Learning to Reason with Neural Networks: Generalization, Unseen Data and Boolean Measures

Emmanuel Abbe[*] EPFL
Samy Bengio Apple
Elisabetta Cornacchia EPFL
Jon Kleinberg Cornell University
Aryo Lotfi EPFL
Maithra Raghu Google Research
Chiyuan Zhang Google Research

Abstract

This paper considers the Pointer Value Retrieval (PVR) benchmark introduced in [ZRKB21], where a 'reasoning' function acts on a string of digits to produce the label. More generally, the paper considers the learning of logical functions with gradient descent (GD) on neural networks. It is first shown that in order to learn logical functions with gradient descent on symmetric neural networks, the generalization error can be lower-bounded in terms of the *noise-stability* of the target function, supporting a conjecture made in [ZRKB21]. It is then shown that in the distribution shift setting, when the data withholding corresponds to freezing a single feature (referred to as canonical holdout), the generalization error of gradient descent admits a tight characterization in terms of the *Boolean influence* for several relevant architectures. This is shown on linear models and supported experimentally on other models such as MLPs and Transformers. In particular, this puts forward the hypothesis that for such architectures and for learning logical functions such as PVR functions, GD tends to have an *implicit bias towards low-degree representations*, which in turn gives the Boolean influence for the generalization error under quadratic loss.

1 Introduction

Recently [ZRKB21] introduced the pointer value retrieval (PVR) benchmark. This benchmark consists of a supervised learning task on MNIST [LCB10] digits with a 'logical' or 'reasoning' component in the label generation. More specifically, the functions to be learned are defined on MNIST digits organized either sequentially or on a grid, and the label is generated by applying some 'reasoning' on these digits, with a specific digit acting as a pointer on a subset of other digits from which a logical/Boolean function is computed to generate the label.

For instance, consider the PVR setting for binary digits in the string format, where a string of MNIST digits is used as input. Consider in particular the case where only 0 and 1 digits are used, such as in the example of Figure 1. The label of this string is defined as follows: the first 3 bits 101 define the pointer in binary expansion, and the pointer points to a window of a given length, say 2 in this example. Specifically, the pointer points at the first bit of the window. To generate the label, one has

[*] Authors are in alphabetical order.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

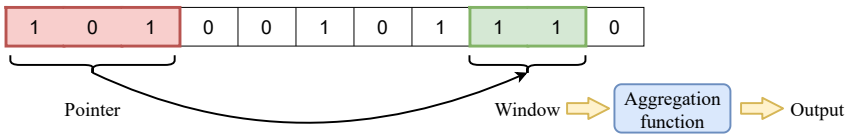


Figure 1: An example of a PVR function with a window size of 2. The first 3 bits are the pointer, which points to a window in the subsequent bits. Specifically, the number indicated by the pointer bits in binary expansion gives the position of the first bit of the window. The label is then produced by applying some fixed aggregation function to the window bits (e.g., parity, majority-vote, etc.).

to thus look the 6th window[2] of length 2 given by 11, and there the label is produced by applying some fixed function, such as the parity (so the label would be 0 in this example). In [ZRKB21], the PVR benchmark is also defined for matrices of digits rather than strings; we focus here on the string version that captures all the purposes of the PVR study. This benchmark is introduced to understand the limits of deep learning on tasks that go beyond classical image recognition, investigating in particular the trade-off between memorization and reasoning by acting with a particular distribution shift at testing (see further details below).

In order to learn such PVR functions, one has to first learn the digit identification and then the logical component on these digits. Handling both tasks successfully at once is naturally more demanding than succeeding at the latter assuming the first is successful. Here, we focus on the 'logical component' as a necessary component to learn, and this corresponds to learning a Boolean function. The overall function that maps the pixels of an image to its label in the PVR is of course also a Boolean function (like any computer-encoded function), but the structural properties of such meta-functions are more challenging to describe, and left for future work. In any case, to understand the limits of deep learning on such benchmarks, we focus on investigating first the limits of deep learning on the logical/Boolean component.

We next re-state formally one of the PVR benchmarks from [ZRKB21], focusing on binary digits for simplicity. We will use the alphabet $\{0, 1\}$ to describe the benchmark to connect to the MNIST dataset, but will later switch to the alphabet $\{+1, -1\}$ for the problem of learning Boolean functions with neural networks. Recall for $d \in \mathbb{N}$, $\mathbb{F}_2^d = \{0, 1\}^d$.

Definition 1 (Boolean PVR with *sliding* windows)**.** *The input x consists of n bits and the label is generated as*

$$f(x_1, \ldots, x_n) = g(x_{P(x^p)}, \ldots, x_{P(x^p)+w-1}). \tag{1}$$

where p is the number of bits in the pointer, w is the window size, $g : \mathbb{F}_2^w \to \mathbb{R}$ is the aggregation function, $P : \mathbb{F}_2^p \to [p+1 : n]$ is the pointer map, x_j denotes the bit in position j and $x^p = (x_1, ..., x_p)$ denotes the pointer bits. We often set $n = p + 2^p$, and hence the last window starts at the last digit.

In words, the first p bits give a pointer to the beginning of a window of w consecutive bits, and the label is produced by applying a Boolean function g on these w bits.

Remark 1. If $w > 1$, for some values of the pointer, $P(x^p) + w - 1$ would exceed the dimension of input n. This issue can be solved by using cyclic indices or by using non-overlapping windows as defined in Appendix D. However, in the experiments of this paper, we mainly truncate windows (if necessary) in order to capture the underlying asymmetries (e.g., the last windows have smaller sizes).

In this paper we consider the problem of learning in the holdout setting, i.e., when some data are withheld (or 'unseen') during training. We focus on a specific type of holdout setting, that we call 'canonical holdout', which we define here. Extensions to other types of holdout are also of interest, and left for future work.

Definition 2 (Canonical holdout)**.** *Let $\mathcal{F}$ be the class of Boolean functions on n bits and let f be a specific function in that class. For $k \in [n]$, consider the problem of learning $\mathcal{F}$ from samples $(X_{-k}, f(X_{-k}))$, where the X_{-k}'s are independently drawn from the distribution that freezes component k to 1 and that draws the other components i.i.d. Bernoulli$(1/2)$. Let $\tilde{f}_{-k}$ be the function*

[2] pointer 000 points at the first window, pointer 001 at the second window, and so on. Thus pointer 101, that is equal to 5 in binary expansion, points at the 6th window.

learned under this training distribution. We are interested in the generalization error with square loss when the k-th bit is not frozen at testing, i.e.,

$$\mathrm{gen}(\mathcal{F}, \tilde{f}_{-k}) = \frac{1}{2}\mathbb{E}_{X\sim_U \mathbb{F}_2^n, f\sim_U \mathcal{F}}(f(X) - \tilde{f}_{-k}(X))^2, \tag{2}$$

where by $X \sim_U \mathbb{F}_2^n$ we mean that X is chosen uniformly at random from $\mathbb{F}_2^n$, and similarly for $f \sim_U \mathcal{F}$. We denote by $\tilde{f}$ the function learned in the case where the k-th component is not frozen at training, i.e., when the train and test distributions are both uniform on the Boolean hypercube, and use the following notation for the corresponding generalization error:

$$\mathrm{gen}(\mathcal{F}, \tilde{f}) = \frac{1}{2}\mathbb{E}_{X\sim_U \mathbb{F}_2^n, f\sim_U \mathcal{F}}(f(X) - \tilde{f}(X))^2. \tag{3}$$

The 'canonical' holdout is thus a special case of holdout where a single feature/bit is frozen at training. In [ZRKB21], different types of holdout are allowed where a given string is absent from the windows rather than a given bit as in the canonical version. We believe that the canonical holdout still captures the essence of the PVR benchmark: other windows will get to see the strings that are withheld due to the bit freezing, and thus the goal is still to investigate whether neural networks trained by GD will manage to patch together the different pieces.

Finally, we will often consider neural network architectures that have invariances on the input, such as permutation invariance. In the PVR setting, this means that we do not assume that the learner has knowledge of which bits are in the window. In such cases, instead of learning a class of function $\mathcal{F}$, one can equivalently talk about learning a single function f (with the implicit $\mathcal{F}$ defined as the orbit of f through the permutation group), and define

$$\mathrm{gen}(f, \tilde{f}_{-k}) = \frac{1}{2}\mathbb{E}_{X\sim_U \mathbb{F}_2^n}(f(X) - \tilde{f}_{-k}(X))^2 \quad \text{and} \quad \mathrm{gen}(f, \tilde{f}) = \frac{1}{2}\mathbb{E}_{X\sim_U \mathbb{F}_2^n}(f(X) - \tilde{f}(X))^2.$$

1.1 Contributions of this paper

The contributions of this paper are:

1. In the matched setting (i.e., train and test distributions are matching), we prove a lower-bound (Theorem 1) on the generalization error for gradient descent[3] (GD), that degrades for functions having large *noise-sensitivity* (or low noise-stability), supporting thereby with a formal result a conjecture put forward in [ZRKB21].
2. In the mismatched setting, specifically in the canonical holdout where a single feature is frozen at training and released to be uniformly distributed at testing, we hypothesize that *(S)GD on the square loss*[4] *and on certain network architectures such as MLPs and Transformers has an implicit bias towards low-degree representations when learning logical functions such as Boolean PVR functions.* This gives a new insight into the implicit bias study of GD on neural networks for the case of logical functions.

   We then show (Lemma 1) that under this hypothesis, the generalization error in the canonical holdout setting is given by the Boolean influence, a central notion in Boolean Fourier analysis.
3. We provide experiments supporting this hypothesis for various target functions and architectures such as MLPs and the Transformers [VSP+17] in Section 3.2 and Appendix F. These rely on mini-batch stochastic gradient descent (see Section 3.2 for more details).
4. We establish formally the hypothesis for GD and linear regression models in Section 3.3, and conduct experiments to support it on multi-layer neural networks of large depths and small initialization scales in Section 3.3 (with further arguments in Appendix G).

1.2 The low-degree bias hypothesis: illustration

We now discuss the hypothesis put forward in this paper that "GD on certain architectures such as MLPs and Transformers has an implicit bias towards lower-degree representations when learning

[3] This part relies on population gradients with polynomial gradient precision as in [AS20, AKM+21]

[4] We do not expect the square loss to be critical, but it makes the connection to the influence more explicit.

PVR and, more generally, logical functions." Before starting our illustration let us recall some basic notions from Boolean analysis (we refer to [O'D14] for details). Any Boolean function $f : \{\pm 1\}^n \to \mathbb{R}$ can be written in terms of its Fourier-Walsh transform: $f(x) = \sum_{T \subseteq [n]} \hat{f}(T)\chi_T(x)$, where $\chi_T(x) = \prod_{i \in T} x_i$ and $\hat{f}(T) = 2^{-n} \sum_{x \in \{\pm 1\}^n} f(x)\chi_T(x)$ are respectively the basis elements and the coefficients of the Fourier-Walsh transform of f.

Definition 3 (Boolean influence [O'D14]). *Let $f : \{\pm 1\}^n \to \mathbb{R}$ be a Boolean function and let $\hat{f}$ be its Fourier-Walsh transform. The Boolean influence of variable $k \in [n]$ on f is defined by $\mathrm{Inf}_k(f) := \sum_{T \subseteq [n] : k \in T} \hat{f}(T)^2$. In particular, if $f : \{\pm 1\}^n \to \{\pm 1\}$, $\mathrm{Inf}_k(f) = \mathbb{P}(f(X) \neq f(X + e_k))$, where $X + e_k$ corresponds to the vector obtained by flipping the k-th component of X.*

Consider the following example of a PVR function with a 1-bit pointer, 2 overlapping windows of length 2, and parity for the aggregation function. We consider $f : \{\pm 1\}^n \to \{\pm 1\}$ (i.e., with ± 1 variables instead of $0, 1$, to simplify the expressions), with f given by

$$f(x_1, x_2, x_3, x_4) = \frac{1 + x_1}{2} x_2 x_3 + \frac{1 - x_1}{2} x_3 x_4. \tag{4}$$

We can rewrite f in terms of its Fourier-Walsh expansion (i.e., pulling out all the multivariate monomials appearing in the function, see Section 3.1 for more details), which gives

$$f(x_1, x_2, x_3, x_4) = \frac{1}{2} x_2 x_3 + \frac{1}{2} x_3 x_4 + \frac{1}{2} x_1 x_2 x_3 - \frac{1}{2} x_1 x_3 x_4. \tag{5}$$

Consider now training a neural network such as a Transformer as in Section 3.2 on this function, with quadratic loss, and a canonical holdout corresponding to freezing $x_2 = 1$ at training. Under this holdout, and under the 'low-degree implicit bias' hypothesis, the low-degree monomials are learned first (see experiments in Section 3.2), resulting in the following function being learned at training:

$$f_{-2}(x_1, x_2, x_3, x_4) = \frac{1}{2} x_3 + \frac{1}{2} x_3 x_4 + \frac{1}{2} x_1 x_3 - \frac{1}{2} x_1 x_3 x_4 = \frac{1 + x_1}{2} x_3 + \frac{1 - x_1}{2} x_3 x_4. \tag{6}$$

Thus, according to Lemma 1 proved in Appendix C, the generalization error is given by

$$\frac{1}{2} \mathbb{E}(f(X) - f_{-2}(X))^2 = \mathbb{P}(f(X) \neq f(X + e_2)) = \frac{1}{2}, \tag{7}$$

which is the probability that flipping the frozen coordinate changes the value of the target function, i.e., the Boolean influence (where we denoted by $X + e_2$ the vector obtained by flipping the second entry in X). As shown in this paper, neural networks tend to follow this trend quite closely, and we can prove this hypothesis on simple linear models. Notice that an ERM-minimizing function could have taken a more general form than a degree minimizing function, i.e.,

$$f_{-2}^{\mathrm{ERM}}(x) := \frac{1 + x_2}{2} f_{-2}(x) + \frac{1 - x_2}{2} r(x) \tag{8}$$

for any choice of $r : \{\pm 1\}^4 \to \{\pm 1\}$. In the special case of $r = 0$, f_{-2}^{ERM} corresponds to f_{-2} (the low-degree representation). For instance, among such ERM-minimizers, one can check that the minimum ℓ_2-norm interpolating solution would be given by

$$f_{-2}^{\ell_2}(x) := \frac{1}{4}(x_3 + x_2 x_3) + \frac{1}{4}(x_3 x_4 + x_2 x_3 x_4) + \frac{1}{4}(x_1 x_3 + x_1 x_2 x_3) - \frac{1}{4}(x_1 x_3 x_4 + x_1 x_2 x_3 x_4). \tag{9}$$

This gives a generalization error of $\frac{1}{2}\mathbb{E}(f(X) - f_{-2}^{\ell_2}(X))^2 = 4(\frac{1}{4})^2 = \frac{1}{4}$, i.e., half the error of f_{-2}, yet still bounded away from 0.

