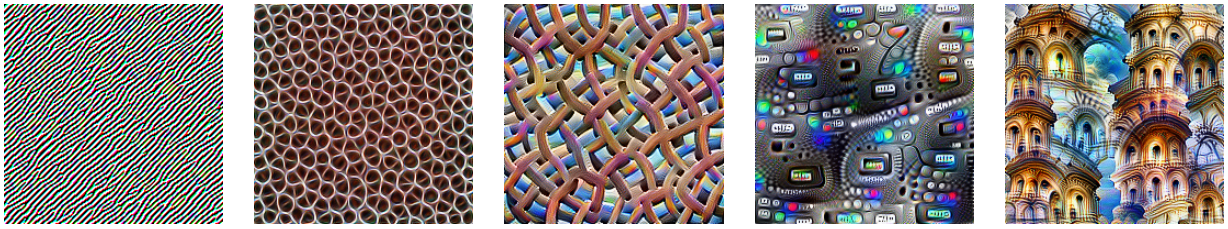
Introduction to Feature Learning

In the first part of the course, we studied how to scale neural networks with width and depth, and one requirement for a good scaling limit was that it allow *feature learning*. At the time, we defined feature learning as a change in the intermediate activations $\|\delta h\|_{\text{RMS}} = \Theta(1)$. This is only a *necessary* condition: when $\|\delta h\|_{\text{RMS}} = o(1)$, the network reduces to a kernel in the limit, which by definition uses a fixed feature map. However, the activations can change $\Theta(1)$ without the network actually learning anything useful. In this section, we'll try to understand the process by which a network extracts relevant features from a dataset, and what types of structure make a feature easy or hard for a neural network to learn.

What makes feature learning much harder to reason about than scaling limits is that we are forced to grapple with the structure of natural data. For scaling limits, we were able to argue that for any fixed sequence of data points, the trained network approached an infinite-width object (the NTK, a mean-field particle system, or a tensor program limit) *trained on the same sequence*. This tells us the finite width network will learn the same features as the infinite width network, but doesn't actually tell us what these features are, or how they depend on the architecture or dataset.

It's worth noting that "feature" has no standard definition. However, it is often used to refer to a function of the input that is computed and later built upon by the network. For example, a feature could be a single neuron in the middle of the network, $x \mapsto h_\ell(x)_i$, or, as is more common in the interpretability literature, a linear combination of neurons, $x \mapsto h_\ell(x) \cdot w$ for some $w \in \mathbb{R}^m$. Here are a few examples from trained networks:

Computer vision. Olah et al. (2017) visualize neurons in vision models by optimizing the input image x to maximally activate them. This process suggests that early layers appear to detect edges and textures, while deeper layers learn more complicated structures.



Edges Textures Patterns Parts Objects

Reinforcement learning. AlphaZero learned chess with no human game data, yet familiar chess concepts can be recovered from its activations (McGrath et al., 2022), along with patterns that look like computations of possible moves. For example, the feature below encodes the opponent's diagonal moves, shown on two different positions.

These are informal course notes and have not undergone peer review. AI tools assisted with simplifying proofs, editing, and typesetting. I take responsibility for the content, and corrections are welcome.





Language models. Templeton et al. (2024) find a direction in the residual stream ($x \mapsto h_\ell(x) \cdot w$) in Claude 3 Sonnet that responds to the Golden Gate Bridge across languages and images. Increasing it also makes the model talk about the bridge in unrelated conversations.



In all three settings, the features learned by the network clearly captured some structure in the dataset. However, we don't have any good mathematical models for such data, which is exactly why generative AI has moved towards neural networks: we don't have a good mathematical model for natural images, so we learn one using a diffusion model, and we don't have a good model for natural language, so we learn one using a transformer. In order to make progress, we will ignore natural data and instead study feature learning in sandbox settings where both the data distribution and the notion of a "feature" are precisely defined. We will then be able to answer questions like:

- Why is feature learning important? (i.e., when do neural networks outperform kernel methods?)
- How many samples does a neural network need to learn a good feature?
- What makes a feature easy or hard for a neural network to learn?

We will begin by analyzing these questions for single-index models before moving to the more interesting class of multi-index models.

Single-index models

Perhaps the simplest nontrivial model of feature learning is a *single-index model*:

$$f^\star(x) = \phi(w^\star \cdot x), \qquad x \in \mathbb{R}^d, \quad \|w^\star\| = 1,$$

where $\phi : \mathbb{R} \to \mathbb{R}$ is the *link function*. The target depends on x only through one unknown direction, so the feature is clear: the direction $w^\star$, or equivalently the map $x \mapsto w^\star \cdot x$.

Single-index models are natural for two layer networks $f(x) = a^\top \sigma(Wx + b)$: the first layer can represent the projection $w^\star \cdot x$ and the second layer can represent the nonlinearity, even when the link ϕ differs from the activation σ. Indeed, if a one-dimensional network $F(z) = \sum_j a_j \sigma(z + \beta_j)$ approximates ϕ, then $F(w^\star \cdot x)$ is a two layer network approximating $\phi(w^\star \cdot x)$.

The simplest example of a single-index model is when $\phi(z) = z$, in which case this reduces to linear regression: $y = w^\star \cdot x$. This already gives us a hint for what we should expect the sample complexity of learning a single index model to be. To recover $w^\star$ from n measurements (x_i, y_i), we just need to solve the linear system $\{x_i \cdot w = y_i\}_{i=1}^n$, which requires $n \geq d$ samples assuming the $\{x_i\}$ are in general position. It turns out that if we ignore computation, any single-index model, regardless of the link ϕ, can also be learned with $n = O(d)$ samples:

Theorem 1 (informal)**.** *Let $y = \phi(w^\star \cdot x) + \xi$, where ϕ, the zero-mean noise ξ, and the input distribution are all reasonably well-behaved. Without knowing ϕ or $w^\star$, we can learn $\phi(w^\star \cdot x)$ to any fixed accuracy from $n = O(d)$ samples.*[a]

[a] For example, it suffices that the inputs are sub-Gaussian and ϕ is Lipschitz. The $O(\cdot)$ hides constants depending on ϕ, ξ, and the accuracy, plus a $\log(1/\delta)$ term for success probability $1 - \delta$.

The estimator is essentially a brute force search over the feature $w^\star$ and the link ϕ. The brute force search over ϕ is easy, since it's a 1D function, but the search over $w^\star$ is hard, since it's a d-dimensional vector and searching over an ϵ-net of S^{d-1} requires $\exp(d)$ time.

Another way to phrase Theorem 1 is that if you view the empirical loss landscape as a function of $\hat{w}$, its global minimizer $\hat{w}_{\text{OPT}}$ is close to $w^\star$ (up to sign if ϕ is even) once $n \gtrsim d$. However, because the loss is not convex it's not always easy to find $w^\star$ from a random initialization. In practice, we initialize $\hat{w}$ randomly, so it has initial correlation $\hat{w} \cdot w^\star \asymp d^{-1/2}$, meaning that you initialize near the equator of this high dimensional sphere, and the hope is that the gradient provides enough signal to find the global minimizer $\hat{w}_{\text{OPT}}$, which is near $w^\star$. We will see that whether or not gradient descent succeeds will depend on both the complexity of ϕ and the number of samples n.

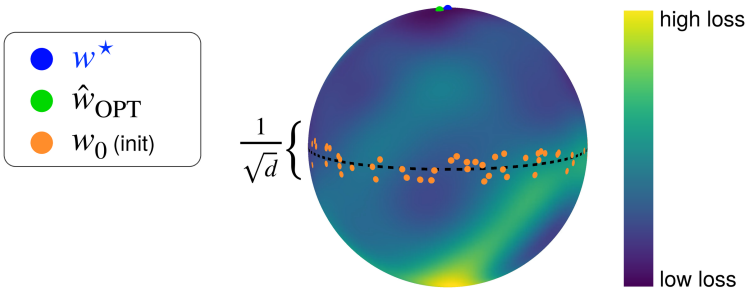
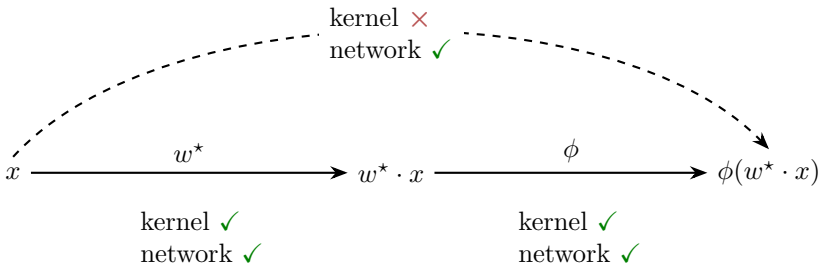


Figure 1: The single-index landscape.

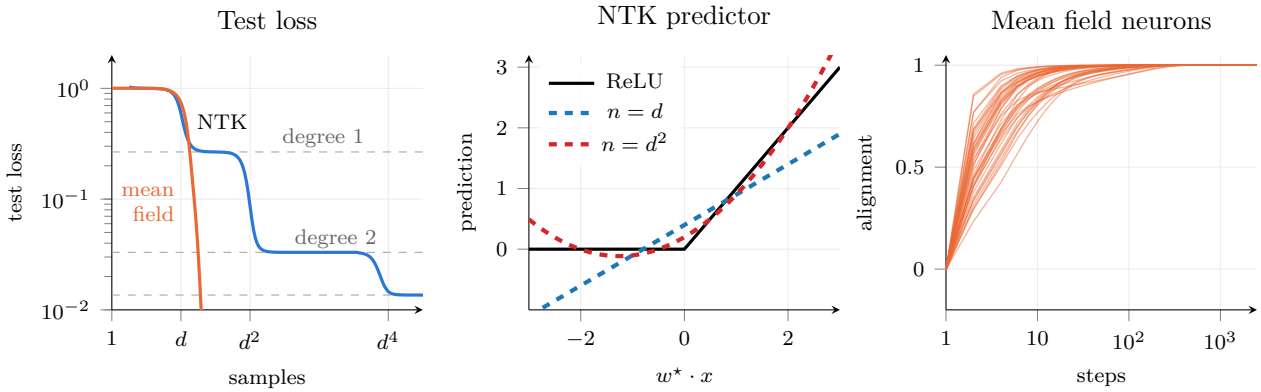
Kernel Lower Bounds

The strongest way to argue that a problem *requires* feature learning is to show that kernel methods must fail, and it turns out that single-index and multi-index models are essentially the "simplest" setting in which neural networks succeed but kernel methods provably fail.

