{E}_r|\mathcal{E}_{r-1}] \geq 1 - 16p^{(r)}$. Conditioned on $\mathcal{E}_{r-1}$ we have that $\|\widehat{g} - w^{(r-1)}\|_0 \leq \frac{k}{8^{r-1}}$. The number of buckets in iteration r of the algorithm is $B^{(r)} = 2^{b^r} \geq \frac{64k}{4^{r-1} \cdot q}$. Hence, it follows from Lemma 5.3.12, that with probability $1 - 32p^{(r)}$, $\|\widehat{g} - w^{(r)}\|_0 \leq \frac{k}{8^r}$. This proves the inductive step.

RUNTIME AND SAMPLE COMPLEXITY: In iteration $r \in [\lceil \log_8 k \rceil]$, the size of the bucket $B^{(r)} = 2^{b^{(r)}} = \frac{64k}{q \cdot 4^r}$ and the error probability $p^{(r)} = \frac{q}{32 \cdot 2^r}$. Moreover at most $\sum_r B^{(r)}$ elements are added to $\widehat{\chi}$, hence we can assume that $\|\widehat{\chi}\|_0 \leq \frac{128k}{q}$. From Lemma 5.3.11 it follows that the total runtime is $O\left(kn \log^2 k \log n \log d\right)$.

The sample complexity of iteration r is $O\left(\frac{kd}{2^r} \log n \log 2^r\right)$ hence the total sample complexity is dominated by the sample complexity of the first iteration which is equal to $O\left(kd \log n\right)$. □

5.4 EXPERIMENTS

We test our ExactSHT algorithm for graph sketching on a real-world data set. We utilize the autonomous systems dataset from the SNAP data collection.[1] In order to compare our methods with [89] we reproduce their experimental setup. The dataset consists of 9 snapshots of an autonomous system in Oregon on 9 different dates. The goal is to detect which edges are added and removed when comparing the system on two different dates. As a pre-processing step, we find the common vertices that exist on all dates and look at the induced subgraphs on these vertices. We take the symmetric differences (over the edges) of dates 7 and 9. Results for other date combinations can be found in the supplementary material. This results in a sparse graph

1 snap.stanford.edu/data/

(sparse in the number of edges). Recall that the running time of our algorithm is $O(kn \log^2 k \log n \log d)$ which reduces to $O(nk \log^2 k \log n)$ for the case of cut functions where $d = 2$.

5.4.1 *Sample and time complexities as number of vertices varies*

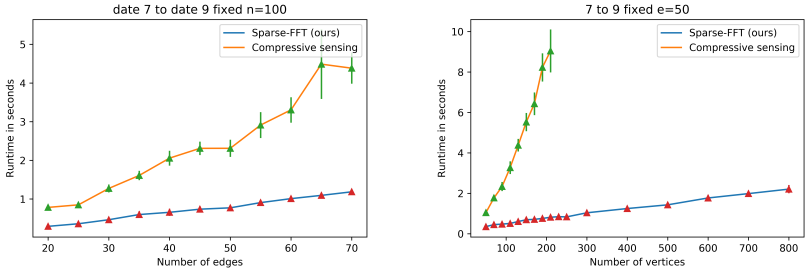
In the first experiment depicted in Figure 5.1b we order the vertices of the graph by their degree and look at the induced subgraph on the n largest vertices in terms of degree where n varies. For each n we pick $e = 50$ edges uniformly at random. The goal is to learn the underlying graph by observing the values of cuts. We choose parameters of our algorithm such that the probability of success is at least 0.9. The parameters tuned in our algorithm to reach this error probability are the initial number of buckets the frequencies are hashed to and the ratio at which they reduce in each iteration. We plot running times as n varies. We compare our algorithm with that of [89] which utilizes a CS approach. We fine-tune their algorithm by tuning the sampling complexity. Both algorithms are run in a way such that each sample (each observation of a cut value) takes the same time. As one can see our algorithm scales *linearly* with n (up to log factors) whereas the CS approach scales *quadratically*. Our algorithm continues to work in a reasonable amount of time for vertex sizes as much as 900 in under 2 seconds. Error bars depict standard deviations.

In Table 5.2 we include both sampling complexities (number of observed cuts) and running times as n varies. Our sampling complexity is equal to $O(k \log n)$. In practice, they perform around a constant factor of 10 worse than the compressive sensing method, which is not provably optimal (see Section 5.1) but performs well in practice. In terms of computational cost, however, the CS approach quickly becomes intractable, taking large amounts of time on instance sizes around 200 and larger [89]. Asterisks in Table 5.2 refer to experiments that have taken too long to be feasible to run.

5.4.2 *Time complexities as number of edges varies*

Here we fix the number of vertices to $n = 100$ and consider the induced subgraph on these vertices. We randomly pick e edges to include in the graph. We plot computational complexities. Our running

time provably scales linearly in the number of edges as can be seen in Figure 5.1a.



(a) Avg. time vs. no. edges

(b) Avg. time vs. no. vertices

5.5 FREQUENCY RECOVERY PRIMITIVES AS LINEAR ERROR-CORRECTING CODES

In Section 5.3.1 and Algorithm 1, we described a novel hashing-based frequency recovery primitive. In this section, we show, that the frequency recovery primitive, when viewed in the correct light, can be reduced to a widely studied problem in the theory of *linear error-correcting codes*. Therefore one can use the rich theory behind linear codes to design optimal measurement (shift) vectors in terms of query complexity. Moreover the resulting frequency recovery primitives are more reliable and easy to tune yielding an overall algorithm that runs in a robust and reliable fashion. We propose two new sets of measurement vectors – based on *Reed-Solomon* codes and *Random* codes respectively. We provide a theoretical analysis, and show through extensive experiments in Section 5.6.1 that indeed, these measurement vectors yield improved query complexities. We continue with a brief introduction to the theory of error correcting codes.

In coding theory, we want to transmit one of 2^g messages $s \in \mathbb{F}_2^g$ over a noisy channel. We embed these message vectors s into a higher dimensional vector space $\mathbb{F}_2^n$, where $n > g$ and call it $\widehat{s}$. These embeddings are chosen such that they have some minimum pairwise hamming distance of say $2d+1$. This allows for error *detection* in case at most $2d$ of these bits are corrupted and for *recovery* when at most d bits become corrupted by the medium of transfer. One such

No. of vertices	CS method		Our method	
	Runtime	Samples	Runtime	Samples
70	1.14	767	0.85	6428
90	1.88	812	0.92	6490
110	3.00	850	0.82	6491
130	4.31	880	1.01	7549
150	5.34	905	1.16	7942
170	6.13	927	1.22	7942
190	7.36	947	1.18	7271
210	8.24	965	1.28	7271
230	*	*	1.38	7942
250	*	*	1.38	7271
300	*	*	1.66	8051
400	*	*	2.06	8794
500	*	*	2.42	8794
600	*	*	3.10	9646
700	*	*	3.35	9646
800	*	*	3.60	9646

TABLE 5.2: Sampling and computational complexity

embedding is the class of "linear" codes where the embeddings form a g-dimensional subspace of $\mathbb{F}_2^n$ and therefore, can be described as a multiplication with a (full-rank) matrix $G \in \mathbb{F}_2^{n \times g}$:

$$\widehat{s} = Gs, s \in \mathbb{F}_2^g$$

The columns of G form a basis for the embedding subspace and the matrix G is called the "generating matrix". To this matrix, we can assign a "parity check" matrix $A \in \mathbb{F}_2^{n \times m}$, where $g + m = n$. By design, the column space of the "parity check" matrix is the orthogonal complement to the generating matrix's columns space (the embedding subspace). Therefore, for any valid code $\widehat{s} = Gs$, it holds that $A^\top \widehat{s} = 0$. Moreover, if the code-word $\widehat{s}$ is corrupted by some error vector $f \in \mathbb{F}_2^n$, i.e., if the code-word $\widehat{s}$ is sent and $\widehat{s} + f$ is received, it holds that:

$$y = A^\top(\widehat{s} + f) = A^\top f$$

and the vector y is called the "syndrome vector". The problem of recovering s given the syndrome vector y under conditions that $\deg(f) \leq d$, is well studied in coding theory and is broadly known as "syndrome decoding". Crucially, a side-effect of finding s is that we also find f, which is exactly what is needed in the "frequency detection" primitive described in Section 5.3.1.

More precisely, our main insight is that we can use the columns of $A \in \mathbb{F}_2^{n \times m}$, denoted by $a_1, \ldots, a_m \in \mathbb{F}_2^n$ as measurement (shift) vectors of the frequency recovery primitive introduced in Section 5.3.1. Syndrome decoders are guaranteed to recover "error" vectors $f \in \mathbb{F}_2^n$ of degree at most d given the syndrome vector $y = A^\top f \in \mathbb{F}_2^m$. In other words, given the measurement vector a.k.a syndrome vector y and measurement matrix $A \in \mathbb{F}_2^{n \times m}$, we can use *any* syndrome decoding algorithm, out of the box, to find the error vector f of degree at most d that satisifes $y = A^\top f$. These "error vectors" are nothing but the "frequency vectors" that we wanted to recover in the frequency recovery primitives.

One set of measurement vectors, that come with syndrome decoders, are called the *Reed-Solomon error detecting codes* [97] and the following Proposition from Das & Vishwanath [98] characterizes its guarantees (Corollary 3.2 in the mentioned paper):

Proposition 5.5.1 (Reed Solomon measurements)**.** *There exist* $m = 2d\lceil \log_2 n + 1 \rceil$ *measurement vectors* $a_1, \ldots, a_m \in \mathbb{F}_2^n$ *such that for any fre-*

quency $f \in \mathbb{F}_2^n$ of degree at most d, if one has the value of the linear measurements $\langle f, a_i \rangle \in \mathbb{F}_2$ then one can recover f using $O(nd\lceil \log_2 n \rceil)$ operations.

Another set of measurement vectors are the so called *random codes*. Here, we pick the measurement vectors $a_i \in \mathbb{F}_2^n$ independently and uniformly at random. We will see later that random codes exhibit query complexity that is lower than Reed-Solomon codes. However they come at the cost of worst case exponential decoding times. We note that there has been significant work on this matter [99–102]. In this work, we take a different approach and cast the decoding problem as ILPs (Integer Linear Programs) and use a modern standard solver SCIP [103, 104] to solve these instances. We are able to go to fairly high dimensions that are not accessible with CS approaches. We elaborate on this further in Section 5.6.1.

We characterize the theoretical achievable bounds for random codes from Theorem 8 of Guruswami [105]. The mentioned source describes how random linear codes can recover error vectors efficiently (in terms of number of measurements). In fact, they are asymptotically optimal and the number of measurements needed matches that of a lower bound called the "Gilbert-Varshamov" bound. We set $q = 2$ and $\delta = \frac{2d+1}{n}$ and use h_2 for the binary entropy function.

Proposition 5.5.2 (Random measurements)**.** *Let $a_1, \ldots, a_m \in \mathbb{F}_2^n$ be chosen uniformly at random. Assume $4d + 2 < n$ and let $0 < \epsilon < 1 - h_2(\frac{2d+1}{n})$ be chosen arbitrarily and let $m = n\lceil h_2(\frac{2d+1}{n}) + \epsilon \rceil$. Then, for sufficiently large n, with probability $1 - e^{-\Omega(n)}$, if one has the value of the linear measurements $\langle f, a_i \rangle \in \mathbb{F}_2$, one can uniquely recover all $f \in \mathbb{F}_2^n$ of degree at most d .*

It remains to show that Propositions 5.5.1 and 5.5.2 imply (analogs of) Lemma 5.3.4. Regarding Proposition 5.5.1, the recovery is with probability one for any frequency f that is of degree at most d, since the algorithm contains no randomness. Therefore, the guarantees in this proposition are strictly stronger than those of Lemma 5.3.4 . Proposition 5.5.2 is also strictly stronger than Lemma 5.3.4 as it *uniformly* guarantees that with high probability, recovery is guaranteed over *all* possible $f \in \mathbb{F}_2^n$ of degree at most d.

5.6 EXPERIMENTS WITH LINEAR ERROR-CORRECTING CODES AS FREQUENCY RECOVERY PRIMITIVES

Firstly, we quantify the performance of our newly proposed frequency recovery primitives based on *random* and *Reed Solomon* codes. Next, we quantify the improvements in the overall end to end HSFT algorithm on a synthetic black-box function and finally on a black-box model trained on data.

5.6.1 *Empirical sampling complexity of the new frequency recovery primitives*

We design experiments to quantify the query complexity of the two new proposed frequency recovery primitives based on *Reed Solomon* and *Random* codes proposed in Propositions 5.5.1 and 5.5.2 respectively. We compare their query complexity to that of prior work, which we call *binning*. The following describes the procedure carried out for each frequency detection primitive.

Random codes: we first generate measurement vectors $a_1, \ldots, a_m \in \mathbb{F}_2^n$ uniformly at random for a fixed $m \in \mathbb{N}$. Next, we generate a frequency $f \in \mathbb{F}_2^n$ of degree at most d uniformly at random. We use the measurement vectors $a_1, \ldots, a_m$ as rows in the observation matrix $A^\top \in \mathbb{F}_2^{m \times n}$. We compute the measurement vector $y = A^\top f$. Next, we form an integer linear program that finds a vector $\widehat{f}$ that satisfies $y = A^\top \widehat{f}$ and $\deg(\widehat{f}) \leq d$. We use an integer program solver, SCIP [103], with the default settings, to find such a feasible solution. For any fixed m we repeat this procedure 10 times and compute the probability of success, where the randomness is over the 10 different instantiations of f and observation matrices A.

Binning We generate measurement vectors $a_1, \ldots, a_m \in \mathbb{F}_2^n$ according to the proposed method and for a fixed set of hyper-parameters. Remember from Section 5.6.1, that this algorithm has three tunable hyper-parameters and each such setting gives rise to a random number of measurements m. For each such set of measurement vectors we generate 100 different frequencies $f \in \mathbb{F}_2^n$ of degree d uniformly at random. We run the algorithm to recover the each frequency and compute its probability of success where the randomness is over the 100 differ-

ent instantiations of the frequencies. We repeat this whole procedure many times for different settings of three different hyper-parameters.

Reed Solomon Note that Reed-Solomon has no tunable hyper-parameters and succeeds with probability one.

Lower-bound This is a trivial combinatorial lower bound on the number of measurements, below which, one can not recover the frequency with probability one. It is computed as $\log_2(\sum_{i=0}^{d}\binom{n}{i})$.

We plot the probability of success vs. number of measurements in Figure 5.2a for $n = 192$ and $d = 5$ for all three primitives. We do the same in Figure 5.2b for the Reed Solomon and binning primitives for $n = 6144$ and $d = 10$. Figures 5.2c 5.2d 5.2e show the respective runtimes it takes to recover exactly one frequency in each of the proposed methods. When compared to binning, both the Reed-Solomon and random primitives achieve higher probabilities of success for the same number of measurements. We can see that random measurements have a phase transition very close to the combinatorial lower bound. Runtimes for Reed Solomon and binning are comparable. As expected, the runtime for the random measurements is an order of magnitude slower. Although slower, recovery comes with the advantage of fewer measurements needed. Therefore random measurements remain relevant when querying the black-box function is very slow and hence the runtime is dominated by these timely queries.

5.6.2 *Learning a synthetic black-box function*

We construct a synthetic black-box function $h : \mathbb{F}_2^n \to \mathbb{R}$, where $n = 500$, by picking $k = 30$ frequencies $f_1, \ldots, f_{30} \in \mathbb{F}_2^{500}$ of degree at most d uniformly at random. We assign to each a frequency f_i, an amplitude $\widehat{h}(f_i)$ chosen uniformly are random in the interval $[-100, 100]$. The function is queried uniformly at random and we minimize the Lagrangian formulation of the sparse recovery problem, using proximal methods [89]. However, even for the case of depth $d = 2$, the algorithm fails to complete within 24 hours.