In order to improve on this, under the same canonical holdout with $x_2 = 1$, one would like to rely on a type of minimum description length bias, since describing f may be more efficient than f_{-2} due to the stronger symmetries of f. Namely, f corresponds to taking the parity on the middle two bits if $x_1 = 1$, and on the last two bits otherwise. On the other hand, f_{-2} requires changing the function depending on $x_1 = 1$ or $x_1 = -1$, since it is once the function x_3 and once the function $x_3 x_4$. So an implicit bias that would exploit such symmetries, featuring in PVR tasks, would give a different solution than the low-degree implicit bias, and could result in lower generalization error. We leave to future work to investigate this 'symmetry compensation' procedure.

1.3 Related literature

GD lower bounds. Several works have investigated the difficulties of learning a class of functions with descent algorithms and identified measures for the complexity in this setting. In particular, [BFJ+94, Kea98] prove that the larger the statistical dimension of a function class is, the more difficult it is for an statistical query (SQ) algorithm to learn. We refer to [AKM+21, DGKP19] for further references on SQ-like lower bounds. For GD with mini-batch and polynomial gradient precision, [AS20] uses the m-Cross-Predictability (CP) of a class of functions to show that classes sufficiently small CP and large batch-size are not efficiently learnable. This is further generalized in [AKM+21] to a broader range of batch size and gradient precision. In [AGNZ18], the noise-stability for deep neural networks is defined as the stability of each layer's computation to noise injected at lower layers, and is used to show correctness of a compression framework. However, no bounds on the generalization error of GD depending on the noise stability of the target function is derived in this work. The noise-stability, the statistical dimension and the cross-predictability are measures for some given function or function class. One can obtain measures that are defined for a dataset and a network architecture. For that purpose, [ACHM22] introduced the "Initial Alignment" (INAL) between the target function and the neural network at initialization and proves that for fully connected networks with Gaussian i.i.d. initialization, if the INAL is negligible, then GD will not learn efficiently. We remark that the problem of approximating and learning Boolean functions appear in other areas as well. For instance, [HPV19] considered the problem of approximating Boolean functions, using functions coming from restricted classes (namely k-juntas and linear Boolean functions); [JSR+19] proposes two algorithms to learn sparse Boolean formulae; and [Udo21] proposes techniques for modeling Boolean functions by mixed-integer linear inequalities.

Implicit bias. The implicit bias of neural networks trained with gradient descent has been extensively studied in recent years [NTS14, NBMS17, MGW+20, JT19, GLSS18a]. In particular, [SHN+17] proved that gradient descent on linearly-separable binary classification problems converges to the maximum ℓ_2-margin direction. Several subsequent works studied the implicit bias in classification tasks on various networks architectures, e.g., homogeneous networks [LL19], two layers networks in the mean field regime [CB20], linear and ReLU fully connected networks [VSS21], and convolutional linear networks [GLSS18b]. Among regression tasks, the problem of implicit bias has been analysed for matrix factorization tasks [GWB+17, RC20, ACHL19], and also gradient flow on diagonal networks [PPVF21]. However, all these works consider functions with real inputs, instead of logical functions which are the focus of this work. On the other hand, as discussed in Section 1.2, the Boolean influence generalization characterization reflects the implicit bias of GD on neural networks to learn low-degree representations. Similar types of phenomena can implicitly be found in [XZX18, XZL+19, RBA+19], in particular as the "spectral bias" in the context of real valued functions decomposed in the classical Fourier basis (where the notion of lower degree is replaced by low frequencies). In [ABAB+21, ABAM22], the case of Boolean functions is considered as in this paper, and it is established that for various 'regular' architectures (having some symmetry in their layers), gradient descent can learn target functions that satisfy a certain 'staircase' property. However, these papers do not investigate lower-bounds in terms of noise stability, neither distribution shift as in the canonical holdout. We note that, in the context of Boolean functions, the problem of regularizing a learning algorithm to avoid overfitting appears in other areas of research, e.g., in the analysis of fitness functions in biology [ANO+21].

Distribution shift. Many works were aimed at characterizing when a classifier trained on a training distribution (also called the "source" distribution) performs well on a different test domain (also called the "target" distribution) [QCSSL09]. On the theoretical side, [BDBC+10] obtains a bound of the target error of a general empirical risk minimization algorithm in terms of the source error and the divergence between the source and target distribution, in a setting where the algorithm has access to a large dataset from the source distribution and few samples from the target distribution. We refer to [SQZY17] for a further result in a similar setting. Instead, in this work we focus on gradient descent on neural networks in the setting where no data from the target distribution is accessible. On the empirical level, several benchmarks have been proposed to evaluate performance for a wide range of models and distribution shifts [WGS+22, MTR+21, SKL+21]. Despite this significant body of works on distribution shift, we did not find works that related the generalization error under holdout shift in terms of the Boolean influence.

2 Matched setting: a formal lower-bound on noise stability and generalization

Our first result provides a lower bound on the generalization error in the setting where the training and test distributions of the inputs are both uniform on the Boolean hypercube, i.e., the "matched setting". We directly relate the generalization error achieved by GD, or SGD with large batch-size, to the complexity of the task, providing theoretical support to the empirical claim in [ZRKB21], that complex tasks are more difficult to learn for GD/SGD on neural networks. The latter work used the noise sensitivity as a dual measure of the target function complexity, whereas we use here the noise stability ($\mathrm{Stab}_\delta[f]$).

Definition 4 (Noise stability)**.** *Let $f : \{\pm 1\}^n \to \mathbb{R}$ and $\delta \in [0, 1/2]$. Let X be uniformly distributed on the Boolean n-dimensional hypercube, and let Y be formed from X by flipping each bit independently with probability δ. We define the δ-noise stability of f by $\mathrm{Stab}_\delta[f] := \mathbb{E}_{(X,Y)}[f(X) \cdot f(Y)]$.*

Intuitively, $\mathrm{Stab}_\delta[f]$ measures how stable the output of f is to a perturbation that flips each input bit with probability δ, independently. The noise stability can be easily related to the noise sensitivity $\mathrm{NS}_\delta[f]$[5], used in [ZRKB21].

The generalization error depends as well on the complexity of the network, which is quantified in terms of the number of edges in the network, the number of time steps, and gradients precision in the gradient descent algorithm. We give one more definition before stating our result.

Definition 5 (N-Extension)**.** *For a function $f : \mathbb{R}^n \to \mathbb{R}$ and for $N > n$, we define its N-extension $\bar{f} : \mathbb{R}^N \to \mathbb{R}$ as $\bar{f}(x_1, x_2, ..., x_n, x_{n+1}, ..., x_N) = f(x_1, x_2, ..., x_n)$.*

Theorem 1. *Consider a fully connected neural network of size E with initialization that is target agnostic in the sense that it is invariant under permutations of the input[6]. Let $f : \{\pm 1\}^n \to \{\pm 1\}$ be a balanced target function. Let $\bar{f}$ be the $2n$-extension of f as defined in Definition 5. Let $f_{\mathrm{NN}}^{(t)}$ be the output of noisy-GD with gradient range A, batch-size b, learning rate γ and noise scale σ (see Def. 8) after t time steps. Then, for δ small enough[7] and for b large enough[8], the generalization error satisfies*

$$\mathrm{gen}\left(\bar{f}, f_{\mathrm{NN}}^{(t)}\right) \geq 1/2 - \frac{\gamma t \sqrt{E} A}{\sigma} \cdot \mathrm{Stab}_\delta[f]^{1/4}. \tag{10}$$

Theorem 1 states a lower bound for learning in the extended input space. We use the input doubling to guarantee that the hypothesis class $\mathcal{F}$ resulting from the orbit of $\bar{f}$, i.e., $\mathrm{orb}(\bar{f}) = \{\bar{f} \circ \pi : \pi \in S_{2n}\}$, is not degenerate to a single function, as this could then be learned using a proper choice of the initialization (i.e., one can simply set the weights of the neural network at initialization to represent the unique function, if the network has enough expressivity). Instead, the input doubling prohibits such representation shortcut and ensures that the structural properties of the function is what creates the difficulty of learning, irrespective of the choice of the initialization. For instance, consider the full parity function on all of the input bits. The orbit of this function contains only that specific function, and one can learn that function by choosing a proper initialization on a neural net of depth 2. However, with an input doubling, this function becomes hard to learn *no matter what* the initialization is [Kea98, AS20]. One can remove this input doubling requirement by assuming that f is non-extremal (i.e., no terms of degree $\theta(n)$ in the Fourier basis) and non-dense (i.e., poly(n)-sized Fourier spectrum), see Appendix B.

The proof of Theorem 1 uses [AS20] which obtains a lower-bound in terms of the cross-predictability (CP) (instead of the noise stability). The CP is a complexity measure of a *class* of functions, rather than a single function. For the orbit of $\bar{f}$, the CP is defined as $\mathrm{CP}(\mathrm{orb}(\bar{f})) = \mathbb{E}_{\pi \sim_U S_{2n}}[\mathbb{E}_{X \sim_U \mathbb{F}_2^{2n}}[\bar{f}(X) \cdot \bar{f} \circ \pi(X)]^2]$. We give a more general definition of CP in Appendix A (we believe that the CP also extends to other invariances than permutations and non i.i.d. distributions).

---

[5] Specifically, for binary-valued functions, $\mathrm{NS}_\delta[f] = \frac{1}{2} - \frac{1}{2}\mathrm{Stab}_\delta[f]$.

[6] One could consider other groups of invariances than the full permutation group; this is left for future work.

[7] Generally, this holds for any δ such that $CP(\mathrm{orb}(\bar{f})) \leq \mathrm{Stab}_\delta[f]$; in particular, this holds for $\delta < 1/4$ under input doubling or for some $\delta > 0$ under non-extremal and non-dense assumptions.

[8] This holds for $b \geq 1/CP(\mathrm{orb}(\bar{f}))$.

Theorem 1 states that if the target function is highly noise unstable, specifically if there exists $\delta < 1/4$ such that $\mathrm{Stab}_\delta[f]$ decreases faster than any inverse polynomial in n, then GD will not learn the $2n$-extension of f (or f itself if it is non-extremal/dense) in polynomial time and with a polynomially sized neural network initialized at random. So in that sense, the noise-stability gives a proxy to generalization error, as observed in [ZRKB21]. More specifically, this result is about failure of the weakest form of learning no matter what the architecture is. One could also consider 'regular' architectures (such as with isotropic layers) and stronger learning requirements; for this it is known that the 'Fourier leap' from [ABAB+21] is a relevant complexity measure, and we leave investigations of regular architectures to future work. More details of the proof of Theorem 1 are in Appendix A. In Appendix D, we explain how the noise stability of PVR functions can be computed, and the implications to the result of Theorem 1.

3 Mismatched setting

In this section, we focus on the generalization error under the distribution shift setting, more specifically the canonical holdout setting defined in Definition 2. Namely, assume that at training component k is frozen to 1, and assume it to be released to $\mathrm{Unif}\{\pm 1\}$ at testing. Our experiments show that in this setting, for some relevant architectures, the generalization error is close to the value of a standard measure in Boolean analysis, namely the Boolean influence of variable k on f.

3.1 Boolean influence and generalization

To explain the connection between generalization error and Boolean influence, we start with a simple lemma relating the Boolean influence to the ℓ_2-distance between the true target function and the function obtained by freezing component k (which we call the "frozen function").

Lemma 1. *Let $f : \{\pm 1\}^n \to \mathbb{R}$ be a Boolean function and let f_{-k} be defined as $f_{-k}(x) := f(x_{-k})$ where $x_{-k}(i) = 1$ if $i = k$ and $x_{-k}(i) = x(i)$ otherwise. Then, $\frac{1}{2}\mathbb{E}_X(f(X) - f_{-k}(X))^2 = \mathrm{Inf}_k(f)$.*

The proof of Lemma 1 can be found in Appendix C. In Section 3.2, we present experiments that demonstrate the relation between the Boolean influence and the generalization error for different architectures. In Section 3.3, we focus on linear models, namely, linear regression and linear networks.

3.2 Experiments

We consider three architectures for our experiments: multi-layer perceptron (MLP) with 4 hidden layers, the Transformer [VSP+17], and MLP-Mixer [THK+21]. For each architecture, we freeze different coordinates separately and evaluate our models. In other words, we train the model while freezing coordinate 1, then coordinate 2 and so on, until coordinate n and compare the generalization error with the Boolean influence of the corresponding coordinates of the target function. We train our models using ℓ_2 loss and mini-batch SGD with momentum and batch-size of 64 as the optimizer. Moreover, we have repeated each experiment 40 times and averaged the results. Furthermore, note that for the MLP model, we pass the Boolean vector directly to the model. However, for the Transformer and MLP-Mixer, we first encode $+1$ and -1 tokens into 256-dimensional vectors using a shared embedding layer, and then we pass the embedded input to the models. More details on training procedure as well as further experiments are presented in Appendix F.[9]

Influence vs. canonical holdout generalization. In this section, we consider a Boolean PVR function (as in Def. 1) with 3 pointer bits to be learned by the neural networks. For this function we set window size to 3 and use majority-vote (defined as $g(x_1, \ldots, x_r) = \mathrm{sign}(x_1 + \cdots + x_r)$, outputting -1, 0, or 1) as the aggregation function on the windows. The generalization error of the models on this PVR function and its comparison with the Boolean influence are presented in Figure 2. The x-axis corresponds to the index of the frozen coordinate, that is from 4 to 11 (we do not freeze the pointer bits). On the y-axis, for each frozen coordinate we report the generalization error obtained in the corresponding holdout setting for MLP, the Transformer, and MLP-Mixer, together with the value of the Boolean influence of the frozen coordinate on the target function. It can be seen that the generalization error of MLP and the Transformer can be well approximated by the Boolean influence.