It might be somewhat surprising that kernel methods struggle to learn single-index models. After all, a kernel can learn any linear map $x \mapsto w^\star \cdot x$ and it can learn any one-dimensional function ϕ. So why can't it learn the composition $\phi(w^\star \cdot x)$? The key is that kernel methods aren't closed under composition: just because a kernel can efficiently learn f and g does not mean it can learn $f \circ g$:



In this section we will focus on the limitations of kernels, and we will start with a simple experiment. Let $x \sim N(0, I_d)$, $w^\star \sim \mathrm{Unif}(S^{d-1})$ and $\phi = \mathrm{ReLU}$. We will compare the performance of a neural network in the mean-field regime with the performance of the NTK:



Note that the test loss for both the NTK and the mean field network drops around $n = d$, the linear regression threshold. However, while the loss for the mean field net immediately plummets to 0, the NTK plateaus until $n = d^2$, and then again until $n = d^4$. Turning to the middle panel, it turns out that with $n \approx d$ samples, the NTK just fits the best linear predictor to ReLU (blue) and with $n \approx d^2$ samples, it fits the best quadratic predictor (red). More generally, it turns out that if $\min(n, \#\text{parameters}) \approx d^p$, the NTK learns the best degree p polynomial approximation to ϕ:

Theorem 2 (informal, Ghorbani et al., 2021; Mei et al., 2022)**.** *Let $x \sim \mathrm{Unif}(S^{d-1}(\sqrt{d}))$ and $f^\star(x) = \phi(w^\star \cdot x)$. Fix an integer $p \geq 0$ and $\epsilon \in (0,1)$. Consider an inner-product kernel of rank N (e.g. random features or the NTK) trained on n samples with $\min\{n, N\} = d^{p+\epsilon}$. Then*

$$\widehat{f} \xrightarrow[d\to\infty]{L^2} P_{\leq p} f^\star,$$

where $P_{\leq p}$ is the projection onto polynomials of degree at most p.

In other words, with $d^p \ll \min(\#\text{samples}, \#\text{parameters}) \ll d^{p+1}$, the NTK just fits the best degree p polynomial to ϕ. We won't prove Theorem 2 in full generality, but we will prove that no kernel can perform better than the best degree p polynomial approximation whenever $\min(n, N) \ll d^{p+1}$.

The first step is to write out what the best degree p approximation to $f^\star$ is, for which we'll introduce some new machinery we'll use throughout the course: Hermite polynomials.

Hermite Polynomials

We measure approximation error in the Gaussian inner product and norm,

$$\langle f, g\rangle := \mathbb{E}[f(z)g(z)], \qquad \|f\| := \sqrt{\langle f, f\rangle}, \qquad z \sim N(0,1),$$

and write $L^2(N(0,1))$ for the functions with $\|f\| < \infty$. The Hermite polynomials are the orthogonal polynomials with respect to this inner product:

Definition 1 (Hermite polynomials)**.** The normalized Hermite polynomials $h_0, h_1, \ldots$ are obtained by applying Gram–Schmidt to $1, z, z^2, \ldots$ in this inner product:

$$h_0(z) = 1, \qquad h_1(z) = z, \qquad h_2(z) = \frac{z^2 - 1}{\sqrt{2}}, \qquad h_3(z) = \frac{z^3 - 3z}{\sqrt{6}}, \qquad \ldots$$

These polynomials form an orthonormal basis, so every function in $L^2(N(0,1))$ can be expanded as an infinite sum of Hermite polynomials:

Proposition 1 (Hermite expansion)**.** *Every $\phi \in L^2(N(0,1))$ has a Hermite expansion:*

$$\phi \overset{L^2}{=} \sum_{k\geq 0} c_k h_k, \qquad c_k = \langle \phi, h_k\rangle = \mathbb{E}[\phi(z)h_k(z)].$$

If also $\tau = \sum_k b_k h_k$, then $\langle \phi, \tau\rangle = \sum_k c_k b_k$, and in particular $\|\phi\|^2 = \sum_k c_k^2$ (Parseval).

Corollary 1. *The best degree p approximation of ϕ is $\sum_{k=0}^p c_k h_k$, with error*

$$\|P_{>p}\phi\|^2 = \sum_{k>p} c_k^2, \qquad \text{where } P_{>p}\phi := \sum_{k>p} c_k h_k.$$

Proof. By Parseval, $\|\phi - \sum_{k=0}^p a_k h_k\|^2 = \sum_{k=0}^p (c_k - a_k)^2 + \sum_{k>p} c_k^2$, minimized at $a_k = c_k$. □

Example: the ReLU plateaus. For $\phi(z) = \max\{0, z\}$, the first Hermite coefficients are

$$c_0 = \frac{1}{\sqrt{2\pi}}, \qquad c_1 = \frac{1}{2}, \qquad c_2 = \frac{1}{2\sqrt{\pi}}, \qquad c_3 = 0, \qquad c_4 = -\frac{1}{4\sqrt{3\pi}}.$$

Every odd coefficient beyond c_1 vanishes, because $\mathrm{ReLU}(z) - z/2 = |z|/2$ is even. Dividing the errors $\sum_{k>p} c_k^2$ by $\mathrm{Var}(\phi(z)) = 1/2 - 1/(2\pi)$ gives approximately $0.267, 0.033, 0.033, 0.014$ for $p = 1, 2, 3, 4$. These are the dashed lines in the NTK staircase figure, and $c_3 = 0$ is why there is no drop at $n = d^3$.

Hermite polynomials are the right tool because they behave nicely under correlated Gaussians:

Proposition 2. *If X, Y are jointly Gaussian with mean 0, variance 1 and correlation ρ, then*

$$\mathbb{E}[h_j(X)h_k(Y)] = \mathbf{1}_{j=k}\rho^k.$$

Proof. By symmetry, suppose $j \le k$. Write $X = \rho Y + \sqrt{1-\rho^2}Z$, where Z is an independent standard Gaussian. Averaging $h_j(X)$ over Z gives a polynomial in Y of degree j, whose leading coefficient is that of h_j multiplied by ρ^j. Thus

$$\mathbb{E}[h_j(X) \mid Y] = \rho^j h_j(Y) + r(Y), \qquad \deg r < j.$$

Multiplying by $h_k(Y)$ and averaging kills r (since $\deg r < j \le k$) and gives $\rho^j \mathbf{1}_{j=k}$. □

For unit vectors w, v and $x \sim N(0, I_d)$, $w \cdot x$ and $v \cdot x$ are standard Gaussians with correlation $w \cdot v$, so expanding both factors gives the identity we will use throughout.

Corollary 2. *If $\|w\| = \|v\| = 1$, $x \sim N(0, I_d)$, $\phi = \sum_k c_k h_k$ and $\tau = \sum_k b_k h_k$, then*

$$\mathbb{E}_x[\phi(w \cdot x)\tau(v \cdot x)] = \sum_{k \ge 0} c_k b_k \, (w \cdot v)^k. \tag{1}$$

The Kernel Lower Bound

The key property of kernels that will allow us to prove a strong lower bound is that the predictor $\widehat{f}$ must lie in a fixed subspace $F \subset L^2$ of dimension at most $\min\{n, N\}$.

To see this, write the kernel as a linear model $f(x) = a \cdot \psi(x)$ on a fixed feature map $\psi : \mathbb{R}^d \to \mathbb{R}^N$, so that $K(x, x') = \langle \psi(x), \psi(x') \rangle$. Recall that the gradient of the training loss is

$$\sum_{i=1}^{n} \ell'(f(x_i), y_i)\, \psi(x_i) \in S := \mathrm{span}\{\psi(x_1), \dots, \psi(x_n)\}.$$

So starting from $a = 0$, any number of (S)GD steps keeps $a \in S$, and the predictor lies in

$$F := \{x \mapsto a \cdot \psi(x) : a \in S\}, \qquad \dim F \le \dim S \le \min\{n, N\},$$

since S is spanned by n vectors in $\mathbb{R}^N$. For a width m neural network this covers random features ($N = m$) and the NTK ($N \asymp md$). The key point is that F depends on the inputs but not the labels: the kernel and any feature randomness are independent of $w^\star$, and the labels only pick $a \in S$. The best the kernel can do is therefore to project the target $f_w(x) = \phi(w \cdot x)$ onto F, with error

$$\min_{\widehat{f} \in F} \|f_w - \widehat{f}\|^2 = \|P_{F^\perp} f_w\|^2 = \|f_w\|^2 - \|P_F f_w\|^2.$$

Our goal is to upper bound the second term $\|P_F f_w\|^2$ by showing that, for a random direction w, the target f_w occupies too many "effective dimensions" to be captured by F. We measure this by how much two independent copies of f_w overlap.

Definition 2 (Participation ratio)**.** For a nonzero random vector V in an inner product space with $\mathbb{E}\|V\|^2 < \infty$ and an independent copy V', define

$$D_{\mathrm{PR}}(V) := \frac{(\mathbb{E}\|V\|^2)^2}{\mathbb{E}[\langle V, V'\rangle^2]} = \frac{(\operatorname{tr}\Sigma)^2}{\|\Sigma\|_{\mathrm{F}}^2}, \qquad \Sigma := \mathbb{E}[VV^\top].$$

For example, $V \sim \mathrm{Unif}(S^{D-1})$ has $\mathbb{E}[\langle V, V'\rangle^2] = 1/D$, so $D_{\mathrm{PR}}(V) = D$. To capture a constant fraction of a random target, a fixed subspace needs $\Omega(D)$ dimensions:

Lemma 1. *Let $F \subset L^2$ be a fixed finite-dimensional subspace and $f_w \in L^2$ a random target with participation ratio $D = D_{\mathrm{PR}}(f_w)$. Then*

$$\mathbb{E}_w\|P_F f_w\|^2 \le \mathbb{E}_w\|f_w\|^2 \sqrt{\frac{\dim F}{D}}.$$

Proof. By rescaling, assume $\mathbb{E}_w\|f_w\|^2 = 1$, and let $\Sigma = \mathbb{E}_w[f_w f_w^\top]$. Cauchy–Schwarz gives[1]

$$\mathbb{E}_w\|P_F f_w\|^2 = \operatorname{tr}(P_F\Sigma) \le \|P_F\|_{\mathrm{F}}\|\Sigma\|_{\mathrm{F}} = \sqrt{\frac{\dim F}{D}},$$

since $\|P_F\|_{\mathrm{F}}^2 = \dim F$ and $\|\Sigma\|_{\mathrm{F}}^2 = 1/D$. □

Therefore, to show that kernels can't learn a single-index model, it suffices to show that the high-degree part of f_w, $P_{>p}f_w$, has large participation ratio, where:

$$P_{>p}f_w(x) = \sum_{k>p} c_k h_k(w\cdot x), \qquad \|P_{>p}f_w\|^2 = \|P_{>p}\phi\|^2 = \sum_{k>p} c_k^2.$$