Next, we consider the HSFT algorithms with the newly proposed frequency recovery primitives. We run the algorithms end to end for varying values of C (specifies the number of buckets we hash into) and different settings of hyper-parameters for each of the frequency

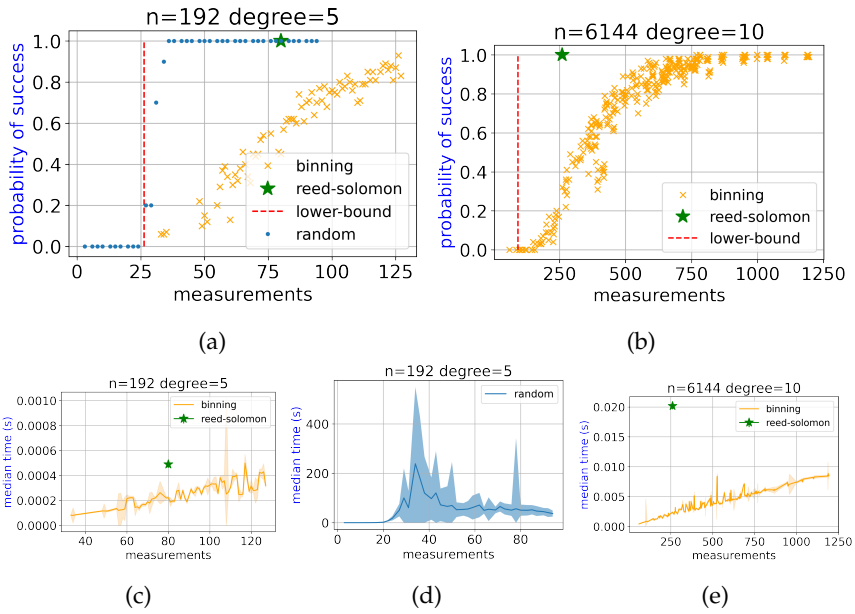


FIGURE 5.2: (a) Probability of success of the three different frequency recovery primitives given number of measurements for the case of $n = 192$, $d = 5$ (b) Probability of success of Reed Solomon vs. binning given number of measurements in much higher dimensions $n = 6144$, $d = 10$ (c) Median runtime of Reed Solomon and binning for recovering a single frequency $n = 192$, $d = 5$ (d) Median runtime of random for recovering a single frequency $n = 192$, $d = 5$ (e) Median runtime of Reed Solomon and binning for recovering a single frequency $n = 6144$, $d = 10$.

recovery primitives. In Figure 5.3d we plot the query complexity and the normalized error of the retrieved Fourier transform defined as:

$$\text{normalized error} \triangleq \frac{\sum_{f \in \mathbb{F}_2^n} (\widehat{h}_{\text{est}}(f) - \widehat{h}(f))^2}{\sum_{f \in \mathbb{F}_2^n} \widehat{h}(f)^2} = \frac{\sum_{t \in \mathbb{F}_2^n} (h_{\text{est}}(t) - h(t))^2}{\sum_{t \in \mathbb{F}_2^n} h(t)^2} \tag{5.3}$$

for the value of degree $d = 3$. Here, $\widehat{h}_{\text{est}}(f)$ is the Fourier transform returned by the algorithm and $\widehat{h}(f)$ is equal to the true Fourier transform. We can see that random measurements is able to nearly perfectly recover the function with less than 1000 measurements. The best result for binning is with 2700 measurements with a normalized

error that is still higher than that of random measurements. Although Reed-Solomon does not necessarily outperform binning in all cases, we can see that it is far more reliable and in almost all cases it outperforms binning in terms of normalized error for a given number of measurements.

5.6.3 *Learning a black-box trained on a dataset*

We now utilize our proposed algorithm for the task of learning a black-box function trained on a real-world dataset. We train a Random Forest model, as our black-box, on the "superconduct" [2] dataset. This dataset has 81 features which are all continuous. Each feature is binned into 16 buckets according to its quantiles, and encoded as a 4-bit binary number. The resulting model is treated as a black-box $h : \mathbb{F}_2^n \to \mathbb{R}$ with $n = 4 \times 81 = 324$. The random forest model is trained with 20 trees for a variety of depths d and its R^2 score is plotted in Figure 5.3a. This figure portrays the importance of depth, and hence the degree of the Fourier transform for models that perform well on this dataset.

We first consider the approach of Stobbe & Krause [89] and try to use find the Fourier transform using Compressive Sensing methods. The black-box is queried uniformly at random and we minimize the Lagrangian formulation of the sparse recovery problem, using proximal methods. However, even for the case of depth $d = 2$ the algorithm fails to converge within 24 hours. As mentioned before, since there is $\theta(n^d)$ basis functions, *CS* methods run in times that are *exponential* with respect to the depth and have rarely been applied to dimensions where n is greater or equal to 100, even for the simple case of $d = 2$. In order to portray this exponential dependence, we utilize feature importance to extract the top $n = 40$ features of the random forest model and we retrain the model on these 40 features. We run the proposed CS method and plot the runtime for a variety of depths and plot the runtime on a *logarithmic* scale in Figure 5.3b.

Next, we consider the HSFT algorithms with the newly proposed frequency recovery primitives. We run the algorithms end to end. In Figures 5.3c and 5.3e, we plot the query complexity and the normalized error of the retrieved Fourier transform defined as defined in

2 https://archive.ics.uci.edu/ml/datasets/superconductivty+data

Equation (5.3). Where $\widehat{h}(f)$ is the Fourier transform returned by the algorithm and $\widehat{h}_{\text{est}(f)}$ is equal to the true Fourier transform which can be computed since one has access to the trees in the random forest. We can see again that Reed-Solomon and random outperform binning in almost all cases in terms of query complexity.

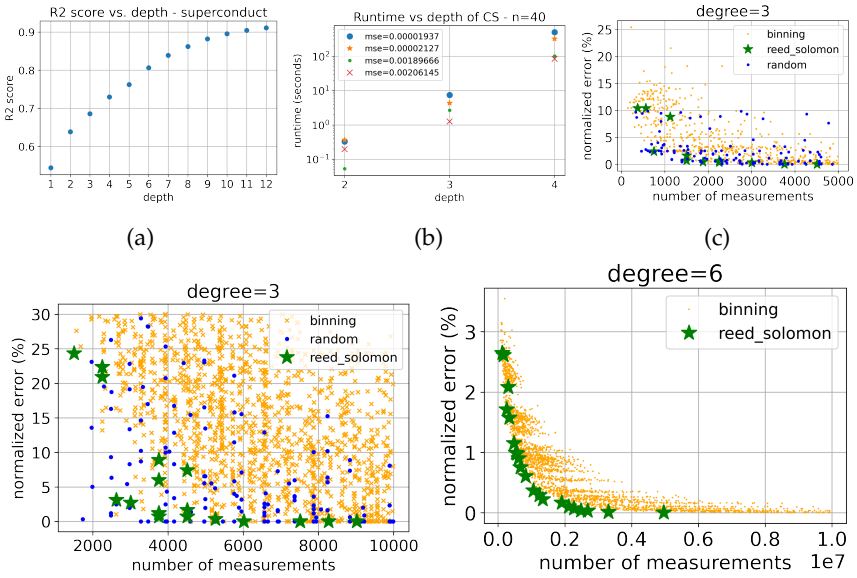


FIGURE 5.3: (a) R^2 scores of the random forest model increase with depth (b) Runtime of CS methods is exponential with respect to depth (c) Normalized error vs. number of measurements for the end to end algorithm on a random forest black-box function of depth 3 (d) Normalized error vs. number of measurements for the end to end algorithm on a synthetic black-box function of degree 3 (e) Normalized error vs. number of measurements for the end to end algorithm on a random forest black-box function of depth 6

5.7 RELATIONSHIP TO NON-ORTHOGONAL FOURIER BASES

So far we have covered computational/query-efficient algorithms for computing the Walsh-Hadamard transform. In light of extensions of

the Fourier basis in signal processing literature [20] we extended the exact version of our algorithm to three new Fourier bases introduced by Püschel [20] in our work [19].

To give a sense of why other Fourier bases might be important we introduce one of the three new bases: the coverage function basis. Before that, we define coverage functions. Let $\{E_1, E_2, \ldots, E_n\}$ be a collection of subsets of some ground set $[k] \triangleq \{1, \ldots, k\}$ i.e. $E_i \subseteq [k]$ let $g : 2^{[n]} \to \mathbb{R}$, be a set function defined as follows:

$$g(S) = \left| \bigcup_{i \in S} E_i \right| : S \subseteq [n]$$

One can think of the subsets E_i as covering sets and the function g saying how many elements of the ground set the union of these subsets covers. Furthermore, one can assign weights $w_i, i \in [k]$ to each of the items in the ground set $[k]$ and define g as follows:

$$g(S) = \sum_{i \in \bigcup_{i \in S} E_i} w_i$$

This is the so-called "generalized coverage function" so long as the weights are non-negative. We show in [19] that such a function (even for negative weights) is k-sparse in the coverage function basis but dense in the Walsh-Hadamard basis. Namely, we show that it can 2^n non-zero Walsh-Hadamard coefficients. We use similar ideas presented in this chapter to compute the non-zero Coverage function Fourier coefficients with $O(nk)$ queries and $\tilde{O}(nk)$ computational complexity given query access to g.

Algorithm 3 SHTInner

input: signal $x \in \mathbb{R}^{2^n}$, signal $\widehat{\chi} \in \mathbb{R}^{2^n}$, failure probability p, integer b, integer d.
output: recovered signal $\widehat{\chi}'$.

Let $\{v^l\}_{l=0}^{\lceil \log_2 n \rceil}$ be binary search vectors on n elements (Definition 5.3.2).
$D \leftarrow$ smallest power of two integer s.t. $D \geq 128d$, $R \leftarrow \lceil 2\log_2(1/p) \rceil$.
For every $r \in \{0, 1, \cdots, \log_4 D\}$ and every $s \in [R]$, let $h_r^s : [n] \to [D/2^r]$ be an independent copy of a pairwise independent hash function.
For every $r \in \{0, 1, \cdots, \log_4 D\}$, every $s \in [R]$, and every $j \in [D/2^r]$ let $w_{r,s}^j \in \mathbb{F}_2^n$ be the binary indicator vector of the set $h_r^s(j)^{-1}$.
For every $s \in [R]$, every $r \in \{0, 1, \cdots, \log_4 D\}$ and every $l \in \{0, 1, \cdots, \lceil \log_2 n \rceil\}$ and every $j \in [D/2^r]$, add $w_{r,s}^j \cdot v^l$ to set A_s.
Let $\sigma \in \mathbb{F}_2^{n \times b}$ be a random matrix. Each entry is independent and uniform on $\mathbb{F}_2$.
For every $a \in \cup_{s \in [R]} A_s$ compute $\widehat{u}_\sigma^a = \textsc{Hash2Bins}(x, \widehat{\chi}, b, \sigma, a)$.
for $j = 1$ to B **do**
    Let L be an empty multi-set.
    for $s \in [R]$ **do**
        for every $r \in \{0, \cdots, \log_4 D\}$, every $i \in [D/2^r]$, and every $l \in \{0, \cdots, \lceil \log_2 n \rceil\}$ **do**
            if $\widehat{u}_\sigma^c(j) \neq 0$, where $c = \{0\}^n$ **then**
                if $\widehat{u}_\sigma^c(j)$ and $\widehat{u}_\sigma^{w_{r,s}^i \cdot v^l}(j)$ have same sign **then** $\phi_r^l(i) \leftarrow 0$. **else** $\phi_r^l(i) \leftarrow 1$.
            end if
        end for
        $\tilde{f} \leftarrow \textsc{RecoverFrequency}\left(D, \{h_r^s\}_{r=0}^{\log_2 D}, \left\{\{\phi_r^l\}_{r=0}^{\log_4 D}\right\}_{l=0}^{\lceil \log_2 n \rceil}\right)$.
        Append $\tilde{f}$ to multi-set L.
    end for
    $f \leftarrow \text{majority}(L)$
    $\widehat{\chi}'_f \leftarrow \widehat{u}^c(j)$, where $c = \{0\}^n$.
end for
return $\widehat{\chi}'$.

Algorithm 4 ExactSHT

input: signal $x \in \mathbb{R}^{2^n}$, failure probability q, sparsity k, integer d.
output: estimate $\widehat{\chi} \in \mathbb{R}^{2^n}$.

$p^{(1)} \leftarrow q/32, b^{(1)} \leftarrow \lceil \log_2 \frac{64k}{q} \rceil, w^{(0)} \leftarrow \{0\}^{2^n}, T \leftarrow \lceil \log_8 k \rceil$.
for $r = 1$ to T **do**
    $\widetilde{\chi} \leftarrow \text{SHTInner}(x, w^{(r-1)}, p^{(r)}, b^{(r)}, d)$
    $w^{(r)} \leftarrow w^{(r-1)} + \widetilde{\chi}$.
    $p^{(r+1)} \leftarrow p^{(r)}/2, b^{(r+1)} \leftarrow b^{(r)} - 2$.
end for
$\widehat{\chi} \leftarrow w^{(T)}$.
return $\widehat{\chi}$.

6

SUMMARY AND FUTURE DIRECTIONS

In conclusion, in this thesis, we delved into the "spectral bias" observed in supervised learning models, particularly neural networks and decision tree ensembles. We have showed that neural networks, despite their capacity to learn arbitrary functions, lean towards simpler functions when trained with methods like stochastic gradient descent. This simplicity is observed through the Walsh-Hadamard transform. More formally, we introduced notions of sparsity and low-degreeness to define what we mean by simplicity for both neural networks and also trees.

The work then presented three main contributions all centered around the notions of low-degreeness and sparsity:

- In this chapter we showed that the simplicity bias of a neural network, more precisely the low-degreeness of its spectrum, can hurt its generalization capabilities. This was shown through extensive synthetic experiments and the notions of Spectral Approximation Error and also on real-world tabular datasets. We introduced a computationally efficient spectral regularization method for addressing this problem bias in neural networks.
- We introduced a faster way to compute interventional SHAP values using compact Fourier representations. This was based on the fact that many models such as ensembles of trees and also deep neural networks have spectral biases. The main point of this chapter was that the computation of SHAP values admits a closed-form expression, avoiding an exponential sum when the predictor function is expressed as a sparse Fourier representation.
- We provided an improved algorithm for computing the Fourier transform given query access to the black-box function. The main advantage of the algorithm we offered was its ability to exploit low-degree assumptions on the black-box. Again, as mentioned above, this assumption holds for black boxes where the function is a tree or a neural network and more: such as graph cut functions discussed in the chapter.

Here are a few open problems worth addressing.

- From Chapter 3 we are missing a more theoretical analysis of a sparsifying L_1-norm. To begin with we can start with the L_1 regularizer applied to the full spectrum of the neural network, as opposed to the computationally friendly down-sampled one we present as a fix in that chapter. Results in compressive sensing provide analytical bounds on how many queries/samples are needed to recover a truly sparse function in some orthogonal basis - such as the Walsh Hadamard basis [93]. One could think of analyzing the function a neural network represents during training, in function space, and through the NTK kernel. Theoretically analyzing how adding a simple L_1 regularizer of the whole spectrum would be a first step. It is worth noting that the Walsh-Hadamard basis functions are the eigenvectors of the NTK matrix [12].
- We can think of improving on the regularization method proposed in Chapter 3. Even though this regularizer is exponentially faster than regularizing the whole spectrum, it is still computationally burdensome compared to training a simple neural network without regularization. A central idea used in Tancik *et al.* [42] is transforming the input space using a transformation that helps the neural network learn higher frequency features. There, they use a random Fourier Features [106] as the transformation. An extension of a feature transformation to higher dimensions, more specifically one to address one-hot encodings is missing. Note that this approach avoids adding a regularizer during training making the training much faster.
- The focus of the thesis was spectral biases of fully connected neural networks and ensembles of trees. One can explore these biases for classes of models more complicated than that. Moreover, nearly all results were for discrete $0, 1$ inputs. One could extend these results to continuous inputs/features. Finally, our focus was mainly on the Walsh-Hadamard basis. One could explore the simplicity biases through lenses other than the Fourier spectrum. For example, regularizing the network through the Haar filter basis.
- The Fourier transform is a "global" transform: it depends on the function values of the entirety of its input domain. Neural network functions tend to have meaningful values on the data

distribution and can be random outside of the data distribution. This is an empirically known fact used for outlier detection and analytically analyzed using the NTK. The regularizer we provided in Chapter 3 is, therefore, not suitable for higher dimensions. Since in higher dimensions, we need a lot of samples to make sure the whole neural network function is sparse. However, what we care about is the sparsity on the data distribution. We need a regularizer that regularizes the neural network function only on the data distribution.

- In Chapter 4 we showed how to compute *interventional* SHAP values very fast. One can try to extend this notion to *observational* SHAP values. Even though the definition of conditional distribution in the observational SHAP values work is not clear [76], the SHAP community seems to have accepted that at least for the case of ensembles of trees, "the conditional distribution produced by the tree" (see [76]) is a good notion of conditional distribution. It seems like a viable path to explore how one can do the tree conditional distribution using a sparse Fourier representation and if that translates to a faster way of computing SHAP values.
- In Chapter 2 we discussed the fact that a tree has a sparse and low-degree Fourier representation. One can think of the same question but the other way around. Namely, say that we have a sparse Fourier representation. We know that the class of functions that admit a sparse Fourier transform is strictly more general than trees [14]. The question is can we provide an algorithm that gives us the most "compact" tree with that sparse representation? Here "compact" means minimal in terms of number of nodes/depth. We attempted this question before and were able to provide results for the case of trees with leaves whose labels are sampled from some continuous distribution. There we proved that from the Fourier representation of the tree, we can recover a tree that has number of nodes and depth at most equal to the original tree. However, we were not able to extend it to more complicated scenarios.

A

APPENDIX: BACKGROUND SECTION PROOFS

Before we start with the proofs we review the Fourier analysis and synthesis equations. As we mentioned in the Background Section 4.2, the Fourier representation of the *pseudo-boolean* function $h : \{0,1\}^n \to \mathbb{R}$ is the unique expansion of h as follows:

$$h(x) = \frac{1}{\sqrt{2^n}} \sum_{f \in \{0,1\}^n} \widehat{h}(f)(-1)^{\langle f,x \rangle}$$

This is the so-called Fourier "synthesis" equation.