[9] Code: https://github.com/aryol/BooleanPVR

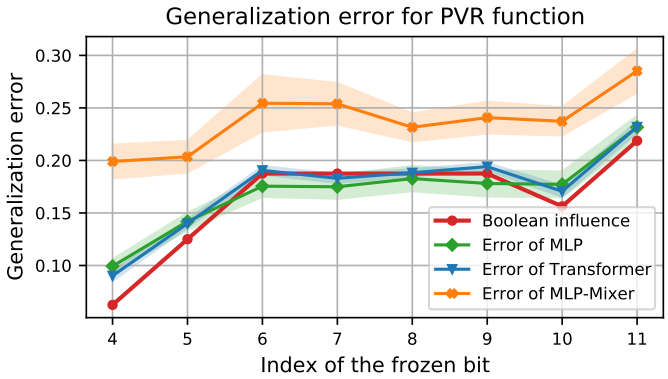


Figure 2: Comparison between the generalization loss in the canonical distribution shift setting and the Boolean influence for a PVR function with 3 pointer bits, window size 3, and majority-vote aggregation function. See Appendix F for further experiments.

Whereas, the generalization error of MLP-Mixer follows the trend of the Boolean influence with an offset. We remark that in Figure 2, the value of the Boolean influence (and gen. error) in the PVR task varies across different indices due to boundary effects and the use of truncated windows (see Def. 1). We refer to Appendix F for further experiments on other target functions.

Implicit bias towards low-degree representation. Consider the problem of learning a function f in the canonical holdout setting freezing $x_k = 1$. Denote the Fourier coefficients of the frozen function f_{-k} (as defined in Lemma 1), by $\hat{f}_{-k}(S)$, for all $S \subseteq [n] := \{1, ..., n\}$ (recall, $\hat{f}_{-k}(S) = \mathbb{E}_X[f_{-k}(X)\chi_S(X)]$). For S such that $k \notin S$, the neural network can learn coefficient $\hat{f}_{-k}(S)$ using either $\chi_S(x)$ or $x_k \cdot \chi_S(x) = \chi_{S\cup\{k\}}(x)$ (since these are indistinguishable at training). The low-degree implicit bias states that neural networks have a preference for the lower degree monomial χ_S. More precisely, χ_S is learned faster than $\chi_{S\cup\{k\}}$ and thus the term $\hat{f}_{-k}(S)\chi_S(x)$ in the Fourier expansion of f_{-k}, is mostly learned by the lower degree monomial. Consequently, according to Lemma 1, the generalization error will be close to the Boolean influence. Figure 3 shows this bias empirically for the above mentioned PVR function and for frozen coordinate $k = 6$. Figure 3 (left) shows that the MLP model has a strong preference for low-degree monomials, Similarly, Figure 3 (bottom) shows that the Transformer also captures the monomials in the original function using monomials that exclude the frozen index. Therefore the generalization errors of the MLP and Transformer are very close to the Boolean influence as seen in Figure 2. Whereas, Figure 3 (right) shows that the MLP-Mixer model has a weaker preference for lower degree monomials (e.g., it learns 1 and x_6 to the same extent) and hence, its generalization error follows the trend of Boolean influence with an offset, which is also presented in Figure 2. We refer to Appendix F for additional experiments.

3.3 Linear Models

In this section, we will focus on linear functions and linear models. First, we state a theorem for linear regression models. Furthermore, we show experiments on the generalization error of linear neural networks and its relation with the depth and initialization of the model.

Theorem 2. *Let $f : \{\pm 1\}^n \to \mathbb{R}$ be a linear function, i.e., $f(x_1, \cdots, x_n) = \hat{f}(\emptyset) + \sum_{i=1}^n \hat{f}(\{i\})x_i$. Consider the canonical holdout where the k-th component is frozen at training for a linear regression model where weights and biases are initialized independently with the same mean and variance σ^2. Also assume the frozen function is unbiased, i.e., $\mathbb{E}_{X_{-k}}[f(X)] = 0$. In this case, the expected generalization error (over different initializations) of the function learned by GD after t time steps is given by $\mathbb{E}_{\Theta^0}[\mathrm{gen}(f, \tilde{f}^{(t)}_{-k})] = \mathrm{Inf}_k(f) + o_{\sigma^2}(1) + O(e^{-ct})$, where c is a constant dependent on the learning rate. Moreover, if the frozen function is biased, the expected generalization error is equal to $\mathbb{E}_{\Theta^0}[\mathrm{gen}(f, \tilde{f}^{(t)}_{-k})] = \frac{(\hat{f}(\emptyset)-\hat{f}(\{k\}))^2}{4} + o_{\sigma^2}(1) + O(e^{-ct})$.*

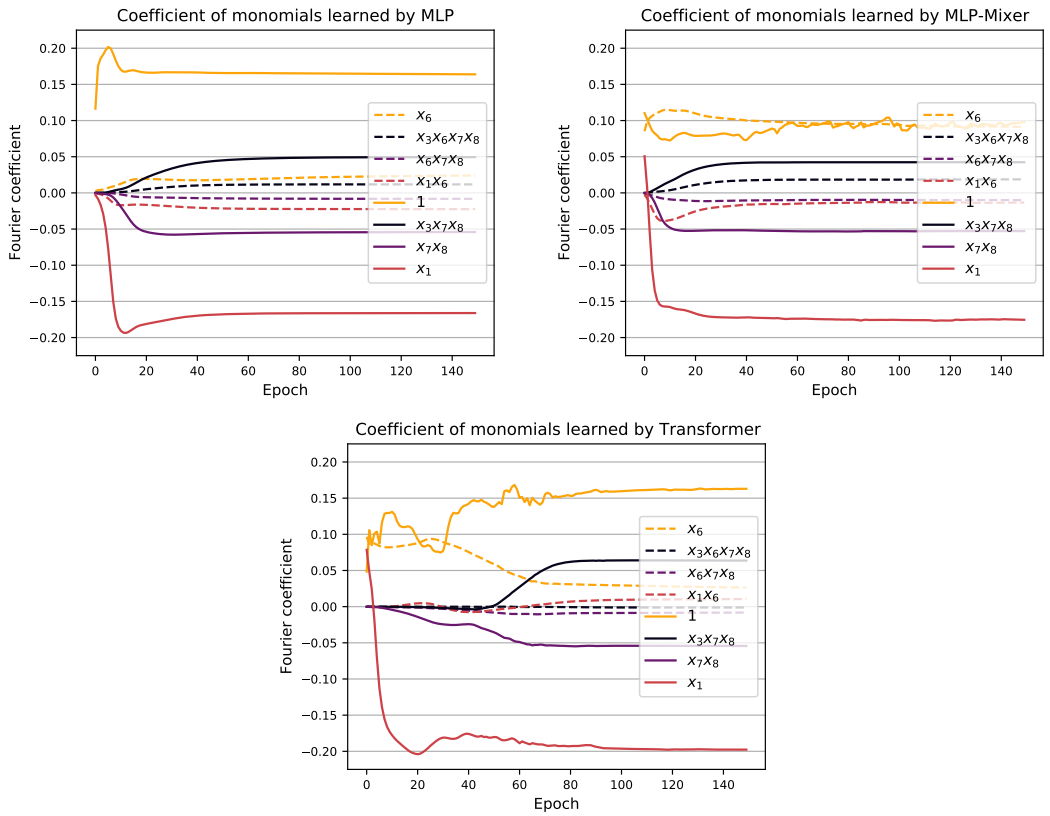


Figure 3: The coefficients of a selected group of monomials learned by the MLP (left), MLP-Mixer (right) and the Transformer (bottom) when learning the aforementioned PVR function, with $x_6 = 1$ frozen during the training. The coefficient of these monomials in the original function are $\hat{f}(\{6\}) = 0.1875$, $\hat{f}(\{3,6,7,8\}) = 0.0625$, $\hat{f}(\{6,7,8\}) = -0.0625$, and $\hat{f}(\{1,6\}) = -0.1875$. One can observe that the monomials of the lowest degree are indeed picked up first during the training of the MLP and Transformer, which explains the tight approximation of the Boolean influence for the generalization error in these cases. In contrast, the MLP-Mixer also picks up some contribution from the higher degree monomials including the frozen bit x_6.

Remark 2 (Kernel regression). Our result would still hold if, instead of linear regression, we performed kernel regression with a kernel K that is invertible under the training distribution, i.e., such that $\mathbb{E}_{X,X'\sim\mathcal{U}_{-k}}[K(X,X')]$ is invertible. Furthermore, one could try to remove the invertibility assumption by adding a regularization term.

Remark 3 (Staircase learning). An attempt to extend the above result to non-linear function consists of considering staircase functions [ABAB+21, ABAM22], and extending [ABAM22] to a setting with bias in order to show that SGD learns the lowest-degree representation under proper mean-field initialization of depth-2 neural networks.

The proof of Theorem 2 is presented in Appendix C. In the rest of this section, we empirically show that the condition of zero bias stated in Theorem 2 is no longer necessary if linear neural networks of large enough depth or small enough initialization are used. In fact, the generalization error of linear neural networks makes a transition from the value proved in Theorem 2 to the Boolean influence as deeper models or smaller scales of initialization are used. We take $f(x_1, x_2, \ldots, x_{11}) = 1 + 2x_1 - 3x_2 + 4x_3 - \cdots - 11x_{10} + 12x_{11}$ as the function that we want to learn. We consider linear neural networks with hidden layers of size 256. Figure 4 (left) shows the effect of depth: we initialize weights and biases independently using the uniform distribution $\mathcal{U}(-\frac{1}{\sqrt{N_{in}}}, \frac{1}{\sqrt{N_{in}}})$ where N_{in} is the input dimension of the respective layer. We plot the generalization error in the holdout setting with respect to the corresponding frozen coordinate at training, together with the value of the Boolean influence of each coordinate. We observe that with the increase of depth, the generalization error tends to the Boolean influence. Figure 4 (right) show the role of initialization: we take a linear neural network with 3 layers and we initialize weights and biases using the uniform distribution

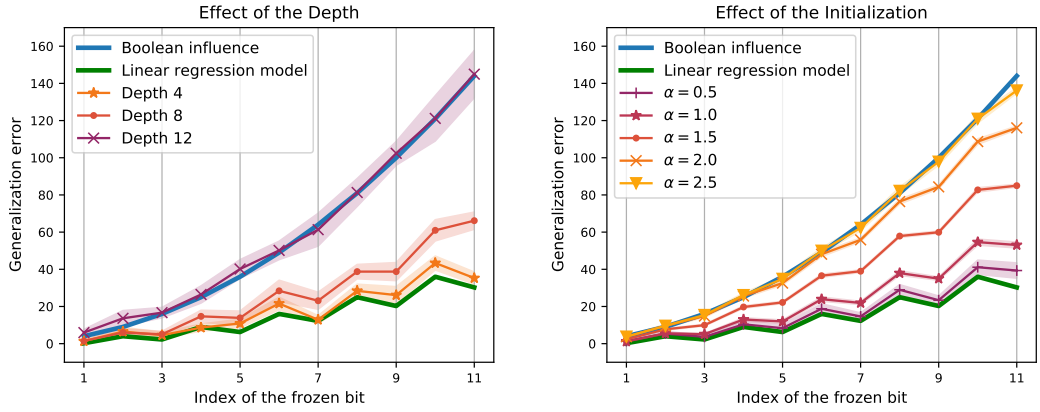


Figure 4: Effect of depth (left) and initialization scale (right) on the out-of-distribution generalization error of linear neural networks. Generalization error tends to the Boolean influence as depth increases (left) or as the initialization scale decreases (right).

$\mathcal{U}(-N_{in}^{-\alpha}, N_{in}^{-\alpha})$ with α taking value in $\{0.5, 1, 1.5, 2, 2.5\}$. It can be seen that as the initialization scale decreases, the generalization error tends to the Boolean influence.

4 Conclusion

This paper first establishes a formal result that supports a conjecture made in [ZRKB21], relating the noise sensitivity of a target function to the generalization error. This gives a first connection between a central measure in Boolean analysis, the noise-sensitivity, and the generalization error when learning Boolean functions with GD. The paper then investigates the generalization error under the canonical holdout. The 'low-degree implicit bias hypothesis' is put forward and supported both experimentally and theoretically for certain architectures. This gives a new insight on the implicit bias of GD when training neural networks, that is specific to Boolean functions such as the Boolean PVR, and that relates to the fact that certain networks tend to greedily learn monomials by incrementing their degree. In particular, this allows to characterize the generalization error in terms of the Boolean influence, a second central notion in Boolean Fourier analysis. Boolean measures thus seem to have a role to play in understanding generalization when learning of 'reasoning' or 'logical' functions. There are now many directions to pursue such as: (1) extending the realm of architectures/models for which we can prove formally the Boolean influence tightness, (2) considering more general holdout or distribution shift models, (3) investigate how the picture changes when the bits/digits are given by MNIST images, (4) better understanding when the low-degree implicit bias is taking place or not, within and beyond PVR, since the Boolean influence is not always tight in our experiments (e.g., MLP-Mixers seem to have a worse performance than the Boolean influence on PVR; see also Appendix F for other functions), (5) investigating how to 'revert' the implicit bias towards low-degree when it is taking place, to compensate for the unseen data; this will require justifying and engineering why certain symmetries are favorable in the learned function.

Acknowledgments and Disclosure of Funding

We thank Raphaël Berthier (EPFL) and Jan Hązła (EPFL) for useful discussions.

