A Scalable Walsh-Hadamard Regularizer to Overcome the Low-degree Spectral Bias of Neural Networks (Supplementary material)

Ali Gorji[*1] **Andisheh Amrollahi**[*1] **Andreas Krause**[1]

[1]Computer Science Department, ETH Zurich, Zurich, Switzerland

A 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).

*These authors contributed equally to this work

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 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 1 from Section 2.1. 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 Alon et al. [1999], Amrollahi et al. [2019]. We finally review EN-S and showcase the superiority and scalability of our method in terms of computation.

Accepted for the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023).

B.1 PROOF OF EQUATION 1

Let

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

as in Section 2.1.

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

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

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

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

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

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

B.2 COLLISIONS FOR HASHWH

We first review the notion of *pairwise independent* families of hash functions introduced by Carter and Wegman [1979]. 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.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., $c_{ij} = \begin{cases} 1 & h(f_i) = h(f_j) \\ 0 & h(f_i) \neq h(f_j) \end{cases}$, for $i \neq j \in [k]$.

Lemma B.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 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.1. If we remove 0 from the domain then it becomes a pairwise independent hashing scheme.

Lemma B.3. *The hash function used in the hashing procedure of our method* HASHWH*, i.e., $h(.) = \sigma^\top(.)$ where $\sigma \sim \mathcal{U}_{\{0,1\}^{n\times b}}$ 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 $\sigma^\top f$ is a linear combination of columns of $\sigma^\top$, where f determines the columns. We denote i^{th} column of $\sigma^\top$ by $\sigma^\top_{\bullet i}$. 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 $\sigma^\top$ that corresponds to those t positions: $\sigma^\top_{\bullet i_1}, \cdots, \sigma^\top_{\bullet i_t}$. 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 and B.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 σ 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.3 EN-S DETAILS

To avoid computing the exact Fourier spectrum of the network at each back-propagation iteration in FULLWH, Aghazadeh et al. [2021] 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 [Boyd, 2011] (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:

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

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:

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

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

Alternating Direction Method of Multipliers [Boyd, 2011], 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:

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

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:

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

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, Aghazadeh et al. [2021] reformulate the FULLWH loss, by introducing a new variable $\boldsymbol{z}$ and adding a constraint such that it is equal to the Fourier spectrum:

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

, 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 (2) followed by a few further adjustments to reach the following alternating scheme for optimization of $\mathcal{L}_{EN-S}$:

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

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 $\gamma \in \mathbb{R}^{O(2^m n)}$ is the dual variable. We will introduce the hash size parameter m momentarily.

Using the optimization scheme (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 [Li et al., 2015]. 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 (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 [Li and Ramchandran, 2015, Scheibler et al., 2015, Li et al., 2015, Amrollahi et al., 2019], including SPRIGHT [Li et al., 2015] 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 ($\boldsymbol{z}$-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.

C DATASETS

We list all the datasets used in the real dataset Section 5.2.

Entacmaea quadricolor fluorescent protein. (Entacmaea) Poelwijk et al. [2019] 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). Nugteren and Codreanu [2015] 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). Wu et al. [2016] 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). Sarkisyan et al. [2016] 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.

D 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 paper. 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 Aghazadeh et al. [2021][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.

E 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.

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.

E.1 FOURIER TRANSFORM OF (ENSEMBLES) OF DECISION TREES

A decision tree, in our context, is a rooted binary tree whose nodes can have either zero or two children. Each leaf node is assigned a real number. Each non-leaf node corresponds to one of n binary features. The tree defines a function $t : \{0, 1\}^n \to \mathbb{R}$ in the following way: To compute $t(x)$ we look at the root, corresponding to, say, feature $i \in [n]$. Next, we check the value of the variable x_i. If the value of the variable is equal to 0 we look at the left child. If it is equal to 1 we look at the right child. Then we repeat this process until we reach a leaf. The value of the function t evaluated at x is the real number assigned to that leaf. In all that follows, when referring to decision trees, we will denote them by the function $t : \{0, 1\}^n \to \mathbb{R}$.

Given a decision tree, we can compute its Fourier transform recursively. Let $i \in [n]$ denote the feature corresponding to its root. Then the tree can be represented as follows:

$$t(x) = \frac{1 + (-1)^{\langle e_i, x\rangle}}{2} t_{\text{left}}(x) + \frac{1 - (-1)^{\langle e_i, x\rangle}}{2} t_{\text{right}}(x) \tag{4}$$

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. Therefore, one can recursively compute the Fourier transform of a decision tree.

This also portrays why a decision tree of depth d is a function of degree d. Moreover, for each tree t, if $|\text{supp}(t_{left})| = k_{\text{left}}$ and $|\text{supp}(t_{right})| = k_{\text{right}}$, then $|\text{supp}(t)| \leq 2(k_{\text{left}} + k_{\text{right}})$. This implies that a decision tree is k-sparse with $k = O(4^d)$. However, in many cases, when the decision tree is not complete or cancellations occur, the Fourier transform is even sparser.

Finally, we can also compute the Fourier transform of an ensemble of trees such as one produced by the random forest and XGBoost algorithms. In the case of regression, the ensemble just predicts the average prediction of its constituent trees. Therefore its Fourier transform is the (normalized) sum of the Fourier transforms of its trees as well. If a random forest model consists of T different trees then its Fourier transform is $k = O(T4^d)$-sparse and of degree equal to its maximum depth.

[1] https://github.com/amirmohan/epistatic-net

[2] https://scikit-learn.org

[3] https://xgboost.readthedocs.io

E.2 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.

F EXTENDED EXPERIMENT RESULTS

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

F.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 6 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 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 7 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 1a, we reported the results for one version of the target function g^* and Figure 7 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 9, 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 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 8, 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, 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.