The Fourier coefficients $\widehat{h}(f)$ are computed by the Fourier "analysis" equation:

$$\widehat{h}(f) = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} h(x)(-1)^{\langle f,x \rangle} \tag{A.1}$$

A.0.1 *Proof of propositions*

We can now prove Proposition 2.1.1:

Proposition 2.1.1. *Assume $g : \{0,1\}^n \to \mathbb{R}$ can be decomposed as follows: $g(x) = \sum_{i=1}^{p} h_i(x_{S_i}), S_i \subseteq [n]$. That is, each function $g_i : \{0,1\}^{|S_i|} \to \mathbb{R}$ depends on at most $|S_i|$ variables. Then, g is $k = O(\sum_{i=1}^{p} 2^{|S_i|})$-sparse and of degree $d = \max(|S_1|, \ldots, |S_p|)$. (Proof in Appendix A.0.1)*

Proof. Let $g : \{0,1\}^n \to \mathbb{R}$ be a function dependent on exactly d variables $x_{i_1}, \ldots, x_{i_d}$, where $i_1, \ldots, i_d \in [n]$ are distinct indices. We show that for any frequency $f \in \{0,1\}^n$, if $f_j = 1$ for some

$j \notin S \triangleq \{i_1, \ldots, i_d\}$, then, $\widehat{g}(f) = 0$. From the Fourier analysis Equation (A.1) we have:

$$\widehat{g}(f) = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} g(x)(-1)^{\langle f,x \rangle} = \sum_{x_i : i \in S} \sum_{x_j : j \in [n] \setminus S} g(x)(-1)^{\langle f, x_S \rangle}(-1)^{\langle f, x_{[n] \setminus S} \rangle}$$

$$\overset{\text{(i)}}{=} \sum_{x_i : i \in S} g(x)(-1)^{\langle f, x_S \rangle} \sum_{x_j : j \in [n] \setminus S} (-1)^{\langle f, x_{[n] \setminus S} \rangle} \overset{\text{(ii)}}{=} \sum_{x_i : i \in S} g(x)(-1)^{\langle f, x_S \rangle} \cdot 0 = 0$$

Where Equation (i) holds because g is only dependent on the variables in S and Equation (ii) holds by checking the inner sum has an equal number of 1 and -1 added together.

The proof of the proposition follows by the linearity of the Fourier transform. □

Moving on to Proposition 2.1.2:

Proposition 2.1.2. *Let, $g : \{0,1\}^n \to \mathbb{R}$ be pseudo-Boolean function and let $d \in \mathbb{N}$ be some constant (w.r.t. n). If g is of degree d, then, it is $k = O(n^d)$-sparse. (Proof in Appendix A.0.1)*

Proof. We simply note that the number of frequencies $f \in \{0,1\}^n$ of degree at most d is equal to $\sum_{i=0}^{d} \binom{n}{i}$. This sum is $O(n^d)$ for d constant w.r.t n. □

B

APPENDIX: WALSH-HADAMARD REGULARIZER FOR THE LOW-DEGREE SPECTRAL BIAS

B.1 WALSH-HADAMARD TRANSFORM MATRIX FORM

The Fourier analysis equation is given by:

$$\widehat{g}(f) = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} g(x)(-1)^{\langle f,x \rangle}$$

Since this transform is linear, it can be represented by matrix multiplication. Let $\mathbf{X} \in \{0,1\}^{2^n \times n}$ be a matrix that has the enumeration over all possible n-dimensional binary sequences ($\{0,1\}^n$) in some arbitrary but fixed order as its rows. Assume $\mathbf{g}(\mathbf{X}) \in \mathbb{R}^{2^n}$ to be the vector of g evaluated on the rows of $\mathbf{X}$. We can compute the Fourier spectrum as:

$$\widehat{\mathbf{g}} = \frac{1}{\sqrt{2^n}} \mathbf{H}_n \mathbf{g}(\mathbf{X})$$

where $\mathbf{H_n} \in \{\pm 1\}^{2^n \times 2^n}$ is an orthogonal matrix given as follows. Each row of $\mathbf{H_n}$ corresponds to some fixed frequency $f \in \{0,1\}^n$ and the elements of that row are given by $(-1)^{\langle f,x \rangle}, \forall x \in \{0,1\}^n$, where the ordering of the x is the same as the fixed order used in the rows of $\mathbf{X}$. The ordering of the rows in $\mathbf{H}_n$, i.e. the ordering of the frequencies considered, is arbitrary and determines the order of the Fourier coefficients in the Fourier spectrum $\widehat{\mathbf{g}}$.

It is common to define the Hadamard matrix $\mathbf{H}_n \in \{\pm 1\}^{2^n \times 2^n}$ through the following recursion:

$$\mathbf{H}_n = \mathbf{H}_2 \otimes \mathbf{H}_{n-1},$$

where $\mathbf{H}_2 := \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$, and $\otimes$ is the Kronecker product. We use this in our implementation. This definition corresponds to the ordering similar to n-bit binary numbers (e.g., $[0,0,0], [0,0,1], [0,1,0], ..., [1,1,1]$ for $n = 3$) for both frequencies and time (input domain).

Computing the Fourier spectrum of a network using a matrix multiplication lets us utilize a GPU and efficiently compute the transform, and its gradient and conveniently apply the back-propagation algorithm.

B.2 ALGORITHM DETAILS

Let $g : \{0,1\}^n \to \mathbb{R}$ be a pseudo-boolean function with Fourier transform $\widehat{g}$. In the context of our work, this pseudo-boolean function is the neural network function. One can sort the Fourier coefficient of g according to magnitude, from biggest to smallest, and consider the top k biggest coefficients as the most important coefficients. This is because they capture the most energy in the Fourier domain and by Parseval's identity also in the time (original input) domain. It is important to us that these k coefficients $\widehat{g}(f_1), \ldots, \widehat{g}(f_k)$ are not hashed into the same bucket. Say for example two large coefficients $\widehat{g}(f_i), \widehat{g}(f_j), i \neq j$ end up in the same bucket, an event which we call a *collision*. If they have different signs, their sum can form a cancellation and the L_1 norm will enforce their sum to be zero. This entails an approximation error in the neural network: Our goal is to sparsify the Fourier spectrum of the neural network and "zero out" the non-important (small-magnitude) coefficients, not to impose wrong constraints on the important (large magnitude) coefficients.

With this in mind, we first prove our hashing result Equation 3.1. Next, we provide guarantees on how increasing the hashing bucket size reduces collisions. Furthermore, we show how independently sampling the hashing matrix over different rounds guarantees that each coefficient does not collide too often. Ideas presented there can also be found in [15, 53]. We finally review EN-S and showcase the superiority and scalability of our method in terms of computation.

B.2.1 *Proof of Equation 3.1*

Let

$$u_{\mathbf{\alpha}}(\tilde{x}) = \sqrt{\frac{2^n}{2^b}} g(\mathbf{\alpha}\tilde{x}), \forall \tilde{x} \in \{0,1\}^b$$

We can compute its Fourier transform $\widehat{u}_{\mathbf{\alpha}}(\tilde{f})$ as:

$$\begin{aligned}\widehat{u}_{\mathbf{\alpha}}(\tilde{f}) &= \frac{1}{\sqrt{2^b}} \sum_{\tilde{x}\in\{0,1\}^b} u_{\mathbf{\alpha}}(\tilde{x})(-1)^{\langle \tilde{f},\tilde{x}\rangle} \\ &= \frac{1}{\sqrt{2^b}} \sum_{\tilde{x}\in\{0,1\}^b} \sqrt{\frac{2^n}{2^b}}\, g(\mathbf{\alpha}\tilde{x})(-1)^{\langle \tilde{f},\tilde{x}\rangle} \\ &= \frac{\sqrt{2^n}}{2^b} \sum_{\tilde{x}\in\{0,1\}^b} g(\mathbf{\alpha}\tilde{x})(-1)^{\langle \tilde{f},\tilde{x}\rangle} \qquad \text{(B.1)}\end{aligned}$$

Inserting the Fourier expansion of g into Equation (B.1) we have:

$$\begin{aligned}\widehat{u}_{\mathbf{\alpha}}(\tilde{f}) &= \frac{1}{2^b} \sum_{\tilde{x}\in\{0,1\}^b} (-1)^{\langle \tilde{f},\tilde{x}\rangle} \sum_{f\in\{0,1\}^n} \widehat{g}(f)(-1)^{\langle f,\mathbf{\alpha}\tilde{x}\rangle} \\ &= \frac{1}{2^b} \sum_{\tilde{x}\in\{0,1\}^b} \sum_{f\in\{0,1\}^n} \widehat{g}(f)(-1)^{\langle \mathbf{\alpha}^\top f,\tilde{x}\rangle}(-1)^{\langle \tilde{f},\tilde{x}\rangle} \\ &= \frac{1}{2^b} \sum_{f\in\{0,1\}^n} \widehat{g}(f) \sum_{\tilde{x}\in\{0,1\}^b} (-1)^{\langle \mathbf{\alpha}^\top f+\tilde{f},\tilde{x}\rangle}\end{aligned}$$

The second summation is always zero unless $\mathbf{\alpha}^\top f + \tilde{f} = 0$, i.e., $\mathbf{\alpha}^\top f = \tilde{f}$, in which case the summation is equal to 2^b. Therefore:

$$\widehat{u}_{\mathbf{\alpha}}(f) = \sum_{\tilde{f}\in\{0,1\}^n:\ \mathbf{\alpha}^T\tilde{f}=f} \widehat{g}(f)$$

B.2.2 *Collisions for* HashWH

We first review the notion of *pairwise independent* families of hash functions introduced by [35]. We compute the expectation of the number of collisions for this family of hash functions. We then show that uniformly sampling $\sigma \in \{0,1\}^{n\times b}$ in our hashing procedure (in HashWH) gives rise to a pairwise independent hashing scheme.

Definition B.2.1 (Pairwise independent hashing)**.** *Let $\mathcal{H} \subseteq \{h|h \in \{0,1\}^n \to \{0,1\}^b\}$ be a family of hash functions. Each hash function maps n-dimensional inputs $x \in \{0,1\}^n$ into a b-dimensional buckets $u = h(f) \in \{0,1\}^b$ and is picked uniformly at random from $\mathcal{H}$. We call this family* pairwise independent *if for any distinct pair of inputs $f_1 \neq f_2 \in \{0,1\}^n$ and an arbitrary pair of buckets $u_1, u_2 \in \{0,1\}^b$:*

1. $P(h(f_1) = u_1) = \frac{1}{2^b}$
2. $P((h(f_1) = u_1) \wedge (h(f_2) = u_2)) = \frac{1}{2^{2b}}$

(randomness is over the sampling of the hash function from $\mathcal{H}$)

Assume $S = \{f_1, ..., f_k\} \subseteq \{0,1\}^n$ is a set of k arbitrary elements to be hashed using the hash function $h \in \{0,1\}^n \to \{0,1\}^b$ which is sampled from a pairwise independent hashing family. Let c_{ij} be an indicator random variable for the collision of $f_i, f_j, i \neq j$, i.e.,

$$c_{ij} = \begin{cases} 1 & h(f_i) = h(f_j) \\ 0 & h(f_i) \neq h(f_j) \end{cases}, \text{ for } i \neq j \in [k].$$

Lemma B.2.2. *The expectation of the total number of collisions* $C = \sum_{i \neq j \in [k]} c_{ij}$ *in a pairwise independent hashing scheme is given by:* $\mathbb{E}[C] = \frac{(k-1)^2}{2^b}$.

Proof.

$$\begin{aligned} \mathbb{E}[C] &= \sum_{i \neq j \in [k]} \mathbb{E}[c_{ij}] \\ &= \sum_{i \neq j \in [k]} \sum_{u \in \{0,1\}^b} P((h(f_i) = u) \wedge (h(f_j) = u)) \\ &= \frac{(k-1)^2}{2^b}, \end{aligned}$$

where we have applied the linearity of expectation. □

The next Lemma shows that the hashing scheme of HashWH introduced in Section 3.4.1 is also a pairwise independent hashing scheme. However, there is one small caveat: the hash function always maps $0 \in \{0,1\}^n$ to $0 \in \{0,1\}^b$ which violates property 1 of the pairwise independence Definition B.2.1. If we remove 0 from the domain then it becomes a pairwise independent hashing scheme.

Lemma B.2.3. *The hash function used in the hashing procedure of our method* HashWH, *i.e.,* $h(.) = \mathbf{\alpha}^\top(.)$ *where* $\mathbf{\alpha} \sim \mathcal{U}_{\{0,1\}^{n \times b}}$ *is a hashing matrix whose elements are sampled independently and uniformly at random (with probability* $\frac{1}{2}$*) from* $\{0,1\}$*, is pairwise independent if we exclude* $f = 0$ *from the domain.*

Proof. Note that for any input $f \in \{0,1\}^n, f \neq 0$, its hash $\mathbf{\alpha}^\top f$ is a linear combination of columns of $\mathbf{\alpha}^\top$, where f determines the columns. We denote i^{th} column of $\mathbf{\alpha}^\top$ by $\mathbf{\alpha}^\top_{\bullet i}$. Let f be non-zero in

$t \geq 1$ positions (bits) $\{i_1, ..., i_t\} \subseteq [n]$. The value of $h(f)$ is equal to the summation of the columns of $\mathbf{œ}^\top$ that corresponds to those t positions: $\mathbf{œ}^\top_{\bullet i_1}, \cdots, \mathbf{œ}^\top_{\bullet i_t}$. Let $u \in \{0,1\}^b$ be an arbitrary bucket. The probability the sum of the columns equals u is $\frac{1}{2^b}$ as all sums are equally likely i.e.

$$P(h(x) = u) = \frac{1}{2^b}$$

Let $f_1, f_2 \neq 0, f_1 \neq f_2 \in \{0,1\}^n$ be a pair of distinct non-zero inputs. Since f_1 and f_2 differ in at least one position (bit), $h(f_1)$ and $h(f_2)$ are independent random variables. Therefore, for any arbitrary $u_1, u_2 \in \{0,1\}^b$

$$\begin{aligned} &P(h(f_1) = u_1 \wedge h(f_2) = u_2) \\ =&P(h(f_1) = u_1)P(h(f_2) = u_2) = \frac{1}{2^{2b}} \end{aligned}$$

□

Lemmas B.2.2 and B.2.3 imply that the expected total number of collisions C in hashing frequencies of the top k coefficients of g in our hashing scheme is also equal to: $\mathbb{E}[C] = \frac{(k-1)^2}{2^b}$. Our guarantee shows that the number of collisions goes down linearly in the number of buckets 2^b.

Finally, let f_1 be an important frequency i.e. one with a large magnitude $|\widehat{g}(f_1)|$. By independently sampling a new hashing matrix $\mathbf{œ}$ at each round of back-prop, we avoid always hashing this frequency into the same bucket as some other important frequency. By a union bound on the pairwise independence property, the probability that a frequency f_1 collides with any other frequency $f_2, \ldots, f_k$ is upper bounded by $\frac{k-1}{2^b}$. Therefore, over T rounds of back-prop the number of times this frequency collides follows a binomial distribution with $p \leq \frac{k-1}{2^b}$ ($\frac{k-1}{2^b} < 1$ for a large enough b). We denote the number of times frequency f_1 collides over the T rounds as C_{f_1}. The expected number of collisions is $\mu \triangleq Tp$ which goes down linearly in the number of buckets. With a Chernoff bound we can say that roughly speaking, the number of collisions we expect can not be too much larger than a fraction p of the T rounds.

By a Chernoff's bound we have:

$$P(C_{f_1} \geq (1+\delta)\mu) \leq e^{-\frac{\delta^2\mu}{2+\delta}}$$

where $\mu = Tp$ as mentioned before

For examples setting $\delta = 1$

$$P(C_{f_1} \geq 2\mu) \leq e^{-\frac{\mu}{3}}$$

As $T \to \infty$ this probability goes to zero. This means that the probability that the number of times the frequency collides during the T rounds to not be more than a fraction $(1+\delta)p = 2p$ of the time is, for all practical purposes, essentially zero. Building on this intuition, we can see that for any fixed $0 < \epsilon < 1$, setting $b = \log_2(\frac{k-1}{\epsilon})$ guarantees that collision of a given frequency happens on average a fraction ϵ of the T rounds and not much more.

B.2.3 EN-S *details*

To avoid computing the exact Fourier spectrum of the network at each back-propagation iteration in FULLWH, [47] suggest an iterative regularization technique to enforce sparsity in the Fourier spectrum of the network called EN-S.

We first briefly describe the Alternating Direction Method of Multipliers [107] (ADMM) which is an algorithm that is used to solve convex optimization problems. This algorithm is used to derive EN-S. Finally, we discuss EN-S itself and highlight the advantages of using our method HASHWH over it.

ADMM. Consider the following separable optimization objective:

$$\min_{\boldsymbol{x}\in\mathbb{R}^n, \boldsymbol{z}\in\mathbb{R}^m} f(\boldsymbol{x}) + g(\boldsymbol{z})$$
$$\text{subject to } \mathbf{A}\boldsymbol{x} + \mathbf{B}\boldsymbol{z} = \mathbf{c},$$

where $\mathbf{A} \in \mathbb{R}^{p\times n}$, $\mathbf{B} \in \mathbb{R}^{p\times m}$, $\mathbf{c} \in \mathbb{R}^p$, and $f \in \mathbb{R}^m \to \mathbb{R}$ and $g \in \mathbb{R}^n \to \mathbb{R}$ are arbitrary *convex* functions The augmented Lagrangian of this objective is formed as:

$$L_\rho(\boldsymbol{x}, \boldsymbol{z}, \mathbf{fl}) = f(\boldsymbol{x}) + g(\boldsymbol{z}) + \mathbf{fl}^\top(\mathbf{A}\boldsymbol{x} + \mathbf{B}\boldsymbol{z} - \mathbf{c}) + \frac{\rho}{2}\|\mathbf{A}\boldsymbol{x} + \mathbf{B}\boldsymbol{z} - \mathbf{c}\|_2^2,$$

where $\mathbf{fl} \in \mathbb{R}^p$ are the dual variables.