References

[ABAB+21] Emmanuel Abbe, Enric Boix-Adsera, Matthew Brennan, Guy Bresler, and Dheeraj Nagaraj. The staircase property: How hierarchical structure can guide deep learning, NeurIPS, 2021. 5, 7, 9, 28

[ABAM22] Emmanuel Abbe, Enric Boix-Adsera, and Theodor Misiakiewicz. The merged-staircase property: a necessary and nearly sufficient condition for sgd learning of sparse functions on two-layer neural networks, 2022. 5, 9, 28

[ACHL19] Sanjeev Arora, Nadav Cohen, Wei Hu, and Yuping Luo. Implicit regularization in deep matrix factorization, 2019. 5

[ACHM22] Emmanuel Abbe, Elisabetta Cornacchia, Jan Hązła, and Christopher Marquis. An initial alignment between neural network and target is needed for gradient descent to learn, 2022. 5, 22, 23

[AGNZ18] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach, 2018. 5

[AKM+21] Emmanuel Abbe, Pritish Kamath, Eran Malach, Colin Sandon, and Nathan Srebro. On the power of differentiable learning versus PAC and SQ learning. In *Advances in Neural Information Processing Systems*, volume 34, 2021. 3, 5, 16

[ANO+21] 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, 2021. 5

[AS20] Emmanuel Abbe and Colin Sandon. On the universality of deep learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20061–20072. Curran Associates, Inc., 2020. 3, 5, 6, 15, 16, 17

[BDBC+10] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Vaughan. A theory of learning from different domains. *Machine Learning*, 79:151–175, 2010. 5

[BFJ+94] Avrim Blum, Merrick Furst, Jeffrey Jackson, Michael Kearns, Yishay Mansour, and Steven Rudich. Weakly learning DNF and characterizing statistical query learning using Fourier analysis. In *Symposium on Theory of Computing (STOC)*, pages 253–262, 1994. 5

[CB20] Lenaic Chizat and Francis Bach. Implicit bias of gradient descent for wide two-layer neural networks trained with the logistic loss. 2020. 5

[DBK+20] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 24

[DGKP19] Abhimanyu Das, Sreenivas Gollapudi, Ravi Kumar, and Rina Panigrahy. On the learnability of deep random networks, 2019. 5

[GLSS18a] Suriya Gunasekar, Jason Lee, Daniel Soudry, and Nathan Srebro. Characterizing implicit bias in terms of optimization geometry, 2018. 5

[GLSS18b] Suriya Gunasekar, Jason Lee, Daniel Soudry, and Nathan Srebro. Implicit bias of gradient descent on linear convolutional networks, 2018. 5

[GWB+17] Suriya Gunasekar, Blake Woodworth, Srinadh Bhojanapalli, Behnam Neyshabur, and Nathan Srebro. Implicit regularization in matrix factorization, 2017. 5

[HPV19] Mohsen Heidari, S. Sandeep Pradhan, and Ramji Venkataramanan. Boolean functions with biased inputs: Approximation and noise sensitivity. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 1192–1196, 2019. 5

[JSR+19] Susmit Jha, Tuhin Sahai, Vasumathi Raman, Alessandro Pinto, and Michael Francis. Explaining ai decisions using efficient methods for learning sparse boolean formulae. *J. Autom. Reason.*, 63(4):1055–1075, dec 2019. 5

[JT19] Ziwei Ji and Matus Telgarsky. Characterizing the implicit bias via a primal-dual analysis, 2019. 5

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 25

[Kea98] Michael Kearns. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM*, 45(6):983–1006, 1998. 5, 6

[LCB10] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010. 1

[LL19] Kaifeng Lyu and Jian Li. Gradient descent maximizes the margin of homogeneous neural networks, 2019. 5

[MGW+20] Edward Moroshko, Suriya Gunasekar, Blake Woodworth, Jason D. Lee, Nathan Srebro, and Daniel Soudry. Implicit bias in deep linear classification: Initialization scale vs training accuracy, 2020. 5

[MTR+21] John Miller, Rohan Taori, Aditi Raghunathan, Shiori Sagawa, Pang Wei Koh, Vaishaal Shankar, Percy Liang, Yair Carmon, and Ludwig Schmidt. Accuracy on the line: On the strong correlation between out-of-distribution and in-distribution generalization, 2021. 5

[NBMS17] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nathan Srebro. Exploring generalization in deep learning, 2017. 5

[NTS14] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning, 2014. 5

[O'D14] Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014. 4, 16, 21, 23

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 25

[PPVF21] Scott Pesme, Loucas Pillaud-Vivien, and Nicolas Flammarion. Implicit bias of sgd for diagonal linear networks: a provable benefit of stochasticity, 2021. 5

[QCSSL09] Joaquin Quionero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D. Lawrence. *Dataset Shift in Machine Learning*. The MIT Press, 2009. 5

[RBA+19] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *International Conference on Machine Learning*, pages 5301–5310. PMLR, 2019. 5

[RC20] Noam Razin and Nadav Cohen. Implicit regularization in deep learning may not be explainable by norms, 2020. 5

[RSR+19] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019. 24

[SHN+17] Daniel Soudry, Elad Hoffer, Mor Shpigel Nacson, Suriya Gunasekar, and Nathan Srebro. The implicit bias of gradient descent on separable data, 2017. 5

[SKL+21] Shiori Sagawa, Pang Wei Koh, Tony Lee, Irena Gao, Sang Michael Xie, Kendrick Shen, Ananya Kumar, Weihua Hu, Michihiro Yasunaga, Henrik Marklund, Sara Beery, Etienne David, Ian Stavness, Wei Guo, Jure Leskovec, Kate Saenko, Tatsunori Hashimoto, Sergey Levine, Chelsea Finn, and Percy Liang. Extending the wilds benchmark for unsupervised adaptation, 2021. 5

[SQZY17] Jian Shen, Yanru Qu, Weinan Zhang, and Yong Yu. Wasserstein distance guided representation learning for domain adaptation, 2017. 5

[THK+21] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in Neural Information Processing Systems*, 34, 2021. 7, 24

[Udo21] Aleksei Udovenko. Milp modeling of boolean functions by minimum number of inequalities. Cryptology ePrint Archive, Paper 2021/1099, 2021. https://eprint.iacr.org/2021/1099. 5

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 3, 7, 24

[VSS21] Gal Vardi, Ohad Shamir, and Nathan Srebro. On margin maximization in linear and relu networks, 2021. 5

[WGS+22] Olivia Wiles, Sven Gowal, Florian Stimberg, Sylvestre-Alvise Rebuffi, Ira Ktena, Krishnamurthy Dj Dvijotham, and Ali Taylan Cemgil. A fine-grained analysis on distribution shift. In *International Conference on Learning Representations*, 2022. 5

[XZL+19] Zhi-Qin John Xu, Yaoyu Zhang, Tao Luo, Yanyang Xiao, and Zheng Ma. Frequency principle: Fourier analysis sheds light on deep neural networks, 2019. 5

[XZX18] Zhi-Qin John Xu, Yaoyu Zhang, and Yanyang Xiao. Training behavior of deep neural network in frequency domain, 2018. 5

[ZRKB21] Chiyuan Zhang, Maithra Raghu, Jon M. Kleinberg, and Samy Bengio. Pointer value retrieval: A new benchmark for understanding the limits of neural network generalization. *ArXiv*, abs/2107.12580, 2021. 1, 2, 3, 6, 7, 10, 23

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] We believe there are no potential negative societal impacts.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Link to the GitHub repository is provided.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 3.2 and Appendix on experiment details.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] The 95% confidence interval is reported if relevant.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix on experiment details

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

   (a) If your work uses existing assets, did you cite the creators? [Yes]

   (b) Did you mention the license of the assets? [Yes]

   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]

   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Proof of Theorem 1

The proof of Theorem 1 goes by two steps. As a first step, we connect the noise stability to another measure of complexity for classes of functions called cross predictability (CP). As a second step, we use the negative result from [AS20], that lower bounds the generalization error of learning a class of functions in terms of its cross predictability.

A.1 From noise stability to cross predictability

We redefine here the cross predictability (CP), for completeness.

Definition 6 (Cross Predictability [AS20])**.** *Let $\mathcal{X}$ be the input space and let $\mathcal{F}$ be a class of functions. Let $P_{\mathcal{X}}$ and $P_{\mathcal{F}}$ be two distributions supported on $\mathcal{X}$ and $\mathcal{F}$ respectively. Their cross-predictability is defined as*

$$\mathrm{CP}(P_{\mathcal{F}}, P_{\mathcal{X}}) = \mathbb{E}_{F,F'\sim P_{\mathcal{F}}}[\mathbb{E}_{X\sim P_{\mathcal{X}}}[F(X)\cdot F'(X)]^2]. \tag{11}$$

Before diving into the proof we give few definitions that will be useful. Given a target function f, we define the "orbit" of f ($\mathrm{orb}(f)$) as the class of all functions generated by composing f with a permutation of the input space:

Definition 7 (Orbit)**.** *For $f : \mathbb{R}^n \to \mathbb{R}$ and a permutation $\pi \in S_n$, we let $(f\circ\pi)(x) = f(x_{\pi(1)}, ..., x_{\pi(n)})$. Then, the* orbit *of f is defined as*

$$\mathrm{orb}(f) := \{f\circ\pi : \pi \in S_n\}. \tag{12}$$

GD (or SGD) on a neural network with initialization that is target agnostic has equivalent behaviour when learning any target function in $\mathrm{orb}(f)$. We believe that one could extend the result to other invariances, beyond permutations.

Recall from Definition 5 that we introduced an augmented input space, to guarantee that the high-degree Fourier coefficients of the target function are sparse enough. Thus, let $\bar{f} : \{\pm1\}^{2n} \to \{\pm1\}$ be the $2n-$extension of f, defined as $\bar{f}(x_1, ..., x_n, x_{n+1}, ..., x_{2n}) = f(x_1, ..., x_n)$. For brevity we make use of the following notation:

$$\mathrm{CP}(\mathrm{orb}(\bar{f})) := \mathrm{CP}(\mathcal{U}_{\mathrm{orb}(\bar{f})}, \mathcal{U}_{\mathbb{F}_2^{2n}}), \tag{13}$$

where $\mathcal{U}_{\mathrm{orb}(\bar{f})}, \mathcal{U}_{\mathbb{F}_2^{2n}}$ denote the uniform distribution over $\mathrm{orb}(\bar{f})$ and over $\mathbb{F}_2^{2n}$ (i.e., the $2n$-dimensional Boolean hypercube), respectively.

Furthermore, recall that every Boolean function f can be written in terms of its Fourier-Walsh expansion $f(x) = \sum_S \hat{f}(S)\chi_S(x)$, where $\chi_S(x) = \prod_{i\in S} x_i$ are the standard Fourier basis elements and $\hat{f}(S)$ are the Fourier coefficients of f. We further denote by

$$W^k(f) = \sum_{S:|S|=k} \hat{f}(S)^2 \quad \text{and} \quad W^{\leq k}(f) = \sum_{S:|S|\leq k} \hat{f}(S)^2, \tag{14}$$

the total weight of the Fourier coefficients of f at degree k and up to degree k, respectively.
Let $\hat{f}$ be the Fourier coefficients of the original function f, and let $\hat{h}$ be the coefficients of the augmented function $\bar{f}$, that are:

$$\hat{h}(T) = \hat{f}(T) \qquad \text{if } T \subseteq [n] \tag{15}$$

$$\hat{h}(T) = 0 \qquad \text{otherwise.} \tag{16}$$

We make use of the following Lemma, that relates the cross-predictability of $\mathrm{orb}(\bar{f})$ to the Stability of f.

Lemma 2. *There exists δ such that for any $\delta' < \delta$*

$$\mathrm{CP}(\mathrm{orb}(\bar{f})) \leq \mathrm{Stab}_{\delta'}(f). \tag{17}$$

Remark 4 (Noise Stability). We remark the following two properties of $\mathrm{Stab}_\delta[f]$:

1. One can show (see e.g., Theorem 2.49 in [O'D14]) that

$$\mathrm{Stab}_\delta(f) = \sum_{k=1}^{n} (1-2\delta)^k W^k[f], \tag{18}$$

where $W^k[f]$ is the Boolean weight at degree k of f;

2. For all $\delta \in [0, 1/2]$, $\mathrm{Stab}_\delta(f) = \mathrm{Stab}_\delta(\bar{f})$. This follows directly from the previous point and (15)-(16).

Proof of Lemma 2. We denote by π a random permutation of $2n$ elements. We can bound the $\mathrm{CP}(\mathrm{orb}(\bar{f}))$ by the following:

$$\begin{aligned}
\mathrm{CP}(\mathrm{orb}(\bar{f})) &= \mathbb{E}_\pi \left[\mathbb{E}_X \left[\bar{f}(X)\bar{f}(\pi(X)) \right]^2 \right] && (19)\\
&= \mathbb{E}_\pi \left(\sum_{T \subseteq [2n]} \hat{h}(T)\hat{h}(\pi(T)) \right)^2 && (20)\\
&= \mathbb{E}_\pi \left(\sum_{T \subseteq [n]} \hat{f}(T)\hat{f}(\pi(T)) \cdot \mathbb{1}\left(\pi(T) \subseteq [n]\right) \right)^2 && (21)\\
&\overset{C.S}{\leq} \mathbb{E}_\pi \left(\sum_{S \subseteq [n]} \hat{f}(\pi(S))^2 \right) \cdot \left(\sum_{T \subseteq [n]} \hat{f}(T)^2 \mathbb{1}\left(\pi(T) \subseteq [n]\right) \right) && (22)\\
&= \sum_{T \subseteq [n]} \hat{f}(T)^2 \cdot \mathbb{P}_\pi\left(\pi(T) \subseteq [n]\right) && (23)\\
&= \sum_{k=1}^{n} W^k[f] \cdot \mathbb{P}_\pi\left(\pi(T) \subseteq [n] \mid |T| = k\right), && (24)
\end{aligned}$$

where (20) is the scalar product in the Fourier basis, (21) follows by applying the formulas of the $\hat{h}$ given in (15)-(16), (22) holds by Cauchy-Schwarz inequality, (23) holds since f is Boolean-valued and for each π by Parseval identity, $\sum_{S \subseteq [n]} \hat{f}(\pi(S))^2 = \mathbb{E}_X[f(X)^2] = 1$, and (24) holds since the second term is invariant for all sets of a given cardinality.
Recalling π is a random permutation over the augmented input space of dimension $2n$, for each $k \in [n]$ we can further bound the second term by

$$\mathbb{P}_\pi\left(\pi(T) \subseteq [n] \mid |T| = k\right) = \frac{\binom{n}{k}}{\binom{2n}{k}} \sim \frac{1}{2^k} \leq (1-2\delta')^k, \quad \text{for all } \delta' \leq 1/4. \tag{25}$$

Thus, for all $\delta' \leq 1/4$,

$$\mathrm{CP}(\mathrm{orb}(\bar{f})) \leq \sum_{k=1}^{n} (1-2\delta')^k W^k[f] = \mathrm{Stab}_{\delta'}[f]. \tag{26}$$

□

Remark 5. Note that the value of δ in Lemma 2 depends on the size of the input extension that we use. In this paper, we defined an input extension of size $2n$ (input doubling), which gives $\delta = 1/4$, however we could have chosen e.g. a $3n$-extension and obtain $\delta = 1/3$, and so on.

A.2 From cross predictability to hardness of learning

For the second step, we make use of Theorem 3 and Corollary 1 in [AS20], that prove a lower bound of learning a class of function in terms of its cross predictability. The lower bound holds for the noisy GD algorithm ([AS20, AKM+21]), of which we give a formal definition here.