F.2 HIGH-DIMENSIONAL SYNTHETIC DATA

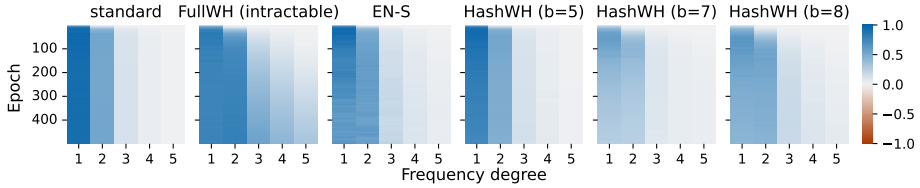
Figure 10 shows the generalization performance of different methods in learning a synthetic degree $d = 5$ function $g^* \in \{0,1\}^n \to \mathbb{R}$, for $n \in \{25, 50, 100\}$, using train sets of different sizes ($c \cdot 25n, c \in [8]$). For each n we sample three different draws of g^*. This is the extended version of Figure 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 11 to 13, 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 4b where we only reported it for $n = 50$

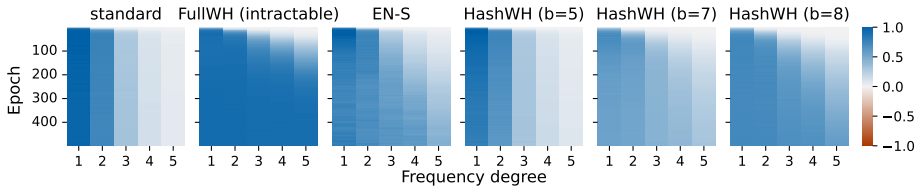
and the sample size multiplier $c = 5$. We consistently see that the trade-off between the generalization performance and the training time can be directly controlled in HASHWH using the parameter b. Furthermore, HASHWH is able to *always* exhibit a significantly better generalization performance in 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.

F.3 REAL DATA

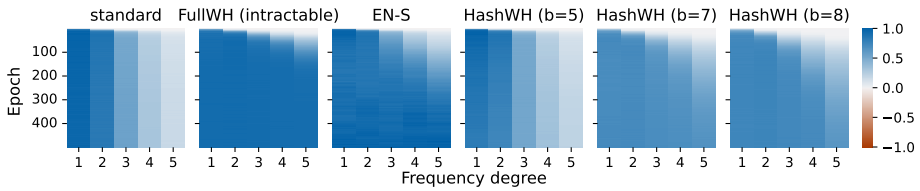
Figure 14 shows the generalization performance and the training time of different methods, including relevant machine learning benchmarks, in learning four real datasets. It is the extended version of Figure 5, where we only reported the generalization performance and not the training time. The training 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.



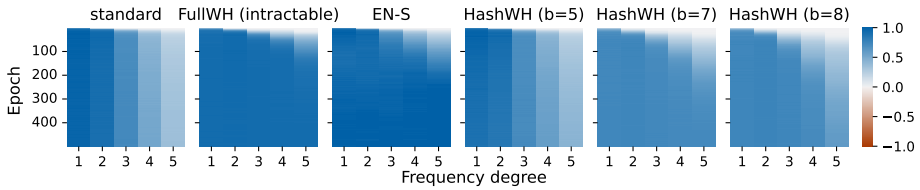
(a) Train size $= 100$



(b) Train size $= 200$



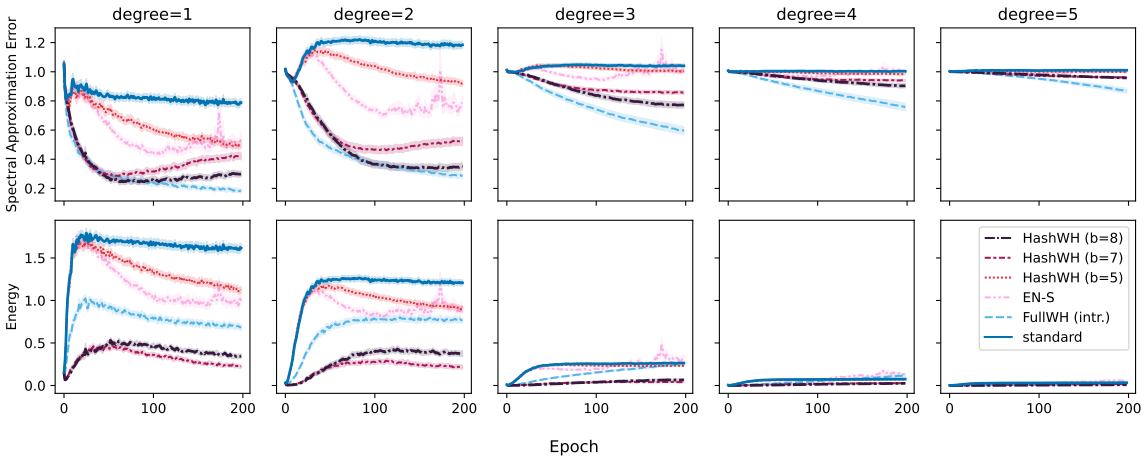
(c) Train size $= 300$



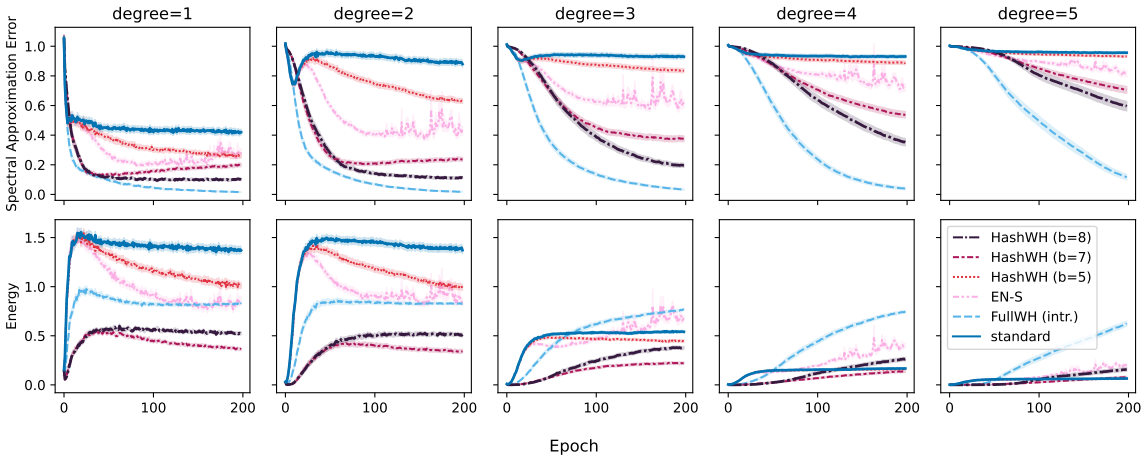
(d) Train size $= 400$

Figure 6: 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 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.