Alternating Direction Method of Multipliers [107], or in short *ADMM*, optimizes the augmented Lagrangian by alternatively minimizing it over the two variables $\boldsymbol{x}$ and $\boldsymbol{z}$ and applying a dual variable update:

$$\begin{aligned}
\boldsymbol{x}^{k+1} &= \operatorname*{argmin}_{\boldsymbol{x}\in\mathbb{R}^n} L_\rho(\boldsymbol{x}, \boldsymbol{z}^k, \mathbf{fl}^k) && (\boldsymbol{x}\text{-minimization})\\
\boldsymbol{z}^{k+1} &= \operatorname*{argmin}_{\boldsymbol{z}\in\mathbb{R}^m} L_\rho(\boldsymbol{x}^{k+1}, \boldsymbol{z}, \mathbf{fl}^k) && (\boldsymbol{z}\text{-minimization})\\
\mathbf{fl}^{k+1} &= \mathbf{fl}^k + \rho(\mathbf{A}\boldsymbol{x}^{k+1} + \mathbf{B}\boldsymbol{z}^{k+1} - \mathbf{c}) && (\text{dual var. update})
\end{aligned}$$

In a slightly different formulation of ADMM, known as *scaled-dual* ADMM, the dual variable can be scaled which results in a similar optimization scheme:

$$\begin{aligned}
\boldsymbol{x}^{k+1} &= \operatorname*{argmin}_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) + \frac{\rho}{2}\|\mathbf{A}\boldsymbol{x} + \mathbf{B}\boldsymbol{z}^k - \mathbf{c} + \mathbf{fl}^k\|_2^2\\
\boldsymbol{z}^{k+1} &= \operatorname*{argmin}_{\boldsymbol{z}\in\mathbb{R}^m} g(\boldsymbol{z}) + \frac{\rho}{2}\|\mathbf{A}\boldsymbol{x}^{k+1} + \mathbf{B}\boldsymbol{z} - \mathbf{c} + \mathbf{fl}^k\|_2^2\\
\mathbf{fl}^{k+1} &= \mathbf{fl}^k + \mathbf{A}\boldsymbol{x}^{k+1} + \mathbf{B}\boldsymbol{z}^{k+1} - \mathbf{c}
\end{aligned} \tag{B.2}$$

Using ADMM, one can decouple the joint optimization of two separable groups of parameters into two alternating separate optimizations for each individual group.

EN-S. To apply ADMM, [47] reformulate the FULLWH loss, by introducing a new variable z and adding a constraint such that it is equal to the Fourier spectrum:

$$\begin{aligned}
&\mathcal{L}_{EN-S} = \mathcal{L}_{net} + \lambda\|z\|_1\\
&\text{subject to: } z = \widehat{\mathbf{g}} = \mathbf{H}_n\mathbf{g}(\mathbf{X})
\end{aligned}$$

, where $g_{\boldsymbol{\theta}}$ is the neural network parameterized by $\boldsymbol{\theta}$, $\mathbf{H}_n \in \{0,1\}^{2^n\times 2^n}$ is the Hadamard matrix, and $\mathbf{X} \in \{0,1\}^{2^n\times n}$ is the matrix of the enumeration over all points on the Boolean cube $\{0,1\}^n$.

They use the scaled-dual ADMM (B.2) followed by a few further adjustments to reach the following alternating scheme for optimization of $\mathcal{L}_{EN-S}$:

$$\begin{aligned}
\boldsymbol{\theta}^{k+1} &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \mathcal{L}_{net} + \frac{\rho}{2}\|\mathbf{g}_{\boldsymbol{\theta}}(\mathbf{X}_T) - \mathbf{H}_T \boldsymbol{z}^k + \mathbf{fl}^k\|_2^2 \\
\boldsymbol{z}^{k+1} &= \underset{\boldsymbol{z}}{\operatorname{argmin}}\, \lambda\|\boldsymbol{z}\|_0 + \frac{\rho}{2}\|\mathbf{g}_{\boldsymbol{\theta}^{k+1}}(\mathbf{X}_T) - \mathbf{H}_T \boldsymbol{z} + \mathbf{fl}^k\|_2^2 \\
\mathbf{fl}^{k+1} &= \mathbf{fl}^k + \mathbf{g}_{\boldsymbol{\theta}^{k+1}}(\mathbf{X}_T) - \mathbf{H}_T \boldsymbol{z}^{k+1},
\end{aligned} \tag{B.3}$$

where $\mathbf{X}_T \in \{0,1\}^{O(2^m n)\times n}$ is the input enumeration matrix $\mathbf{X} \in \{0,1\}^{2^n \times n}$ sub-sampled at $O(2^m n)$ rows, $\mathbf{H}_T \in \{0,1\}^{O(2^m n)\times n}$ is the Hadamard matrix $\mathbf{H}_n \in \{0,1\}^{2^n \times 2^n}$ subsampled at similar $O(2^m n)$ rows, and $\mathbf{fl} \in \mathbb{R}^{O(2^m n)}$ is the dual variable. We will introduce the hash size parameter m momentarily.

Using the optimization scheme (B.3), they decouple the optimization of $\mathcal{L}_{EN-S}$ into two separate alternating optimizations: 1) minimizing $\mathcal{L}_{net}$ by fixing $\boldsymbol{z}$ and optimizing network parameters using SGD for an epoch ($\boldsymbol{\theta}$-minimization), 2) fixing θ and computing a sparse Fourier spectrum approximation of the network at the end of each epoch and updating the dual variable ($\boldsymbol{z}$-minimization).

To approximate the sparse Fourier spectrum of the network at $\boldsymbol{z}$-minimization step, they use the "SPRIGHT" algorithm [18]. SPRIGHT requires $O(2^m n)$ samples from the network to approximate its Fourier spectrum and runs with the complexity of $O(2^m n^3)$, where m is the hash size used in the algorithm (the equivalent of b in our setting). In EN-S optimization scheme (B.3), these $O(2^m n)$ inputs are denoted by the matrix $\mathbf{X}_T \in \{0,1\}^{O(2^m n)\times n}$, and are fixed during the whole optimization process. This requires the computation of the network output on these $O(2^m n)$ inputs at each back-prop iteration in $\boldsymbol{\theta}$-minimization, as well as at the end of each epoch to run SPRIGHT in $\boldsymbol{z}$-minimization.

EN-S VS. HASHWH. The hashing done in our method, HASHWH, is basically the first step of many (if not all) sparse Walsh-Hadamard transform approximation methods [15, 16, 18, 84], including SPRIGHT [18] that is used in EN-S. In the task of sparse Fourier spectrum approximation, further, extra steps are done to infer the *exact* frequencies of the support and their associated Fourier coefficients. These steps are usually computationally expensive. Here, since we are only interested

in the $L1$-norm of the Fourier spectrum of the network and are not necessarily interested in retrieving the exact frequencies in its support, we found the idea of approximating it with the $L1$-norm of the Fourier spectrum of our hash function compelling. This is the core idea behind HASHWH which lets us stick to the FULLWH formulation using a scalable approximation of the $L1$-norm of the network's Fourier spectrum.

From the mere computational cost perspective, EN-S requires a rather expensive sparse Fourier spectrum approximation of the network at the end of each epoch. We realized, one bottleneck of their algorithm was the evaluation of the neural network on the required time samples of their sparse Fourier approximation algorithm. We re-implemented this part on a GPU to make it substantially faster. Still, we empirically observe that more than half of the run time of each EN-S epoch is spent on the Fourier transform approximation. Furthermore, in EN-S, the network output needs to be computed for $\Omega(2^m n)$ samples at each back-prop iteration.

On the contrary, in HASHWH, the network Fourier transform approximation is not needed anymore. We only compute the network output on precisely 2^b samples at each round of back-propagation to compute the Fourier spectrum of our sub-sampled neural network. Remember that our b is roughly equivalent to their m. Since the very first step in their sparse Fourier approximation step is a hashing step into 2^m buckets.

Let us compare our method with EN-S more concretely. For the sake of simplicity, we ignore the network sparse Fourier approximation step (z-minimization) that happens at the end of each epoch for EN-S and assume their computational complexity is only dominated by the $\Omega(2^m n)$ evaluations made during back-prop. In order to use the same number of samples as EN-S, we can set our hashing size to $b = m + log(n) + c$, where c is a constant which we found in practice to be at least $c \geq 3$. In the case of our avGFP experiment, this would be for instance $b \geq 18$ in HASHWH for EN-S with $m = 7$. There, we outperformed EN-S using $b \in \{7, 10, 13, 16\}$ in terms of R^2-score. Note that even with $b = 18$ we are still at least two times faster than EN-S as we do not go the extra mile of approximating the Fourier spectrum of the network at each epoch.

B.3 DATASETS

We list all the datasets used in the real dataset Section 3.5.2.

ENTACMAEA QUADRICOLOR FLUORESCENT PROTEIN. (ENTACMAEA) [52] study the fluorescence brightness of all 2^{13} distinct variants of the Entacmaea quadricolor fluorescent protein, mutated at 13 different sites. They examine the goodness of fit (R^2-score) when only using a limited set of frequencies of the highest amplitude. They report that only 1% of the frequencies are enough to describe data with a high goodness of fit ($R^2 = 0.96$), among which multiple high-degree frequencies exist.

GPU KERNEL PERFORMANCE (SGEMM). [57] measures the running time of a matrix product using a parameterizable SGEMM GPU kernel, configured with different parameter combinations. The input has 14 categorical features that we one-hot encode into 40-dimensional binary vectors.

IMMUNOGLOBULIN-BINDING DOMAIN OF PROTEIN G (GB1). [56] study the "fitness" of variants of protein GB1, that are mutated at four different sites. Fitness, in this work, is a quantitative measure of the stability and functionality of a protein variant. Given the 20 possible amino acids at each site, they report the fitness for $20^4 = 160,000$ possible variants, which we represent with one-hot encoded 80-dimensional binary vectors. In a noise reduction step, they included 149,361 data points as is and replaced the rest with imputed fitness values. We use the former, the untouched portion, for our study.

GREEN FLUORESCENT PROTEIN FROM AEQUOREA VICTORIA (AVGFP). [55] estimate the fluorescence brightness of random mutations over the green fluorescent protein sequence of Aequorea victoria (avGFP) at 236 amino acid sites. We transform the data into the boolean space of the absence or presence of a mutation at each amino acid site by averaging the brightness for the mutations with similar binary representations. This converts the original 54,024 distinct amino acid mutations into 49,089 236-dimensional binary data points.

B.4 IMPLEMENTATION TECHNICAL DETAILS

NEURAL NETWORK ARCHITECTURE AND TRAINING We used a 5-layer fully connected neural network including both weights and biases and LeakyReLu as activations in all settings. For training, we used MSE loss as the loss of the network in all settings. We always initialized the networks with Xavier uniform distribution. We fixed 5 random seeds in order to make sure the initialization was the same over different settings. The Adam optimizer with a learning rate of 0.01 was used for training all models. We always used a single Nvidia GeForce RTX 3090 to train each model to be able to fairly compare the runtime of different methods. We did not utilize other regularization techniques such as Batch Normalization or Dropout to limit our studies to analyze the mere effect of Fourier spectrum sparsification. We use networks of different widths in different experiments which we detail in the following:

- *Fourier spectrum evolution:* The architecture of the network is $10 \times 100 \times 100 \times 10 \times 1$.
- *High-dimensional synthetic data:* For each $n \in \{25, 50, 100\}$, the architecture of the network is $n \times 2n \times 2n \times n \times 1$.
- *Real data:* Assuming n to be the dimensionality of the input space, we used the network architecture of $n \times 10n \times 10n \times n \times 1$ for all the experiments except avGFP. For avGFP with $n = 236$, we had to down-size the network to $n \times n \times n \times n \times 1$ to be able to run EN-S on GPU as it requires a significant amount of samples to compute the Fourier transform at each epoch in this dimension scale.

DATA SPLITS In the Fourier spectrum evolution experiment, where we do not report R^2 of the predictions, we split the data into training and validation sets (used for hyperparameter tuning). For the rest of the experiments, we split the data into three splits training, validation, and test sets. We use the validation set for the hyperparameter tuning (mainly the regularizer multiplier λ and details to be explained later) and early stopping. We stop each training after 10 consecutive epochs without any improvements over the best validation loss achieved and use the epoch with the lowest loss for testing the model. All the R^2s reported are the performance of the model on the (hold-out) test set.

For each experiment, we used different training dataset sizes that are explicitly mentioned in the main body of the thesis. Here we list the validation and test dataset sizes:

- *Fourier spectrum evolution:* Given that $n = 10$ and the Boolean cube is of size $2^n = 1024$, we always use the whole data and split it into training and validation sets. For example, for the training set of size 200, we use the rest of the 824 data points as the validation set.
- *High-dimensional synthetic data:* For each training set, we use validation and test sets of five times the size of the training set. That is, for a training set of size $c \cdot 25n$, both of our validation and test sets are of size $c \cdot 125n$.
- *Real data:* After taking out the training points from the dataset, we split the remaining points into two sets of equal sizes one for validation and one for test.

HYPER-PARAMETER TUNING. In all experiments, we hand-picked candidates for important hyper-parameters of each method studied and tested every combination of them, and picked the version with the best performance on the validation set. This includes testing different $\lambda \in \{0.0001, 0.001, 0.01, 0.1\}$ for HashWH, $\lambda \in \{0.01, 0.1, 1\}$ and $\rho \in \{0.001, 0.01, 0.1\}$ for EN-S, and $\lambda \in \{0.01, 0.1, 1\}$ for FULLWH. Furthermore, we also used the following hyper-parameters for the individual experiments:

- *Fourier spectrum evolution:* We used $b \in \{5, 7, 8\}$ for HashWH and $m = 5$ for the EN-S. We did not tune b for HashWH as we reported all the results in order to show the graceful dependence with increasing the hashing matrix size.
- *High-dimensional synthetic data:* We used $b \in \{7, 10, 13\}$ for HashWH and $m = 7$ for the EN-S. We did not tune b for HashWH as we reported each individually.
- *Real data:* We used $b \in \{7, 10, 13\}$ for HashWH and $m = 7$ for EN-S in the Entacmaea, SGEMM, and GB1 experiments. Furthermore, for avGFP, we also considered $b = 16$ for HashWH. Unlike the synthetic experiments, where we reported results for each b individually, we treated b as a hyper-parameter in real data experiments. For Lasso, we tested different L1 norm coefficients of $\lambda \in \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1\}$. For Random

Forest, we tested different numbers of estimators $n_{estimators} \in \{100, 200, 500, 1000\}$, and different maximum depths of estimators $max_{depth} \in \{5, 10, 15\}$ for Entacmaea experiments and $max_{depth} \in \{10, 20, 30, 40, 50\}$ for the rest of experiments. We tested the exact same hyper-parameter candidates we considered for Random Forest in our XGBoost models.

Like common practice, we always picked the hyper-parameter combination resulting in the minimum loss on the validation set, and reported the model's performance on the test (hold-out) dataset.

CODE REPOSITORIES. All the implementations for the methods as well as the experiments are publicly accessible through:

https://github.com/agorji/WHRegularizer.

For EN-S and FullWH regularizers, we used the implementation shared by [47][1]. We applied minor changes so to compute samples needed for the Fourier transform approximation in EN-S on GPU, making it run faster and fairer to compare our method with.

We used the python implementation of `scikit-learn`[2] for our Lasso and Random Forest experiments. We also used the XGBoost[3] python library for our XGBoost experiments.

B.5 ABLATION STUDY DETAILS

To study the effect of the low-degree simplicity bias on generalization on the real-data distribution, we conduct an ablation study by fitting a sparse Fourier transform to two of our datasets. To this end, we fit Random Forest models on Entecamaa and SGEMM datasets, such that they achieve test R^2 of nearly 1 on an independent test set not used in the training. Then, we compute the exact sparse Fourier transform of each Random Forest model, which essentially results in a sparse Fourier function that has been fitted to the training dataset. In our ablation study, finally, we remove frequencies based on two distinct regimes of lower-amplitudes-first and higher-degrees-first and show that the former harms the generalization more. This is against the assumption of simplicity bias being always helpful.

1 https://github.com/amirmohan/epistatic-net
2 https://scikit-learn.org
3 https://xgboost.readthedocs.io

In the next two subsections, we provide the details on how to compute the exact sparse Fourier transform of a Random Forest model as well as finer details of the study setup.

B.5.1 *Ablation study setup*

For the Entacmae dataset, we used a training set of size $5,000$ and a test set of size $2,000$, for which we trained a Random Forest model with 100 trees with maximum depths of 7. For the SGEMM dataset, we used a training set of size $100,000$ and a test set of size $5,000$, for which we trained a Random Forest model with 100 trees with a maximum depth of 10.

B.6 EXTENDED EXPERIMENT RESULTS

Here, we report the extended experiment results containing variations not reported in the main body of the thesis.

B.6.1 *Fourier spectrum evolution*

We randomly generated five synthetic target functions $g^* \in \{0,1\}^{10}$ of degree $d = 5$, each having a single frequency of each degree in its support (the randomness is over the choice of support). We create a dataset by randomly sampling the Boolean cube. Figure B.1 shows the evolution of the Fourier spectrum of the learned neural network function for different methods over training on datasets of multiple sizes $(100, 200, 300, 400)$ limited to the target support. This is the extended version of Figure 3.1a, where we only reported results for the train size of 200. We observe that, quite unsurprisingly, each method shows better performance when trained on a larger training set in terms of converging at earlier epochs and also converging to the true Fourier amplitude it is supposed to. It can also be observed that the Fourier-sparsity-inducing (regularized) methods are *always* better than the standard neural network in picking up the higher-degree frequencies, regardless of the training size.

Figure B.2 goes a step further and shows the evolution of the full Fourier spectrum (not just the target frequencies) over the course of

training. Here, unlike the previous isolated setting where we were able to aggregate the results from different target functions (because of always having a single frequency of each degree in the support), we have to separate the results for each target function $g^* \in \{0,1\}^{10}$, as each has a unique set of frequencies in its support. In Figure 3.1a, we reported the results for one version of the target function g^* and Figure B.2 shows the Fourier spectrum evolution for the other four. We observe that in addition to the spotted inability of the standard neural network in learning higher-degree frequencies, it seems to start picking up erroneous low-degree frequencies as well.

