Leap Complexity: How Stochastic Gradient Descent Exploits Hierarchical Structure in Functions to Learn Efficiently

Arif Kerem Dayi

December 2023

1 Motivation

Neural networks have proven extremely powerful in practice due to their ability to learn very complex functions, despite their high dimensionality and complex loss landscape. It is commonly conjectured that neural networks exploit (1) low dimensional latent structures, and (2) hierarchical structures to learn effectively. We would like to ideally characterize the 'hierarchical structure' of functions: [AAM23] proposes the leap complexity, which is a notion of hierarchy in functions, and matches the CSQ lower bounds for learning 2-layer neural networks. The leap complexity generalizes the previous notion of the 'staircase property' as given in [ABAB$^+$21].

The paper improves the existing bounds on the runtime of online-SGD (which are of order d^D where D is the degree of a function's hermite representation), and proposes a bound of $d^{\max\{\mathrm{Leap}(f_*)-1,1\}}$ which can be significantly smaller than d^D. Furthermore, the paper uses an analysis technique that analyzes the whole trajectory of training, rather than a 1-step gradient descent analysis. This allows more insight into what is happening during online SGD training and how SGD picks up the support of f_* over time. This analysis technique uses tools from hermite analysis and stochastic processes to obtain a drift-martingale decomposition of learning dynamics. Studying this analysis technique is interesting on its own as it can prove better bounds than one-step analyses and lead to more insight about stochastic optimization algorithms.

2 Setup

We are interested in learning functions $f_*(\mathbf{x})$ that have low-latent dimension. That is, $f_*(\mathbf{x})$ only depends on $\mathbf{x}$ through a low-dimensional subspace. Formally, we have $f_*(\mathbf{x}) = h_*(\mathbf{z})$ where $\mathbf{z} = \mathbf{U}\mathbf{x}$, and $\mathbf{U} \in \mathbb{R}^{P \times d}$ with $\mathbf{U}\mathbf{U}^\top = \mathrm{Id}_P$ represents a $P = O(1)$ dimensional subspace[1]. Throughout [AAM23], the authors furthermore assume that $\mathbf{U}$ is simply the projection onto the first P coordinates to simplify the analysis. That is, we only work with the first P coordinates of the input. This will not be true in general, and is only used to simplify the analysis. We also assume $h_*(\mathbf{z})$ is a degree D-polynomial, so that our target function is a degree-D polynomial of the first P coordinates of the input.

In the paper, the authors consider two distributions for the inputs $\mathbf{x}$. One distributed gaussian and the other boolean uniform over the hypercube. In this project, we consider the gaussian case where $\mathbf{x} \sim \mu^{\otimes d}$ where $\mu = \mathcal{N}(0,1)$, but the analysis for both cases are similar. We consider the online learning case, where at each time step t we sample a new data point $(\mathbf{x}^t, y^t)$ with $\mathbf{x}^t \sim \mu^{\otimes d}$ and $y^t = f_*(\mathbf{x}^t) + \epsilon^t$. Here, ϵ^t is independent of $\mathbf{x}^t$ and is K-sub Gaussian.

[1] $P = O(1)$ as d grows

We want to use a 2-layer network which we call $\hat{f}_{\mathsf{NN}}$ to learn f_*. We have M neurons and the weights of our network are represented by $\theta = (a_j, b_j, \mathbf{w}_j)_{j=1}^m$. Therefore, we have

$$\hat{f}_{\mathsf{NN}}(\mathbf{x};\theta) = \sum_{j\in[M]} a_j \sigma(\langle \mathbf{x}, \mathbf{w}_j\rangle + b_j) \tag{1}$$

3 Leap Complexity

The leap complexity formalizes the notion of a 'hierarchical structure' in functions, and is closely related to the runtime of online SGD while training neural networks. Now, we state the leap complexity and also a general conjecture relating leap complexity to the runtime of online SGD. Let $h_* \in L_2(\mu^{\otimes P})$ be any function and let $\mathrm{He}_k(z)$ denote the k'th hermite polynomial. Recall that $\{\mathrm{He}_k\}_{k\geq 0}$ forms an orthonormal basis for $L_2(\mu)$. Similarly, let $S \in \mathbb{Z}^P$ denote a set of 'degrees' and let $\chi_S(\mathbf{z}) = \prod_{i\in[P]} \mathrm{He}_{S_i}(z_i)$. Here, the collection $\{\chi_S\}_{S\in\mathbb{Z}^P}$ form an orthonormal basis of $L_2(\mu^{\otimes P})$. Hence, for any given function $h_* \in L_2(\mu^{\otimes P})$, we have the hermite coefficient $\hat{h}_*(S) = \mathbb{E}_{\mathbf{z}\sim\mu^{\otimes P}}[\chi_S(\mathbf{z})h_*(\mathbf{z})]$ and we have

$$h_*(\mathbf{z}) = \sum_{S\in\mathbb{Z}^P} \hat{h}_*(S)\chi_S(\mathbf{z}) \tag{2}$$

Using this hermite representation of a function, we define the leap complexity.