Definition 8 (Noisy GD with batches)**.** *Consider a neural network of size E, with a differentiable non-linearity and initialization of the weights $W^{(0)}$. Given a differentiable loss function, the updates of the noisy GD algorithm with learning rate γ_t and gradient precision A are defined by*

$$W^{(t)} = W^{(t-1)} - \gamma_t \mathbb{E}_{X\sim S^{(t)}}[\nabla L(f(X), f_{NN}^{(t)})]_A + Z^{(t)}, \qquad t = 1, ..., T, \tag{27}$$

where for all t, $Z^{(t)}$ are i.i.d. $\mathcal{N}(0, \sigma^2)$, for some σ, and they are independent from other variables, $S^{(t)} = (X_1^{(t)}, ..., X_m^{(t)})$ has independent components drawn from the input distribution $P_{\mathcal{X}}$ and independent from other time steps, and f is the target function, from which the labels are generated, and by $[.]_A$ we mean that whenever the argument is exceeding A (resp. $-A$) it is rounded to A (resp. $-A$).

Theorem 3 and Corollary 1 in [AS20] imply that for any distribution over the Boolean hypercube $P_{\mathcal{X}}$ and Boolean functions $P_{\mathcal{F}}$, it holds that

$$\mathbb{P}_{X,F\sim P_{\mathcal{F}}, f_{NN}^{(T)}}(F(X) \neq f_{NN}^{(T)}(X)) \geq 1/2 - \frac{\gamma T\sqrt{E}A}{\sigma}\left(1/m + \mathrm{CP}(P_{\mathcal{F}}, P_{\mathcal{X}})\right)^{1/4}, \tag{28}$$

where γ, E, A, σ, m have the same meaning as in Definition 8. As observed by them, in our case since the initialization is invariant under permutations of the input, then learning the orbit of $\bar{f}$ under uniform distribution is equivalent to learning $\bar{f}$, thus the following bound holds:

$$\mathbb{P}_{X, f_{NN}^{(T)}}(\bar{f}(X) \neq f_{NN}^{(T)}(X)) \geq 1/2 - \frac{\gamma T\sqrt{E}A}{\sigma}\left(1/m + \mathrm{CP}(\mathrm{orb}(\bar{f}))\right)^{1/4}. \tag{29}$$

B Removing the input doubling

One can prove a similar result to the one of Theorem 1, without using the input extension technique. However, we need some additional assumptions on f. To introduce them, let us first fix some notation. In the following, we say that a sequence a_n is *noticeable* if there exists $c \in \mathbb{N}$ such that $a_n = \Omega(n^{-c})$. On the other hand, we say that f is *negligible* if $\lim_{n\to\infty} n^c a_n = 0$ for every $c \in \mathbb{N}$ (which we also write $a_n = n^{-\omega(1)}$).

Assumption 1 (Non-dense and non-extremal function)**.**

a) We say that f is "non-dense" if there exists c such that $W\{T : \hat{f}(T)^2 \leq n^{-c}\} = n^{-\omega(1)}$, i.e., the negligible Fourier coefficients do not bring a noticeable contribution if taken all together;

b) We say f is "non-extremal" if for any positive constant D, $W^{\geq n-D}[f] = n^{-\omega(1)}$, i.e., f does not have noticeable Fourier weight on terms of degree $n - O(1)$.

With such additional assumptions, we can conclude the following.

Proposition 1. *Let $f : \{\pm 1\}^n \to \{\pm 1\}$ be a balanced target function, let $\mathrm{Stab}_\delta(f)$ be its noise stability and let $f_{NN}^{(t)}$ be the output of GD with gradient precision A after t time steps, trained on a neural network of size E with initialization that is target agnostic. Assume f satisfies Assumption 1. Then, there exist $c, C > 0$ and $D > 0$ such that if $\delta < D/n$*

$$\mathrm{gen}(f, f_{NN}^{(t)}) \geq 1/2 - C \cdot t \cdot \sqrt{E} \cdot \left(n^c \cdot \mathrm{Stab}_\delta(f) + n^{-\omega(1)}\right)^{1/4}. \tag{30}$$

The proof of Proposition 1 resembles the proof of Theorem 1. The only modification required is in Lemma 2, which is replaced by the following Lemma.

Lemma 3. *Let f be a Boolean function that satisfies Assumption 1. There exists $c, D > 0$ such that for $\delta < D/n$,*

$$\mathrm{CP}(\mathrm{orb}(f)) \leq 2 \cdot n^c \cdot \mathrm{Stab}_\delta(f) + n^{-\omega(1)}. \tag{31}$$

Proof of Lemma 3. Let $c > 0$ be such that $W\{T : \hat{f}(T)^2 \leq n^{-c}\} = n^{-\omega(1)}$. This c exists because of Assumption 1a.

$$
\begin{aligned}
&\mathrm{CP}(\mathrm{orb}(f)) = && (32)\\
&\quad = \mathbb{E}_\pi \left[\mathbb{E}_X \left[f(X) f(\pi(X)) \right]^2 \right] && (33)\\
&\quad = \mathbb{E}_\pi \left(\sum_{T \subseteq [n]} \hat{f}(T) \hat{f}(\pi(T)) \right)^2 && (34)\\
&\quad = \mathbb{E}_\pi \left(\sum_{T \subseteq [n]} \hat{f}(T) \hat{f}(\pi(T)) \cdot \left(\mathbb{1}\left(\hat{f}(\pi(T))^2 \leq n^{-c} \right) + \mathbb{1}\left(\hat{f}(\pi(T))^2 > n^{-c} \right) \right) \right)^2 && (35)\\
&\quad \leq 2\mathbb{E}_\pi \left(\sum_{T \subseteq [n]} \hat{f}(T) \hat{f}(\pi(T)) \mathbb{1}\left(\hat{f}(\pi(T))^2 \leq n^{-c} \right) \right)^2 + && (36)\\
&\qquad\qquad + 2\mathbb{E}_\pi \left(\sum_{T \subseteq [n]} \hat{f}(T) \hat{f}(\pi(T)) \mathbb{1}\left(\hat{f}(\pi(T))^2 > n^{-c} \right) \right)^2, && (37)
\end{aligned}
$$

where in the last inequality we used $(a+b)^2 \leq 2(a^2 + b^2)$. Let us first focus on the second term on the right.

$$
\begin{aligned}
&\mathbb{E}_\pi \left(\sum_{T \subseteq [n]} \hat{f}(T) \hat{f}(\pi(T)) \mathbb{1}\left(\hat{f}(\pi(T))^2 > n^{-c} \right) \right)^2 && (38)\\
&\quad \overset{C.S}{\leq} \mathbb{E}_\pi \left(\sum_{S \subseteq [n]} \hat{f}(\pi(S))^2 \right) \cdot \left(\sum_{T \subseteq [n]} \hat{f}(T)^2 \mathbb{1}\left(\hat{f}(\pi(T))^2 > n^{-c} \right) \right) && (39)\\
&\quad \leq \sum_{T \subseteq [n]} \hat{f}(T)^2 \cdot \mathbb{P}_\pi \left(\hat{f}(\pi(T))^2 > n^{-c} \right) && (40)\\
&\quad = \sum_{k=1}^{n} W^k[f] \cdot \mathbb{P}_\pi \left(\hat{f}(\pi(T))^2 > n^{-c} \mid |T| = k \right) && (41)\\
&\quad = \sum_{k=1}^{n-D} W^k[f] \cdot \mathbb{P}_\pi \left(\hat{f}(\pi(T))^2 > n^{-c} \mid |T| = k \right) + && (42)\\
&\qquad\qquad + \sum_{k=n-D+1}^{n} W^k[f] \cdot \mathbb{P}_\pi \left(\hat{f}(\pi(T))^2 > n^{-c} \mid |T| = k \right)\\
&\quad \leq \sum_{k=1}^{n-D} W^k[f] \cdot \mathbb{P}_\pi \left(\hat{f}(\pi(T))^2 > n^{-c} \mid |T| = k \right) + W^{\geq n-D+1}[f]. && (43)
\end{aligned}
$$

where D is an arbitrary positive constant. Because of Assumption 1b, $W^{\geq n-D+1}[f] = n^{-\omega(1)}$. On the other hand, since f is a Boolean valued function,

$$
\sum_{T} \hat{f}(T)^2 = \mathbb{E}_X[f(X)^2] = 1, \tag{44}
$$

which implies that there are at most n^c sets T such that $\hat{f}(T)^2 > n^{-c}$. Thus, recalling π is a random permutation over the input space of dimension n, we get

$$\mathbb{P}_\pi\left(\hat{f}(\pi(T))^2 > n^{-c} \mid |T| = k\right) \leq \frac{n^c}{\binom{n}{k}} \tag{45}$$

$$\leq n^c \left(\frac{k}{n}\right)^k \tag{46}$$

$$\leq n^c \left(\frac{n-D}{n}\right)^k \tag{47}$$

$$\leq n^c \left(1-2\delta\right)^k \qquad \text{if } \delta \leq \frac{D}{2n}, \tag{48}$$

where in (46) we used that $\binom{n}{k} \geq (\frac{n}{k})^k$ for all $k \geq 1$. Going back to the first term in (37) we get

$$\mathbb{E}_\pi\left(\sum_{T\subseteq[n]} \hat{f}(T)\hat{f}(\pi(T))\mathbb{1}\left(\hat{f}(\pi(T))^2 \leq n^{-c}\right)\right)^2 \tag{49}$$

$$\overset{C.S}{\leq} \mathbb{E}_\pi\left(\sum_{S\subseteq[n]} \hat{f}(S)^2\right) \cdot \left(\sum_{T\subseteq[n]} \hat{f}(\pi(T))^2\mathbb{1}\left(\hat{f}(\pi(T))^2 > n^{-c}\right)\right) \tag{50}$$

$$\leq \sum_{T\subseteq[n]} \hat{f}(\pi(T))^2\mathbb{1}\left(\hat{f}(\pi(T))^2 > n^{-c}\right) \tag{51}$$

$$= n^{-\omega(1)}, \tag{52}$$

by Assumption 1a. Hence overall,

$$\mathrm{CP}(\mathrm{orb}(f)) \leq 2n^c \sum_{k=1}^{n-D} W^k[f](1-2\delta)^k + n^{-\omega(1)} \tag{53}$$

$$\leq 2n^c\,\mathrm{Stab}_\delta(f) + n^{-\omega(1)}. \tag{54}$$

□

C Proof for Lemma 1 and Theorem 2

In this section, we present proofs for results mentioned in Section 3, namely, Lemma 1 and Theorem 2.

C.1 Proof of Lemma 1

Proof of Lemma 1. Let $f(x) = \sum_{T\subseteq[n]} \hat{f}(T)\chi_T(x)$ be the Fourier expansion of the function where $\chi_T(x) = \prod_{i\in T} x_i$. Therefore, the Fourier expansion of the frozen function will become

$$f_{-k}(x) = \sum_{T\subseteq[n]\setminus k} (\hat{f}(T) + \hat{f}(T\cup k))\chi_T(x). \tag{55}$$

Thus, the difference between functions is equal to

$$(f - f_{-k})(x) = \sum_{T\subseteq[n]:k\in T} \hat{f}(T)\chi_T(x) - \sum_{T\subseteq[n]\setminus k} \hat{f}(T\cup k)\chi_T(x). \tag{56}$$

Hence, using Parseval's Theorem we have the following:

$$\mathbb{E}_{U^n}(f - f_{-k})_2^2 = \sum_{T\subseteq[n]:k\in T} \hat{f}(T)^2 + \sum_{T\subseteq[n]\setminus k} \hat{f}(T\cup k)^2 = 2\sum_{T\subseteq[n]:k\in T} \hat{f}(T)^2. \tag{57}$$

Therefore,

$$\mathbb{E}_{U^n}\frac{1}{2}(f - f_{-k})_2^2 = \sum_{T\subseteq[n]:k\in T} \hat{f}(T)^2 = \mathrm{Inf}_k(f), \tag{58}$$

and the lemma is proved. □

C.2 Proof of Theorem 2

Proof of Theorem 2. Assume $\tilde{f}^{(t)}_{-k}(x, \Theta^{(t)}) := x^T W^{(t)} + b^{(t)}$ to be our linear model where $\Theta^{(t)} = (W^{(t)}, b^{(t)})$ are the model parameters at time t. In the following, the super-script t and T denote the time-step and transpose respectively. Also, we use $\mathbb{E}_{x_{-k}}$ to denote the expectation of x taken uniformly on the Boolean hypercube while $x_k = 1$. Using the square loss, we have

$$L(\Theta^{(t)}, x, f) = (x^T W^{(t)} + b^{(t)} - f(x))^2, \tag{59}$$

and the gradients will be

$$\nabla_W L(\Theta^{(t)}, x, f) = 2x \left(x^T W^{(t)} + b^{(t)} - f(x) \right), \tag{60}$$

$$\partial_b L(\Theta^{(t)}, x, f) = 2 \left(x^T W^{(t)} + b^{(t)} - f(x) \right). \tag{61}$$

The GD update rule will then become

$$W^{(t+1)} = W^{(t)} - 2\gamma \left(\mathbb{E}_{x_{-k}} \left[x x^T \right] W^{(t)} + \mathbb{E}_{x_{-k}}[x] b^{(t)} - \mathbb{E}_{x_{-k}}[x f(x)] \right), \tag{62}$$

$$b^{(t+1)} = b^{(t)} - 2\gamma \left(\mathbb{E}_{x_{-k}}[x^T] W^{(t)} + b^{(t)} - \mathbb{E}_{x_{-k}}[f(x)] \right). \tag{63}$$

Note that $\mathbb{E}_{x_{-k}} \left[x x^T \right] = \mathbb{I}_n$, $\mathbb{E}_{x_{-k}}[x] = \vec{e}_k$. So we have

$$\forall j \neq k: \; W^{(t+1)}_j = W^{(t)}_j (1 - 2\gamma) + 2\gamma \mathbb{E}_{x_{-k}}[x_j \cdot f(x)], \tag{64}$$

$$W^{(t+1)}_k = W^{(t)}_k - 2\gamma (W^{(t)}_k + b^{(t)}) + 2\gamma \mathbb{E}_{x_{-k}}[f(x)], \tag{65}$$

$$b^{(t+1)} = b^{(t)} - 2\gamma (W^{(t)}_k + b^{(t)}) + 2\gamma \mathbb{E}_{x_{-k}}[f(x)]. \tag{66}$$