For an independent direction $v \sim \mathrm{Unif}(S^{d-1})$, Corollary 2 and $|w\cdot v| \le 1$ give

$$|\langle P_{>p}f_w, P_{>p}f_v\rangle| = \left|\sum_{k>p} c_k^2 (w\cdot v)^k\right| \le \|P_{>p}\phi\|^2 |w\cdot v|^{p+1}.$$

Squaring and averaging, using that $\mathbb{E}[(w\cdot v)^{2m}] = \Theta_m(d^{-m})$, gives

$$\mathbb{E}_{w,v}[\langle P_{>p}f_w, P_{>p}f_v\rangle^2] \le \|P_{>p}\phi\|^4 \underbrace{\mathbb{E}[(w\cdot v)^{2p+2}]}_{\Theta_p(d^{-(p+1)})},$$

so $D_{\mathrm{PR}}(P_{>p}f_w) = \Omega_p(d^{p+1})$. Dropping the low-degree part can only decrease the error, $\|\widehat{f} - f_w\| \ge \|P_{>p}\widehat{f} - P_{>p}f_w\|$, and $P_{>p}\widehat{f}$ lies in $P_{>p}F$, which has dimension at most $\dim F$. Since $\dim F \le \min\{n, N\}$, combining Lemma 1 with the error formula above gives

$$\mathbb{E}\|\widehat{f} - f^\star\|^2 \ge \|P_{>p}\phi\|^2 \left(1 - C_p\sqrt{\frac{\min\{n,N\}}{d^{p+1}}}\right).$$

[1] For functions in L^2, $\|\cdot\|_{\mathrm{F}}$ denotes the Hilbert–Schmidt norm, the analogue of the Frobenius norm.

So if $\min\{n, N\} \ll d^{p+1}$, the expected error is at least that of the best degree p polynomial.

A Single Neuron Beats Kernels

We will now show that neural networks can learn single-index models with $O(d)$ samples even when the NTK fails. The simplest case is when the link ϕ is monotone, in which case a single neuron can learn the direction $w^\star$ with $n \geq d$ samples. Specifically, let:

$$f_w(x) = \sigma(w \cdot x), \qquad y_i = \sigma(w^\star \cdot x_i).$$

And assume σ is strictly monotone with slope at least $\gamma > 0$:

$$(\sigma(a) - \sigma(b))(a - b) \geq \gamma(a - b)^2 \quad \text{for all } a, b. \tag{2}$$

For example, the leaky ReLU $\sigma(z) = \max\{z, \gamma z\}$ satisfies this. Monotonicity is mild for the learner's activation, since we can pick a leaky ReLU, but strong for the target.

The inputs $x_1, \ldots, x_n \in \mathbb{R}^d$ are arbitrary, and we run gradient flow on the empirical square loss:

$$\frac{dw_t}{dt} = -\nabla L(w_t), \qquad L(w) = \frac{1}{2n}\sum_{i=1}^n (\sigma(w \cdot x_i) - y_i)^2.$$

How many samples should we expect? Recall that when $\sigma(z) = z$, this reduces to linear regression and $w^\star$ is identifiable whenever the data points span $\mathbb{R}^d$ (i.e. $n \geq d$ with the $\{x_i\}_{i=1}^n$ in general position). The same condition turns out to be sufficient to learn any monotone link.

Theorem 3. *Suppose σ satisfies (2), and that $\lambda := \lambda_{\min}(\widehat{\Sigma}) > 0$ where $\widehat{\Sigma} = \frac{1}{n}\sum_{i=1}^n x_i x_i^T$, i.e. the data points span $\mathbb{R}^d$. Then gradient flow from any initialization satisfies*

$$\|w_t - w^\star\|^2 \leq e^{-2\gamma^2\lambda t}\|w_0 - w^\star\|^2.$$

The challenge in proving Theorem 3 is that the loss is not convex. However, we will show that the negative gradient is always positively aligned with $w^\star - w$, a condition known as *one-point convexity*, and this is sufficient to show that $w \to w^\star$.

Proof. By the chain rule, $\ell' = \sigma(w \cdot x_i) - y_i$ and $\nabla_w f_w(x_i) = \sigma'(w \cdot x_i)x_i$, so by (2),

$$\langle -\nabla L(w), w^\star - w\rangle = \frac{1}{n}\sum_{i=1}^n \underbrace{\sigma'(w \cdot x_i)}_{\geq \gamma} \underbrace{(\sigma(w^\star \cdot x_i) - \sigma(w \cdot x_i))(w^\star \cdot x_i - w \cdot x_i)}_{\geq \gamma(w^\star \cdot x_i - w \cdot x_i)^2},$$

Therefore, using that $\hat{\Sigma} = \frac{1}{n}\sum_{i=1}^n x_i x_i^\top$ has minimal eigenvalue λ,

$$\langle -\nabla L(w), w^\star - w\rangle \geq \frac{\gamma^2}{n}\sum_{i=1}^n ((w^\star - w) \cdot x_i)^2 = \gamma^2(w^\star - w)^\top \widehat{\Sigma}(w^\star - w) \geq \gamma^2\lambda\|w - w^\star\|^2. \tag{3}$$

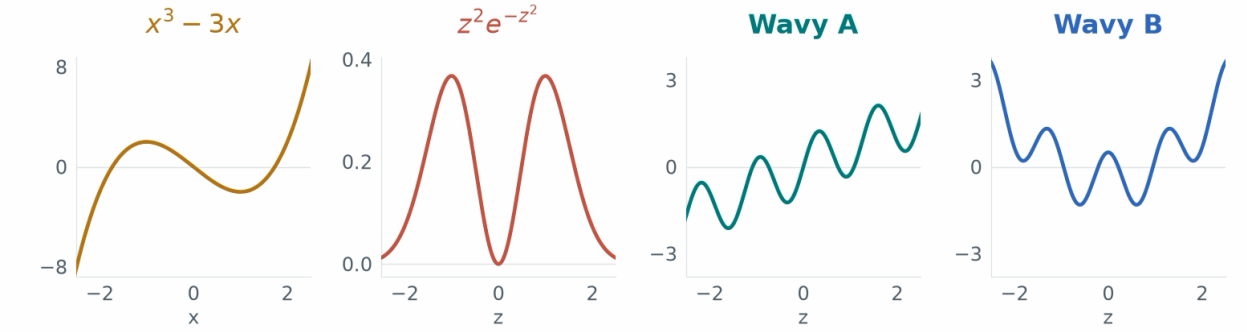
So along gradient flow,

$$\frac{d}{dt}\|w_t - w^\star\|^2 = 2\left\langle \frac{dw_t}{dt}, w_t - w^\star\right\rangle = -2\langle -\nabla L(w_t), w^\star - w_t\rangle \leq -2\gamma^2\lambda\|w_t - w^\star\|^2.$$

Integrating gives the result. □

General Link Functions

Consider the four non-monotone links below. Which one do you think will be learned first? The cubic and the double hump look benign, and the two wavy functions look more complicated.



It turns out that the neural network learns Wavy A first, then Wavy B, then the cubic, and finally the double hump, and the double hump takes quite a long time to learn:

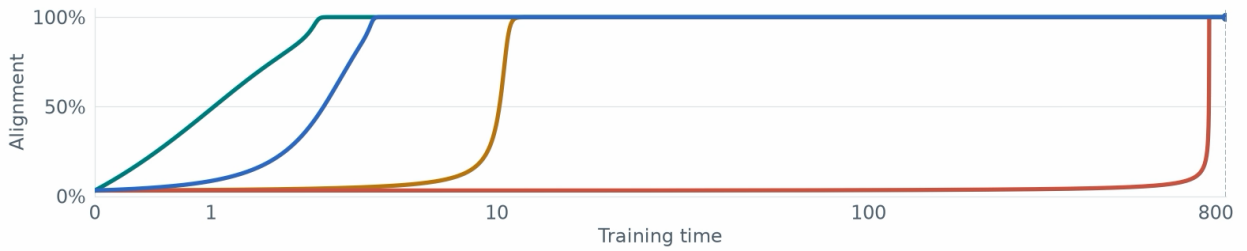


Figure 2: Alignment $\rho = w \cdot w^\star$ during training.

Why is that? What's so special about the double hump function that makes it so hard and why are the wavy functions, which a priori look more complicated, actually much easier for the network to learn? The key is the correlation of each of these functions with low-degree polynomials:

- Wavy A is reasonably well-approximated by a linear function, so the network learns the feature $x \mapsto w^\star \cdot x$ and then reuses it to fit the rest of the wavy function.
- Wavy B is symmetric, so it can't be represented by a linear function, but it is still reasonably well-approximated by a quadratic, so the network learns $x \mapsto w^\star \cdot x \mapsto (w^\star \cdot x)^2$, and then reuses the intermediate feature $x \mapsto w^\star \cdot x$ to fit the rest of the function.
- The cubic is proportional to h_3, the third Hermite polynomial, which by definition is orthogonal to all polynomials of degree at most 2. So the network can't learn any low-degree polynomial approximation to the cubic, and it takes longer to learn.
- The double hump was carefully designed to be orthogonal to h_1, h_2, h_3, so the network can't learn any cubic approximation to the double hump, and it takes even longer to learn.

We will now describe why this polynomial approximation viewpoint is relevant and how this manifests in the loss landscape and training dynamics.