Definition 1 (Leap Complexity)**.** *Let $h_* \in L_2(\mu^{\otimes P})$ and let $\mathcal{S}(h_*) = \{S_1, \dots, S_m\}$ denote the non-zero basis elements as in the expansion Equation* (2)*. The leap complexity is defined as*

$$\mathrm{Leap}(h_*) = \min_{\pi\in\Pi_m} \max_{i\in[m]} \left\| S_{\pi(i)} \setminus \cup_{j=0}^{i-1} S_{\pi(j)} \right\|_1$$

where $\left\| S_{\pi(i)} \setminus \cup_{j=0}^{i-1} S_{\pi(j)} \right\|_1 \triangleq \sum_{k\in[P]} S_{\pi(i)} \mathbb{1}\{S_{\pi(j)}(k) = 0,\ \forall j \in [i-1]\}$ with $S_{\pi(0)} = 0^P$. If $\mathrm{Leap}(h_*) = l$ we say that h_* is a Leap-l function.

This definition deserves some explanation. Intuitively, it is saying that we are looking at all possible orderings of the terms of our function, and finding the one that gives us the minimum 'leap.' Here, we think of the leap as follows. Suppose you start with 0 terms and keep adding terms to your function. At each step, the leap associated with that step is the sum of the degrees of the coordinates you added that were not present earlier in your terms. Then, the leap of a permutation is the maximum leap at any given step, and the leap of a function is the minimum leap achievable by any permutation. This definition suggests that the hardness of training comes from large degree terms in the function that do not have corresponding lower degree terms. For instance, we have

$$\mathrm{Leap}(\mathrm{He}_1(x_1)\mathrm{He}_1(x_2)\mathrm{He}_2(x_3)) = \mathrm{Leap}(\mathrm{He}_3(x_1)) = 3$$
$$\mathrm{Leap}(\mathrm{He}_2(x_1) + \mathrm{He}_2(x_2) + \mathrm{He}_{19}(x_1)\mathrm{He}_{23}(x_2)) = 2$$

So, even though the second polynomial has higher degree, it has lower leap, and therefore 'easier' to learn due to its hierarchical structure. Furthermore, if you picked a 'random' function, it is likely to have lower order terms, which means it will be easy to learn.

Now, we state a general conjecture relating the leap complexity to the runtime of online SGD on functions.

Conjecture 1. *Let $f_* : \mathbb{R}^d \to \mathbb{R}$ in $L_2(\mu^{\otimes d})$ be a function satisfying the low-latent dimension hypothesis. I.e. $f_*(\mathbf{x}) = h_*(\mathbf{U}\mathbf{x})$ for some $P = O(1)$. Let $\hat{f}^t_{\mathsf{NN}}$ a fully connected neural network with $\mathrm{poly}(d)$ edges that is the output of a training process using t steps of online SGD on the square loss. Then, for all but a measure-0 set of functions, we have*

$$\mathbb{E}_{\mathbf{x}}[(\hat{f}^t_{\mathsf{NN}}(\mathbf{x}) - f_*(\mathbf{x}))] \leq \epsilon \text{ if and only if } t = \tilde{\Omega}(d^{\max\{1, \mathrm{Leap}(h_*)-1\}})\mathrm{poly}(1/\epsilon)$$

The conjecture says that, to achieve ϵ error for functions that map d dimensions to 1 dimension, we need polynomial number of steps in d with the exponent relating to the leap complexity.

4 Algorithm

Throughout the analysis, we make use of some regularity properties of the activation function σ. Specifically, we have the following assumption

Assumption 1 (Activation Function). *Let $\sigma : \mathbb{R} \to \mathbb{R}$ be as follows. There exists a constant $K > 0$ such that σ is $D+3$-differentiable with $\left\|\sigma^{(k)}\right\|_\infty \leq K$ for $k = 0, 1, \ldots, D+3$ and $|\mu_k(\sigma)| > \frac{1}{K}$ for $k = 0, 1, \ldots, D$ where $\mu_k(\sigma)$ is the k'th hermite coefficient of σ.*

Mainly, we want the magnitude of all the derivatives be bounded above while the hermite do not vanish. This assumption is satisfied for the shifted sigmoid function $\sigma(z) = \frac{1}{1+e^{-z+c}}$. However, it is not satisfied for ReLU as ReLU is not differentiable. We will be performing online-SGD to minimize the population loss

$$R(\theta) = \mathbb{E}_x[\ell(f_*(\mathbf{x}), f_{\mathsf{NN}}(x; \theta)]$$

However, we will not be using vanilla SGD. We need to make some crucial modifications to SGD to make our analysis viable. First, we train layerwise, so that we can analyze each of the layers separately. Furthermore, we keep the bias terms frozen. Secondly, we do some projections while training the first layer to make sure that the weights $\mathbf{w}_j$ do not blow up to infinity.

Specifically, we do 2 types of projection on the weights $\mathbf{w}_j$. First, we project them onto a ℓ_∞ ball of radius Δ to bound the maximum magnitude. Then, we take the 'small' coordinates (those that had magnitude less than r until the current time step) and project those coordinates to the unit sphere. Hence, the large coordinates are clipped and the small coordinates are kept on the unit sphere.