Using above equations, we have

$$W^{(t+1)}_k - b^{(t+1)} = W^{(t)}_k - b^{(t)} = W^{(0)}_k - b^{(0)}, \tag{67}$$

$$W^{(t+1)}_k + b^{(t+1)} = (1 - 4\gamma)(W^{(t)}_k + b^{(t)}) + 4\gamma \mathbb{E}_{x_{-k}}[f(x)]. \tag{68}$$

Assume $\gamma < \frac{1}{4}$ and define $0 < c = -\log(1 - 2\gamma) < -\log(1 - 4\gamma)$, then we have

$$\begin{aligned} W^{(t)}_k + b^{(t)} &= (1 - 4\gamma)^t (W^{(0)}_k + b^{(0)} - \mathbb{E}_{x_{-k}}[f(x)]) + \mathbb{E}_{x_{-k}}[f(x)] \\ &= O((1 - 4\gamma)^t) + \mathbb{E}_{x_{-k}}[f(x)] = O(e^{-ct}) + \mathbb{E}_{x_{-k}}[f(x)] \\ &= O(e^{-ct}) + \hat{f}(\emptyset) + \hat{f}(\{k\}), \end{aligned} \tag{69}$$

$$\begin{aligned} \forall j \neq k: \; W^{(t)}_j &= (1 - 2\gamma)^t (W^{(0)}_j - \mathbb{E}_{x_{-k}}[x_j \cdot f(x)]) + \mathbb{E}_{x_{-k}}[x_j \cdot f(x)] \\ &= O((1 - 2\gamma)^t) + \mathbb{E}_{x_{-k}}[x_j \cdot f(x)] = O(e^{-ct}) + \mathbb{E}_{x_{-k}}[x_j \cdot f(x)] \\ &= O(e^{-ct}) + \hat{f}(\{j\}). \end{aligned} \tag{70}$$

So the learned function is

$$\begin{aligned} \tilde{f}_{-k}(x; \Theta^{(t)}) = \frac{b^{(0)} - W^{(0)}_k + \hat{f}(\emptyset) + \hat{f}(\{k\})}{2} &+ \frac{W^{(0)}_k - b^{(0)} + \hat{f}(\emptyset) + \hat{f}(\{k\})}{2} x_k \\ &+ \sum_{j \neq k} \hat{f}(\{j\}) \cdot x_j + O(e^{-ct}) \end{aligned} \tag{71}$$

and the generalization error can be computed using Parseval Theorem:

$$\mathrm{gen}(f, \tilde{f}^{(t)}_{-k}) = \frac{1}{2} \mathbb{E}_{x \sim U^n} \left[\left(f(x) - \tilde{f}^{(t)}_{-k}(x; \Theta^{\infty}) \right)^2 \right] \tag{72}$$

$$= \frac{1}{2} \left(\frac{(b^{(0)} - W^{(0)}_k - \hat{f}(\emptyset) + \hat{f}(\{k\}))^2 + (W^{(0)}_k - b^{(0)} + \hat{f}(\emptyset) - \hat{f}(\{k\}))^2}{4} \right) + O(e^{-ct}) \tag{73}$$

$$= \frac{(b^{(0)} - W^{(0)}_k - \hat{f}(\emptyset) + \hat{f}(\{k\}))^2}{4} + O(e^{-ct}) \tag{74}$$

$$= \frac{(b^{(0)} - W^{(0)}_k)^2}{4} + \frac{(\hat{f}(\emptyset) - \hat{f}(\{k\}))^2}{4} - 2 \frac{(b^{(0)} - W^{(0)}_k)(\hat{f}(\emptyset) - \hat{f}(\{k\})}{4} + O(e^{-ct}). \tag{75}$$

Therefore, the expected generalization loss over different initializations is given by

$$\mathbb{E}_{\Theta^0}[\mathrm{gen}(f, \tilde{f}^{(t)}_{-k})] = \mathbb{E}_{\Theta^0}\left[\frac{(b^{(0)} - W_k^{(0)})^2 + (\hat{f}(\emptyset) - \hat{f}(\{k\}))^2}{4}\right] + O(e^{-ct}) \tag{76}$$

$$= \frac{(\hat{f}(\emptyset) - \hat{f}(\{k\}))^2}{4} + \frac{\sigma^2}{2} + O(e^{-ct}). \tag{77}$$

Particularly, if the frozen function is unbiased, i.e., $\hat{f}(\emptyset) + \hat{f}(\{k\}) = 0$, we have

$$\mathbb{E}_{\Theta^0}[\mathrm{gen}(f, \tilde{f}^{(t)}_{-k})] = \frac{(2\hat{f}(\{k\}))^2}{4} + \frac{\sigma^2}{2} + O(e^{-ct})$$

$$= \hat{f}(\{k\})^2 + \frac{\sigma^2}{2} + O(e^{-ct}) = \mathrm{Inf}_k(f) + \frac{\sigma^2}{2} + O(e^{-ct}). \tag{78}$$

□

D Further details on noise stability

D.1 Noise stability of PVR functions

As mentioned above, a PVR function consists of a pointer (the first bits of the input) and an aggregation function that acts on a specific window indicated by the pointer. We denote by p the number of bits that define the pointer, and by w the size of each window. For simplicity, we consider a slight variation of Boolean PVR task with non-overlapping windows, defined as follows:

- PVR with *non-overlapping* windows: the 2^p windows pointed by the pointer bits are non-overlapping, i.e., the first window is formed by bits $x_{p+1}, ..., x_{p+w}$, the second window is formed by bits $x_{p+w+1}, ..., x_{p+2w}$, and so forth.

The input size is thus given by $n := p + 2^p w$ and $p = O(\log(n))$. We denote by $g : \{\pm 1\}^w \to \{\pm 1\}$ the aggregation function, which we assume to be balanced (i.e., $\mathbb{E}_X[g(X)] = 0$). One can verify (see details below) that the noise stability of the PVR function f is given by

$$\mathrm{Stab}_\delta[f] = (1-\delta)^{p+w} + (1-\delta)^p(1-(1-\delta)^w) \cdot \mathrm{Stab}_\delta[g]. \tag{79}$$

We notice that the $\mathrm{Stab}_\delta[f]$ is given by two terms: the first one depends on the window size and the second one on the stability of the aggregation function. For large enough window size, the second term in (79) is the dominant one, and $\mathrm{Stab}_\delta[f]$ depends on the stability of g. Thus from Theorem 1, f is not learned by GD (in the extended input space) in poly(n) time if the stability of the aggregation function is $n^{-\omega(1)}$. On the other hand, for small window size (specifically for $w = O(\log(n))$), the $\mathrm{Stab}_\delta(f)$ is 'noticeable' (as defined in Appendix B) for every aggregation function, since the function value itself depends on a limited number of input bits. Thus, noise unstable aggregation functions (e.g. parities) can form a PVR function with 'noticeable' stability, if the window size is $O(\log(n))$. As examples, we consider the specific cases of pairty and majority vote as aggregation functions.

- Parity: If we choose $g(x_1, ..., x_w) = \prod_{i=1}^w x_i$, one can observe that $\mathrm{Stab}_\delta(g) = (1-2\delta)^w$. Then, eq. (79) becomes $\mathrm{Stab}_\delta(f) = (1-\delta)^{w+p}[1-(1-2\delta)^w] + (1-\delta)^p$, and $\mathrm{Stab}_\delta(f)$ is decreasing with w.
- Majority: If we choose g to be $g(x_1, ..., x_w) = \mathrm{sgn}(\sum_{i=1}^w x_i)$, then, for w large, $\mathrm{Stab}_\delta(g) \sim 1 - 2/\pi \cdot \arccos(1-2\delta)$ (see e.g. [O'D14]). Plugging this in eq. (79), one can observe that also for majority vote $\mathrm{Stab}_\delta(f)$ is decreasing with w.

Computation of (79). We compute the expression in (79) with the following:

$$\mathrm{Stab}_\delta[f] = 1 - 2\,\mathrm{NS}_\delta[f], \tag{80}$$

where $\mathrm{NS}_\delta[f] := \mathbb{P}(f(X) \neq f(Y))$ is the Noise sensitivity of f, defined as the probability that perturbing each input bit independently with probability δ changes the output of f and where we denoted by Y the vector obtained from X by flipping each component with prob. δ independently. To

compute $\mathrm{NS}_\delta[f]$, we can first distinguish depending on whether the perturbation affects the pointer bit:

$$\begin{aligned}\mathrm{NS}_\delta[f] :&= \mathbb{P}(f(X) \neq f(Y)) \\ &= (1-\delta)^p \cdot \mathbb{P}(f(X) \neq f(Y) \mid X^p = Y^p) + (1-(1-\delta)^p)\mathbb{P}(f(X) \neq f(Y) \mid X^p \neq Y^p) \\ &= (1-\delta)^p \cdot \mathbb{P}(f(X) \neq f(Y) \mid X^p = Y^p) + (1-(1-\delta)^p)\frac{1}{2},\end{aligned}$$

where the last inequality holds since we are using non-overlapping windows and we assumed g to be balanced. To compute the first term, we can condition on whether any bit in the window pointed by X and Y is changed:

$$\begin{aligned}&\mathbb{P}(f(X) \neq f(Y) \mid X^p = Y^p) \\ &\quad= (1-\delta)^w \cdot \mathbb{P}(f(X) \neq f(Y) \mid X^p = Y^p, X_{P(X^p)} = Y_{P(Y^p)}) + \\ &\qquad + (1-(1-\delta)^w) \cdot \mathbb{P}(f(X) \neq f(Y) \mid X^p = Y^p, X_{P(X^p)} \neq Y_{P(Y^p)}) \\ &\quad= (1-(1-\delta)^w) \cdot \mathbb{P}(f(X) \neq f(Y) \mid X^p = Y^p, X_{P(X^p)} \neq Y_{P(Y^p)}) \\ &\quad= (1-(1-\delta)^w) \cdot \mathrm{NS}_\delta[g],\end{aligned}$$

where the last inequality holds because g is unbalanced. By replacing $\mathrm{NS}_\delta[g] = \frac{1}{2} - \frac{1}{2}\mathrm{Stab}_\delta[g]$ and rearranging terms one can obtain (79).

D.2 Noise stability and initial alignment [ACHM22]

[ACHM22] introduced the notion of Initial Alignment (INAL) between a target function $f : \mathcal{X} \to \mathcal{Y}$ and a neural network $\mathrm{NN} : \mathcal{X} \to \mathcal{Y}$ with random initialization Θ_0 and neuron set V_{NN}. The INAL is defined as

$$\mathrm{INAL}(f, \mathrm{NN}) := \max_{v \in V_{\mathrm{NN}}} \mathbb{E}_{\Theta^0}\mathbb{E}_X \left[f(X) \cdot \mathrm{NN}^{(v)}_{\Theta^0}(X)\right]^2, \tag{81}$$

where $\mathrm{NN}^{(v)}_{\Theta^0}$ denotes the output of neuron v of the network at initialization. In [ACHM22], it is shown that GD cannot learn functions that have negligible initial alignment with a fully connected architectures with i.i.d. Gaussian initialization (with rescaled variance) and ReLU activation. Here, we show how the INAL can be related to the noise sensitivity of the target function. We remark that both noise sensitivity and INAL are related by the cross-predictability (CP). Let us first give two definitions. Recall that for $f : \{\pm 1\}^n \to \{\pm 1\}$, $\mathrm{NS}_\delta[f] = \frac{1}{2} - \frac{1}{2}\mathrm{Stab}_\delta[f]$.

Definition 9 (High-Degree.)**.** *We say that a family of functions $f_n : \{\pm 1\}^n \to \mathbb{R}$ is "high-degree" if for any fixed k, $W^{\leq k}(f_n)$ is negligible.*

Definition 10 (Noise sensitive function)**.** *We say that a family of functions $f_n : \{\pm 1\}^n \to \{\pm 1\}$ is noise sensitive if for any $\delta \in (0, 1/2]$, $\mathrm{NS}_\delta[f_n] = 1/2 - o_n(1)$.*

Definition 11 (Strongly noise sensitive function)**.** *We say that a family on functions $f_n : \{\pm 1\}^n \to \{\pm 1\}$ is strongly noise sensitive if for any $\delta \in (0, 1/2]$, $\mathrm{NS}_\delta[f_n] = 1/2 - n^{-\omega(1)}$.*

Then we can prove the following.