The population loss

For simplicity, we will again consider a single neuron whose activation matches the link, $\phi = \sigma$.[2] With Gaussian inputs $x \sim N(0, I_d)$ and $\|w\| = \|w^\star\| = 1$, the population loss is

$$L(w) = \frac{1}{2}\mathbb{E}_x\left[\sigma(w \cdot x) - \sigma(w^\star \cdot x)\right]^2.$$

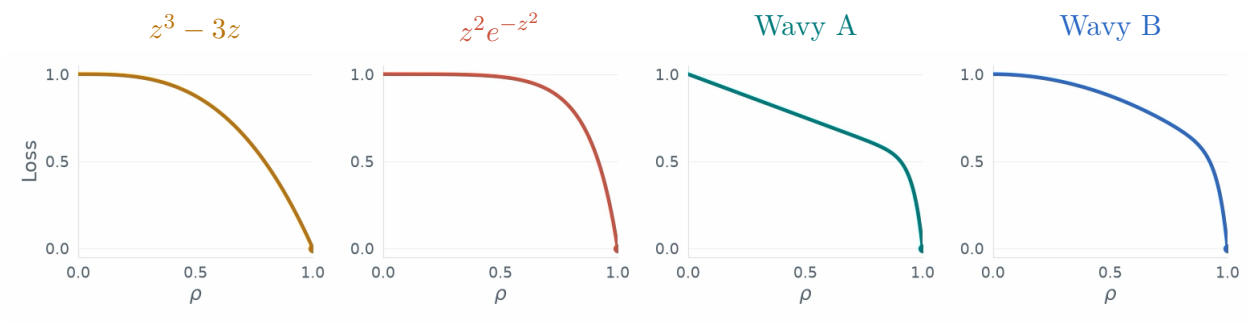
Here $w \cdot x$ and $w^\star \cdot x$ are standard Gaussians with correlation $\rho := w \cdot w^\star$, the *overlap*. Expanding the square, each squared term gives $\|\sigma\|^2 = \sum_k c_k^2$, and Corollary 2 gives the cross term:

$$L(w) = \frac{1}{2}\sum_k c_k^2 + \frac{1}{2}\sum_k c_k^2 - \sum_k c_k^2 \rho^k.$$

So the loss depends on w only through ρ:

$$L(\rho) = \sum_{k \geq 1} c_k^2 (1 - \rho^k). \tag{4}$$

We normalize $\sum_{k \geq 1} c_k^2 = 1$, so that $L(0) = 1$. We plot $L(\rho)$ for our four examples below, with the first nonzero term of (4) underneath:



$1 - \rho^3$ $\quad$ $1 - 0.16\rho^4 + O(\rho^6)$ $\quad$ $1 - \frac{1}{2}\rho + O(\rho^2)$ $\quad$ $1 - \frac{1}{2}\rho^2 + O(\rho^4)$

Near $\rho = 0$ the loss is dominated by its first nonzero term, $L \approx 1 - c_k^2 \rho^k$. For Wavy A this is a straight line, and for Wavy B a saddle $1 - c_2^2 \rho^2$ that is easy to escape. The cubic $1 - c_3^2 \rho^3$ is very flat near zero, and the double hump is basically flat until $\rho \approx 0.5$.

In general, the flatness of the loss landscape near $\rho = 0$ is determined by the index of the first nonzero Hermite coefficient:

Definition 3 (Information exponent, Ben Arous et al., 2021)**.** The *information exponent* of σ is the index of its first nonzero Hermite coefficient after the constant,

$$k^\star = \min\{k \geq 1 : c_k \neq 0\}, \quad c_k = \langle \sigma, h_k \rangle .$$

Equivalently, $k^\star$ is the smallest degree of a non-constant polynomial correlated with σ.

The higher the information exponent, the flatter the loss landscape is near $\rho = 0$.

[2] If the activation σ differs from the link ϕ, the same calculation goes through with c_k^2 replaced by $c_k b_k$, where c_k and b_k are the Hermite coefficients of ϕ and σ.

The population gradient. We keep w on the unit sphere, so ∇_w is the gradient on the sphere.[3] Since $\nabla_w \rho = w^\star - \rho w$, the chain rule gives

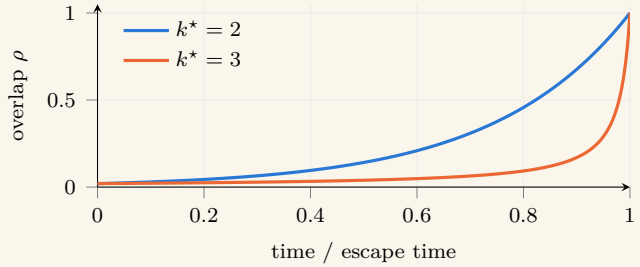
$$-\nabla_w L(w) = (w^\star - \rho w) \sum_{k \geq 1} k c_k^2 \rho^{k-1}. \tag{5}$$

Thus, when $\rho > 0$ the population gradient always points towards $w^\star$, but it may be vanishingly small at initialization. Recall that at initialization $|\rho| = |w \cdot w^\star| = \Theta(1/\sqrt{d})$. The sum in (5) is then dominated by its first nonzero term $k^\star c_{k^\star}^2 \rho^{k^\star - 1}$, so the size of the gradient at initialization is

$$\|\nabla_w L(w)\| = \Theta\left(d^{-(k^\star-1)/2}\right).$$

Once ρ is of order one, the gradient is of order one for every $k^\star$, so all the difficulty is in leaving the equator.

Why the loss drops suddenly. Near the equator the loss looks like $1 - c\rho^{k^\star}$, so gradient flow gives $\dot{\rho} \approx c\rho^{k^\star - 1}$. For $k^\star = 2$ this is exponential growth, $\rho_0 e^{ct}$. For $k^\star = 3$ it is $\rho_0/(1 - c\rho_0 t)$, which blows up at time $1/(c\rho_0)$. From $\rho_0 \asymp d^{-1/2}$, almost nothing happens for a long time and then ρ jumps. This is why the alignment curves jump so abruptly.



For Wavy A we can see $k^\star = 1$ just by drawing a line through it. For our examples,

$\sigma(z)$	z	$z^3 - 3z$	$z^2 e^{-z^2}$	Wavy A	Wavy B
$k^\star$	1	3	4	1	2

Monotone links have $k^\star = 1$. If σ satisfies (2), then $(\sigma(z) - \sigma(0))z \geq \gamma z^2$, so

$$c_1 = \mathbb{E}[\sigma(z)z] = \mathbb{E}[(\sigma(z) - \sigma(0))z] \geq \gamma \mathbb{E}[z^2] = \gamma > 0.$$

From gradients to samples. How many samples do we need for the empirical gradient to point in the right direction? At initialization the population gradient along $w^\star$ has size $\Theta(d^{-(k^\star-1)/2})$, and with n samples the empirical gradient fluctuates around it by $\Theta(n^{-1/2})$. For the signal to beat the fluctuation we need

$$n^{-1/2} \lesssim d^{-(k^\star-1)/2} \qquad \Longleftrightarrow \qquad n \gtrsim d^{k^\star - 1}.$$

Theorem 4 formalizes this heuristic by showing that $n \gtrsim d^{\max(1, k^\star - 1)}$ is necessary and sufficient for online SGD to learn a single-index model with information exponent $k^\star$:

[3] This is the ordinary gradient with its component along w removed (the Riemannian gradient).

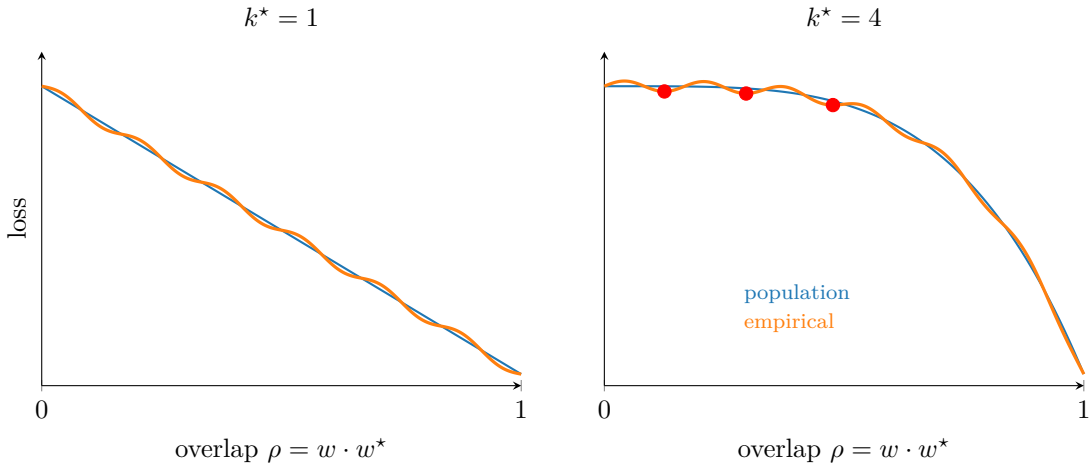
Theorem 4 (informal, Ben Arous et al., 2021; Damian et al., 2023)**.** *Let $x \sim N(0, I_d)$ and $y = f^\star(x) = \sigma(w^\star \cdot x)$, where σ has information exponent $k^\star$. Train $f_w(x) = \sigma(w \cdot x)$ by online SGD on the sphere, from an initialization with $\rho_0 \gtrsim d^{-1/2}$. Then after*

$$T \gtrsim T_\star(d) + \frac{d}{\varepsilon^2}, \qquad T_\star(d) = \begin{cases} d, & k^\star = 1, \\ d \log d, & k^\star = 2, \\ d^{k^\star - 1}, & k^\star \geq 3, \end{cases}$$

steps (and fresh samples), $\|f_{w_T} - f^\star\| \leq \varepsilon$ with high probability, up to logarithmic factors.

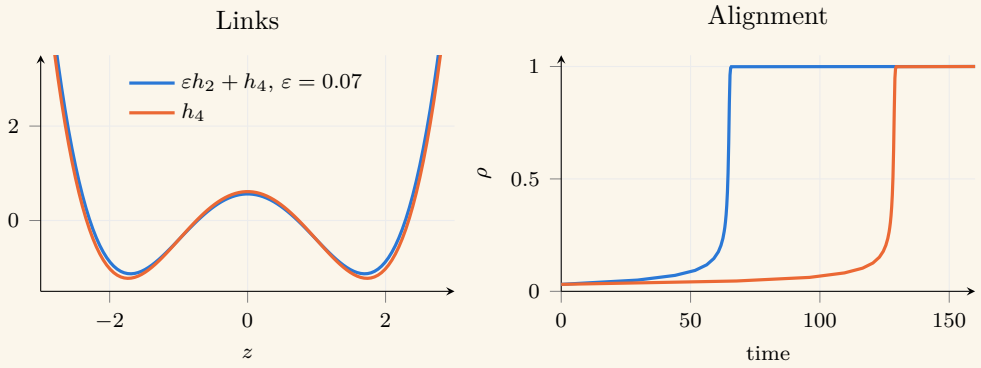
On the sphere of Figure 1, $T_\star(d)$ is the time to **escape the equator**, and d/ε^2 is the time to converge near the north pole (Damian et al., 2023). The proof technique for Theorem 4 is essentially that the population loss is so flat near the equator that it takes many steps to escape, and for online SGD the number of steps is equal to the number of samples.

However, a similar picture emerges if we allow the learner to reuse samples. Given n samples, we can view the empirical loss as a perturbed version of the population loss, where the size of the perturbation shrinks as n grows. For a rapidly decaying loss, these fluctuations can be benign, while for an extremely flat loss they can create spurious local minima that trap gradient descent near the equator. This is illustrated in the following cartoon in which the red dots denote local minima:



By analogy with related results in tensor PCA (Ben Arous et al., 2019, 2020), the critical threshold for n past which these local minima near the equator disappear is given by $n \gtrsim d^{k^\star - 1}$.