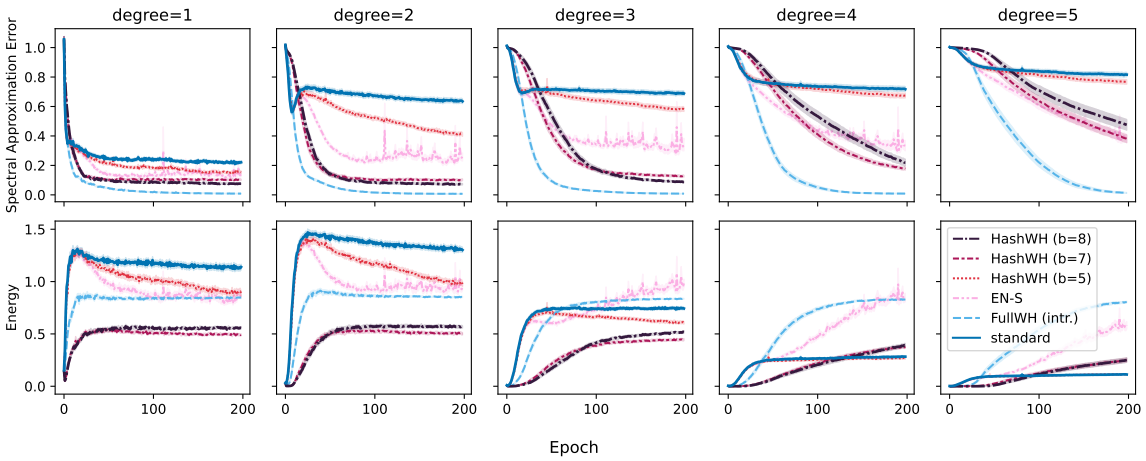
Figure 7: Evolution of the Fourier spectrum in learning a synthetic function $g^* \in \{0,1\}^{10}$ of degree 5 during training, categorized by frequency degree. All synthetic functions used have single frequencies of each degree in their support that are all given the amplitude of 1. We reported the results for one draw of g^* in Figure 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 $= 100$

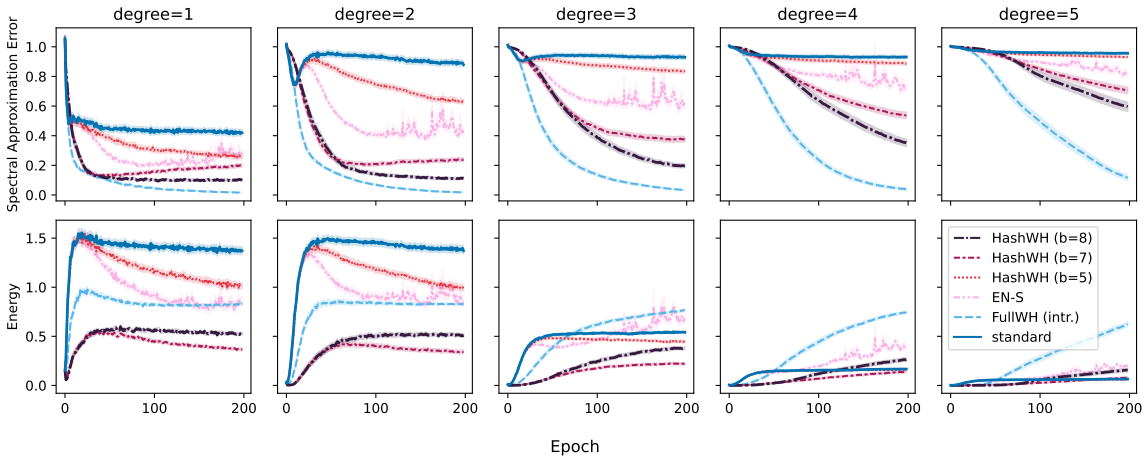


(b) Train size $= 200$



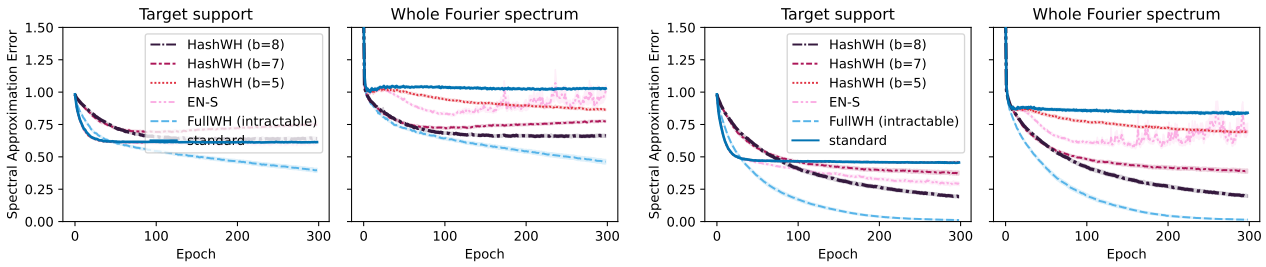
(c) Train size $= 300$

Figure 8: Evolution of the Spectral Approximation Error (SAE) and energy of the network during training, categorized by frequency degree (continued in the next page).