Let $0 < r$ be some parameter. We define the small coordinates up to time t for neuron j as

$$S_{j,t} = \{i \in [d] : |\tilde{w}^{t'}_{j,i}| < r \text{ for all } 1 \leq t' \leq t\}$$

where $\tilde{\mathbf{w}}_j$ is the neuron weights before projection. Then, define $\mathbf{P}_\infty(\mathbf{w})_i = \mathrm{sign}(w_i)\min\{\Delta, |w_i|\}$. Similarly, if $\mathcal{S}_t(\mathbf{w})$ denotes the vector obtained by setting all the components of $\mathbf{w}$ that have magnitude at least r to 0, we have

$$\mathbf{P}_t(\mathbf{w})_i = \begin{cases} w_i & \text{if } i \in S_t \\ \frac{w_i}{\|\mathcal{S}_t(\mathbf{w})\|} & \text{otherwise} \end{cases} \tag{3}$$

Then, our projection is $\mathrm{Proj}_t(\mathbf{w}) \triangleq \mathbf{P}_t\mathbf{P}_\infty(\mathbf{w})$. Finally, because of the projection onto the unit sphere defined by the small coordinates, we use the spherical gradient. Specifically, for any function f the spherical gradient is defined as

$$\mathrm{grad}_{\mathbf{w}_j^t} f(\mathbf{w}_j^t) \triangleq \nabla_{\mathbf{w}_j^t} f(\mathbf{w}_j^t) - \mathcal{S}_t(\mathbf{w}_j^t)\langle \mathcal{S}_t(\mathbf{w}_j^t), \nabla_{\mathbf{w}_j^t} f(\mathbf{w}_j^t)\rangle \tag{4}$$

This gives the gradient that is tangential to the direction of projection. The full training algorithm is given in Algorithm 1. The main challenge in analyzing the algorithm is the first layer. Once the first layer is frozen, the second layer is simply a linear model.

There are important caveats about this algorithm selection. First, the uniform projection onto the ℓ_∞ ball makes the algorithm non-rotationally invariant. Because of this, the analysis is limited to the specific latent structure (only using first P coordinates) instead of the more general latent structure given by $\mathbf{z} = \mathbf{Mx}$. Second, the clipping of each of the neurons by the same amount decreases the diversity of the neuron values. This is because the entries of neurons that are in the support of the latent space all converge to Δ up to a sign. This limits which functions could be learned in the second layer training by a linear model since we do not have a great diversity of features at the end of first round training. Therefore, the results we have are simplied cases, and are important for modeling purposes; there are still important theoretical challenges remaining in this direction.

Algorithm 1 Online SGD with Projection Steps. Parameters: initialization scales $\kappa, \rho >$, step sizes η_1, η_2, number of iterations $\overline{T_1}, \overline{T_2}$, second layer ℓ_2 regularization parameter λ_a and projection parameters $r, \Delta > 0$.

```
Initialize $a_j^0 \sim \mathrm{Unif}(\{\pm\kappa\}), b_j \sim \mathrm{Unif}([-\rho, \rho]), w_j^0 \sim \mathrm{Unif}(\{\pm 1/\sqrt{d}\}^d)$
for $t = 0, \dots, \overline{T_1} - 1$ and all $j \in [M]$ do
    $\tilde{\mathbf{w}}_j^{t+1} \leftarrow \mathbf{w}_j^t - \eta_1 \mathrm{grad}_{\mathbf{w}_j^t} \ell(y^t, \hat{f}_{\mathsf{NN}})(\mathbf{x}^t, \theta^t)$
    $\mathbf{w}_j^{t+1} \leftarrow \mathrm{Proj}_t(\tilde{\mathbf{w}}_j^{t+1})$
    Leave other parameters untouched, $a_j^{t+1} = a_j^t, b_j^{t+1} = b_j^t$.
end for
for for $t = \overline{T_1}, \dots, \overline{T_1} + \overline{T_2} - 1$ and all $j \in [M]$ do
    Freeze first layer: $\mathbf{w}_j^{t+1} = \mathbf{w}_j^t$, $b_j^{t+1}, b_j^t$
    Perform GD with ridge regularization: $a_j^{t+1} = (1 - \lambda_a) a_j^t - \eta_2 \frac{\partial}{\partial a_j^t} \ell(y^t, \hat{f}_{\mathsf{NN}}(\mathbf{x}^t, \theta^t)$
end for
```

5 Analysis

Here, we initially consider learning a single monomial. Let $h_*(\mathbf{x}) = \prod_{i \in [P]} \mathrm{He}_{k_i}(x_i)$. We call $[P]$ the support of this function. Our goal is to prove the following theorem:

Theorem 1 (Learning a monomial, first layer)**.** *Let σ be an activation that satisfies assumption 1. Then, for $0 < r < \Delta$ sufficiently small constants that depend on D, K and $\rho \leq \Delta$, the following holds: For any $C_* > 0$ there exist constant $C_i, i = 0, 1, \dots, 6$ such that*

$$\overline{T}_1 = C_0 d^{D-1} \log(d)^{C_0}, \quad \eta_1 = \frac{1}{C_1 \kappa d^{D/2} \log(d)^{C_1}}, \quad \kappa \leq \frac{1}{C_2 d^{C_2}}$$

and for sufficiently large d such that $r \geq C_0 \log(d)^{C_0}/\sqrt{d}$, the following event holds with probability at least $1 - Md^{-C_*}$. For any neuron $j \in [M]$ such that $a_j^0 \mu_D(\sigma) \prod_{i \in [P]} (w_{j,i}^{k_i}) > 0$, we have

1. *On the support (for $i \in [P]$), we have $|w_{j,i}^{\overline{T}_1} - \mathrm{sign}(w_{j,i}^0) \cdot \Delta| \leq C_5/\sqrt{d \log(d)}$*

2. *Outside the support (for $i = P+1, \ldots, d$) and for $\sum_{i>P}(w_{j,i}^{\overline{T}_1})^2 = 1$, we have $|w_{j,i}^{\overline{T}_1} - w_{j,i}^0| \leq C_6 r^2 \sqrt{d}$.*

This theorem says that for neurons that are initially aligned with the function, the coordinates in the support are amplified whereas the coordinates outside the support remain small. Additionally, this theorem says that if we stop early (i.e. for $t \leq \overline{T}_1/(C_4 \log(d)^{C_4})$) the neuron weights are close to their initialization. This means that the learning happens towards the end of the dynamics, and not in the beginning. Another important thing to note is that $\eta_1 = o(1/\sqrt{\overline{T}_1})$. This is because the noise terms (coming from the stochastic sampling of the gradients) form a martingale that grows at rate $\sqrt{\overline{T}_1}$, and the drift terms (coming from population gradients), grow at rate $\overline{T}_1$. This choice of η_1 will eliminate the contribution of the noise terms while still keeping the drift term.