The information exponent is not robust. Adding a tiny quadratic part to $\phi = h_4$ gives $\phi_\varepsilon = \varepsilon h_2 + h_4$ which has $k^\star = 2$, and gradient descent escapes the equator much faster:



The General Picture: Kernels vs Neural Networks

Both a kernel and a neural network learn from the lowest-degree part of $f^\star$ first, say of degree k.[4] The difference is what happens next. The kernel starts from scratch on the next degree, which costs d^{k+1} samples, and then again on the next degree, which costs d^{k+2} samples, and so on, while the network uses the fact that it has already found $w^\star$ to compute the feature map $x \mapsto w^\star \cdot x$ and then just has to fit the remaining 1D function $z \mapsto \phi(z)$, which it can do with $O(1)$ time and samples. In other words, the network reuses $w^\star$ once it has extracted it from the lowest-degree signal, while a kernel pays d^k samples separately for every degree k.

[4] The kernel needs $n = d^k$ samples to learn from the degree k part, while the network only needs $n = d^{\max(1,k-1)}$.

Multi-Index Models

So far the target depended on a single direction $w^\star$. We will now ask what happens when the target depends on *multiple directions*, so that we have multiple features to learn:

$$y = \phi(w_1^\star \cdot x, \dots, w_r^\star \cdot x)$$

where $w_1^\star, \dots, w_r^\star$ are orthonormal vectors. The corresponding feature maps are $x \mapsto w_i^\star \cdot x$ for $i = 1, \dots, r$.

Juntas on the Boolean cube

It turns out that multi-index models are easiest to study on the Boolean cube: $x \sim \text{Unif}\{\pm 1\}^d$. We will primarily focus on this setting and only briefly discuss Gaussian multi-index models at the end.

The same way every function over Gaussians has a Hermite expansion, every function over the Boolean cube has a Fourier expansion:

$$f(x) = \sum_{S \subseteq [d]} \widehat{f}(S)\chi_S(x), \qquad \chi_S(x) = \prod_{i \in S} x_i, \qquad \widehat{f}(S) = \mathbb{E}_x[f(x)\chi_S(x)]. \tag{6}$$

The monomials χ_S are called *parities*, since they compute the parity of $\{x_i\}_{i \in S}$. They are orthonormal and they play the role that Hermite polynomials played for Gaussians. The *degree* of χ_S is $|S|$.

A *junta* is a target that depends on only r of the d coordinates, $f^\star = \sum_{S \subseteq S^\star} \widehat{f^\star}(S)\chi_S$ with $|S^\star| = r$. We keep r and the coefficients fixed and let $d \to \infty$. The features are the relevant coordinates: we write $x_1x_2x_3$, but for all the network knows the target is $x_{307}x_{472}x_{1056}$.

Statistically this is easy. A finite function class $\mathcal{F}$ can be learned with about $\log|\mathcal{F}|$ samples, and for a ± 1-valued target there are at most $d^r 2^{2^r}$ candidates, so $O_r(\log d)$ samples suffice. The question is computational: which juntas are hard to learn? The natural guess is the degree. This is the right answer for kernels, but not for networks.

Theorem 5 (informal, Mei et al., 2022; Abbe et al., 2022)**.** *Let $x \sim \text{Unif}\{\pm 1\}^d$ and let $f^\star$ be a junta on a uniformly random set of r coordinates. Consider an inner-product kernel of rank N (e.g. random features or the NTK) trained on n samples with $d^p \ll \min\{n, N\} \ll d^{p+1}$. Then*

$$\widehat{f} \xrightarrow[d \to \infty]{L^2} P_{\le p} f^\star,$$

where $P_{\le p}$ keeps the parities of degree at most p.

The lower bound is the same argument as for Theorem 2. For a uniformly chosen S, the parities are orthonormal, so $\mathbb{E}[\langle \chi_S, \chi_{S'} \rangle^2] = \Pr(S = S') = 1/\binom{d}{k}$ and $D_{\text{PR}}(\chi_S) = \binom{d}{k} \asymp d^k$. By Lemma 1, a kernel captures a vanishing fraction of every degree $k > p$ part.

Parities are hard. The hardest junta of a given size is a single parity $\chi_{S^\star}$ with $|S^\star| = k$. Without noise, Gaussian elimination over $\mathbb{F}_2$ finds $S^\star$ from about d samples, but this trick breaks as soon as we flip each label independently with probability $\eta < 1/2$. With noise, the best known algorithms take time d^{ck} for a constant $c < 1$ (Valiant, 2015), and it is conjectured that the exponent must grow linearly in k. We will see that neural networks need about d^{k-1} samples.

Following one neuron

Train $f_\theta(x) = \sum_j a_j \sigma(w_j \cdot x)$ on the square loss,

$$L(\theta) = \frac{1}{2}\mathbb{E}[y^2] - \mathbb{E}[y f_\theta(x)] + \frac{1}{2}\mathbb{E}[f_\theta(x)^2].$$

The first term is constant. The last term contains no labels, so it cannot point the weights toward the relevant coordinates, and we drop it for the purpose of finding features. The middle term is a sum over neurons, so each neuron w sees the correlation

$$C(w) = \mathbb{E}[f^\star(x)\sigma(w \cdot x)]. \tag{7}$$

Label noise only rescales C by $1 - 2\eta$, so we ignore it from now on. For single-index models, we computed this by Hermite expanding $f^\star$ and $\sigma(w \cdot x)$, and we will now do the same calculation using the Fourier basis. The full calculation for the Fourier expansion of $\sigma(w \cdot x)$ is fairly involved, and the following simple heuristic, and much of the staircase discussion below, follow Misiakiewicz (2025):

Heuristic 1. *Let $x \sim \mathrm{Unif}\{\pm 1\}^d$, $\|w\|_2 = 1$, and let $\sigma = \sum_k c_k h_k$ be smooth. When all the w_i are small, the Fourier coefficients of $\sigma(w \cdot x)$ are*

$$\mathbb{E}_x[\sigma(w \cdot x)\chi_S(x)] \approx \sqrt{|S|!}\, c_{|S|} \prod_{i \in S} w_i. \tag{8}$$

Proof sketch. Write $w \cdot x = Z + w_1 x_1$ with Z independent of x_1. Averaging over x_1 gives

$$\mathbb{E}[x_1 \sigma(w \cdot x)] = \frac{1}{2}\mathbb{E}[\sigma(Z + w_1) - \sigma(Z - w_1)] \approx w_1 \mathbb{E}[\sigma'(Z)].$$

Doing this for each $i \in S$ gives $(\prod_{i \in S} w_i)\mathbb{E}[\sigma^{(q)}(Z)]$ with $q = |S|$. The sum Z of the remaining coordinates is approximately a standard Gaussian G by the central limit theorem, and Gaussian integration by parts gives $\mathbb{E}[\sigma^{(q)}(G)] = \sqrt{q!}\, c_q$. □

Thus, to get the correlation up to constants, we can just take the Fourier expansion of the target and replace every x_i by w_i:

$$C(w) \approx \sum_S \widehat{f^\star}(S)\sqrt{|S|!}\, c_{|S|} \prod_{i \in S} w_i. \tag{9}$$

In particular the gradient lies (up to small corrections) in the span of the relevant coordinates.

Parities. For $f^\star = x_1 \cdots x_k$, $C(w) \approx \sqrt{k!}\, c_k\, w_1 \cdots w_k$, so

$$\nabla C(w) \approx \sqrt{k!}\, c_k \begin{bmatrix} w_2 w_3 \cdots w_k \\ w_1 w_3 \cdots w_k \\ \vdots \\ w_1 w_2 \cdots w_{k-1} \\ 0 \\ \vdots \end{bmatrix} \overset{\text{init}}{\approx} \begin{bmatrix} d^{-(k-1)/2} \\ d^{-(k-1)/2} \\ \vdots \\ d^{-(k-1)/2} \\ 0 \\ \vdots \end{bmatrix}. \tag{10}$$

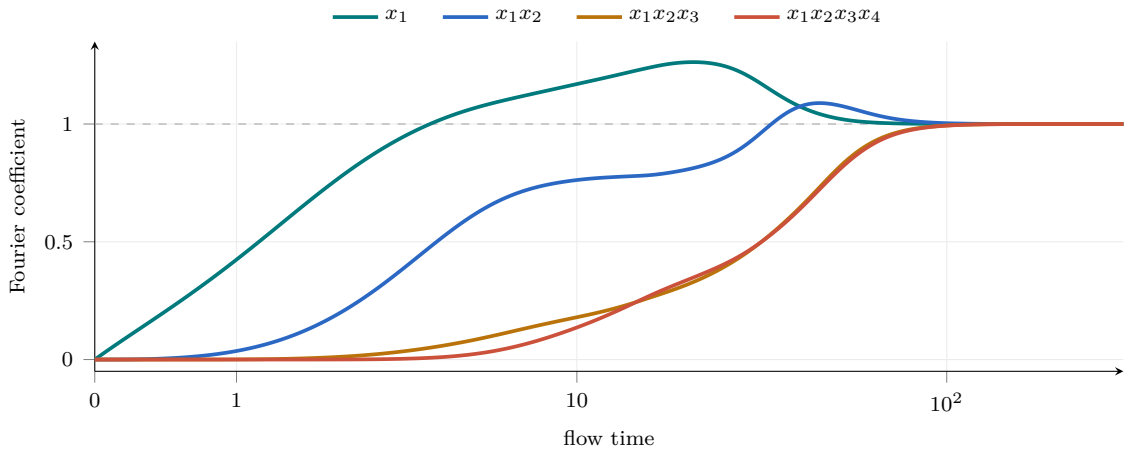
Each relevant coordinate gets a product of $k-1$ weights of size $d^{-1/2}$, so $\|\nabla C(w)\| \asymp d^{-(k-1)/2}$ at initialization. This is the single-index answer with $k^\star$ replaced by k, and the same heuristic gives that you need $n \approx d^{\max\{k-1,1\}}$ samples for the gradient to point towards the relevant coordinates.

Staircases

Compare

$$f_1(x) = x_1x_2x_3x_4, \qquad f_2(x) = x_1 + x_1x_2 + x_1x_2x_3 + x_1x_2x_3x_4.$$

Both have degree four, so a kernel needs d^4 samples for either one, and even for the neural network the parity will cost d^3 samples. Even though the parity is one of the terms in f_2, the network will actually learn it much faster in only $O(d)$ samples. This plot gives the learning order:



We can understand this better by computing the gradient. By (9) (and dropping constants), $C(w) \approx w_1 + w_1w_2 + w_1w_2w_3 + w_1w_2w_3w_4$, so

$$\nabla C(w) \approx \begin{bmatrix} 1 + w_2 + w_2w_3 + w_2w_3w_4 \\ w_1 + w_1w_3 + w_1w_3w_4 \\ w_1w_2 + w_1w_2w_4 \\ w_1w_2w_3 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \overset{\text{init}}{\approx} \begin{bmatrix} \Theta(1) \\ d^{-1/2} \\ d^{-1} \\ d^{-3/2} \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{11}$$

At first only w_1 moves. Once w_1 is of order one, the second entry is of order one and w_2 grows. Then the third entry w_1w_2 is of order one, and so on. The coordinates are unlocked one at a time, and each step is as easy as learning a single-index model with information exponent 1, which takes d samples. Thus, the total cost is $O(d)$ samples, instead of the d^3 required by the full parity.