To quantitatively validate our findings, in Figure B.4, we show the evolution of Spectral Approximation Error (SAE) during training on both target support and the whole Fourier spectrum. This is an extended version of Figure 3.2, where we report the results for the train size of 200. Here we also include results when using training datasets of three other train sizes $\{100, 300, 400\}$. We observe that even though the standard neural network exhibits comparable performance to HashWH on the target support when the training dataset size is 100 and 400, it is always underperforming HashWH when broadening our view to the whole Fourier spectrum, regardless of the train size and the hashing size.

From a more fine-grained perspective, in Figure B.3, we categorize the frequencies into subsets of the same degree and show the evolution of SAE and energy on each individual degree. This is an extended version of Figure 3.3, where we reported the results for the training dataset size of 200. Firstly, we observe that using more data aids the standard neural network to eventually put more energy on higher-degree frequencies. But it is still incapable of appropriately learning higher-degree frequencies. Fourier-sparsity inducing methods, including ours, show significantly higher energy in the higher degrees. Secondly, No matter the train size, we note that the SAE on low-degree frequencies first decreases and then increases and the standard neural network starts to overfit. This validates our previous conclusion that the standard neural network learns erroneous low-degree frequencies. Our regularizer prevents overfitting in lower degrees. Its performance of which can be scaled using the hashing size parameter b.

B.6.2 *High-dimensional synthetic data*

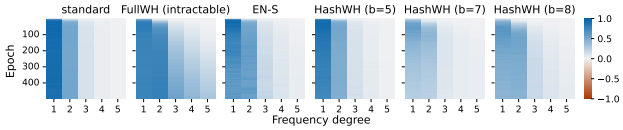
Figure B.5 shows the generalization performance of different methods in learning a synthetic degree $d = 5$ function $g^* \in \{0,1\}^n \to \mathbb{R}$, for $n \in \{25, 50, 100\}$, using train sets of different sizes ($c \cdot 25n, c \in [8]$). For each n we sample three different draws of g^*. This is the extended version of Figure 3.4a, where we only reported the results for the first draw of g^* for each input dimension n. Our regularization method, HashWH, outperforms the standard network and EN-S in all possible combinations of input dimension and dataset sizes, regardless of the draw of g^*. We observe that increasing b in HashWH, i.e. increasing the number of hashing buckets, almost always improves the generalization performance. EN-S, on the other hand, does not show significant superiority over the standard neural network rather than marginally outperforming it in a few cases when $n = 25$. This does not match its performance in the previous section and conveys that it is not able to perform well when increasing the input dimension, i.e., having more features in the data.

To both showcase the computational scalability of our method, HashWH, and compare it to EN-S, we show the achievable performance by the number of training epochs and training time in Figures B.6 to B.8, for all train set sizes and input dimensions individually and limited to the first draw of g^* for each input dimension. This is the extended version of Figure 3.4b where we only reported it for $n = 50$ and the sample size multiplier $c = 5$. We consistently see that the trade-off between the generalization performance and the training time can be directly controlled in HashWH using the parameter b. Furthermore, HashWH is able to *always* exhibit a significantly better generalization performance in remarkably less time, in all versions of b tested. This emphasizes the advantage of our method in not directly computing the approximate Fourier spectrum of the network, which resulted in this gap with EN-S in the run time, that increases as the input dimension n grows.

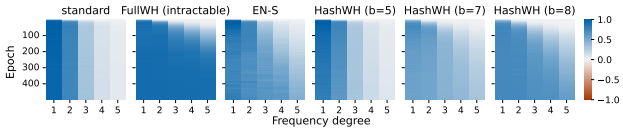
B.6.3 *Real data*

Figure B.9 shows the generalization performance and the training time of different methods, including relevant machine learning benchmarks, in learning four real datasets. It is the extended version of

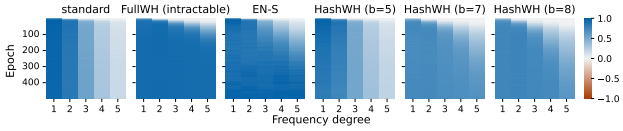
Figure 3.5, where we only reported the generalization performance and not the training time. The training time for neural nets is considered to be the time until overfitting occurs i.e. we do early stopping. *In addition* to superior generalization performance of our method, HASHWH, in most settings, again, we see that it is able to achieve it in significantly less time than EN-S. LASSO is the fastest among the methods but usually shows poor generalization performance.



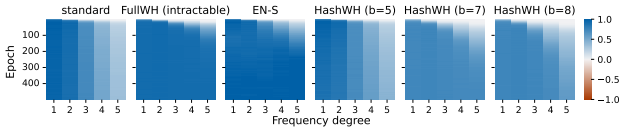
(a) Train size = 100



(b) Train size = 200



(c) Train size = 300



(d) Train size = 400

FIGURE B.1: Evolution of the Fourier spectrum during training limited to the target support, using training sets of different sizes. All synthetic functions have single frequencies of each degree in their support that are all given the amplitude of 1. This is an extended version of Figure 3.1a, where we only reported the results for the train set size 200. It can be observed that the Fourier-sparsity-inducing (regularized) methods are *always* better than the standard neural network in picking up the higher-degree frequencies, regardless of the training size. Each method shows better performance when trained on a larger training set in terms of converging at earlier epochs and also converging to the true Fourier amplitude it is supposed to.

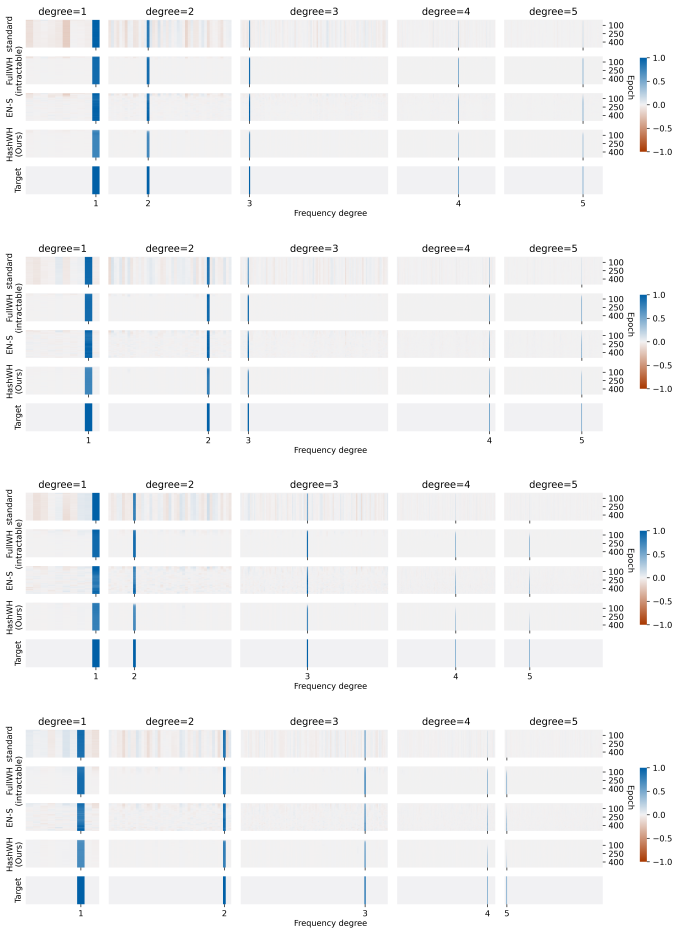
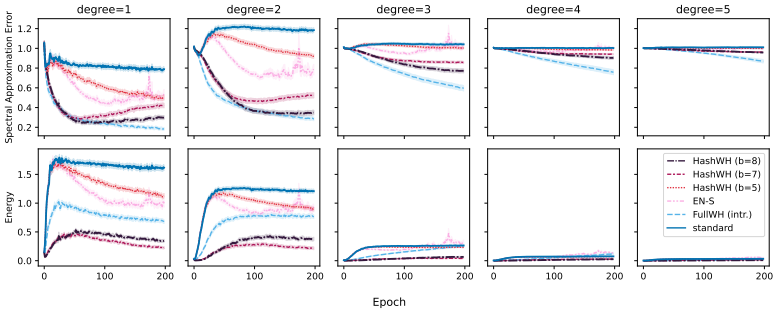
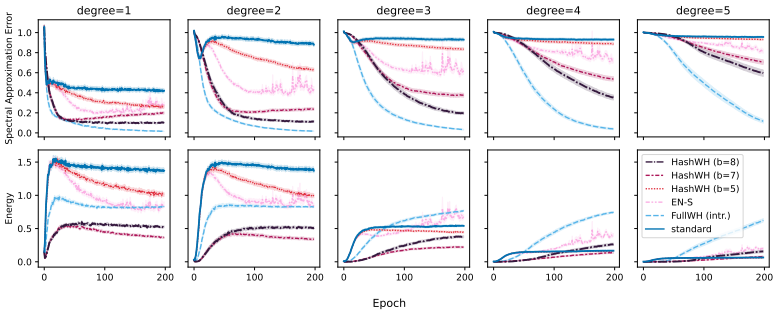


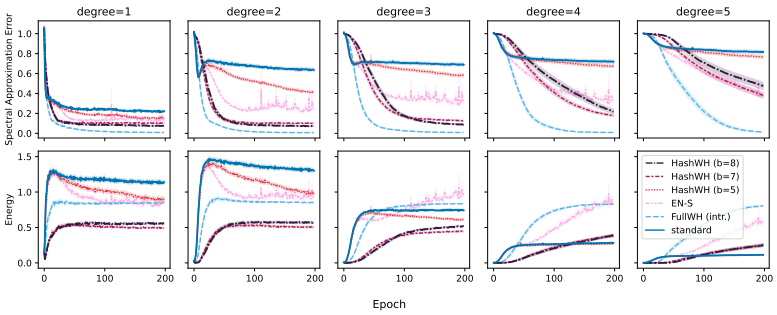
FIGURE B.2: Evolution of the Fourier spectrum in learning a synthetic function $g^* \in \{0,1\}^{10}$ of degree 5 during training, categorized by frequency degree. All synthetic functions used have single frequencies of each degree in their support that are all given the amplitude of 1. We reported the results for one draw of g^* in Figure 3.1b and the four others here, for the training dataset size of 200. In addition to the incapability of the standard neural network in learning high-degree frequencies, they tend to consistently pick up wrong low-degree frequencies. Both of the problems are remedied through our regularizer.



(a) Train size = 100

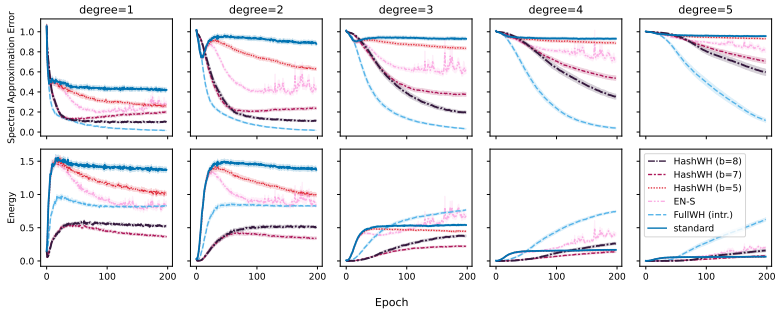


(b) Train size = 200



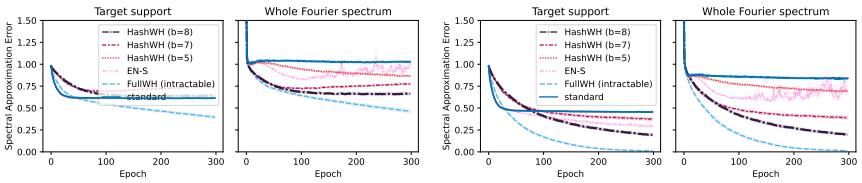
(c) Train size = 300

FIGURE B.3: Evolution of the Spectral Approximation Error (SAE) and energy of the network during training, categorized by frequency degree (continued in the next page).



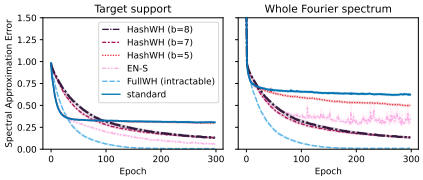
(d) Train size = 400

FIGURE B.3: Evolution of the Spectral Approximation Error (SAE) and energy of the network during training, categorized by frequency degree. This is an extended version of Figure 3.3, where we only reported results for training dataset size 200. Firstly, in a standard neural network, the energy is mostly put on low-degree frequencies as compared to the high-degree frequencies. The energy slightly shifts towards high-degree frequencies when increasing the training dataset size. Our regularizer facilitates the learning of higher degrees in all cases. Secondly, over the lower-degree and regardless of the train size, the standard neural network's energy continues to increase while the SAE first decreases then reverts and increases. This shows that the standard neural network emphasizes energy on erroneous low-degree frequencies and overfits. Our regularizer prevents overfitting in lower degrees.

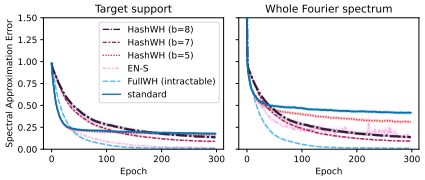


(a) Train size = 100

(b) Train size = 200

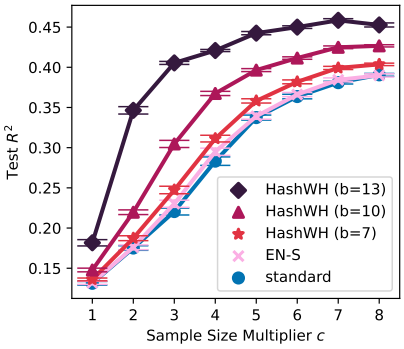


(c) Train size = 300

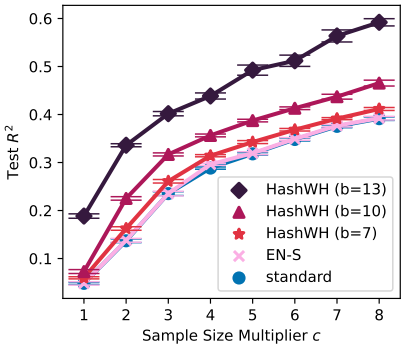


(d) Train size = 400

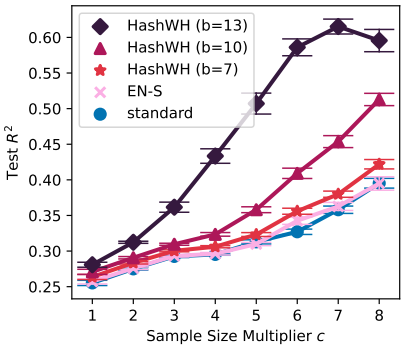
FIGURE B.4: Evolution of the spectral approximation error (SAE) during training. The left plot limits the error to the target support, while the right one considers the whole Fourier spectrum. This is an extended version of Figure 3.2, where we only reported results for train size 200. The standard neural network is able to achieve a lower (better) (train size 100) or somewhat similar (train size 400) SAE on the *target support* compared to our method. However, our method always achieves lower SAE on the *whole Fourier spectrum*, regardless of b used. This shows how our regularisation method is effective in preventing the network from learning the wrong frequencies that are not in the support.