First, we will soon show that the training dynamics can be well approximated by a correlation dynamics, where each neuron's update is independent of other neuron's updates. Because of this, we will analyze each neuron separately. Therefore, from now on, we will drop the subscript j in our neurons and use $(\mathbf{w}, a)$ instead of $(\mathbf{w}_j, a_j)$. Furthermore, because we analyze the whole dynamics, we would like to be able to condition on where we are at during the dynamics. Therefore, we will define some stopping times that we will use for conditioning.

Before we do that, define the negative gradient at time t to be

$$\mathbf{v}^t \triangleq (y^t - \hat{f}_{\mathsf{NN}}(\mathbf{x}^t; \theta^t))a^0 \sigma'(\langle \mathbf{w}^t, boldx^t \rangle)\mathbf{x}^t \tag{5}$$

Then, define the spherical gradient $\tilde{\mathbf{v}}^t$ we use in our training as

$$\tilde{\mathbf{v}}^t \triangleq -\mathrm{grad}_{\mathbf{w}^t} \ell(y^t, \hat{f}_{\mathsf{NN}}(\mathbf{x}^t; \theta^t)) = \mathbf{v}^t - \mathcal{S}_t(\mathbf{w}^t)\langle \mathcal{S}_t(\mathbf{w}^t), \mathbf{v}^t \rangle \tag{6}$$

Hence, the update equations become $\tilde{\mathbf{w}}^{t+1} = \mathbf{w}^t + \eta_1 \tilde{\mathbf{v}}^t$, $\overline{\mathbf{w}}^{t+1} = \mathbf{P}_\infty \tilde{\mathbf{w}}^{t+1}$, and $\mathbf{w}^{t+1} = \mathbf{P}_{t+1}\overline{\mathbf{w}}^{t+1}$.

5.1 Simplifying Assumptions

In our proof, we make some important simplifying assumptions. First, because of assumption 1, one can show that small shifts in the activation function do not affect the hermite coefficients of the activation a lot. Then, we can choose $b_j \leq \rho$ small such that the bias terms are small. The bias terms are also frozen during training, so the effect of the bias terms can be assumed to be negligible, and we assume $b_j = 0$. We furthermore assume $w_{j,i} > 0$ for our proof. This is because we can flip the sign of the $w_{j,i}$ and also the sign of the second layer weights and the input weights, without affecting the distribution they are each sampled from, up to signs. This is mainly to not keep track of the signs of the neuron weights throughout our analysis.

5.2 Stopping Times on Dynamic

Because we analyze the whole training dynamic, we will need to keep track of where the current time step t is. We initially define the following stopping times that will help us condition on the sizes of the neuron weights, and also the sizes of other variables such as the gradients, inputs, labels, etc.

$$\tau^+ \triangleq \inf\left\{t \geq 0 : \max_{i=P+1,\ldots,d} \{|\tilde{w}_i^{t+1}| \vee |w_i^{t+1}| \geq 3/(2\sqrt{d})\}\right\} \tag{7}$$

$$\tau^- \triangleq \inf\left\{t \geq 0 : \min_{i\in[d]} \{|\tilde{w}_i^{t+1}| \wedge |w_i^{t+1}|\} \leq 1/(2\sqrt{d})\right\} \tag{8}$$

$$\tau^0 \triangleq \inf\left\{t \geq 0 : \max(\left\|\mathbf{v}^t/a^0\right\|_\infty, \left\|\tilde{\mathbf{v}}^t/a^0\right\|, |y^t|, \left\|\mathbf{x}^t\right\|_\infty) \geq C_0 \log(d)^{C_0}\right\} \tag{9}$$

We will adopt the notation from the paper and use $\vee$ and $\max$ interchangeably, and also $\wedge$ and $\min$ interchangeably. Ideally, we would like τ^+ to be larger than our training epoch $\bar{T}_1$ since we want the weights not in the support (i.e. w_i for $i = P+1, \dots, d$) to be small. In fact, we will later show that $\tau^+ \wedge \tau^- \wedge \tau^0 > \overline{T}_1$ with high probability. We will condition t being smaller than these stopping times in most of our proof. First, we show that we have $\tau^0 \le \tau^+ \wedge d^D$ with high probability.

Lemma 1 (Bound on τ_0)**.** *Suppose $\Delta \le 1$. Then, for any constant $C_* > 0$, there exists some other large enough constant C_0 that only depends on C_*, D, K such that for $d \ge 2$, we have*

$$\Pr[\tau^0 \le \tau^+ \wedge d^D] \le d^{-C_*}$$

The proof of this lemma directly follows from summing all the probabilities of $\tau_0 = t$ for $t \le \tau^+ \wedge d^D$ and some tail bounds shown on ϵ^t. By making C_0 larger, we decrease the probability of $\tau^0 = t$ to achieve the probability bound on the right. We omit the full proof, as it is peripheral to this paper. This result will be extremely useful as it helps us bound the magnitudes of several quantities by conditioning $t < \tau^0$.