This is an example of a *staircase function*, which is a function in which the relevant coordinates can be learned one at a time with $O(d)$ samples (Abbe et al., 2021, 2022).

Heuristic 2 (Iterative Conditioning)**.** *Once we know x_1 matters, keep only the samples with $x_1 = 1$. On those, $f_2 = 1 + x_2 + x_2x_3 + x_2x_3x_4$, a staircase on one fewer coordinate. In general, conditioning on the known coordinates x_U turns χ_S into a parity on $S \setminus U$, so it should cost what a parity of degree $|S \setminus U|$ costs. Each step of the staircase is a degree-one problem, about d samples, so the staircase costs $n \approx d$ instead of d^3.*

Leap complexity

More generally, when we have an arbitrarily complex $f^\star$, the cost of learning it is determined by the *leap complexity* (Abbe et al., 2023a), which is the maximum number of new coordinates the network is forced to learn at once. For example, $x_1 + x_1x_2 + x_1x_2x_3 + x_1x_2x_3x_4$ has leap one because the learning order is $x_1 \to x_2 \to x_3 \to x_4$, while $x_1x_2 + x_1x_2x_3x_4$ has two leaps of size 2 (first learning x_1, x_2 then learning x_3, x_4 given x_1, x_2). The precise definition is given by:

Definition 4 (Leap complexity)**.** Start with $U = \varnothing$, the set of coordinates learned so far. While some $S \in \operatorname{supp} f^\star$ is not contained in U:

1. Let $\ell_t = \min\{|S \setminus U| : S \in \operatorname{supp} f^\star,\ S \not\subseteq U\}$. This round is a *leap of size* ℓ_t.
2. Add to U all coordinates of every term with $|S \setminus U| = \ell_t$.

The *leap* of $f^\star$ is $\ell = \mathrm{Leap}(f^\star) = \max_t \ell_t$.

Conjecture 1 (Abbe et al., 2023a)**.** *For a target $f^\star$ with generic coefficients, on Boolean inputs, a two-layer network trained by online SGD learns $f^\star$ with*

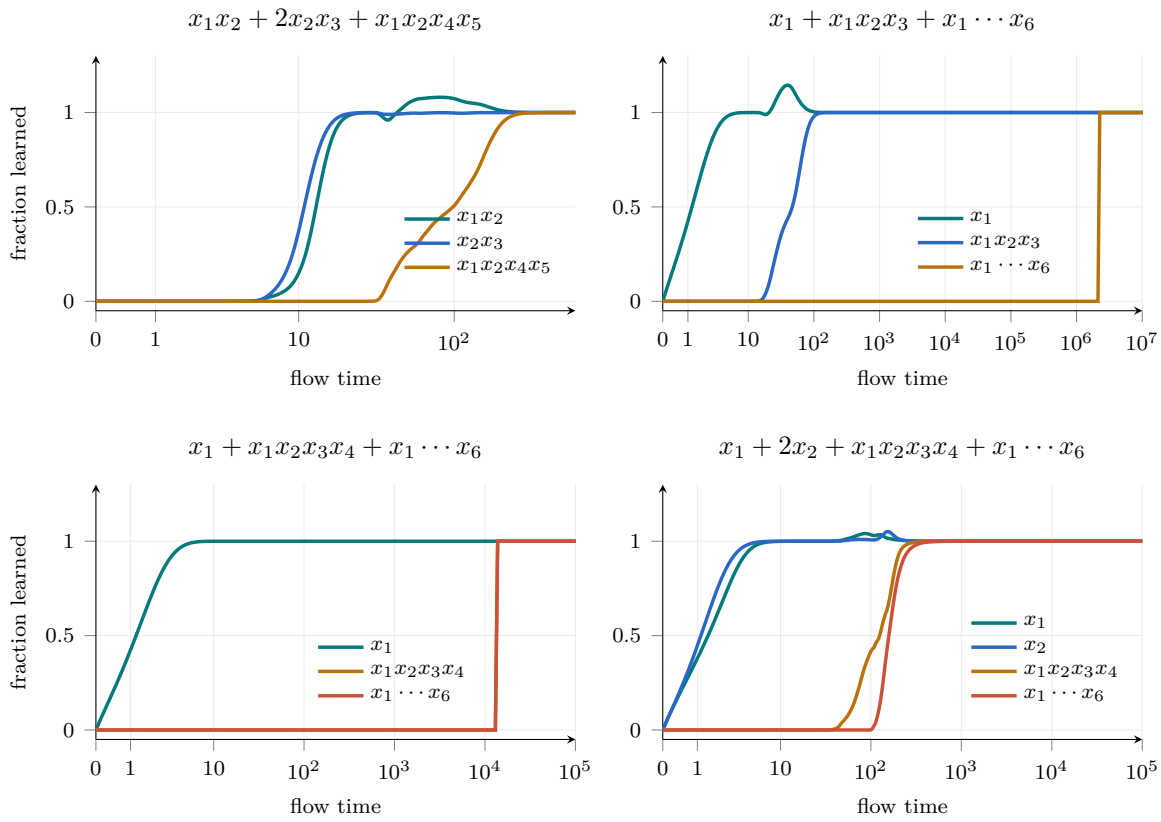
$$n = \widetilde{\Theta}\left(d^{\max\{\ell-1,1\}}\right)$$

samples, where $\ell = \mathrm{Leap}(f^\star)$.

A version of this is proved in Abbe et al. (2022, 2023a) for a non-standard form of SGD (layer-wise training, non-standard learning rates, etc.), but the conjecture has been proved for standard SGD on a two-layer network for 2-parity, with the logistic loss (Glasgow, 2024).

Examples. Let's compute the leap for a few targets:

- $x_1x_2 + 2x_2x_3 + x_1x_2x_4x_5$. The first round learns both degree-two terms, which reveals x_1, x_2, x_3. Given these, $x_1x_2x_4x_5$ only needs x_4, x_5. The leaps are $2, 2$, so $\ell = 2$ and $n \approx d$, even though the degree is 4.
- $x_1 + x_1x_2x_3 + x_1 \cdots x_6$. The leaps are $1, 2, 3$, so $\ell = 3$ and $n \approx d^2$.
- $x_1 + x_1x_2x_3x_4 + x_1 \cdots x_6$. The leaps are $1, 3, 2$, so $\ell = 3$ and $n \approx d^2$. Adding $2x_2$ changes the leaps to $1, 2, 2$, so $\ell = 2$ and $n \approx d$. Adding a term makes the target easier to learn.



Gaussian multi-index models

A Gaussian multi-index model is defined by $x \sim N(0, I_d)$ and $f^\star(x) = g(Ux)$ with $U \in \mathbb{R}^{r\times d}$, $UU^\top = I_r$. The same ideas apply, but rotation invariance makes life weirder. Consider $f^\star = z_1 + z_2 + z_1z_2$ with $z = Ux$ (Abbe et al., 2023a, Appendix B.2). On the Boolean cube this has leap one.

However, for Gaussians the story is weirder. The network will first fit the best linear fit, which is $z_1 + z_2$. It therefore learns the feature $u = (z_1 + z_2)/\sqrt{2}$. The critical subspace is therefore spanned by u, v where $v = (z_1 - z_2)/\sqrt{2}$ is the orthogonal direction. In the (u, v) basis, we can rewrite $f^\star$ as:

$$f^\star = \sqrt{2}\, u + \frac{u^2 - v^2}{2}.$$

Therefore, after we've learned the u direction, the remaining term is $-v^2/2$, which is a single-index problem with $k^\star = 2$. Thus, $z_1 + z_2 + z_1z_2$ has leap one for the Boolean cube, but leap two for Gaussians. For Gaussians we can't read the leap off the formula in the given basis. We have to rotate into the basis the network actually learns in. See Abbe et al. (2023a); Bietti et al. (2023) for the Gaussian staircase, and Damian et al. (2025) for the sharp sample complexity when the algorithm may also transform the labels.

Staircases in the wild

A parity is hard because the network needs to learn several directions at once, and even if it learns all directions but one, it still doesn't achieve a non-trivial loss. This is a symptom of a saddle in the

population loss landscape. This story generalizes to other tasks as well: whenever multiple parts of the network must coordinate to solve a task, and there is little reward from learning only a subset of them, the network will struggle to learn, and building a staircase can help the network escape the saddle. Such situations are common with synthetic data, but seem rarer in natural data.

Natural data is not uniform. Consider sentences of the form

$$\text{The movie was } \{\varnothing,\ \text{not}\}\ \{\text{good},\ \text{bad}\},\ \text{I}\ \{\varnothing,\ \text{didn't}\}\ \{\text{love},\ \text{hate}\}\ \text{it.}$$

Let $x_1 = -1$ if "not" is present, $x_2 = +1$ for good, $x_3 = -1$ if "didn't" is present, and $y = +1$ for love. Each negation flips the answer, so $y = x_1x_2x_3$, a three-parity. But negations are rare. If "not" appears with probability 0.2 and "didn't" with probability 0.3, then $\mathbb{E}[y \mid x_2] = 0.24\,x_2$, so the first step is a leap of one, and each remaining coordinate is one more leap of one.[5] Biased inputs build the staircase for us, and training on biased inputs first provably makes parities learnable by SGD (Abbe et al., 2023b).

Multi-hop lookup. Answering "on what day does John play?" from a document that says "John plays quarterback" and "the football game is on Sunday" requires chaining lookups. Wang et al. (2025) model this as k-fold composition $\sigma_k \circ \pi_k \circ \cdots \circ \sigma_1 \circ \pi_1(x)$, where the π_j are memorized and the σ_j are given in context. Getting some of the hops right gives no correlation with the answer, so this is a parity again, and for SQ algorithms, learning from k-hop data alone needs exponentially many samples in k. Adding shorter hops to the training data builds a staircase and makes it learnable with polynomially many samples (see also Sanford et al., 2024).

Internalizing chain of thought. Transformers trained directly on multiplication fail to learn the correct mechanism, but learn easily with a chain of thought (CoT). Deng et al. (2024) start from a CoT model and remove CoT tokens a few at a time while fine-tuning. GPT-2 Small trained this way multiplies 9-digit numbers with 99% accuracy and no intermediate tokens, while training directly without CoT gets essentially zero. Thus, even though the network could *express* the correct multiplication algorithm, SGD failed to find it without a staircase.