Proposition 2. *Let $\mathrm{NN}_n : \mathbb{R}^n \to \mathbb{R}$ be a fully connected neural network with Gaussian i.i.d. initialization and expressive activation (as in Theorem 2.7 in [ACHM22]). If $\mathrm{INAL}(\mathrm{NN}_n, f_n) = n^{-\omega(1)}$, then f_n is noise sensitive.*

Proof. We need to show that for any $\delta \in [0, 1/2]$, $\sum_{k=0}^n (1-2\delta)^k W^k(f_n) = o_n(1)$, or analogously that for any $\epsilon > 0$ and for n large enough $\sum_{k=0}^n (1-2\delta)^k W^k(f_n) < \epsilon$. Fix δ and let $\epsilon > 0$. Let k_0 be such that $(1-2\delta)^{k_0} < \epsilon/2$. Then,

$$\sum_{k=0}^n (1-2\delta)^k W^k(f_n) = \sum_{k=0}^{k_0} (1-2\delta)^k W^k(f_n) + \sum_{k=k_0+1}^n (1-2\delta)^k W^k(f_n) \tag{82}$$

$$\leq W^{\leq k_0}(f_n) + (1-2\delta)^{k_0+1} \sum_{k=k_0+1}^n W^k(f_n). \tag{83}$$

By Proposition 4.3 and Corollary 4.4 in [ACHM22], if $\mathrm{INAL}(f_n, \sigma) = n^{-\omega(1)}$ then f_n is high degree. Thus, $W^{\leq k_0}(f_n) = n^{-\omega(1)}$, and clearly for n large enough $W^{\leq k_0}(f_n) < \epsilon/2$. On the other hand, $\sum_{k=k_0+1}^{n} W^k(f_n) < 1$, since f is Boolean-valued. Thus, $\sum_{k=0}^{n}(1-2\delta)^k W^k(f_n) < \epsilon$, and the Proposition is proven. □

Proposition 3. *If f_n is strongly noise sensitive, then f_n is high degree.*

Proof. We need to show that if $\sum_{k=0}^{n}(1-2\delta)^k W^k(f_n) = n^{-\omega(1)}$ then for any constant k, $W^{\leq k}(f_n) = n^{-\omega(1)}$. Take $k_0 \in \mathbb{N}$, then

$$n^{-\omega(1)} = \sum_{k=0}^{n}(1-2\delta)^k W^k(f_n) \geq \sum_{k=0}^{k_0}(1-2\delta)^k W^k(f_n) \geq (1-2\delta)^{k_0} W^{\leq k_0}(f_n). \tag{84}$$

Clearly this implies that $W^{\leq k_0}(f_n) = n^{-\omega(1)}$, and the proof is concluded. □

E Computation of the Boolean influence for PVR functions

In this section, we compute the Boolean influence for PVR functions. Here, we consider PVR functions with sliding windows and cyclic indices (i.e., $x_{n+1} = x_{p+1}$). The Boolean influence for PVR tasks with truncated windows or non-overlapping windows can be calculated in a similar manner. Also note that we never freeze pointer bits in this paper as done in [ZRKB21]; therefore, we skip the calculation of the Boolean influence for pointer bits. Consider a bit at k-th position ($k > p$). Note that this bit appears in w windows. We denote by U^n the uniform distribution over the n-dimensional hypercube. Using Lemma 1, we have:

$$\begin{aligned}
\mathrm{Inf}_k(f) &= \mathbb{E}_{x\sim U^n}\frac{1}{2}\left(f(x) - f_{-k}(x)\right)^2 && (85)\\
&= \mathbb{E}_{x\sim U^n}\frac{1}{2}\Bigg(\sum_{i=0}^{w-1}\mathbb{1}(P(x^p) = k-i)\Big(g\left(x_{k-i},\ldots,x_k,\ldots,x_{k-i+w-1}\right) && (86)\\
&\qquad - g\left(x_{k-i},\ldots,1,\ldots,x_{k-i+w-1}\right)\Big)\Bigg)^2\\
&= \mathbb{E}_{x\sim U^n}\frac{1}{2}\Bigg(\sum_{i=0}^{w-1}\mathbb{1}(P(x^p) = k-i)\Big(g\left(x_{k-i},\ldots,x_k,\ldots,x_{k-i+w-1}\right) && (87)\\
&\qquad - g\left(x_{k-i},\ldots,1,\ldots,x_{k-i+w-1}\right)\Big)^2\Bigg)\\
&= \frac{1}{2^p}\sum_{i=1}^{w}\mathrm{Inf}_i(g). && (88)
\end{aligned}$$

Note that the expression $\sum_{i=1}^{w}\mathrm{Inf}_i(g)$ in Equation (88) is known as the *total influence* of the aggregation function g [O'D14]. Below follows the value of the Boolean influence of the PVR task f, depending on different aggregation functions:

- **Parity.** If we choose g to be the parity function, i.e., $g(x_1,\ldots,x_w) = x_1x_2\cdots x_w$ then $\mathrm{Inf}_i(g) = \mathbb{P}(g(x) \neq g(x+e_i)) = 1$. Therefore, $\mathrm{Inf_k}(f) = \frac{w}{2^p}$.
- **Median/Majority vote.** We define the majority vote function as $g(x_1,\ldots,x_w) = \mathrm{sign}(x_1+\cdots+x_w)$ where the sign function outputs $+1$, -1, and 0. First assume w is odd. In this case, flipping the i-th bit matters only in the case where exactly $\frac{w-1}{2}$ other bits have the same sign as the i-th bit. Therefore, $\mathrm{Inf}_i(g) = \mathbb{P}(g(x) \neq g(x+e_i)) = 2^{-(w-1)}\binom{w-1}{\frac{w-1}{2}}$. Similarly, if w is even, flipping the i-th bit only matters if there are exactly $\frac{w}{2}$ or $\frac{w}{2}-1$ other bits with the same sign. Using Lemma 1, $\mathrm{Inf}_i(g) = \mathbb{E}_{x\sim U^w}\frac{1}{2}\left(g(x) - g_{-i}(x)\right)^2 = 2^{-(w+1)}\left(\binom{w-1}{\frac{w}{2}} + \binom{w-1}{\frac{w}{2}-1}\right) = 2^{-w}\binom{w-1}{\frac{w}{2}}$. Therefore, for odd w, $\mathrm{Inf_k}(f) = \frac{w}{2^{(p+w-1)}}\binom{w-1}{\frac{w-1}{2}}$ and for even w, $\mathrm{Inf_k}(f) = \frac{w}{2^{(w+p)}}\binom{w-1}{\frac{w}{2}}$.

- **Min/Max.** Here we consider the $\min$ function, $g(x_1, \ldots, x_w) = \min(x_1, \ldots, x_w)$. By symmetry, the Boolean influence values are the same for the $\max$ function. In this case, flipping the i-th bit only matters if all bits other than x_i are equal to $+1$. Thus, the Boolean influence is given by $\mathrm{Inf}_i(g) = 2^{-(w-1)}$ and hence, $\mathrm{Inf}_{\mathrm{k}}(f) = \frac{w}{2^{(p+w-1)}}$.

One can see how different parameters of the Boolean PVR functions, such as p, w, and g affect the Boolean influence. Assuming fixed window size, w, each bit is less likely to appear in a window if the number of pointer bits, p, is increased. Hence for fixed w and g, an increase in p results in smaller influence for all the bits. On the other hand a change of w has a two-fold effect. First, since each bit appears in w windows, the increase of w makes each bit more likely to appear in a window. On the other hand, for some functions such as majority-vote and min/max, the increase of w reduces the Boolean influence of the aggregation function for all bits. Thus, the increase of w can result in either an increase of the Boolean influence (for example, if parity is used) or a decrease of the Boolean influence (for instance, if min/max aggregation is used). We refer to Appendix F for experiments on PVR tasks with varying window size.

F Experiment Details and Additional Experiments

In this section, we describe the experiments in more detail. Furthermore, we demonstrate more experiments on the comparison of the out-of-distribution generalization error and the Boolean influence.

F.1 Architectures and Procedure

We first explain the general experimentation setup for PVR tasks and other functions. Afterward, we describe the procedure used for linear neural networks and results presented in Section 3.3.

Architectures. Three architectures have been used for the main experiments of this paper: MLP, the Transformer [VSP+17], and MLP-Mixer [THK+21]. Below, we describe each of these architectures:

- **MLP.** The MLP model consists of 4 fully connected hidden layers of sizes 512, 1024, 512, and 64. We used ReLU as the activation function for all layers except the last layer.
- **Transformer.** We follow the standard decoder-only Transformer architectures [RSR+19] that are commonly used for language modeling, and are also the backbone of Vision Transformers (ViTs) [DBK+20]. Specifically, an embedding layer is used to embed the binary $+1$ and -1 values into 256 dimensional vectors, and a shared embedding layer is used for all the binary tokens in the input sequence. Then, the embedded input is passed through 12 transformer layers [VSP+17]. In each transformer layer, the hidden dimension of MLP block is also 256. Moreover, 6 heads are used for each self-attention block. At the end, a linear layer is used to compute the output of the model.
- **MLP-Mixer.** Similar to the Transformer based model, first we embed $+1$ and -1 tokens into a 256 dimensional vector using a shared embedding layer for all the binary input tokens. Then, the embedded input is passed through a standard 12-layer MLP-Mixer model [THK+21]. Finally, a linear layer is used to compute the output. The MLP-Mixer architectures are similar to the decoder-only Transformers, except that "mixer layers" based on MLPs are used instead of the attention mechanism. Please see [THK+21] for details.

Procedure. To perform each of the experiments, we first fix a dimension to be frozen during the training. Afterward, we train the model on the frozen training set to make the model learn the frozen function. Finally, we evaluate the trained model uniformly on the Boolean hypercube ($\{\pm 1\}^n$) to compute the out-of-distribution generalization error.

Now, we explain the hyperparameters used for the experiments. Note that the experiments are aimed to exhibit an implicit bias towards low degree monomials and consequently to show that the generalization error is close to the Boolean influence. Therefore, the experiments are not focused on the learning of the frozen function itself and the in-distribution generalization error. In other words, we are interested in the setting that the frozen function is learned during the training, and then we want to examine the out-of-distribution generalization. Due to this reason, we have always used a

relatively large number of training samples. Moreover, we have not used a learning rate scheduler and we have not done extensive hyperparameter tuning. Generally, we have considered two optimizers for the training of the models: mini-batch SGD and Adam [KB14]. We describe the setting used for each of them below:

- **SGD.** When using SGD, we tried using 0 and 0.9 momentum. We observed that the use of momentum remarkably accelerates the training process, and hence we continued with 0.9 momentum. Nonetheless, we also performed experiments using SGD without momentum, and we did not notice any difference in their preference towards low degree monomials and therefore the out-of-distribution generalization error. For learning rate of SGD, we tried values in $\{10^{-3}, 5 \times 10^{-4}, 2.5 \times 10^{-4}, 10^{-4}\}$ and selected the learning rate dependent on the model and task (more on this below). Additionally, we always used the mini-batch size of 64 with SGD.
- **Adam.** In our experiments with Adam, we used default values of the optimizer and only changed the learning rate. For learning rate, we tried values in $\{10^{-4}, 5 \times 10^{-5}, 10^{-5}\}$ and finally selected 5×10^{-4}. While employing Adam, we used mini-batch size of 64 for PVR tasks with 3 pointer bits (11 bits in total) and mini-batch size of 1024 for PVR tasks with 4 pointer bits (20 bits in total).

We selected the learning rate (and in general, hyperparameters) based on the speed of the convergence and its stability. Note that we set the number of epochs for each task to a value to ensure the training loss and in-distribution generalization error are small enough.[10]

Finally, we note that all of our experiments are implemented using PyTorch framework [PGM+19], and the training has been done on NVIDIA A100 GPUs. The experiments presented in this paper took approximately 250 GPU hours. Note that we have repeated PVR experiments with 3 pointer bits 40 times and the rest of the experiments 20 times, and have reported the averaged results and 95% confidence interval. Please refer to the code for more details on the experiments.

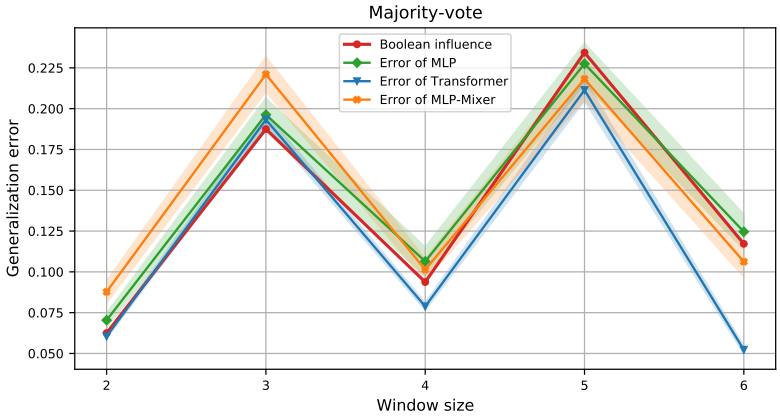
Linear neural networks. For the experiments on linear models, we considered fully connected linear neural networks with fixed hidden layer size of 256. As presented in Figure 4, we varied the initialization and depth of these networks. For optimizing linear neural networks, we used mini-batch SGD with 64 and 10^{-5} as the batch size and learning rate respectively. Note that we trained linear models on CPU and stopped the training when the loss became less than 10^{-8}.

F.2 Additional Results

More PVR tasks. First, we compare the generalization error and the Boolean influence for more PVR functions. In the additional experiments, we consider the cyclic version of the PVR (i.e., $x_{n+1} = x_{p+1}$), and due to the symmetry, we only freeze one dimension of the input. Also, we use Adam to optimize the models (instead of SGD) due to faster convergence. As a first example, we consider PVR tasks with 3 pointer bits (11 bits in total) and varying window sizes. We use majority-vote and parity as the aggregation functions (see Appendix E for computation of the Boolean influence for such functions). In Figure 5, the window size of the aforementioned PVR tasks is varied in the x-axis and the averaged generalization error over 40 experiments is shown. Figure 5 (top) corresponds to the case where majority-vote is used as the aggregation function whereas Figure 5 (bottom) shows the results when parity is the aggregation function. It can be seen that in this setting, the generalization error of all models follow the Boolean influence closely. Note that learning parity function becomes increasingly difficult as the window size is increased. Even for $w = 4$, the MLP and MLP-Mixer models could not learn the frozen function completely and their in-distribution generalization loss was between 0.05 and 0.10.

Furthermore, we experimented on PVR tasks of larger scales. To this end, we consider PVR tasks with 4 pointer bits (20 bits in total) and different window sizes and aggregation functions. For these experiments, we also used Adam optimizer with batch-size of 1024. We repeated each experiment 20 times. The generalization error and Boolean influence for these functions are given in Table 1. It can be observed that for these experiments, the generalization errors of MLP and Transformer are well approximated by the Boolean influence; while MLP-Mixer has higher generalization error. The

[10] This is problem dependent; nonetheless, we generally refer to errors of order of magnitude 10^{-2} or less.



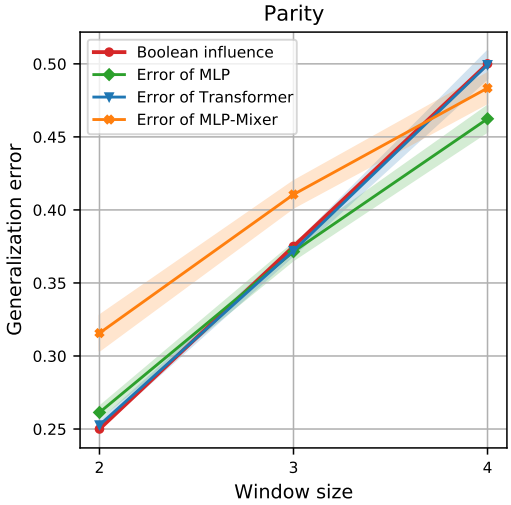


Figure 5: PVR tasks with 3 pointer bits and varying window sizes where the aggregation function is majority-vote (top) and parity (bottom). X-axis represents the window size of the PVR task, and y-axis shows the value of the generalization error and the Boolean influence.

results of Figure 5 and Table 1 indicate that the implicit bias towards low-degree monomials also exists when Adam is used as the optimizer and therefore is not limited to SGD.