5.3 Reduction to Correlation Dynamics

Now, we prove the earlier claim that the dynamics is closely approximated by a correlation dynamics. Define the correlation dynamics gradient and its spherical counterpart as follows:

$$\mathbf{u}^t = a^0 y^t \sigma'(\langle \mathbf{w}^t, \mathbf{x}^t \rangle)\mathbf{x}^t \quad \text{and} \quad \tilde{\mathbf{u}}^t = \mathbf{u}^t - \mathcal{S}_t(\mathbf{w}^t)\langle \mathcal{S}_t(\mathbf{w}^t), \mathbf{u}^t \rangle \tag{10}$$

We will show that $\mathbf{u}^t$ and $\tilde{\mathbf{u}}^t$ are close to $\mathbf{v}^t$ and $\tilde{\mathbf{v}}^t$ respectively. The idea is that we can make the magnitude of $\hat{f}_{\mathsf{NN}}(\mathbf{x}^t;\theta^t)$ arbitrarily small by making κ (initialization size of second layer) small. For $t < \tau^0$, notice that

$$\left\|\mathbf{v}^t - \mathbf{u}^t\right\|_\infty / |a_0| \le \left\|\hat{f}_{\mathsf{NN}}(\mathbf{x}^t;\theta^t)\sigma'(\langle \mathbf{w}^t, \mathbf{x}^t\rangle)\mathbf{x}^t\right\|_\infty \overset{(a)}{\le} \kappa K^2 M C_0 \log(d)^{C_0} \le \tilde{\kappa} \tag{11}$$

for $\tilde{\kappa} = 2\kappa d K^2 M C_0$. In (a), we bound $\hat{f}_{\mathsf{NN}}(\mathbf{x}^t;\theta^t)$ using $\|\sigma\|_\infty \le K$ and σ' similarly using the assumption on the activation (Assumption 1), and $\|x\|_\infty \le C_0 \log(d)^{C_0}$ using $t \le \tau^0$. Similarly, for spherical gradients, we have

$$\begin{aligned} \left\|\tilde{\mathbf{v}}^t - \tilde{\mathbf{u}}^t\right\|_\infty / |a_0| &\le \left\|\mathbf{v}^t - \mathbf{u}^t - \mathcal{S}_t(\mathbf{w}^t)\langle \mathcal{S}_t(\mathbf{w}^t), \mathbf{v}^t - \mathbf{u}^t\rangle\right\|_\infty / |a_0| & (12) \\ &\overset{(b)}{\le} (1+d)\left\|\mathbf{v}^t - \mathbf{u}^t\right\|_\infty / |a_0| \le \tilde{\kappa} & (13) \end{aligned}$$

where we use $\|\mathcal{S}_t(\mathbf{w}^t)\|_\infty \le 1$ and $|\langle \mathcal{S}_t(\mathbf{w}^t), \mathbf{v}^t - \mathbf{u}^t\rangle| \le \|\mathcal{S}_t(\mathbf{w}^t)\|_1 \left\|\mathbf{v}^t - \mathbf{u}^t\right\|_\infty \le d\left\|\mathbf{v}^t - \mathbf{u}^t\right\|_\infty$ in (b). Notice that we divide the difference by $|a_0| = \kappa$. We can interpret this as follows: The contribution of other neurons to $\mathbf{v}^t$ scale with κ^2 whereas the effect of a neuron on its own scales with κ. Hence, we can reduce the contribution of other neurons relative to own contribution by scaling κ appropriately.

5.4 Rewriting Projection Step

We first define additional stopping times that are related to the projection steps. For all $i \in [d]$, define

$$\begin{aligned} \tau_i^r &\triangleq \inf\left\{t \ge 0 : |\tilde{w}_i^{t+1}| \ge r\right\} & (14) \\ \tau_i^\Delta &\triangleq \inf\left\{t \ge 0 : |w_i^{t+1}| \ge \Delta - |a^0|\eta_1 C_0 \log(d)^{C_0}\right\} & (15) \\ \tau^r &\triangleq \sup_{i\in[P]} \tau_i^r \quad \text{and} \quad \tau_i^\Delta \triangleq \sup_{i\in[P]} \tau_i^\Delta & (16) \end{aligned}$$

where C_0 is the same C_0 as before. Here, τ_i^r is the first time a coordinate is not used in the spherical projection (and it is never included again), and τ_i^Δ similarly gives us a way of conditioning that a coordinate is not projected onto the Δ-ℓ_∞ ball.

Then, notice that if $t < \tau_i^\Delta \wedge \tau^0$, we have that $|\tilde{w}_i^{t+1}| = |w_i^t + \eta_1 \tilde{v}_i^t| \leq \Delta$. This is because $w_i^t \leq \Delta - |a_0|\eta_1 C_0 \log(d)^{C_0}$ using $t < \tau_i^\Delta$ and $\tilde{v}_i^t < |a_0| C_0 \log(d)^{C_0}$ using $t < \tau^0$. Hence, we do not have projection on the i'th coordinate when $t < \tau_i^\Delta$. And whenever $t \geq \tau_i^\Delta$, we have that the weights were close to Δ at some point, and it can be shown that they will stay close to Δ. Now, we will get rid of the ℓ_∞ projection term by replacing it by a time-varying multiplicative constant. Define $\gamma_i^t = \min\left(1, \frac{\Delta - \mathrm{sign}(\tilde{v}_i^t) w_i^t}{\eta_1 |\tilde{v}_i^t|}\right)$ and let the truncated gradient be $g_i^t = \gamma_i^t \tilde{v}_i^t$. Then, the update rule becomes $\overline{\mathbf{w}^{t+1}} = \mathbf{w}^t + \eta_1 \mathbf{g}^t$ and $\mathbf{w}^{t+1} = \mathbf{P}_{t+1} \overline{\mathbf{w}^{t+1}}$. This helps us work with the gradients without worrying about the Δ-clipping projection.