> **Takeaway.** In general, when a network should be able to represent something but fails to learn it, there is often a saddle where several things have to be learned at once and the easiest fix is to create a staircase by modifying the data or architecture in some way.

Lower Bounds

Neural networks need about d^{k-1} samples to learn k-parity or a single-index model with information exponent k. Is there any evidence that these problems are genuinely hard, and not just hard for the specific networks, learning rates, and parameterizations we analyzed?

The ideal theorem would be that any neural network needs at least some number of samples to learn these targets. Without restrictions on what counts as a neural network, this statement is false.

[5] Formally, the leap is defined in the Fourier basis of the biased cube, $\phi_i = (x_i - \mu_i)/\sqrt{1-\mu_i^2}$, where $y \approx 0.24\,\phi_2 + 0.55\,\phi_2\phi_3 + 0.32\,\phi_1\phi_2 + 0.73\,\phi_1\phi_2\phi_3$.

Abbe and Sandon (2020) showed that for any polynomial-time algorithm there is a "polynomial-size network" on which SGD, from a carefully constructed initialization, emulates that algorithm. So a lower bound against all networks trained by SGD is a lower bound against all polynomial-time algorithms.

The restriction to polynomial time is essential. By Theorem 1, a brute force search over an ϵ-net of the sphere learns any single-index model from $O(d)$ samples, just like linear regression, but the net has size $(3/\epsilon)^d$. All of the difficulty is in optimizing a nonconvex landscape, and it disappears if we can afford exponential time. Proving that no polynomial time algorithm can solve a problem that is easy to check (just compute the loss) is P versus NP territory, and far out of reach.

Instead, the community has focused on proving lower bounds for restricted classes of algorithms, each of which constrains how the algorithm may look at the samples. The most common class is the statistical query (SQ) model, and there is a natural variant called the correlational statistical query (CSQ) model that is particularly relevant for neural networks. We will also briefly discuss the low-degree polynomial framework. These lower bounds are good evidence of hardness because almost every noise robust polynomial time algorithm we know for these problems (including spectral/tensor methods and large-batch gradient descent), lies in one of these classes.

We will begin by defining the three classes:

Definition 5 (SQ, Kearns, 1998)**.** A *statistical query* (SQ) algorithm repeatedly chooses a query $h : \mathbb{R}^d \times \mathbb{R} \to [-1, 1]$ and receives an arbitrary value $\hat{h}$ with $\left|\hat{h} - \mathbb{E}[h(x, y)]\right| \leq \tau$. The parameter τ is the *tolerance*.

Definition 6 (CSQ, Bshouty and Feldman, 2002)**.** A *correlational statistical query* (CSQ) algorithm is the same as a statistical query, but its queries must have the form $h(x, y) = y\, h(x)$.

Definition 7 (Low-degree polynomials, Hopkins, 2018)**.** A *low-degree* algorithm sees the samples $\{(x_i, y_i)\}_{i=1}^n$, and its output must be a polynomial in them of degree at most $(\log d)^{1.01}$.

The query h may be any function, even one that is intractable to compute. However, it can only look at a single sample at a time, so it can't execute the brute force search that gives you the information theoretic threshold.

An SQ algorithm can be simulated with samples. Given n fresh samples, the empirical average $\frac{1}{n}\sum_i h(x_i, y_i)$ is within $\tau \approx n^{-1/2}$ of $\mathbb{E}[h]$ with high probability by Hoeffding's inequality, since h is bounded. So we think of τ as $n^{-1/2}$, where n is the number of samples per query. The only difference is that sampling noise is random, while the oracle's error is adversarial.

Why CSQ? CSQ restricts SQ by forcing the query to be linear in the labels. There are two reasons why this is reasonable. The first is that if $y \in \{\pm 1\}$, then any function of y is affine in y, so CSQ and SQ are equivalent. When y is real-valued, SQ can be significantly more powerful because it can transform the labels, e.g. $h(x, y) = T(y)h(x)$ for some label transformation T.

The second reason to care about correlational queries is that the gradient of a neural network trained with square loss is a correlational query:

$$\nabla_\theta \tfrac{1}{2}\mathbb{E}[(f_\theta(x) - y)^2] = \mathbb{E}[f_\theta(x)\nabla_\theta f_\theta(x)] - \mathbb{E}[y\,\nabla_\theta f_\theta(x)].$$

Assuming the learner knows the input distribution (e.g. $N(0, I_d)$ or $\mathrm{Unif}\{\pm 1\}^d$), the first term can be computed without labels, and the second term is a correlational query.

CSQ lower bounds

Parities. Let $x \sim \mathrm{Unif}\{\pm 1\}^d$ and $y = \chi_S(x)$, where the adversary picks S uniformly among the $N = \binom{d}{k}$ sets of size k. The adversary answers 0 to every query, which is a valid answer unless $|\mathbb{E}[y\, h(x)]| = |\langle h, \chi_S\rangle| > \tau$. If we knew S, the best query would be $h = \chi_S$. But we don't, so a query has to hedge across many candidate sets, and it can't correlate with many at once: by Parseval,

$$\sum_S \langle h, \chi_S\rangle^2 \le \mathbb{E}[h^2] \le 1,$$

so at most $1/\tau^2 = n$ sets satisfy $|\langle h, \chi_S\rangle| > \tau$. Each query therefore rules out at most n of the N candidates, and on all the others the learner gains no information. After q queries with $qn \le N/2$, at least half of the candidates are still consistent with every answer, so the learner fails with probability at least $1/2$. Hence any CSQ algorithm that learns k-parity with probability at least $1/2$ must have

$$qn \gtrsim \binom{d}{k} \approx d^k.$$

Since the labels are ± 1, CSQ and SQ coincide so this is also an SQ lower bound (Kearns, 1998).

It turns out SGD already matches this lower bound! A gradient is d queries, and with batch size $n \approx d^{k-1}$ a network learns in $O(\log d)$ steps (HW2, Problem 1), so $qn \approx d^k$ up to logarithms.

General targets. The same game works for any prior over targets f_w with $\|f_w\| = 1$: the adversary answers 0, and a query h rules out only the targets it correlates with,

$$B_h = \{w : |\langle h, f_w\rangle| > \tau\}.$$

The only place we used parities was Parseval, which needs the targets to be orthogonal. We can generalize the parity argument by studying the distribution of the overlap

$$\rho(w, v) = \langle f_w, f_v\rangle$$

of two targets w, v drawn independently from the prior. For parities $\rho = \mathbf{1}\{S = S'\}$. The following lemma replaces Parseval when the overlaps are small but nonzero, which we will use to prove lower bounds for single-index models.

Lemma 2. *Suppose that for every w, $\Pr_v(|\rho(w, v)| > \varepsilon) \le \delta$. If $\|h\| \le 1$ and $\tau^2 \ge 2\varepsilon$, then*

$$\Pr_w(w \in B_h) \le 2\delta/\tau^2.$$

Proof. Let $\mu = \Pr(B_h)$ and $s_w = \mathrm{sign}\,\langle h, f_w\rangle$, so that $s_w \langle h, f_w\rangle > \tau$ on B_h. By Cauchy–Schwarz,

$$\tau\mu \le \mathbb{E}_w[\mathbf{1}_{B_h}(w)\, s_w \langle h, f_w\rangle] = \left\langle h, \mathbb{E}_w[\mathbf{1}_{B_h}(w)\, s_w f_w]\right\rangle \le \left\|\mathbb{E}_w[\mathbf{1}_{B_h}(w)\, s_w f_w]\right\|.$$

Squaring, with $v \overset{\text{i.i.d.}}{\sim} w$ and splitting on whether $|\rho(w, v)| \le \varepsilon$,

$$\tau^2\mu^2 \le \mathbb{E}_{w,v}[\mathbf{1}_{B_h}(w)\mathbf{1}_{B_h}(v)|\rho(w, v)|] \le \varepsilon\mu^2 + \delta\mu.$$

And rearranging gives $\mu \le \delta/(\tau^2 - \varepsilon) \le 2\delta/\tau^2$. □

For parities, $\varepsilon = 0$ and $\delta = 1/N$, and the lemma recovers the bound n/N from Parseval.

The easiest way to interpret Lemma 2 is that if $|\rho| \le \epsilon$ with very high probability over the prior, then $n \gtrsim 1/\epsilon$ samples are necessary to learn the target. It turns out that this criterion (when applied to the likelihood ratios below instead of to f_w directly) also implies a lower bound for low-degree algorithms (Brennan et al., 2021), so it gives a fairly general criterion for hardness.

Single-index models. Let $x \sim N(0, I_d)$ and $y = \phi(w \cdot x)$ with w uniform on the sphere, $\phi = \sum_{k \ge k^\star} c_k h_k$, and $\sum_k c_k^2 = 1$. By Corollary 2,

$$\rho = \sum_{k \ge k^\star} c_k^2 (w \cdot v)^k, \qquad \text{so} \qquad |\rho| \le |w \cdot v|^{k^\star}.$$

Therefore the typical overlap has size $d^{-k^\star/2}$. Then as long as $n \ll d^{k^\star/2}$, Lemma 2 says each query kills only a tiny fraction of the sphere, so polynomially many queries leave almost every direction consistent with all-zero answers. The algorithm then outputs the same $\hat{w}$ for all of these directions, and $\hat{w}$ is close to only an exponentially small fraction of them, implying CSQ learners need $n \ge \tilde{\Omega}(d^{k^\star/2})$ samples (Damian et al., 2022).

Recall that standard SGD needs $d^{k^\star - 1}$ samples, which can be significantly worse than $d^{k^\star/2}$ when $k^\star$ is large. However, this better rate is actually tight. SGD on a smoothed loss landscape $\tilde{L}(w) = \mathbb{E}_z L(w + \lambda z)$ only needs $d^{\max(1, k^\star/2)}$ (Damian et al., 2023), and the intuition is that the smoothing eliminates many of the shallow local minima near the equator.