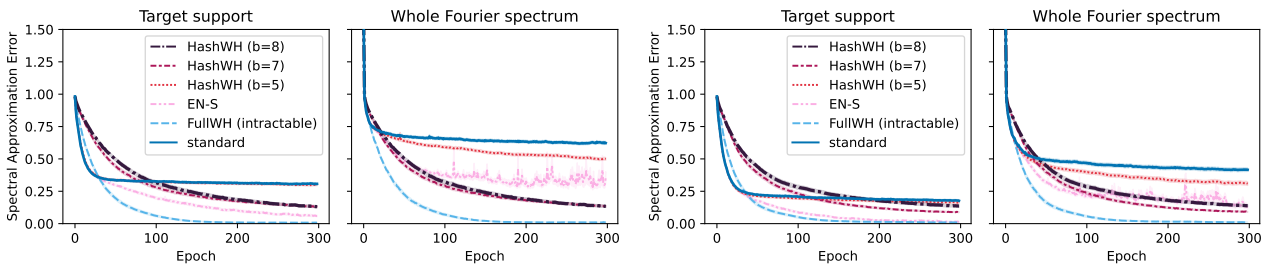
(d) Train size $= 400$

Figure 8: 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, 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 $= 100$

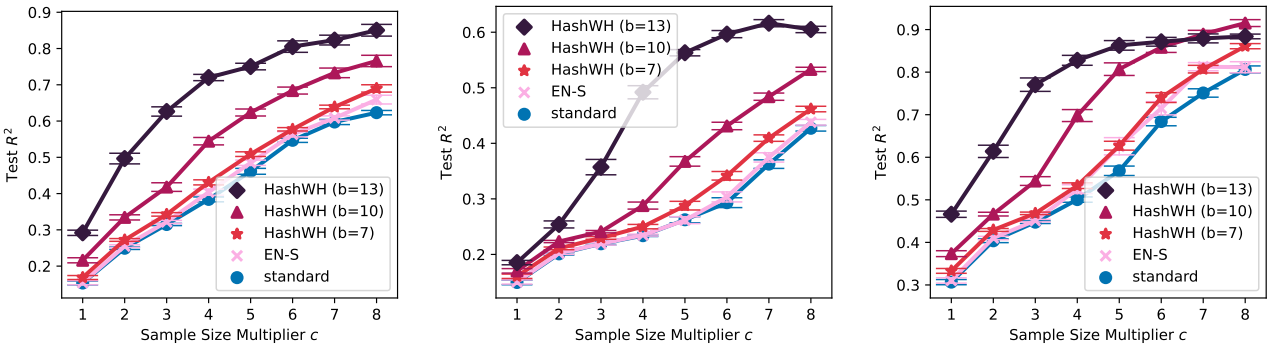
(b) Train size $= 200$



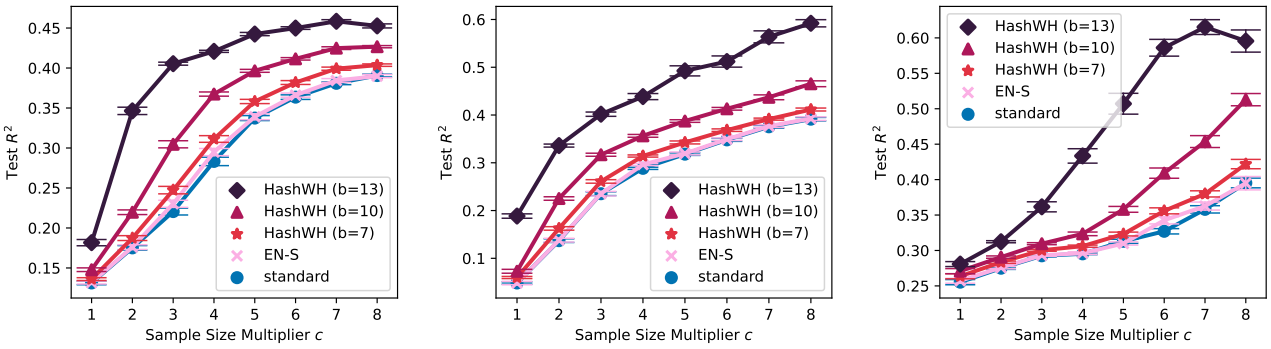
(c) Train size $= 300$

(d) Train size $= 400$

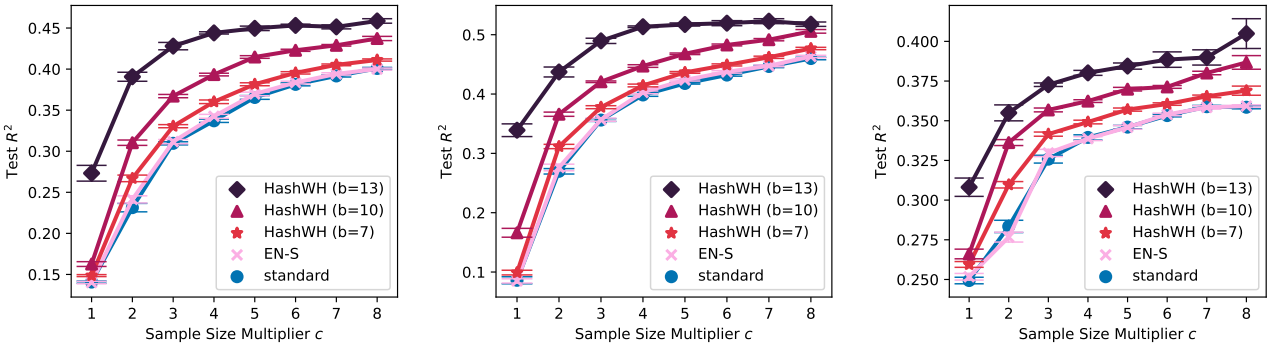
Figure 9: Evolution of the spectral approximation error (SAE) during training. The left plot limits the error to the target support, while the right one considers the whole Fourier spectrum. This is an extended version of Figure 2, where we only reported results for train size 200. The standard neural 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) $n = 25$, first draw of g^* (b) $n = 25$, second draw of g^* (c) $n = 25$, third draw of g^*