Table 1: Generalization error for PVR tasks with 4 pointer bits

PVR task			Generalization error		
Aggregation	Window size	Boolean influence	MLP	Transformer	MLP-Mixer
Min	2	0.0625	0.062 ± 0.004	0.068 ± 0.006	0.118 ± 0.016
Parity	3	0.1875	0.206 ± 0.004	0.198 ± 0.015	0.329 ± 0.017
Majority	3	0.09375	0.099 ± 0.004	0.095 ± 0.001	0.194 ± 0.022
Majority	4	0.046875	0.051 ± 0.004	0.049 ± 0.002	0.094 ± 0.019

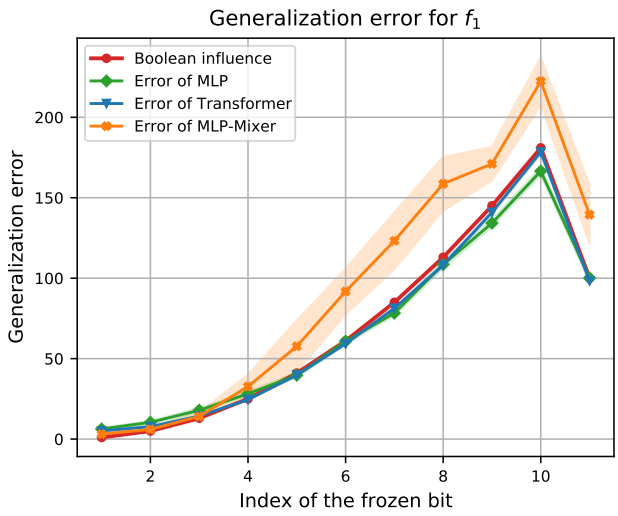


Figure 6: Comparison between the Boolean Influence and generalization error for $f_1(x_1, \ldots, x_{11}) = x_1x_2 + 2x_2x_3 + 3x_3x_4 + \cdots + 10x_{10}x_{11}$. Frozen coordinates are represented by the x-axis; while the y-axis represents the value of generalization error and the Boolean influence.

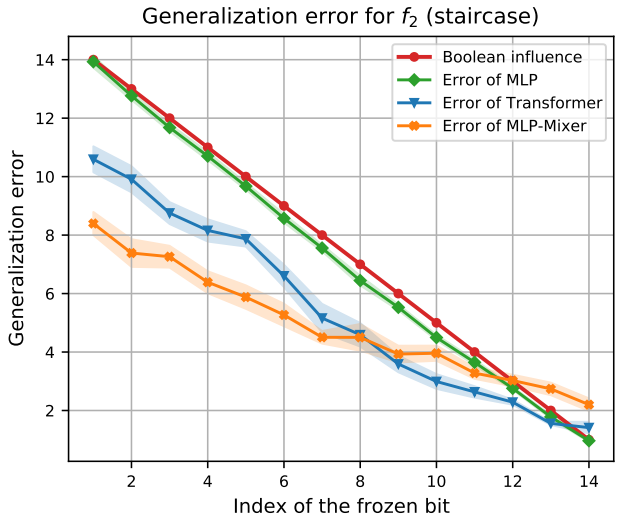
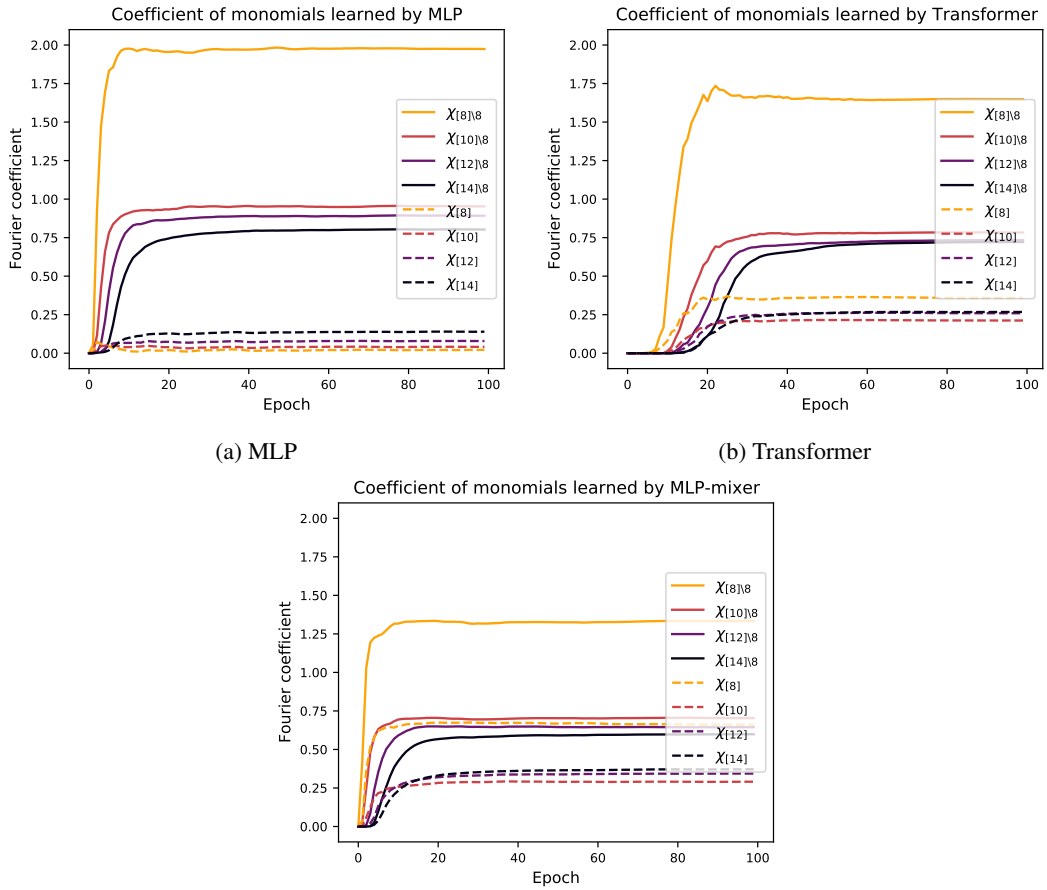


Figure 7: Comparison between the generalization loss in the canonical distribution shift setting and the Boolean Influence for $f_2(x_1, x_2, \ldots, x_{14}) = x_1 + x_1x_2 + x_1x_2x_3 + \cdots + x_1x_2x_3 \cdots x_{14}$.

Non-PVR examples. We also experimented on non-PVR functions. As the first example, we consider the target function $f_1(x_1, \ldots, x_{11}) = x_1x_2 + 2x_2x_3 + 3x_3x_4 + 4x_4x_5 + \cdots + 10x_{10}x_{11}$ which is a sum of second degree monomials. For each of the architectures, we freeze a coordinate (ranging from 1 to 11), train the model on the frozen samples using mini-batch SGD and evaluate the generalization loss. The relation between the Boolean influence and the averaged generalization error over 20 runs for f_1 is demonstrated in Figure 6. It can be seen that the generalization errors of the MLP and the Transformer model are again well approximated by the Boolean influence. However, the generalization error of the MLP-Mixer follows the trend of Boolean influence with an offset. This implies that the MLP and Transformer have a stronger preference for low-degree monomials than the MLP-Mixer in this case. As the last example, we consider the vanilla staircase



(a) MLP

(b) Transformer

(c) MLP-Mixer

Figure 8: The coefficients of different monomials learned by the MLP, Transformer, and MLP-Mixer when learning the staircase function, with $x_8 = 1$ frozen during the training. Note that $\chi_T(x) = \prod_{i \in T} x_i$, e.g., $\chi_{[8]} = x_1 x_2 \cdots x_8$. Generally, monomials of lower degrees are learned faster by the models. Consequently, the models prefer to learn the monomials which exclude the frozen index.

function for 14 bits, i.e., $f_2(x_1, x_2, \ldots, x_{14}) = x_1 + x_1x_2 + x_1x_2x_3 + \cdots + x_1x_2x_3 \cdots x_{14}$ (see [ABAB+21, ABAM22] for theoretical results on such staircase functions). We train models for this function using mini-batch SGD. In Figure 7, we report the generalization errors of the MLP, Transformer, and MLP-mixer models for each frozen coordinate of f_2, as well as the values of the Boolean influence of the corresponding index. Note that the generalization errors have been averaged over 20 runs. It can be observed that the generalization loss of MLP is very close to the Boolean influence in this case as well. However, the generalization errors of the Transformer and MLP-Mixer follow the Boolean influence with an offset. It is worth noting that the previous two functions are quite different than the PVR function: the PVR function has strong 'symmetries' given by the fact that each window is treated similarly with the aggregation function, and thus one may expect that certain architectures would exploit such symmetries. Thus, the PVR function is still a staircase function [ABAM22], of leap 2 in the example of Section 1.2, but it is a staircase function with the related symmetries. Instead, the two functions considered here are staircase functions that do not have any such symmetries.

Figure 7 shows that in some cases the Boolean influence may not always give a tight characterization of the generalization error. However, it appears that in such cases the offset still maintains the general trend of the influence. As an attempt to better understand this offset, recall the relation between the

Boolean influence and the generalization error in terms of the implicit low-degree bias: the stronger the preference for low-degree monomials is, the closer the generalization error is to the Boolean influence. We thus plot the coefficient of different monomials for f_2 and these three models while $x_8 = 1$ is frozen during the training in Figure 8. One can observe that for this staircase function, the bias towards low-degree monomials is stronger for MLP and it is weaker for Transformer and MLP-mixer. This well explains the relation between the Boolean influence and the generalization error of different models depicted in Figure 7, where the generalization error of MLP is significantly closer to the Boolean influence, compared to the generalization error of Transformer and MLP-Mixer.

G Intuition on the linear neural networks

At last, we provide heuristic justifications for the effect of depth and initialization on the generalization error and its closeness to the Boolean influence in the case of linear neural networks. Let $f_{NN}(x;\Theta) = w_L^T(W_{L-1}^T(\cdots(W_1^T x + b_1)\cdots) + b_{L-1}) + b_L$ be a linear neural network with depth L, after training in the canonical holdout setting where the k-th bit is frozen to 1. Assume the target function to be linear. After training, the neural network learns the frozen function $f_{-k}(x) = f(x_{-k})$. Note that the bias of the frozen function is $\hat{f}(\{\emptyset\}) + \hat{f}(\{k\})$ (where with $\hat{f}$ we denote the Fourier coefficients of the target function f), that is expressed by the neural network by the following:

$$B := (w_L^T W_{L-1}^T \cdots W_2^T b_1 + w_L^T W_{L-1}^T \cdots W_3^T b_2 + \cdots + w_L^T b_{L-1} + b_L) + w_L^T W_{L-1}^T \cdots W_2^T w_{1,k}^T \tag{89}$$

where by $w_{1,k}$ we indicate the weights in the first layer of the frozen dimension k. Assuming the neural network has learned the function, we have

$$\hat{f}_{NN}(\{i\}) = \hat{f}(\{i\}) \qquad \text{for all } i \neq k, \tag{90}$$

$$\hat{f}_{NN}(\{\emptyset\}) + \hat{f}_{NN}(\{k\}) = B = \hat{f}(\{\emptyset\}) + \hat{f}(\{k\}), \tag{91}$$

where we denoted by $\hat{f}_{NN}$ the Fourier coefficients of f_{NN}. Therefore, applying Parseval identity we find that the generalization error equals

$$\begin{aligned}
\mathrm{gen}(f, f_{NN}) &= \frac{1}{2}\mathbb{E}_X(f(X) - \hat{f}_{NN}(X))^2 && (92)\\
&= \frac{1}{2}\left(\hat{f}(\{\emptyset\}) - \hat{f}_{NN}(\{\emptyset\})\right)^2 + \frac{1}{2}\left(\hat{f}(\{k\}) - \hat{f}_{NN}(\{k\})\right)^2 && (93)\\
&= \frac{1}{2}\left(\hat{f}(\{\emptyset\}) - (w_L^T \cdots W_2^T b_1 + \cdots + b_L)\right)^2 + \frac{1}{2}\left(\hat{f}(\{k\}) - w_L^T \cdots W_2^T w_{1,k}^T\right)^2 && (94)\\
&= (\hat{f}(\{k\}) - w_L^T \cdots W_2^T w_{1,k}^T)^2. && (95)
\end{aligned}$$

Therefore, the amount of bias captured by $w_L^T W_{L-1}^T \cdots W_2^T w_{1,k}^T$ determines the generalization error. Particularly, if $w_L^T \cdots W_2^T w_{1,k}^T$ goes to zero, the generalization error will become equal to the Boolean influence. Note that $x_k = 1$ during the training, therefore $w_{1,k}$ has the same training dynamics as the bias of the first layer b_1.

Effect of depth. From (89), we note that there are $L+1$ terms that contribute to B, and one of them is indeed $w_L^T \cdots W_2^T w_{1,k}^T$. Therefore as the depth L increases, if those terms are appropriately aligned, one can expect that the contribution of each term, including $w_L^T \cdots W_2^T w_{1,k}^T$, decreases; thus, the generalization error becomes closer to the Boolean influence.

Effect of initialization. The gradients of the parameters for a sample x are given by

$$\begin{aligned}
\nabla_{b_L} L(\Theta, x, f) &= (f_{NN}(x;\Theta)(x) - f_{-k}(x)),\\
\nabla_{b_{L-1}} L(\Theta, x, f) &= (f_{NN}(x;\Theta)(x) - f_{-k}(x))w_L,\\
&\vdots\\
\nabla_{b_1} L(\Theta, x, f) &= (f_{NN}(x;\Theta)(x) - f_{-k}(x))W_2 W_3 \cdots w_L.
\end{aligned}$$

Now, consider the first update of the parameters. As we decrease the scale of initialization, the ratio of $\frac{\nabla_{b_L}}{\nabla_{b_{L-1}}}, \cdots, \frac{\nabla_{b_2}}{\nabla_{b_1}}$ increases which implies that b_1 would have the smallest update and b_L will have the largest update. Since the dynamics of $w_{1,k}$ and b_1 are the same, the frozen dimension would contribute the least to the bias after the first iteration. Our experiments on decreasing the scale of initialization suggest that this argument is not limited to the first iteration. In other words, using small enough initialization the bias will be mostly captured by the bias terms in other layers, which results in generalization error being close to the Boolean influence.