5.5 Approximating Population Gradient

In this section, we will show that the gradients for the coordinates in the support and outside the support differ. The gradients for the support are positive, bounded below, whereas the gradients outside the support are close to 0. First, we will find an expression approximating the population gradient for $i \in [P]$. We can do this using Hermite analysis, and the fact that the 'correlation gradient' $\tilde{\mathbf{u}}^t$ is close to the actual gradient. We have the following lemma that lets us approximate the population gradient:

Lemma 2. *Let $\chi_*(\mathbf{w}^t) = \prod_{j \in [P]} (\mathbf{w}_j^t)^{k_j}$. Let $i \in [P]$ and $t < \tau_i^\Delta \wedge \tau^0$. Furthermore, let $t < \tau_i^r$ such that $i \in \mathcal{S}_t(\mathbf{w}^t)$. Then, we have*

$$\left| \overline{g}_i^t - a^0 \frac{\chi_*(\mathbf{w}^t)}{w_i^t} \left(k_i - (w_i^t)^2 \sum_{j \in S_t \cap [P]} k_j \right) \mathbb{E}_G[\sigma^{(D)}(\left\|\mathbf{w}^t\right\|_2 G)] \right| \leq |a^0| \tilde{\kappa}$$

We have similar (but different) expressions for the cases $t \geq \tau_i^r$ and $i > P$ which have similar proofs. We omit those here as the derivation is quite similar to this expression. Now, let us prove this lemma. From hermite analysis, we have the following identities:

$$\mathbb{E}_G[\mathrm{He}_k(G) f(G)] = \mathbb{E}_G[f^{(k)}(G)], \quad x\mathrm{He}_k(x) = \mathrm{He}_{k+1}(x) + k\mathrm{He}_{k-1}(x) \tag{17}$$

Now, we know that $|\overline{g}_i^t - \mathbb{E}_{(x^t, y^t)}[\tilde{u}_i^t]| \leq \tilde{\kappa}|a^0|$ as we showed that the correlation gradient approximates the true gradient. Now, by transforming the integral to standard gaussian, iteratively expanding the expectation and using the hermite-function product formula (eq. (17)), we have

$$\mathbb{E}\left[\prod_{j \in [P]} \mathrm{He}_{v_j}(x_j) \sigma'(\langle \mathbf{w}^t, \mathbf{x}^t \rangle)\right] = \left(\prod_{j \in [P]} (w_j^t)^{v_j}\right) \mathbb{E}_G\left[\sigma^{(1 + \sum_{i \in [P]} v_i)}(\left\|\mathbf{w}^t\right\|_2 G)\right]$$

Furthermore, from the second hermite formula (eq. (17)) we have

$$x_i f_*(\mathbf{x}) = \left(\prod_{j \neq i, j \in [P]} \mathrm{He}_{k_j}(x_j)\right) (k_i \mathrm{He}_{k_i - 1}(x_i) + \mathrm{He}_{k_i + 1}(x_i))$$

Then, writing out the definition of $\tilde{\mathbf{u}}^t$, we get

$$\begin{aligned}\mathbb{E}_{(\mathbf{x}^t,y^t)} &\overset{(a)}{=} \mathbb{E}_{(\mathbf{x}^t,y^t)}\left[a^0(f_*(\mathbf{x}^t)+\epsilon^t)\left(x_i^t - w_i^t\sum_{j\in S_t} w_j^t x_j^t\right)\sigma'(\langle \mathbf{w}^t,\mathbf{x}^t\rangle)\right] \\ &\overset{(b)}{=} a^0\frac{\chi_*(\mathbf{w}^t)}{w_i^t}\left(k_i - (w_i^t)^2\sum_{j\in S_t\cap[P]} k_j\right)\mathbb{E}_G[\sigma^{(D)}(\left\|\mathbf{w}^t\right\|_2 G)] \\ &\quad + a^0\chi_*(\mathbf{w}^t)\left(w_i^t - w_i^t\sum_{j\in S_t}(w_j^t)^2\right)\mathbb{E}_G[\sigma^{(D+2)}(\left\|\mathbf{w}^t\right\|_2 G)]\end{aligned}$$

where in (a), we expanded the definition of y^t and $\langle \mathbf{w}^t, \mathbf{x}^t\rangle$. In (b), we used the fact that ϵ^t is independent and has 0 mean to cancel the terms involving ϵ, and then simplified the remaining two terms using the hermite formula. Now, because $\|S_t(\mathbf{w}^t)\|_2 = 1$ ($\mathbf{w}^t$ is after normalization), the second term in the sum is equal to 0, and we get the expression in the formula in lemma 2.