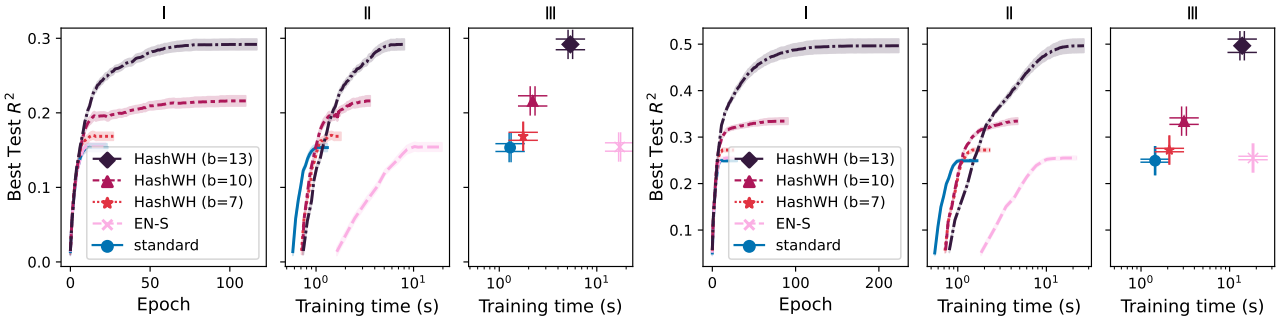


(d) $n = 50$, first draw of g^* (e) $n = 50$, second draw of g^* (f) $n = 50$, third draw of g^*



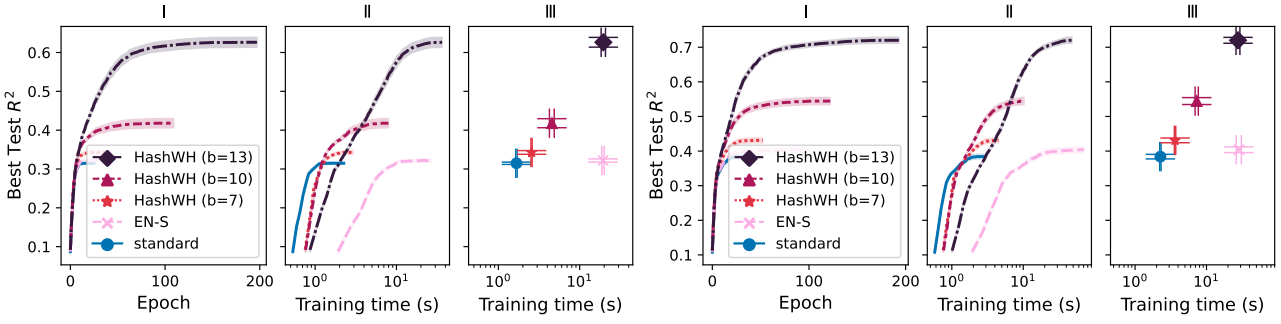
(g) $n = 100$, first draw of g^* (h) $n = 100$, second draw of g^* (i) $n = 100$, third draw of g^*

Figure 10: 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 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.