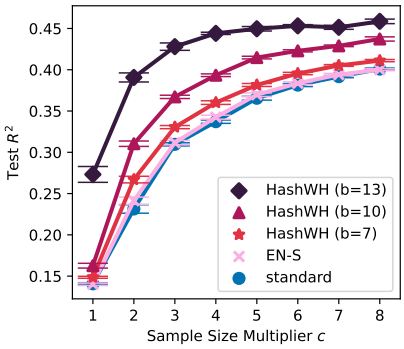
(a) $n = 50$, first draw of g^*



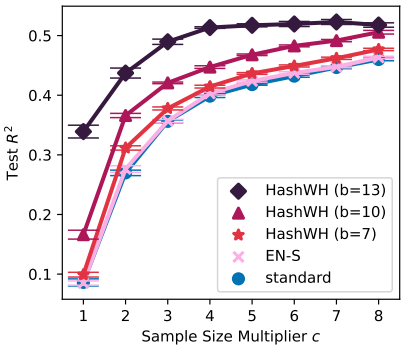
(b) $n = 50$, second draw of g^*



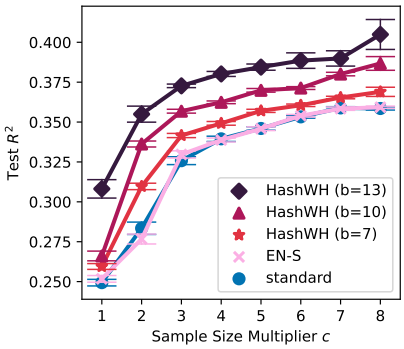
(c) $n = 50$, third draw of g^*



(d) $n = 100$, first draw of g^*

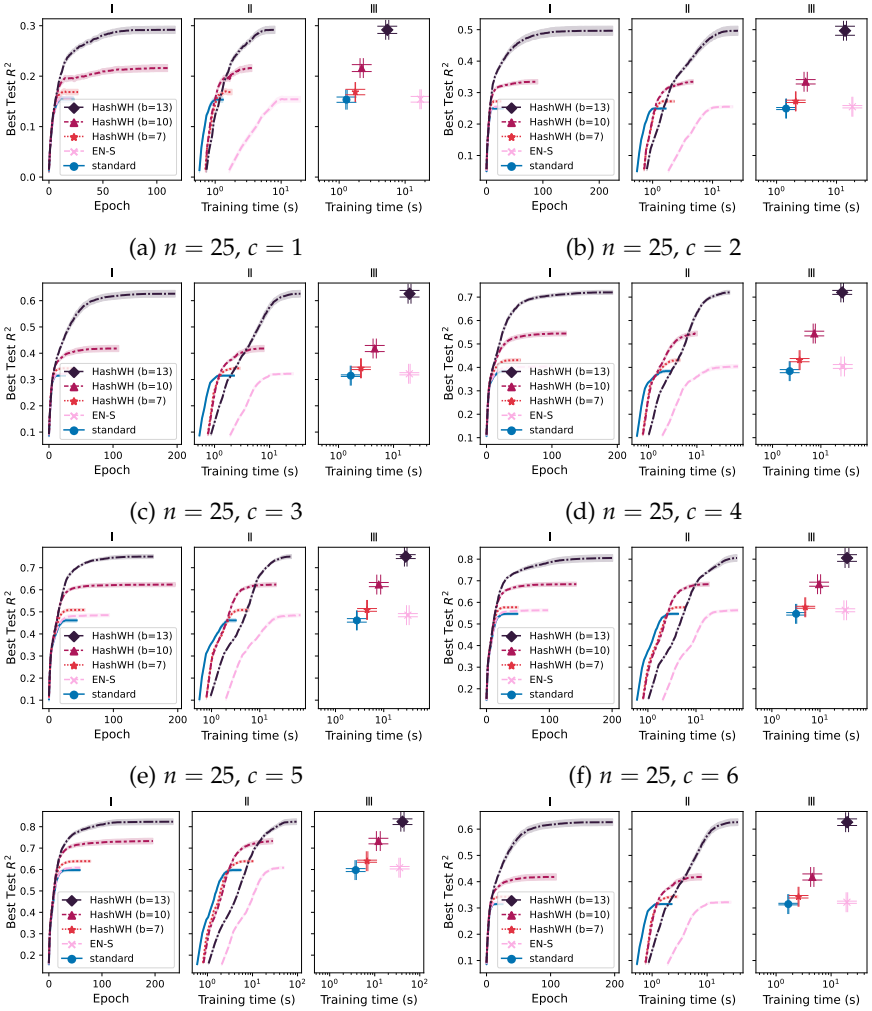


(e) $n = 100$, second draw of g^*



(f) $n = 100$, third draw of g^*

FIGURE B.5: Generalization performance R^2 on a hold-out test set, in learning a synthetic degree 5 function $g^* \in \{0,1\}^n$ for $n \in \{25, 50, 100\}$, using datasets of size $c \cdot 25n$. We report the results of the first draws of g^* for each input dimension in Figure 3.4a and the extended version for all three draws of g^* of different dimensions here. Our method, HASHWH, *always* outperforms the standard neural network and EN-S. We are capable of significantly increasing the outperformance margin by increasing b. EN-S, however, does not show improvement over the standard network in most cases which indicates its diminishing effectiveness as the size of the input dimension grows, i.e., the number of features increases.



(a) $n = 25$, $c = 1$

(b) $n = 25$, $c = 2$

(c) $n = 25$, $c = 3$

(d) $n = 25$, $c = 4$

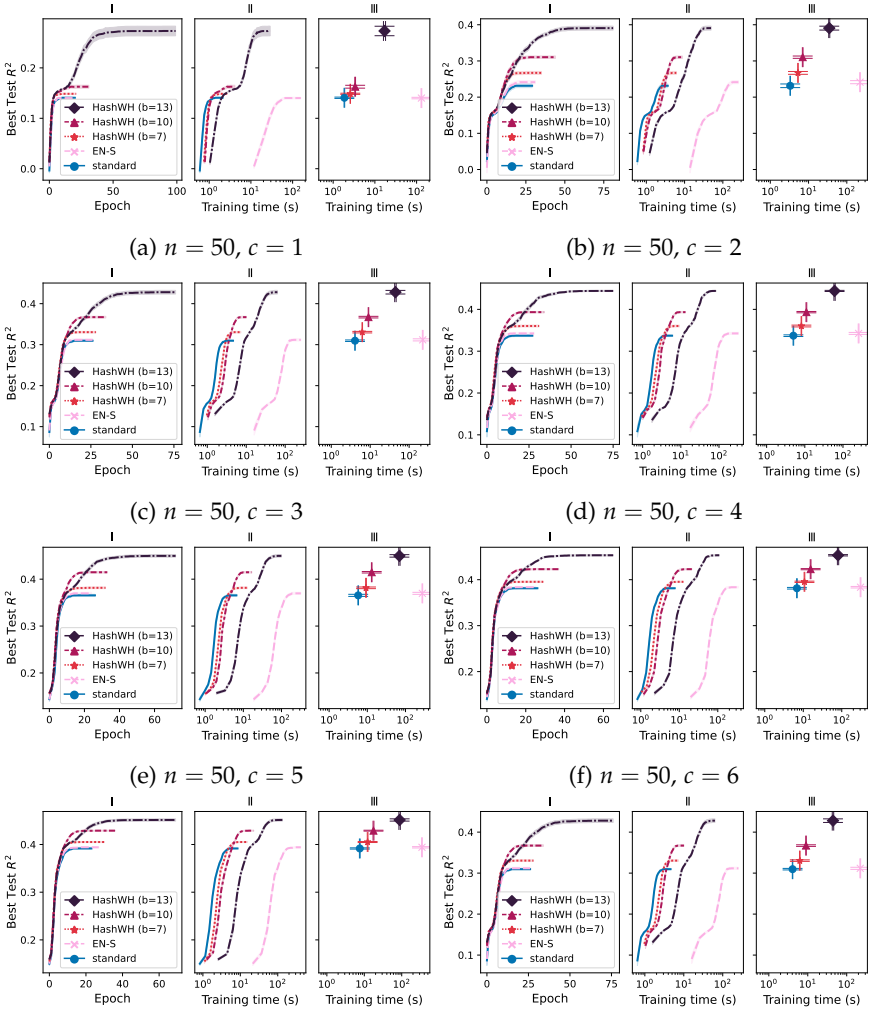
(e) $n = 25$, $c = 5$

(f) $n = 25$, $c = 6$

(g) $n = 25$, $c = 7$

(h) $n = 25$, $c = 8$

FIGURE B.6: Best achievable generalization performance R^2 up to a certain epoch or training time (seconds), in learning a synthetic degree 5 function $g^* \in \{0,1\}^n$, using datasets of size $c \cdot 25n$. This figure is an extended version of Figure 3.4b, where we reported similar plots for $n = 50$ and $c = 5$. Here we report the results for the first draw of g^* with $n = 25$. Our method, HASHWH, *always* outperforms EN-S R^2 score in significantly less time. HASHWH can also be scaled by the choice of b to achieve better generalization performance at th price of higher training times.



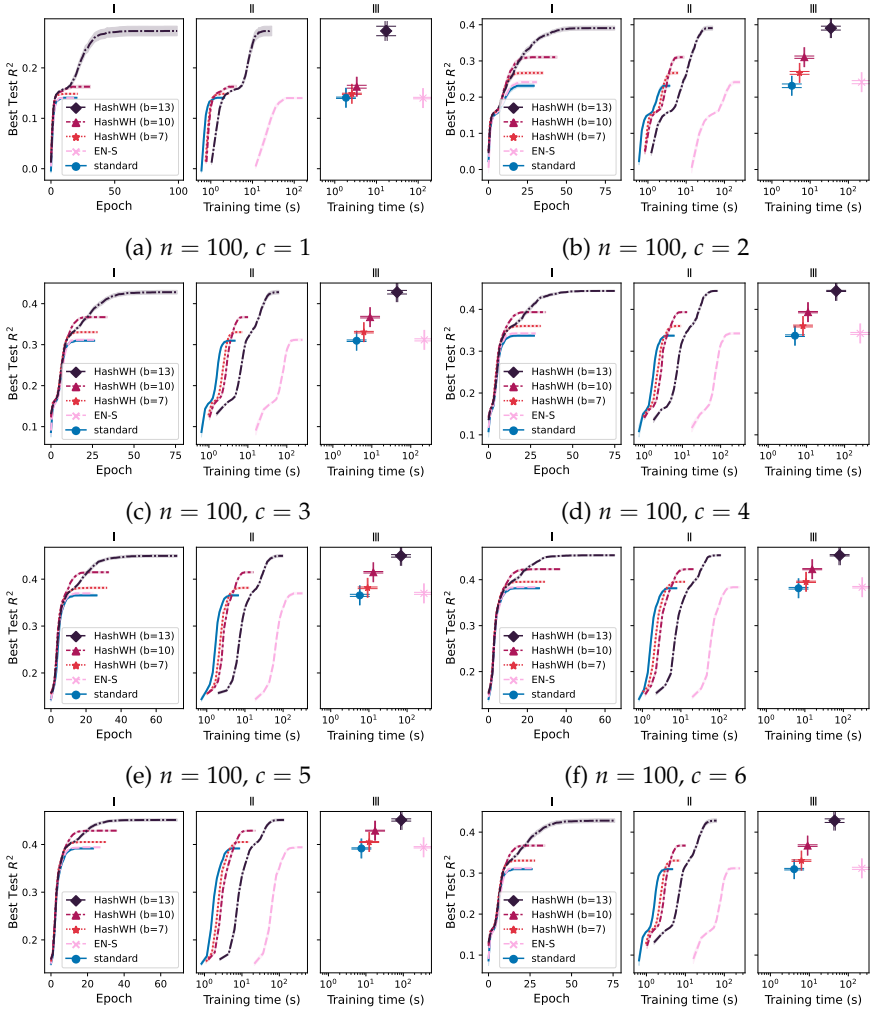
(a) $n = 50$, $c = 1$ (b) $n = 50$, $c = 2$

(c) $n = 50$, $c = 3$ (d) $n = 50$, $c = 4$

(e) $n = 50$, $c = 5$ (f) $n = 50$, $c = 6$

(g) $n = 50$, $c = 7$ (h) $n = 50$, $c = 8$

FIGURE B.7: Best achievable generalization performance R^2 up to a certain epoch or training time (seconds), in learning a synthetic degree 5 function $g^* \in \{0,1\}^n$, using datasets of size $c \cdot 25n$. This figure is an extended version of Figure 3.4b, where we reported similar plots for $n = 50$ and $c = 5$. Here we report the results for the first draw of g^* with $n = 50$. Our method, HASHWH, *always* outperforms EN-S R^2 score in significantly less time. HASHWH can also be scaled by the choice of b to achieve better generalization performance athe price of higher training times.



(a) $n = 100$, $c = 1$

(b) $n = 100$, $c = 2$

(c) $n = 100$, $c = 3$

(d) $n = 100$, $c = 4$

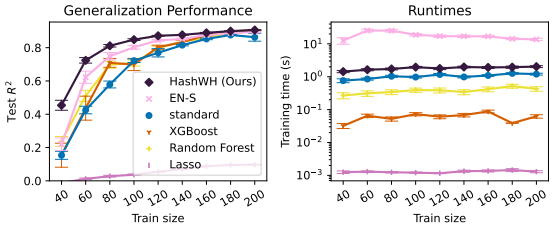
(e) $n = 100$, $c = 5$

(f) $n = 100$, $c = 6$

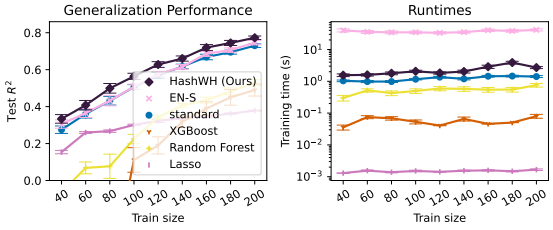
(g) $n = 100$, $c = 7$

(h) $n = 100$, $c = 8$

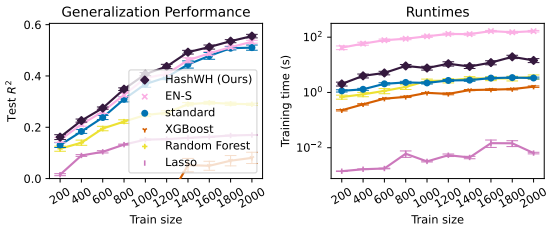
FIGURE B.8: Best achievable generalization performance R^2 up to a certain epoch or training time (seconds), in learning a synthetic degree 5 function $g^* \in \{0,1\}^n$, using datasets of size $c \cdot 25n$. This figure is an extended version of Figure 3.4b, where we reported similar plots for $n = 50$ and $c = 5$. Here we report the results for the first draw of g^* with $n = 100$. Our method, HASHWH, *always* outperforms EN-S R^2 score in significantly less time. HASHWH can also be scaled by the choice of b to achieve better generalization performance at the price of higher training times.



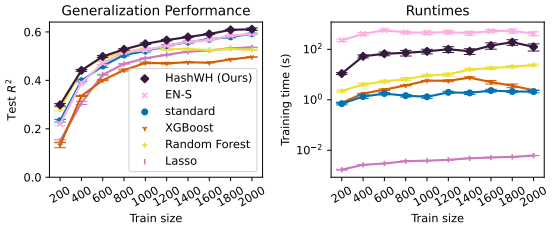
(a) Entacmaea (n=13)



(b) SGEMM (n=40)



(c) GB1 (n=80)



(d) avGFP (n=236)

FIGURE B.9: Generalization performance of standard/regularized neural networks and benchmark ML models on four real datasets. This figure is an extended version of Figure 3.5. It also includes the training times (logarithmically scaled in the plot). Our method is able to achieve the best test R^2s while always training significantly faster than EN-S.

C

APPENDIX: AMORTIZED SHAP VALUES VIA FUNCTION APPROXIMATION

C.1 RELEVANT WORK

C.1.1 *Sparse and low degree Fourier transform algorithms*

We now discuss algorithms that efficiently approximate *general* black-box predictors by a Fourier sparse representation. Let $h : \{0,1\}^n \to \mathbb{R}$ be a any function. We assume we have query access to h. That is, we can arbitrarily pick $x \in \{0,1\}^n$ and query h for its value $h(x)$. Without any further assumptions, computing the Fourier transform requires us to query *exponentially*, to be precise 2^n, many queries: one for every $x \in \{0,1\}^n$. Furthermore, classical Fast Fourier Transform (FTT) algorithms are known to take at least $\Omega(2^n \log(2^n))$ time.

Under the additional assumption that h is k-sparse, works such as [16, 32, 84–86, 108] provide algorithms that obtain the Fourier transform more efficiently. In particular, [85] provide algorithms with query complexity $O(nk)$ and time complexity $O(nk \log k)$ time. Assuming further that the function is of degree $d = o(n)$, the query complexity reduces to $O(kd \log n)$, with run time still polynomial in n, k, d. Crucially, even if the function h is not k-sparse, Algorithm RobustSWHT of [85] yields the best k-sparse approximation in the $\ell_2 - \ell_2$ sense. More precisely, let us denote by $h_k : \{0,1\}^n \to$ the function that is formed by only keeping the top k non-zero Fourier coefficients of h and setting the rest to zero. Then the algorithm returns a $O(k)$-sparse function g such that:

$$\sum_{f \in \{0,1\}^n} (\widehat{g}(f) - \widehat{h}(f))^2 \leq C(1+\epsilon) \min_{\text{all } k\text{-sparse } g} \sum_{f \in \{0,1\}^n} (\widehat{g}(f) - \widehat{h_k}(f))^2,$$

where C is a universal constant. By Parseval's identity, the same holds if the summations were over the time (input) domain instead of the frequency domain.