Now, we furthermore simplify the population gradient. Notice that $\mathbb{E}_G[\sigma^{(D)}(\|\mathbf{w}^t\|_2 G)] = \mathbb{E}_G[\mathrm{He}_D(G)f(G)]$ where $f(G) = \sigma^{(D)}(\|\mathbf{w}^t\|_2 G)$. Then, by assumption 1, shifts and scales do not change the hermite coefficients of the activation function significantly (by choosing Δ small enough so that the scaling $\|\mathbf{w}^t\|$ is close to 1). Then, we have $\mathbb{E}_G[\mathrm{He}_D(G)f(G)] = \Theta(\mu_D(\sigma))$. Furthermore, by choosing Δ small enough (with respect to D) and using $(w_i)^2 \leq \Delta^2$, $k_j \leq D$ and $k_i \geq 1$ we can ensure that

$$D \geq k_i - (w_i^t)^2 \sum_{j\in S_t\cap[P]} k_j \geq 1 - \Delta^2 D > 0$$

Then, we can combine our expression for the population gradient (and the ones we omitted) to get lower and upper bounds on the gradient when $i \in [P]$. There exists constant $c, C > 0$ that only depend on D and K,such that for all $t < \tau_i^{\Delta} \wedge \tau^+ \wedge \tau^-$, for all $i \in [P]$, we have

$$ca^0\frac{\chi_*(\mathbf{w}^t)}{w_i^t} - |a^0|\kappa\mu_D(\sigma) \leq \overline{g}_i^t \leq Ca^0\frac{\chi_*(\mathbf{w}^t)}{w_i^t}\mu_D(\sigma) + |a^0|\tilde{\kappa}$$

where this follows as we have $\mathrm{sign}(w_i^t) = 1$ since $t < \tau^-$ (the w's were lower bounded below at all times, so they kept their initialization sign). Now, because we have a lower and upper bounds for the w_i^t (using $t < \tau^+ \wedge \tau^-$) that depend on d, we can choose $\tilde{\kappa} \lesssim (1/\sqrt{d})^{D-1}$ to absorb the $|a^0|\tilde{\kappa}$ term into the other terms. This is because the w_i^t are of order $\sqrt{d}$ and $\chi_*(\mathbf{w}^t)$ is of order $(w_i^t)^D$. Hence, for $i \in [P]$ and $t < \tau_i^{\Delta} \wedge \tau^+ \wedge \tau^-$, for some new $c, C > 0$ we have

$$ca^0\frac{\chi_*(\mathbf{w}^t)}{w_i^t}\mu_D(\sigma) \leq \overline{g}_i^t \leq Ca^0\frac{\chi_*(\mathbf{w}^t)}{w_i^t}\mu_D(\sigma)$$

Similarly, if we were to do the derivations in this section for $i > P$, we would get

$$|\overline{g_i^t}| \leq Ca^0\mu_D(\sigma)w_i^t\chi_*(\mathbf{w}^t)\left(\sum_{j\in S_t\cap[P]} k_j\right) + |a_0|\tilde{\kappa}$$

The main intuition here is that for $t \geq \tau^r + 1$ (if w_i^t is large for $i. > P$), and $i > P$, we have $S_t \cap [P] = \emptyset$, so that the magnitude of $\overline{g}_i^t$ will be bounded above by a $|a_0|\tilde{\kappa}$. On the other hand, if

$i \in [P]$ and w_i^t is still small ($t < t_i^{\Delta}$) then the gradients for these coordinates will be bounded away from 0. Then, the learning dynamics will be pushing the first P coordinates to be large, whereas the gradients will vanish for the rest of the coordinates. Hence, only the coordinates in the support will grow until they reach Δ, which is how the dynamics picks up only the support.

5.6 Final result

Now, we will explain (but not rigorously prove) how the results above contribute to Theorem 1. We have shown that the population gradients for the coordinates in the support (i.e. $i \in [P]$) will be positive, bounded below, while the gradients for the coordinates not in the support will be bounded above by a small constant when the weights exceed r, which is a small constant. If we were performing gradient descent using the population gradients (instead of the stochastic gradient), the coordinates in the support would grow, while the others would remain small.

However, we have stochastic gradients g_i^t that are estimates for the population gradients $\overline{g_i}^t$, which will give us a drift + martingale dynamics. The drift term comes from the population gradients, whereas the martingale term comes from summing terms that are ϵ^t multiplied by some term at each step, which has expectation 0. Because the spherical projection term effects g_i^t, we would need to bound its impact to be able to compare $\overline{g_i}^t$ to g_i^t. Hence, we have the following lemma that bounds the impact of the spherical projection.

Lemma 3. *Suppose $C_0 \log(d)^{C_0}/\sqrt{d} \leq r \leq \Delta/2 \leq 1/(8\sqrt{P})$ and $\eta_1 \leq d$. Then, we have constants $C, C' > 0/$ that only depend on D, K, C_0 such that for $d \geq C'$ and all $t < \tau^0 \wedge \tau^+$, if $S_{t+1} = S_t$*

$$\frac{1}{2} \leq 1 - C_1\eta_1^2 \left\|\mathbf{g}^t\right\|_2^2 \leq \frac{1}{\left\|S_{t+1}(\overline{\mathbf{w}}^{t+1})\right\|_2} \leq 1 + C_1\eta_1^2 \left\|\mathbf{g}^t\right\|_2^2$$

otherwise,

$$\frac{1}{2} \leq 1 - Cr^2 \leq \frac{1}{\left\|S_{t+1}(\overline{\mathbf{w}}^{t+1})\right\|} \leq 1 + Cr^2$$

This lemma is used to approximate the gradient dynamics with the spherical projection using dynamics without projection. Now, we will outline the rest of the proof. We will decompose the learning dynamics into a drift and martingale term. Specifically, for $i \in [P]$ and $t > \tau_i^r$, we have[2]

$$w_i^t = w_i^0 + \eta_1 D_i^{\tau_i^r,t} + \eta_1 M_i^{\tau^r,t} \tag{18}$$

where $D_i^{\tau_i^r,t}$ is a drift term that involves the sum of the population gradients over time, and $\eta_1 M_i^{\tau^r,t}$ is a martingale term that sums up the 'noises' that we generate at each step due to the stochasticity of the gradient. The key result is that the magnitude of the martingale term will grow with the square root of number of iterations, while the drift term grows linearly. Then, choosing $\eta_1 = o(1/\sqrt{\overline{T}_1})$, will make sure that the martingale term vanishes, while the drift term has unbounded contribution (in $\overline{T}_1$). Then, the rest of the proof is about bounding the effect of the martingale term. These rely on probabilistic tools (such as Doob maximal inequality for martingales) and bounding the specific distortions due to projections.