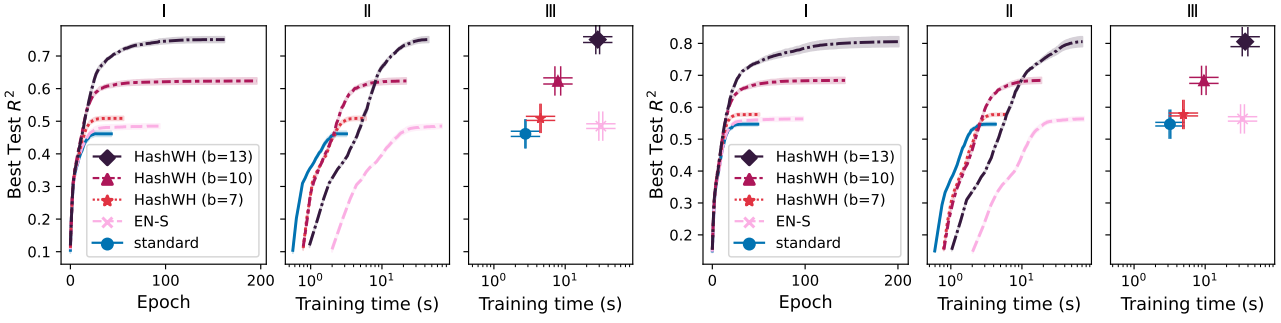
(a) $n = 25$, $c = 1$

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



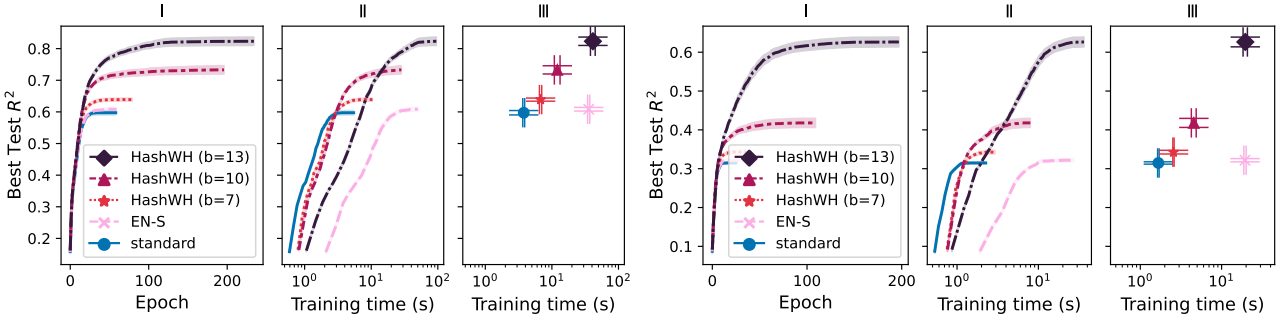
(c) $n = 25$, $c = 3$

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



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

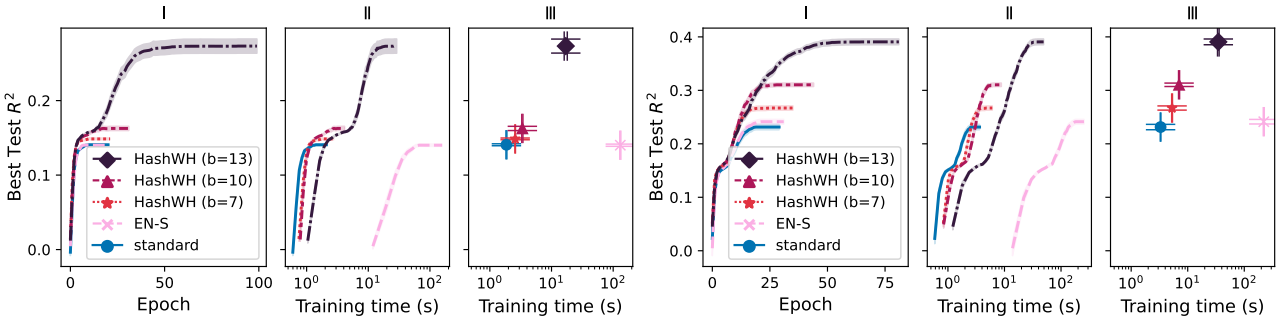
(f) $n = 25$, $c = 6$



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

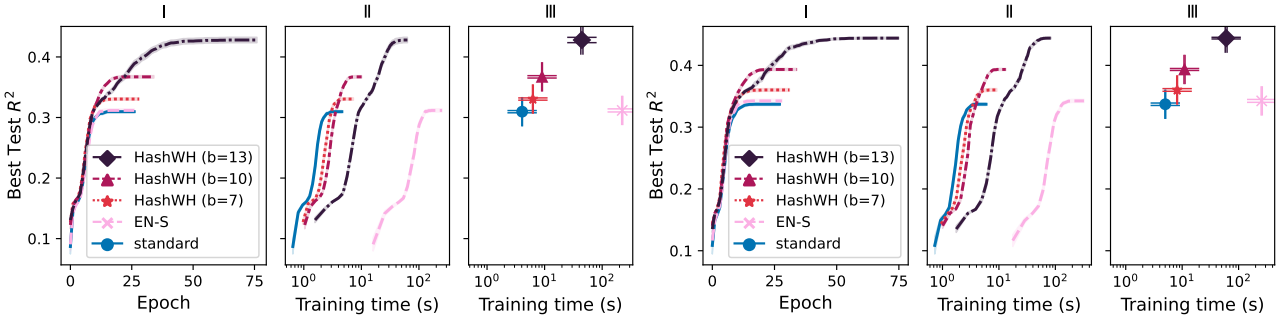
(h) $n = 25$, $c = 8$

Figure 11: 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 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.