C.2 PROOFS

C.2.1 *Proof of Lemma 4.3.1*

Proof. We start from Equation (4.2):

$$\begin{aligned}\phi_i^{\Psi_f} &= \sum_{S\subseteq N\setminus\{i\}} \frac{|S|!(n-|S|-1)!}{n!}\cdot\frac{1}{|\mathcal{D}|} \\ &\quad \sum_{(x,y)\in\mathcal{D}} \left((-1)^{\langle f, x^*_{S\cup\{i\}}\oplus x_{N\setminus S\cup\{i\}}\rangle} - (-1)^{\langle f, x^*_S\oplus x_{N\setminus S}\rangle}\right) \\ &= \frac{1}{|\mathcal{D}|}\sum_{(x,y)\in\mathcal{D}}\sum_{S\subseteq N\setminus\{i\}} \frac{|S|!(n-|S|-1)!}{n!} \\ &\quad (-1)^{\langle f_{-i}, x^*_S\oplus x_{N\setminus S\cup\{i\}}\rangle}\left((-1)^{f_i x^*_i} - (-1)^{f_i x_i}\right)\end{aligned}$$

One can see by checking for different values of $x_i, x^*_i, f_i \in \{0,1\}$ that $(-1)^{f_i x^*_i} - (-1)^{f_i x_i} = 2f_i(x_i - x^*_i)$. To determine $(-1)^{\langle f_{-i}, x^*_S\oplus x_{N\setminus S\cup\{i\}}\rangle}$, we partition $N\setminus\{i\}$ into two subsets $A \triangleq \{j\in N | x_j \neq x^*_j, j\neq i, f_j = 1\}$ and $B \triangleq N\setminus A\cup\{i\}$. Doing this, we can factor out $(-1)^{\langle f_{-i}, x_{-i}\rangle}$ and determine the rest of the sign based on the number of indices in S where x and x^* disagree and $f_i = 1$. This is equal to $|A\cap S|$:

$$\phi_i^{\Psi_f} = \frac{2f_i}{|\mathcal{D}|}\sum_{(x,y)\in\mathcal{D}}(x_i - x^*_i)(-1)^{\langle f_{-i}, x_{-i}\rangle}\sum_{S\subseteq N\setminus\{i\}}\frac{|S|!(n-|S|-1)!}{n!}(-1)^{|A\cap S|}$$

A and B partition $N\setminus\{i\}$, therefore we split the inner sum as follows:

$$\begin{aligned}\phi_i^{\Psi_f} &= \frac{2f_i}{n}\sum_{(x,y)\in\mathcal{D}}(x_i - x^*_i)(-1)^{\langle f_{-i}, x_{-i}\rangle} \\ &\quad \sum_{\tilde{B}\subseteq B}\sum_{\tilde{A}\subseteq A}\frac{(|\tilde{A}|+|\tilde{B}|)!(n-(|\tilde{A}|+|\tilde{B}|)-1)!}{n!}(-1)^{|\tilde{A}|}\end{aligned}$$

Since the inner expression only depends on the cardinalities of $\tilde{A}$ and $\tilde{B}$ we can recast the inner sum to be over numbers instead of

subsets by counting the number of times each cardinality appears in the summation:

$$\phi_i^{\Psi_f} = \frac{2f_i}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} (x_i - x_i^*)(-1)^{\langle f_{-i}, x_{-i}\rangle} \tag{C.1}$$

$$\sum_{b=0}^{n-|A|-1} \sum_{a=0}^{|A|} \binom{n-|A|-1}{b} \binom{|A|}{a} \frac{(a+b)!(n-a-b-1)!}{n!} (-1)^a$$

$$= \frac{2f_i}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} (x_i - x_i^*)(-1)^{\langle f_{-i}, x_{-i}\rangle} \sum_{a=0}^{|A|} (-1)^a \sum_{b=0}^{n-|A|-1} \frac{\binom{|A|}{a}\binom{n-|A|-1}{b}}{n\binom{n-1}{a+b}} \tag{C.2}$$

Now we find a closed-form expression for the innermost summation in the above Equation, which is a summation over b where a is fixed:

$$\sum_{b=0}^{n-|A|-1} \frac{\binom{|A|}{a}\binom{n-|A|-1}{b}}{n\binom{n-1}{a+b}} \overset{\text{(i)}}{=} \sum_{b=0}^{n-|A|-1} \frac{\binom{|A|}{a}\binom{n-|A|-1}{b}}{n\frac{\binom{n-1}{|A|}}{\binom{a+b}{a}\binom{n-a-b-1}{|A|-a}}\binom{|A|}{a}\binom{n-|A|-1}{b}}$$

$$= \frac{1}{n\binom{n-1}{|A|}} \sum_{b=0}^{n-|A|-1} \binom{a+b}{a}\binom{n-a-b-1}{|A|-a}$$

$$\overset{\text{(ii)}}{=} \frac{1}{n\binom{n-1}{|A|}} \binom{n}{|A|+1}$$

$$= \frac{1}{|A|+1} \tag{C.3}$$

In Equation (i), we use the following identity $\binom{n-1}{a+b}\binom{a+b}{a}\binom{n-a-b-1}{|A|-a} = \binom{n-1}{|A|}\binom{|A|}{a}\binom{n-|A|-1}{b}$ which can be checked algebraically by the reader by simply writing down each binomial term as factorials and doing the cancellations.

In Equation (ii), we use the following identity $\sum_{b=0}^{n-|A|-1} \binom{a+b}{a}\binom{n-a-b-1}{|A|-a} = \binom{n}{|A|+1}$ which holds because of the following double-counting argument. We want to pick $|A|+1$ items out of n items numbered $1, \ldots, n$. Say we pick $|A|+1$ items and order them according to their number. We condition on the $(a+1)^{th}$ chosen item (after ordering). The $(a+1)^{th}$ chosen item's number can be equal to $(a+b+1)$, where b ranges from 0 to $n-|A|-1$. Choosing the preceding a and trailing $|A|-a$ items can be done in $\binom{a+b}{a}\binom{n-a-b-1}{|A|-a}$ ways.

Based on Equation (C.3), we see that the innermost summation in Equation (C.1) is only dependent on $|A|$. Thus, we rewrite Equation (C.1) as follows:

$$\phi_i^{\Psi_f} = \frac{2f_i}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} (x_i - x_i^*)(-1)^{\langle f_{-i}, x_{-i}\rangle} \frac{(|A|+1) \mod 2}{|A|+1}$$

By absorbing the sign of $(x_i - x_i^*)$ into the $(-1)^{\langle f_{-i}, x_{-i}\rangle}$ term we arrive at Equation (4.3):

$$\phi_i^{\Psi_f} = \frac{2f_i}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} \mathbb{1}_{x_i \neq x_i^*} (-1)^{\langle f, x\rangle} \frac{(|A|+1) \mod 2}{|A|+1}$$

□

C.2.2 *Proof of Theorem 4.3.2*

Proof. Proof of Equation(4.4) simply follows from the fact that SHAP values are linear w.r.t. the explained function. Regarding the computational complexity we restate Equation(4.4)

$$\phi_i^h = \frac{2}{n} \sum_{f\in\text{supp}(h)} \widehat{h}(f) \cdot f_i \sum_{(x,y)\in\mathcal{D}} \mathbb{1}_{x_i \neq x_i^*} (-1)^{\langle f, x\rangle} \frac{(|A|+1) \mod 2}{|A|+1}$$

where $A \triangleq \{j \in N | x_j \neq x_j^*, j \neq i, f_j = 1\}$.

We first do a pre-processing step for amortizing the cost of computing $|A|$: we compute $\tilde{A} \triangleq \{j \in N | x_j \neq x_j^*, f_j = 1\}$ which takes $\Theta(|\mathcal{D}|n)$ flops.

We assume we are computing the whole vector $\Phi^h = (\Phi_1^h, \ldots, \Phi_n^h)$, that is we are compute SHAP values for all $i \in [n]$ at the same time. Going back to the inner summation above, computing A (and $|A|$) for different values of $i \in [n]$ is $\Theta(n)$ if we utilize the pre-computed $\tilde{A}$. The inner product $\langle f, x\rangle$ is not dependent on i and is $\Theta(n)$ flops. Computing $\mathbb{1}_{x_i \neq x_i^*}$ for different values of $i \in [n]$ is $\Theta(n)$. Therefore, the inner expression of the summand takes $\Theta(n)$ flops for a fixed data-point $x \in \mathcal{D}$ and $f \in \text{supp}(h)$.

Computing the inner sum for any fixed frequency $f \in \text{supp}(h)$ is $\Theta(n|\mathcal{D}|)$, because we are summing over $|\mathcal{D}|$ vectors each of size n

(the vector which holds SHAP value for each $i \in [n]$). Moving on to the outer sum each evaluation of the inner sum is $\Theta(n|\mathcal{D}|)$ and it results in a vector of size n (one element for each SHAP value). The multiplication of $\widehat{h}(f) \cdot f_i$ is $\Theta(n)$. Therefore the cost of the inner sum dominates i.e. $\Theta(n|\mathcal{D}|)$. Since we are summing over the whole support the total number of flops is: $\Theta(n|\mathcal{D}|k)$ where $k = |\text{supp}(h)|$. □

C.3 DATASETS

We list all the datasets used in the Experiments Section 4.4.

ENTACMAEA QUADRICOLOR FLUORESCENT PROTEIN. (ENTACMAEA) [52] study the fluorescence brightness of all 2^{13} distinct variants of the Entacmaea quadricolor fluorescent protein, mutated at 13 different sites.

GPU KERNEL PERFORMANCE (SGEMM). [57] measures the running time of a matrix product using a parameterizable SGEMM GPU kernel, configured with different parameter combinations. The input has 14 categorical features. After one-hot encoding the dataset is 40-dimensional.

IMMUNOGLOBULIN-BINDING DOMAIN OF PROTEIN G (GB1). [56] study the "fitness" of variants of protein GB1, that are mutated at four different sites. Fitness, in this work, is a quantitative measure of the stability and functionality of a protein variant. Given the 20 possible amino acids at each site, they report the fitness for $20^4 = 160,000$ possible variants, which we represent with one-hot encoded 80-dimensional binary vectors.

GREEN FLUORESCENT PROTEIN FROM AEQUOREA VICTORIA (AVGFP). [55] estimate the fluorescence brightness of random mutations over the green fluorescent protein sequence of Aequorea victoria (avGFP) at 236 amino acid sites. We transform the amino acid features into binary features indicating the absence or presence of a mutation at each amino acid site. This converts the original 54,024 distinct amino acid sequences of length 236 into 49,089 236-dimensional binary data points.

C.4 EXPERIMENT DETAILS

All experiment code will be open-sourced once the double-blind review process is over. We have access to one NVIDIA GeForce GTX 1080 Ti GPU.

For the Entacmaea and SGEMM datasets, we train fully connected neural networks with 3 hidden layers containing 300 neurons each. The network is trained using the means-squared loss and ADAM optimizer with a learning rate of 0.01. For GB1 we train ensembles of trees models of varying depths using the random forest algorithm using the sklearn library [109]. For avGFP we train again, ensembles of trees models with 10 trees of varying depths using the cat-boost algorithm/library[33]. All other setting are set to the default in both cases. Model accuracy for different depths are plotted in Figure C.1.

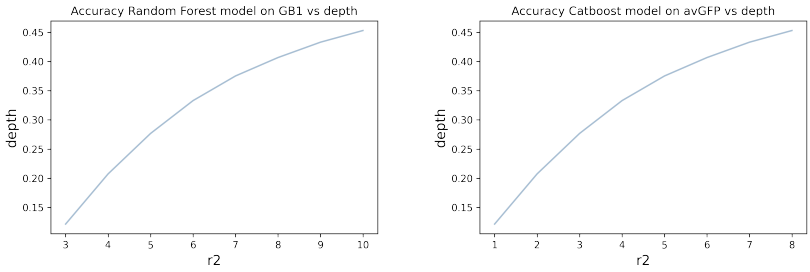


FIGURE C.1: Model accuracy of tree models evaluated on the test set for different depths

C.4.1 *Black-box*

Regarding the first step of the FOURIERSHAP algorithm which is computing a sparse Fourier approximation of the black-box model. We use a GPU implementation, written by ourselves using the JAX library, of a sparse Walsh-Hadamard Transform (sparse WHT) a.k.a Fourier transform algorithm [85] for each of the four trained models.

Regarding the second step of FOURIERSHAP which utilizes the Fourier approximation to compute SHAP values using Equation (4.4). We again implement this step on a GPU using the JAX [87] library. For

each model to be explained, we choose four different values for the number of background samples and four different values for the number of query points to be explained, resulting in a total of 16 runs of FourierSHAP for each model. Error bars capture these variations. The set of values used for number of background samples and number of query points for different models is as follows:

ENTACMAEA, SGEMM, AND GB1: Number of background samples: $\{10, 20, 30, 40\}$, Number of query points: $\{10, 20, 30, 40\}$.

AVGFP: Number of background samples: $\{10, 20, 30, 40\}$, Number of query points: $\{1, 2, 3, 4\}$.

For KernelShap, we use the standard library provided by the writers of the paper [65] [1] with its default settings. KernelShap is written in C and is to our knowledge the fastest implementation of this algorithm.

LinRegShap, is a variance-reduced version of KernelSHAP [72]. We again use the implementation of the original writers [2]. Their implemntation includes automatic detection of the convergence of stochastic sampling which is meant to speed up the algorithm by taking less samples from the black box

For DeepLift we again we use the library of Lundberg & Lee [65] [3] with default settings. In this setting the neural network is passed to the algorithm on a GPU to make sure this algorithm is as fast as possible.

C.4.2 *White-box*

We fit random forests of depths ranging from 3 to 8 with 50 estimators on 90% of the Entacmaea dataset, and use the rest of the data to measure the model accuracy R^2 (reported in Table 4.1). For each model, we compute SHAP values using three methods of TreeSHAP [4], FastTreeSHAP [5], and our FastFourierSHAP in five independant runs, and report the average speedup as well as the standard deviation

1 https://github.com/slundberg/shap
2 https://github.com/iancovert/shapley-regression
3 https://github.com/slundberg/shap
4 https://github.com/slundberg/shap
5 https://github.com/linkedin/FastTreeSHAP

in Table 4.1. We always use 100 datapoints as the background data, and 100 datapoints as the query inputs to compute the SHAP values for.

BIBLIOGRAPHY

1. Hornik, K., Stinchcombe, M. & White, H. Multilayer feedforward networks are universal approximators. *Neural networks* **2**, 359 (1989).
2. Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems* **2**, 303 (1989).
3. Domingos, P. Every model learned by gradient descent is approximately a kernel machine. *arXiv preprint arXiv:2012.00152* (2020).
4. Lee, J., Bahri, Y., Novak, R., Schoenholz, S. S., Pennington, J. & Sohl-Dickstein, J. Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165* (2017).
5. Lee, J., Bahri, Y., Novak, R., Schoenholz, S. S., Pennington, J. & Sohl-Dickstein, J. *Deep Neural Networks as Gaussian Processes* in *International Conference on Learning Representations* (2018).
6. Lee, J., Xiao, L., Schoenholz, S., Bahri, Y., Novak, R., Sohl-Dickstein, J. & Pennington, J. Wide neural networks of any depth evolve as linear models under gradient descent. *Advances in neural information processing systems* **32** (2019).
7. Jacot, A., Gabriel, F. & Hongler, C. *Neural Tangent Kernel: Convergence and Generalization in Neural Networks* in *Advances in Neural Information Processing Systems* **31** (Curran Associates, Inc., 2018).
8. Arpit, D., Jastrzębski, S., Ballas, N., Krueger, D., Bengio, E., Kanwal, M. S., Maharaj, T., Fischer, A., Courville, A., Bengio, Y. & Lacoste-Julien, S. *A Closer Look at Memorization in Deep Networks* in *Proceedings of the 34th International Conference on Machine Learning* ISSN: 2640-3498 (PMLR, 2017), 233.
9. Nakkiran, P., Kaplun, G., Kalimeris, D., Yang, T., Edelman, B. L., Zhang, F. & Barak, B. *SGD on neural networks learns functions of increasing complexity* in *Proceedings of the 33rd International Conference on Neural Information Processing Systems* (2019), 3496.

10. Valle-Perez, G., Camargo, C. Q. & Louis, A. A. *Deep learning generalizes because the parameter-function map is biased towards simple functions* in *International Conference on Learning Representations* (2019).

11. Kalimeris, D., Kaplun, G., Nakkiran, P., Edelman, B., Yang, T., Barak, B. & Zhang, H. *SGD on Neural Networks Learns Functions of Increasing Complexity* in *Advances in Neural Information Processing Systems* **32** (Curran Associates, Inc., 2019).

12. Yang, G. & Salman, H. *A Fine-Grained Spectral Perspective on Neural Networks* arXiv:1907.10599 [cs, stat]. 2020.

13. Mansour, Y. *Learning Boolean functions via the Fourier transform* in *Theoretical advances in neural computation and learning* (Springer, 1994), 391.

14. Kushilevitz, E. & Mansour, Y. *Learning decision trees using the Fourier spectrum* in *Proceedings of the twenty-third annual ACM symposium on Theory of computing* (1991), 455.

15. Amrollahi, A., Zandieh, A., Kapralov, M. & Krause, A. *Efficiently Learning Fourier Sparse Set Functions* in *Advances in Neural Information Processing Systems* **32** (Curran Associates, Inc., 2019).

16. Scheibler, R., Haghighatshoar, S. & Vetterli, M. A fast Hadamard transform for signals with sublinear sparsity in the transform domain. *IEEE Transactions on Information Theory* **61**, 2115 (2015).

17. Li, X. & Ramchandran, K. *An Active Learning Framework using Sparse-Graph Codes for Sparse Polynomials and Graph Sketching* in *Advances in Neural Information Processing Systems* **28** (Curran Associates, Inc., 2015).

18. Li, X., Bradley, J. K., Pawar, S. & Ramchandran, K. *SPRIGHT: A Fast and Robust Framework for Sparse Walsh-Hadamard Transform* arXiv:1508.06336 [cs, math]. 2015.

19. Wendler, C., Amrollahi, A., Seifert, B., Krause, A. & Püschel, M. *Learning set functions that are sparse in non-orthogonal Fourier bases* in *Proceedings of the AAAI Conference on Artificial Intelligence* **35** (2021), 10283.

20. Püschel, M. *A Discrete Signal Processing Framework for Set Functions* in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2018), 4359.

21. Valentin, R., Ferrari, C., Scheurer, J., Amrollahi, A., Wendler, C. & Paulus, M. B. Instance-wise algorithm configuration with graph neural networks. *arXiv preprint arXiv:2202.04910* (2022).

22. Gorji, A., Amrollahi, A. & Krause, A. *A scalable Walsh-Hadamard regularizer to overcome the low-degree spectral bias of neural networks* in *The 39th Conference on Uncertainty in Artificial Intelligence* (2023).

23. Amrollahi, A., Gorji, A. & Krause, A. *Amortized SHAP values via function approximation* in (2023).

24. O'Donnell, R. *Analysis of boolean functions* (Cambridge University Press, 2014).

25. Rasmussen, C. E. *Gaussian processes in machine learning* in *Summer school on machine learning* (2004), 63.

26. Daniely, A., Frostig, R. & Singer, Y. Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity. *Advances in neural information processing systems* **29** (2016).

27. Chizat, L., Oyallon, E. & Bach, F. On lazy training in differentiable programming. *Advances in Neural Information Processing Systems* **32** (2019).

28. Du, S. S., Zhai, X., Poczos, B. & Singh, A. *Gradient Descent Provably Optimizes Over-parameterized Neural Networks* in *International Conference on Learning Representations* (2019).