Finally, we need to prove the runtime of our algorithm, (i.e. the bound for $\overline{T}_1$ in theorem 1). We give a heuristic argument to why $\overline{T}_1$ will scale with d^D. Notice that the $\chi_*(\mathbf{w}^t)$ term scales with $1/d^{D/2}$

[2] We have similar expressions for $i > P$ or $t \leq \tau_i^r$

since the w_i^t scale with $1/\sqrt{d}$ (lower bound). Because the gradients scale with χ_*, we need that $\overline{T}_1$ scales with $1/(\eta_1\chi_*)$ to reach a total effect of Δ (i.e. $O(1)$). We furthermore want that η_1 is $o(1/\sqrt{\overline{T}_1})$, which gives us that $\sqrt{\overline{T}_1}$ has to scale with $d^{D/2}$, giving us the d^D runtime.

6 Extension to sum of monomials

Because of the way the analysis is structured, the proof for a single monomial extends nicely to the case where the target function is a sum of nested monomials. We can do this by conditioning on learning the support for the first P_l terms, and finding the runtime to learn the next terms. Specifically, now let h_* be a nested sum of hermite polynomials such that

$$h_*(\mathbf{z}) = \sum_{l=1}^{L} \prod_{i\in[P_l]} \mathrm{He}_{k_i}(z_i) \tag{19}$$

with $0 = P_0 < P_1 < \cdots < P_L$, and k_i are positive integers denoting the degrees of the hermite polynomials as before. Now, we have the same result as theorem 1, except $D = \mathrm{Leap}(h_*)$. Hence, the 'runtime' of our algorithm not scales with the leap complexity, instead of the degree of the target function. This shows that adding smaller order terms to the target function can make learning easier, as the training algorithm will pick up the support of the lower degree terms quickly, due to larger contribution coming from the $\chi_*(\mathbf{w}^t)$ term in the population gradient.

The reason the leap shows up in the runtime is as follows. Because lower order terms are learned more quickly than the higher order terms, when we get to a high order term, the w_i^t for the lower order terms that are already learned will be $\Theta(\Delta) = \Theta(r) = \Theta(1)$. Then, consider $\chi_{*,l}(\mathbf{w}^t) = \prod_{i\in[P_l]}(w_i^t)^{k_i}$. We can write

$$\chi_{*,l}(\mathbf{w}^t) = \prod_{i\in[P_{l-1}]}(w_i^t)^{k_i} \prod_{i\in[P_l]\setminus[P_{l-1}]}(w_i^t)^{k_i} \geq r^{\sum_{i\in[P_{l-1}]}k_i} \prod_{i\in[P_l]\setminus[P_{l-1}]}(w_i^t)^{k_i}$$

Because during the learning process ($t < \tau^-$), we have $w_i \geq \frac{1}{2\sqrt{d}}$, The rightmost product will give a $d^{-D_l/2}$ scaling, where D_l is the leap in the target function from P_{l-1} to P_l. Then, $\chi_{*,l}$ term contributing to the gradient scales with $d^{-\mathrm{Leap}(h_*)/2}$, instead of $d^{D/2}$ in the previous section. Furthermore choosing $\eta_1 = O(\frac{1}{d^{\mathrm{Leap}(h_*)/2}})$ as before gives the runtime of $d^{\mathrm{Leap}(h_*)}$.

7 Concluding Remarks

In this paper, we've seen how a full dynamic analysis can prove new insights into the learning trajectory. This analysis technique was also very useful in generalizing the result for the single monomial case to the sum of monomials case, since we can iteratively condition on learning the lowest degree monomials. Having a mix of lower degree and higher degree terms makes the learning of higher degree terms easier. This is because the algorithm can first learn the lower degree terms quickly, which makes learning the higher term degrees easier.

There is still a lot to explore in this direction. First, we haven't touched much on how the martingale term is analyzed in this project. Second, there are many caveats to the learning algorithm we used: We have strong assumptions on the activation σ that do not hold with common activations like ReLU, and we used projection terms that are not needed in practice. Finally, we have not looked at the training of the second layer, which has many caveats due to our restriction of the weights to a small Δ-ball.

References

[AAM23] Emmanuel Abbe, Enric Boix Adsera, and Theodor Misiakiewicz. Sgd learning on neural networks: leap complexity and saddle-to-saddle dynamics. In *The Thirty Sixth Annual Conference on Learning Theory*, pages 2552–2623. PMLR, 2023.

[ABAB+21] Emmanuel Abbe, Enric Boix-Adsera, Matthew S Brennan, Guy Bresler, and Dheeraj Nagaraj. The staircase property: How hierarchical structure can guide deep learning. *Advances in Neural Information Processing Systems*, 34:26989–27002, 2021.