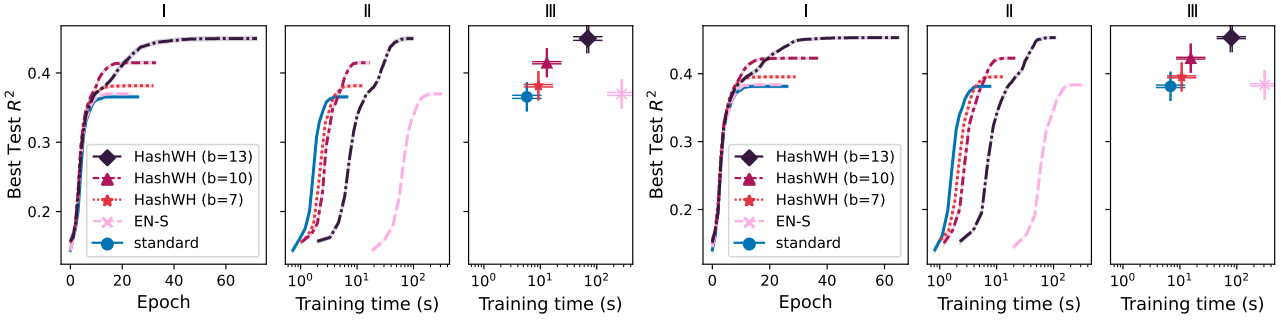
(a) $n = 50$, $c = 1$

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



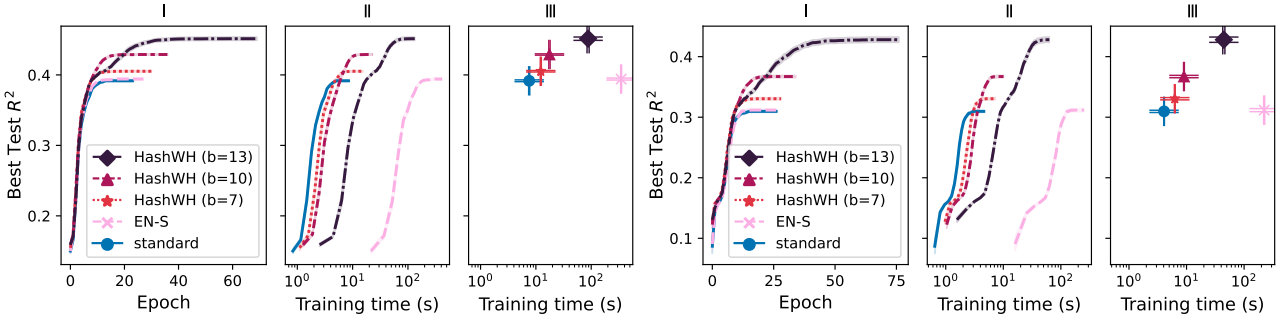
(c) $n = 50$, $c = 3$

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



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

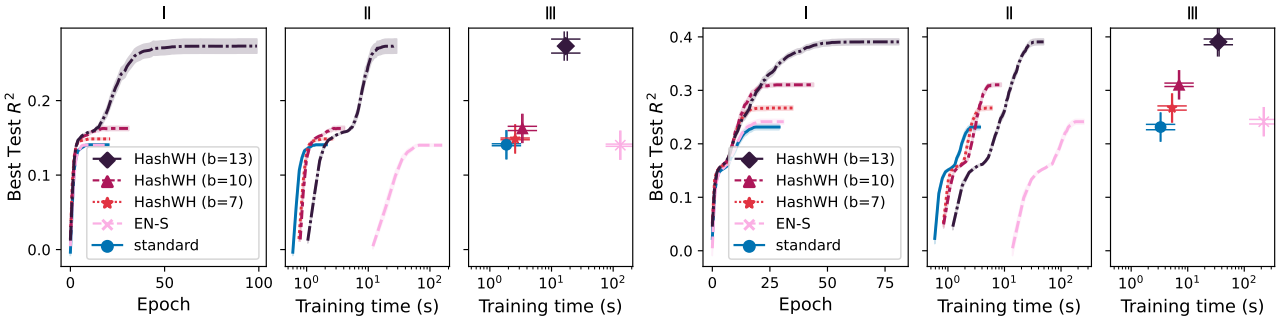
(f) $n = 50$, $c = 6$



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

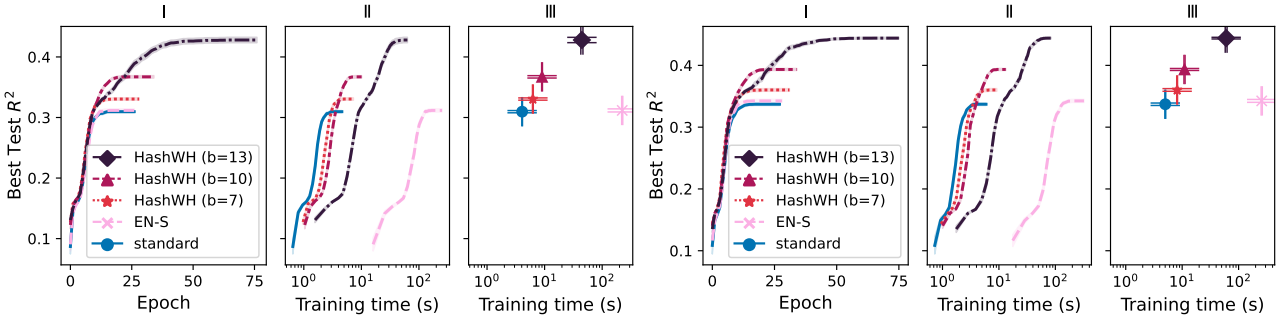
(h) $n = 50$, $c = 8$

Figure 12: 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 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.