29. Allen-Zhu, Z., Li, Y. & Song, Z. *A convergence theory for deep learning via over-parameterization* in *International Conference on Machine Learning* (2019), 242.

30. Allen-Zhu, Z., Li, Y. & Song, Z. On the convergence rate of training recurrent neural networks. *Advances in neural information processing systems* **32** (2019).

31. Yang, G. & Salman, H. A fine-grained spectral perspective on neural networks. *arXiv preprint arXiv:1907.10599* (2019).

32. Kushilevitz, E. & Mansour, Y. Learning decision trees using the Fourier spectrum. *SIAM Journal on Computing* **22**, 1331 (1993).

33. Dorogush, A. V., Ershov, V. & Gulin, A. CatBoost: gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363* (2018).

34. Chen, T. & Guestrin, C. *Xgboost: A scalable tree boosting system* in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (2016), 785.
35. Carter, J. L. & Wegman, M. N. Universal classes of hash functions. *Journal of computer and system sciences* **18**, 143 (1979).
36. Daniely, A. SGD learns the conjugate kernel class of the network. *Advances in Neural Information Processing Systems* **30** (2017).
37. Rahaman, N., Baratin, A., Arpit, D., Draxler, F., Lin, M., Hamprecht, F., Bengio, Y. & Courville, A. *On the Spectral Bias of Neural Networks* in *Proceedings of the 36th International Conference on Machine Learning* ISSN: 2640-3498 (PMLR, 2019), 5301.
38. Ronen, B., Jacobs, D., Kasten, Y. & Kritchman, S. *The Convergence Rate of Neural Networks for Learned Functions of Different Frequencies* in *Advances in Neural Information Processing Systems* **32** (Curran Associates, Inc., 2019).
39. Basri, R., Galun, M., Geifman, A., Jacobs, D., Kasten, Y. & Kritchman, S. *Frequency bias in neural networks for input of non-uniform density* in *International Conference on Machine Learning* (2020), 685.
40. Neyshabur, B., Bhojanapalli, S., McAllester, D. & Srebro, N. *Exploring Generalization in Deep Learning* arXiv:1706.08947 [cs]. 2017.
41. Poggio, T., Kawaguchi, K., Liao, Q., Miranda, B., Rosasco, L., Boix, X., Hidary, J. & Mhaskar, H. *Theory of Deep Learning III: explaining the non-overfitting puzzle* arXiv:1801.00173 [cs]. 2018.
42. Tancik, M., Srinivasan, P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J. & Ng, R. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems* **33**, 7537 (2020).
43. Shah, H., Tamuly, K., Raghunathan, A., Jain, P. & Netrapalli, P. *The Pitfalls of Simplicity Bias in Neural Networks* in *Advances in Neural Information Processing Systems* **33** (Curran Associates, Inc., 2020), 9573.
44. Benjamin, A., Rolnick, D. & Kording, K. *Measuring and regularizing networks in function space* in *International Conference on Learning Representations* (2019).

45. Sun, S., Zhang, G., Shi, J. & Grosse, R. *Functional Variational Bayesian Neural Networks* in *International Conference on Learning Representations* (2019).

46. Wang, Z., Ren, T., Zhu, J. & Zhang, B. *Function Space Particle Optimization for Bayesian Neural Networks* in *International Conference on Learning Representations* (2019).

47. Aghazadeh, A., Nisonoff, H., Ocal, O., Brookes, D. H., Huang, Y., Koyluoglu, O. O., Listgarten, J. & Ramchandran, K. Epistatic Net allows the sparse spectral regularization of deep neural networks for inferring fitness functions. *Nature Communications* **12**. Number: 1 Publisher: Nature Publishing Group, 5225 (2021).

48. Sailer, Z. R. & Harms, M. J. Detecting High-Order Epistasis in Nonlinear Genotype-Phenotype Maps. *Genetics* **205**, 1079 (2017).

49. Yang, G., Anderson, D. W., Baier, F., Dohmen, E., Hong, N., Carr, P. D., Kamerlin, S. C. L., Jackson, C. J., Bornberg-Bauer, E. & Tokuriki, N. Higher-order epistasis shapes the fitness landscape of a xenobiotic-degrading enzyme. *Nature Chemical Biology* **15**. Number: 11 Publisher: Nature Publishing Group, 1120 (2019).

50. Brookes, D. H., Aghazadeh, A. & Listgarten, J. On the sparsity of fitness functions and implications for learning. eng. *Proceedings of the National Academy of Sciences of the United States of America* **119**, e2109649118 (2022).

51. Ballal, A., Laurendon, C., Salmon, M., Vardakou, M., Cheema, J., Defernez, M., O'Maille, P. E. & Morozov, A. V. Sparse Epistatic Patterns in the Evolution of Terpene Synthases. *Molecular Biology and Evolution* **37**, 1907 (2020).

52. Poelwijk, F. J., Socolich, M. & Ranganathan, R. Learning the pattern of epistasis linking genotype and phenotype in a protein. *Nature Communications* **10**. Number: 1 Publisher: Nature Publishing Group, 4213 (2019).

53. Alon, N., Dietzfelbinger, M., Miltersen, P. B., Petrank, E. & Tardos, G. Linear hash functions. *Journal of the ACM (JACM)* **46**, 667 (1999).

54. Arik, S. Ö. & Pfister, T. *Tabnet: Attentive interpretable tabular learning* in *Proceedings of the AAAI Conference on Artificial Intelligence* **35** (2021), 6679.

55. Sarkisyan, K. S., Bolotin, D. A., Meer, M. V., Usmanova, D. R., Mishin, A. S., Sharonov, G. V., Ivankov, D. N., Bozhanova, N. G., Baranov, M. S., Soylemez, O., Bogatyreva, N. S., Vlasov, P. K., Egorov, E. S., Logacheva, M. D., Kondrashov, A. S., Chudakov, D. M., Putintseva, E. V., Mamedov, I. Z., Tawfik, D. S., Lukyanov, K. A. & Kondrashov, F. A. Local fitness landscape of the green fluorescent protein. *Nature* **533**. Number: 7603 Publisher: Nature Publishing Group, 397 (2016).

56. Wu, N. C., Dai, L., Olson, C. A., Lloyd-Smith, J. O. & Sun, R. Adaptation in protein fitness landscapes is facilitated by indirect paths. *eLife* **5** (ed Neher, R. A.) Publisher: eLife Sciences Publications, Ltd, e16965 (2016).

57. Nugteren, C. & Codreanu, V. *CLTune: A Generic Auto-Tuner for OpenCL Kernels* in *2015 IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip* (2015), 195.

58. Voigt, P. & Von dem Bussche, A. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing* **10**, 10 (2017).

59. Shapley, L. S. *A Value for N-Person Games* (RAND Corporation, Santa Monica, CA, 1952).

60. Gromping, U. Estimators of relative importance in linear regression based on variance decomposition. *The American Statistician* **61**, 139 (2007).

61. Štrumbelj, E., Kononenko, I. & Šikonja, M. R. Explaining instance classifications with interactions of subsets of feature values. *Data & Knowledge Engineering* **68**, 886 (2009).

62. Owen, A. B. Sobol indices and Shapley value. *SIAM/ASA Journal on Uncertainty Quantification* **2**, 245 (2014).

63. Datta, A., Sen, S. & Zick, Y. *Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems* in *2016 IEEE symposium on security and privacy (SP)* (2016), 598.

64. Owen, A. B. & Prieur, C. On Shapley value for measuring importance of dependent inputs. *SIAM/ASA Journal on Uncertainty Quantification* **5**, 986 (2017).

65. Lundberg, S. M. & Lee, S.-I. *A Unified Approach to Interpreting Model Predictions* in *Advances in Neural Information Processing Systems* **30** (2017).

66. Lundberg, S. M., Erion, G., Chen, H., DeGrave, A., Prutkin, J. M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N. & Lee, S.-I. From local explanations to global understanding with explainable AI for trees. *Nature machine intelligence* **2**, 56 (2020).

67. Aas, K., Jullum, M. & Løland, A. Explaining individual predictions when features are dependent: More accurate approximations to Shapley values. *Artificial Intelligence* **298**, 103502 (2021).

68. Yang, J. Fast treeshap: Accelerating shap value computation for trees. *arXiv preprint arXiv:2109.09847* (2021).

69. Bifet, A., Read, J., Xu, C., *et al.* Linear tree shap. *Advances in Neural Information Processing Systems* **35**, 25818 (2022).

70. Shrikumar, A., Greenside, P. & Kundaje, A. *Learning important features through propagating activation differences* in *International conference on machine learning* (2017), 3145.

71. Jethani, N., Sudarshan, M., Covert, I. C., Lee, S.-I. & Ranganath, R. *FastSHAP: Real-Time Shapley Value Estimation* in *International Conference on Learning Representations* (2021).

72. Covert, I. & Lee, S.-I. Improving kernelshap: Practical shapley value estimation via linear regression. *arXiv preprint arXiv:2012.01536* (2020).

73. Mitchell, R., Cooper, J., Frank, E. & Holmes, G. Sampling Permutations for Shapley Value Estimation. *Journal of Machine Learning Research* **23**, 1 (2022).

74. Janzing, D., Minorics, L. & Bloebaum, P. *Feature relevance quantification in explainable AI: A causal problem* in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics* (eds Chiappa, S. & Calandra, R.) **108** (PMLR, 2020), 2907.

75. Van den Broeck, G., Lykov, A., Schleich, M. & Suciu, D. *On the tractability of SHAP explanations* in *Proceedings of the 35th Conference on Artificial Intelligence (AAAI)* (2021).

76. Sundararajan, M. & Najmi, A. *The many Shapley values for model explanation* in *International conference on machine learning* (2020), 9269.

77. Pearl, J. *Causality* (Cambridge university press, 2009).

78. Arenas, M., Barceló, P., Bertossi, L. & Monet, M. *The tractability of SHAP-score-based explanations for classification over deterministic and decomposable boolean circuits* in *Proceedings of the AAAI Conference on Artificial Intelligence* **35** (2021), 6670.

79. Kwon, Y., Rivas, M. A. & Zou, J. *Efficient computation and analysis of distributional Shapley values* in *International Conference on Artificial Intelligence and Statistics* (2021), 793.

80. Chen, H., Janizek, J. D., Lundberg, S. & Lee, S.-I. True to the Model or True to the Data? *arXiv preprint arXiv:2006.16234* (2020).

81. Wendler, C. *Machine learning on non-euclidean domains: powersets, lattices, posets* PhD thesis (ETH Zurich, 2023).

82. Huh, M., Mobahi, H., Zhang, R., Cheung, B., Agrawal, P. & Isola, P. *The Low-Rank Simplicity Bias in Deep Networks* arXiv:2103.10427 [cs]. 2022.

83. Durvasula, K. & Liter, A. There is a simplicity bias when generalising from ambiguous data. *Phonology* **37**. Publisher: Cambridge University Press, 177 (2020).

84. Li, X. & Ramchandran, K. An active learning framework using sparse-graph codes for sparse polynomials and graph sketching. *Advances in Neural Information Processing Systems* **28** (2015).

85. Amrollahi, A., Zandieh, A., Kapralov, M. & Krause, A. *Efficiently learning fourier sparse set functions* in *Advances in Neural Information Processing Systems* (2019), 15120.

86. Li, X., Bradley, J. K., Pawar, S. & Ramchandran, K. SPRIGHT: A fast and robust framework for sparse Walsh-Hadamard transform. *arXiv preprint arXiv:1508.06336* (2015).

87. Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S. & Zhang, Q. *JAX: composable transformations of Python+NumPy programs* version 0.3.13. 2018.

88. Goldreich, O. & Levin, L. A. *A hard-core predicate for all one-way functions* in *Proceedings of the twenty-first annual ACM symposium on Theory of computing* (1989), 25.

89. Stobbe, P. & Krause, A. *Learning Fourier Sparse Set Functions* in *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics* ISSN: 1938-7228 (PMLR, 2012), 1125.

90. Hazan, E., Klivans, A. & Yuan, Y. *Hyperparameter optimization: a spectral approach* in *International Conference on Learning Representations* (2018).

91. Rudelson, M. & Vershynin, R. On sparse reconstruction from Fourier and Gaussian measurements. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences* **61**, 1025 (2008).

92. Vershynin, R. Introduction to the non-asymptotic analysis of random matrices. *arXiv e-prints*, arXiv:1011.3027 (2010).

93. Haviv, I. & Regev, O. in *Geometric Aspects of Functional Analysis* 163 (Springer, 2017).

94. Hassanieh, H., Indyk, P., Katabi, D. & Price, E. *Nearly optimal sparse fourier transform* in *Proceedings of the forty-fourth annual ACM symposium on Theory of computing* (2012), 563.

95. Indyk, P., Kapralov, M. & Price, E. *(Nearly) sample-optimal sparse Fourier transform* in *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms* (2014), 480.

96. Das, A. K. & Vishwanath, S. *On finite alphabet compressive sensing* in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing* (2013), 5890.

97. Reed, I. S. & Solomon, G. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics* **8**, 300 (1960).

98. Das, A. K. & Vishwanath, S. *On finite alphabet compressive sensing* in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing* (2013), 5890.

99. May, A., Meurer, A. & Thomae, E. *Decoding random linear codes in \tilde {\mathcal {O}}(2^{0.054 n})* in *International Conference on the Theory and Application of Cryptology and Information Security* (2011), 107.

100. Leon, J. S. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Transactions on Information Theory* **34**, 1354 (1988).

101. Stern, J. *A method for finding codewords of small weight* in *International Colloquium on Coding Theory and Applications* (1988), 106.

102. Bernstein, D. J., Lange, T. & Peters, C. *Smaller decoding exponents: ball-collision decoding* in *Annual Cryptology Conference* (2011), 743.

103. Bestuzheva, K., Besançon, M., Chen, W.-K., Chmiela, A., Donkiewicz, T., van Doornmalen, J., Eifler, L., Gaul, O., Gamrath, G., Gleixner, A., Gottwald, L., Graczyk, C., Halbig, K., Hoen, A., Hojny, C., van der Hulst, R., Koch, T., Lübbecke, M., Maher, S. J., Matter, F., Mühmer, E., Müller, B., Pfetsch, M. E., Rehfeldt, D., Schlein, S., Schlösser, F., Serrano, F., Shinano, Y., Sofranac, B., Turner, M., Vigerske, S., Wegscheider, F., Wellner, P., Weninger, D. & Witzig, J. *The SCIP Optimization Suite 8.0* Technical Report (Optimization Online, 2021).

104. Bestuzheva, K., Besançon, M., Chen, W.-K., Chmiela, A., Donkiewicz, T., van Doornmalen, J., Eifler, L., Gaul, O., Gamrath, G., Gleixner, A., Gottwald, L., Graczyk, C., Halbig, K., Hoen, A., Hojny, C., van der Hulst, R., Koch, T., Lübbecke, M., Maher, S. J., Matter, F., Mühmer, E., Müller, B., Pfetsch, M. E., Rehfeldt, D., Schlein, S., Schlösser, F., Serrano, F., Shinano, Y., Sofranac, B., Turner, M., Vigerske, S., Wegscheider, F., Wellner, P., Weninger, D. & Witzig, J. *The SCIP Optimization Suite 8.0* ZIB-Report 21-41 (Zuse Institute Berlin, 2021).

105. Guruswami, V. *Lecture notes for 15-859V, Introduction to Coding Theory, Spring 2010 at CMU, Notes 2, Gilbert Varshamov bound* 2010.

106. Rahimi, A. & Recht, B. Random features for large-scale kernel machines. *Advances in neural information processing systems* **20** (2007).

107. Boyd, S. P. *Distributed optimization and statistical learning via the alternating direction method of multipliers* (Now Publishers Inc, Hanover, MA, 2011).

108. Cheraghchi, M. & Indyk, P. Nearly optimal deterministic algorithm for sparse Walsh-Hadamard transform. *ACM Transactions on Algorithms (TALG)* **13**, 1 (2017).

109. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. & Duchesnay, E. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* **12**, 2825 (2011).

CURRICULUM VITAE

PERSONAL DATA

Name	Andisheh Amrollahi
Date of Birth	March 31, 1994
Place of Birth	Tehran, Iran
Citizen of	Iran

EDUCATION

2012 – 2017	Sharif University, Tehran, Iran *Final degree:* Bachelor in Electrical Engineering, Minor in Computer Engineering
2017 – 2019	ETH Zurich Switzerland *Final degree:* Master degree in Computer science
2019– 2023	ETH Zurich Switzerland *Final degree:* PhD in Computer science

PUBLICATIONS

Conference contributions:

1. Amrollahi, A., Zandieh, A., Kapralov, M. & Krause, A. *Efficiently learning fourier sparse set functions* in *Advances in Neural Information Processing Systems* (2019), 15120.
2. Gorji, A., Amrollahi, A. & Krause, A. *A scalable Walsh-Hadamard regularizer to overcome the low-degree spectral bias of neural networks* in *The 39th Conference on Uncertainty in Artificial Intelligence* (2023).
3. Amrollahi, A., Gorji, A. & Krause, A. *Amortized SHAP values via function approximation* in *Advances in Neural Information Processing Systems (under review)* (2023).
4. Wendler, C., Amrollahi, A., Seifert, B., Krause, A. & Püschel, M. *Learning set functions that are sparse in non-orthogonal Fourier bases* in *Proceedings of the AAAI Conference on Artificial Intelligence* **35** (2021), 10283.

Other contributions:

5. Valentin, R., Ferrari, C., Scheurer, J., Amrollahi, A., Wendler, C. & Paulus, M. B. Instance-wise algorithm configuration with graph neural networks. *arXiv preprint arXiv:2202.04910* (2022).