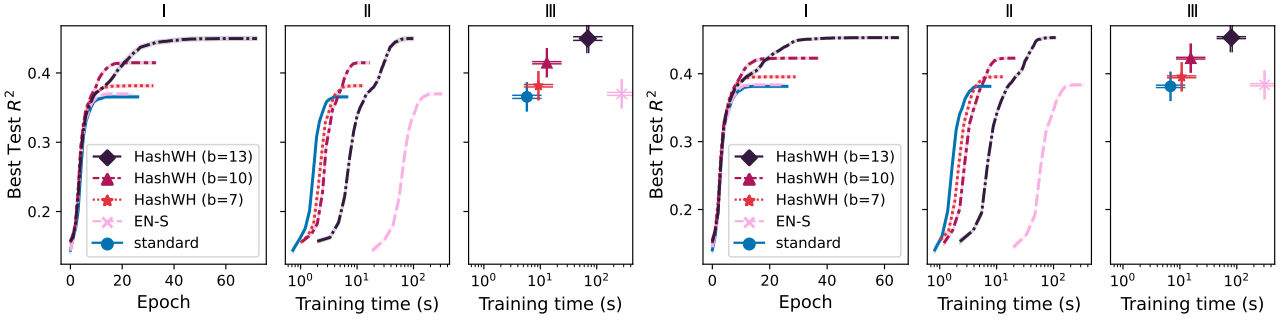
(a) $n = 100$, $c = 1$

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



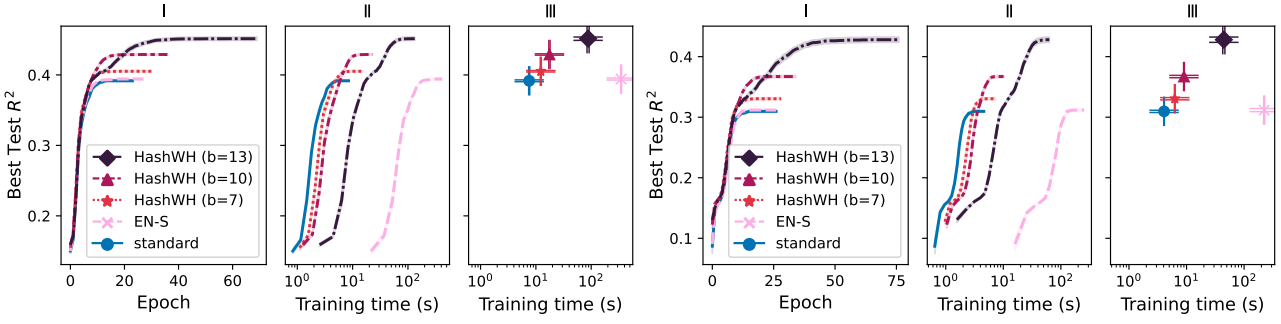
(c) $n = 100$, $c = 3$

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



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

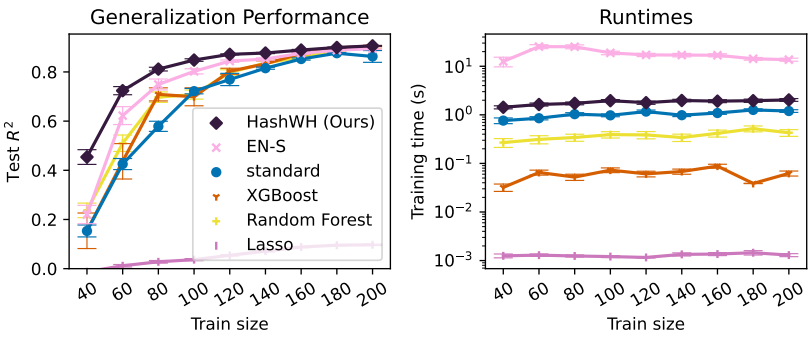
(f) $n = 100$, $c = 6$



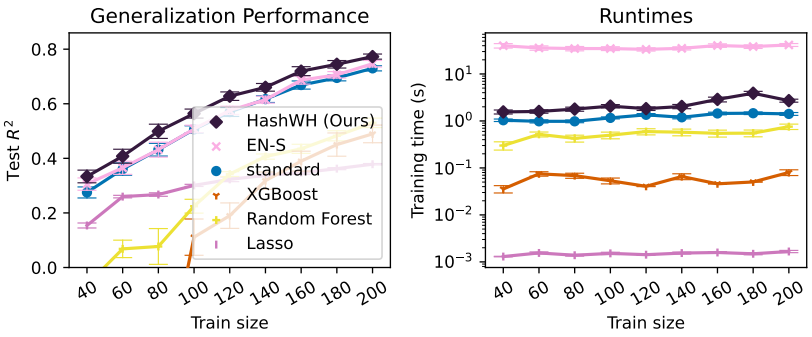
(g) $n = 100$, $c = 7$

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

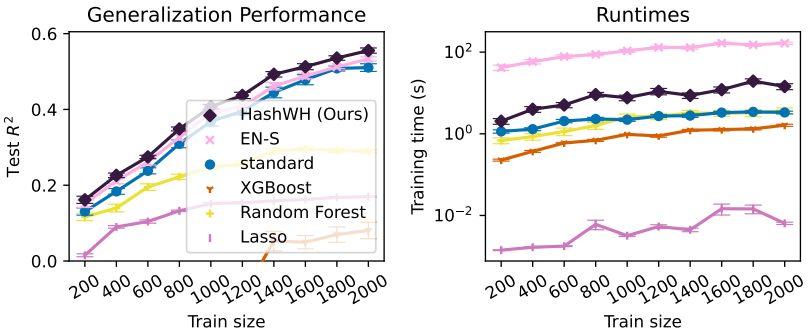
Figure 13: Best achievable generalization performance R^2 up to a certain epoch or training time (seconds), in learning a synthetic degree 5 function $g^* \in \{0,1\}^n$, using datasets of size $c \cdot 25n$. This figure is an extended version of Figure 4b, where we reported similar plots for $n = 50$ 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.



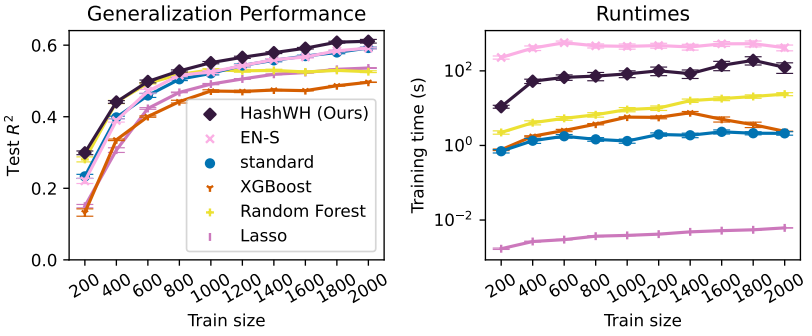
(a) Entacmaea (n=13)



(b) SGEMM (n=40)



(c) GB1 (n=80)



(d) avGFP (n=236)

Figure 14: Generalization performance of standard/regularized neural networks and benchmark ML models on four real datasets. This figure is an extended version of Figure 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.

References

Amirali Aghazadeh, Hunter Nisonoff, Orhan Ocal, David H. Brookes, Yijie Huang, O. Ozan Koyluoglu, Jennifer Listgarten, and Kannan Ramchandran. Epistatic Net allows the sparse spectral regularization of deep neural networks for inferring fitness functions. *Nature Communications*, 12(1):5225, September 2021. ISSN 2041-1723. doi: 10.1038/s41467-021-25371-3. Number: 1 Publisher: Nature Publishing Group.

Noga Alon, Martin Dietzfelbinger, Peter Bro Miltersen, Erez Petrank, and Gábor Tardos. Linear hash functions. *Journal of the ACM (JACM)*, 46(5):667–683, 1999.

Andisheh Amrollahi, Amir Zandieh, Michael Kapralov, and Andreas Krause. Efficiently Learning Fourier Sparse Set Functions. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Stephen P. Boyd. *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, Hanover, MA, 2011. ISBN 978-1-60198-460-9.

J Lawrence Carter and Mark N Wegman. Universal classes of hash functions. *Journal of computer and system sciences*, 18(2):143–154, 1979.

Xiao Li and Kannan Ramchandran. An active learning framework using sparse-graph codes for sparse polynomials and graph sketching. *Advances in Neural Information Processing Systems*, 28, 2015.

Xiao Li, Joseph K. Bradley, Sameer Pawar, and Kannan Ramchandran. SPRIGHT: A Fast and Robust Framework for Sparse Walsh-Hadamard Transform, August 2015. arXiv:1508.06336 [cs, math].

Cedric Nugteren and Valeriu Codreanu. CLTune: A Generic Auto-Tuner for OpenCL Kernels. In *2015 IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip*, pages 195–202, September 2015. doi: 10.1109/MCSoC.2015.10.

Frank J. Poelwijk, Michael Socolich, and Rama Ranganathan. Learning the pattern of epistasis linking genotype and phenotype in a protein. *Nature Communications*, 10(1):4213, September 2019. ISSN 2041-1723. doi: 10.1038/s41467-019-12130-8. Number: 1 Publisher: Nature Publishing Group.

Karen S. Sarkisyan, Dmitry A. Bolotin, Margarita V. Meer, Dinara R. Usmanova, Alexander S. Mishin, George V. Sharonov, Dmitry N. Ivankov, Nina G. Bozhanova, Mikhail S. Baranov, Onuralp Soylemez, Natalya S. Bogatyreva, Peter K. Vlasov, Evgeny S. Egorov, Maria D. Logacheva, Alexey S. Kondrashov, Dmitry M. Chudakov, Ekaterina V. Putintseva, Ilgar Z. Mamedov, Dan S. Tawfik, Konstantin A. Lukyanov, and Fyodor A. Kondrashov. Local fitness landscape of the green fluorescent protein. *Nature*, 533(7603):397–401, May 2016. ISSN 1476-4687. doi: 10.1038/nature17995. Number: 7603 Publisher: Nature Publishing Group.

Robin Scheibler, Saeid Haghighatshoar, and Martin Vetterli. A fast hadamard transform for signals with sublinear sparsity in the transform domain. *IEEE Transactions on Information Theory*, 61(4):2115–2132, 2015.

Nicholas C Wu, Lei Dai, C Anders Olson, James O Lloyd-Smith, and Ren Sun. Adaptation in protein fitness landscapes is facilitated by indirect paths. *eLife*, 5:e16965, July 2016. ISSN 2050-084X. doi: 10.7554/eLife.16965. Publisher: eLife Sciences Publications, Ltd.