SQ lower bounds

For an SQ algorithm, the oracle can no longer adversarially answer 0 to every query since even the trivial query $h(x, y) = 1$ has expectation 1. Instead, the oracle answers with the expectation under a *null distribution* D_0 in which x and y are independent with the same marginals. For a single-index model, sample $x, x' \sim N(0, I_d)$ independently, set $y = \phi(w \cdot x)$, and output (x', y). The law of y does not depend on w, since $w \cdot x \sim N(0, 1)$ for every w, so answering every query with $\mathbb{E}_{(x,y) \sim D_0}[h(x, y)]$ reveals nothing about w. This answer is valid unless $|\mathbb{E}_{P_w}[h] - \mathbb{E}_{D_0}[h]| > \tau$, where P_w is the true distribution of (x, y). Changing measure with the likelihood ratio $L_w = dP_w/dD_0$,[6]

$$\mathbb{E}_{P_w}[h] - \mathbb{E}_{D_0}[h] = \mathbb{E}_{D_0}\left[h\,(L_w - 1)\right] = \langle h, L_w - 1 \rangle_{D_0}.$$

So a CSQ query has to correlate with the target f_w, while an SQ query has to correlate with $L_w - 1$, and the CSQ argument goes through verbatim with f_w replaced by $L_w - 1$. By symmetry $\|L_w - 1\|_{D_0}^2 =: A$ does not depend on w, and Lemma 2 holds with τ replaced by $\tau/\sqrt{A}$ and overlap

$$\rho = \frac{1}{A} \langle L_w - 1, L_v - 1 \rangle_{D_0}.$$

For a single-index model $y \sim p(\cdot \mid w \cdot x)$, write $z = w \cdot x$. By Bayes' rule $L_w(x, y) = p(z \mid y)/\gamma(z)$, whose Hermite coefficients in z are $\mathbb{E}[h_k(z) \mid y]$. So

$$L_w(x, y) = \sum_{k \ge 0} \mathbb{E}[h_k(z) \mid y]\, h_k(w \cdot x), \qquad \rho = \frac{1}{A} \sum_{k \ge 1} \lambda_k^2 (w \cdot v)^k, \qquad \lambda_k^2 = \mathbb{E}_y\left[\mathbb{E}[h_k(z) \mid y]^2\right].$$

[6] This assumes L_w exists and $\|L_w - 1\|_{D_0} < \infty$, which fails for noiseless real-valued labels such as $y = h_3(w \cdot x)$. Adding a little label noise fixes this, and Damian et al. (2024) handle the general case.

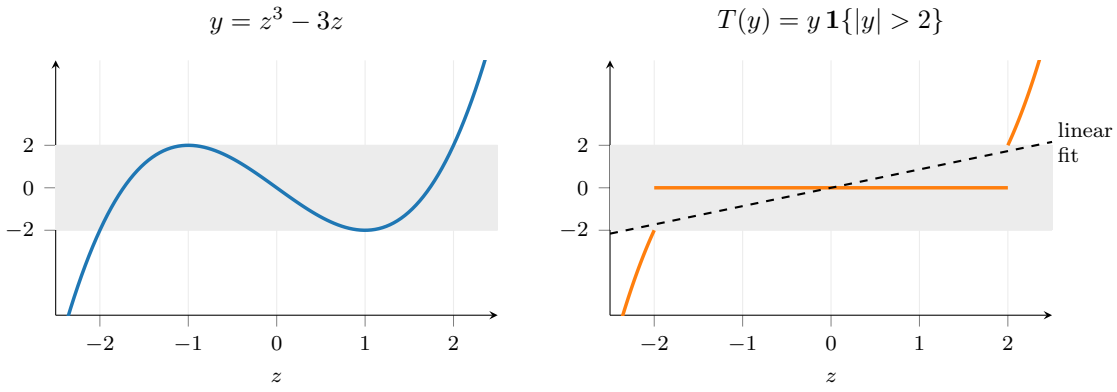
This is the CSQ overlap with c_k^2 replaced by λ_k^2. The *generative exponent* $k^\star_{\text{gen}}$ is the first $k \geq 1$ with $\lambda_k \neq 0$, i.e. the information exponent of the likelihood ratio. Since $\mathbb{E}[T(y)h_k(z)] = \mathbb{E}[T(y)\,\mathbb{E}[h_k(z) \mid y]]$, it is also the smallest information exponent of $T(y)$ over all label transformations T.

Corollary 3 (Damian et al., 2024). *SQ algorithms with* $\text{poly}(d)$ *queries need* $n \gtrsim (d/\log d)^{k^\star_{\text{gen}}/2}$ *samples to learn a single-index model with generative exponent* $k^\star_{\text{gen}}$.

The same bound holds for low-degree polynomials (Hopkins, 2018; Damian et al., 2024), and a polynomial-time algorithm matches it using $d^{\max(1,k^\star_{\text{gen}}/2)}$ samples (Damian et al., 2024).

It's already rare to encounter a link function with high information exponent and it is much rarer to encounter one with high generative exponent. For example, every polynomial link has $k^\star_{\text{gen}} = 1$ unless it is even (i.e., $\phi(z) = \phi(-z)$ for all z), in which case $k^\star_{\text{gen}} = 2$.

To give an explicit example, consider the cubic $y = z^3 - 3z \propto h_3(z)$, which has information exponent 3. If we apply the label transformation T which zeros out the labels with $|y| \leq 2$, where y has three preimages, we are left with a function that is monotone in its tails, and therefore has nonzero correlation with the linear function z. This means that the generative exponent $k^\star_{\text{gen}}$ is actually 1 for this function, despite the information exponent being 3.



References

E. Abbe and C. Sandon. On the universality of deep learning. In *Advances in Neural Information Processing Systems*, volume 33, 2020. arXiv:2001.02992.

E. Abbe, E. Boix-Adserà, M. Brennan, G. Bresler, and D. Nagaraj. The staircase property: How hierarchical structure can guide deep learning. In *Advances in Neural Information Processing Systems*, 2021. URL https://arxiv.org/abs/2108.10573.

E. Abbe, E. Boix-Adserà, and T. Misiakiewicz. The merged-staircase property: A necessary and nearly sufficient condition for SGD learning of sparse functions on two-layer neural networks. In *Proceedings of the 35th Conference on Learning Theory*, volume 178 of *Proceedings of Machine Learning Research*, pages 4782–4887, 2022. URL https://proceedings.mlr.press/v178/abbe22a.html.

E. Abbe, E. Boix-Adserà, and T. Misiakiewicz. SGD learning on neural networks: Leap complexity and saddle-to-saddle dynamics. In *Proceedings of the 36th Conference on Learning Theory*, volume 195 of *Proceedings of Machine Learning Research*, pages 2552–2623, 2023a. URL https://proceedings.mlr.press/v195/abbe23a.html.

E. Abbe, E. Cornacchia, and A. Lotfi. Provable advantage of curriculum learning on parity targets with mixed inputs. In *Advances in Neural Information Processing Systems*, volume 36, 2023b. URL https://arxiv.org/abs/2306.16921.

G. Ben Arous, S. Mei, A. Montanari, and M. Nica. The landscape of the spiked tensor model. *Communications on Pure and Applied Mathematics*, 72:2282–2330, 2019. doi: 10.1002/cpa.21861.

G. Ben Arous, R. Gheissari, and A. Jagannath. Algorithmic thresholds for tensor PCA. *The Annals of Probability*, 48(4):2052–2087, 2020. doi: 10.1214/19-AOP1415.

G. Ben Arous, R. Gheissari, and A. Jagannath. Online Stochastic Gradient Descent on Non-Convex Losses from High-Dimensional Inference. *Journal of Machine Learning Research*, 22(106):1–51, 2021. URL https://arxiv.org/abs/2003.10409.

A. Bietti, J. Bruna, and L. Pillaud-Vivien. On learning Gaussian multi-index models with gradient flow, 2023. URL https://arxiv.org/abs/2310.19793.

M. Brennan, G. Bresler, S. B. Hopkins, J. Li, and T. Schramm. Statistical query algorithms and low-degree tests are almost equivalent. In *Proceedings of the 34th Conference on Learning Theory*, 2021. arXiv:2009.06107.

N. H. Bshouty and V. Feldman. On using extended statistical queries to avoid membership queries. *Journal of Machine Learning Research*, 2:359–395, 2002.

A. Damian, J. D. Lee, and M. Soltanolkotabi. Neural Networks Can Learn Representations with Gradient Descent. In *COLT*, 2022. URL https://arxiv.org/abs/2206.15144.

A. Damian, E. Nichani, R. Ge, and J. D. Lee. Smoothing the landscape boosts the signal for SGD: Optimal sample complexity for learning single index models. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

A. Damian, L. Pillaud-Vivien, J. D. Lee, and J. Bruna. Computational-statistical gaps in Gaussian single-index models. In *COLT*, page 1262, 2024. URL https://arxiv.org/abs/2403.05529. Extended abstract in PMLR 247; full version at the arXiv link.

A. Damian, J. D. Lee, and J. Bruna. The generative leap: Sharp sample complexity for efficiently learning Gaussian multi-index models, 2025. URL https://arxiv.org/abs/2506.05500.

Y. Deng, Y. Choi, and S. Shieber. From explicit CoT to implicit CoT: Learning to internalize CoT step by step. arXiv:2405.14838, 2024. URL https://arxiv.org/abs/2405.14838.

B. Ghorbani, S. Mei, T. Misiakiewicz, and A. Montanari. Linearized Two-Layers Neural Networks in High Dimension. *The Annals of Statistics*, 49(2):1029–1054, 2021. URL https://arxiv.org/abs/1904.12191.

M. Glasgow. SGD finds then tunes features in two-layer neural networks with near-optimal sample complexity: A case study in the XOR problem. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2309.15111.

S. B. Hopkins. *Statistical Inference and the Sum of Squares Method*. PhD thesis, Cornell University, 2018. URL https://www.samuelbhopkins.com/thesis.pdf.

M. Kearns. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM*, 45(6):983–1006, 1998. doi: 10.1145/293347.293351. URL https://doi.org/10.1145/293347.293351.

T. McGrath, A. Kapishnikov, N. Tomašev, A. Pearce, M. Wattenberg, D. Hassabis, B. Kim, U. Paquet, and V. Kramnik. Acquisition of chess knowledge in AlphaZero. *Proceedings of the National Academy of Sciences*, 119(47):e2206625119, 2022. doi: 10.1073/pnas.2206625119. URL https://arxiv.org/abs/2111.09259.

S. Mei, T. Misiakiewicz, and A. Montanari. Generalization Error of Random Feature and Kernel Methods: Hypercontractivity and Kernel Matrix Concentration. *Applied and Computational Harmonic Analysis*, 59: 3–84, 2022. URL https://arxiv.org/abs/2101.10588.

T. Misiakiewicz. Lecture 9: Learning multi-index functions. SDS 659 lecture notes, 2025. URL https://misiakie.github.io/SDS659_notes_2025/Lecture%209_%20multi-index.pdf.

C. Olah, A. Mordvintsev, and L. Schubert. Feature visualization. *Distill*, 2017. doi: 10.23915/distill.00007. URL https://distill.pub/2017/feature-visualization/. Images selected and arranged from the original under CC BY 4.0.

C. Sanford, D. Hsu, and M. Telgarsky. Transformers, parallel computation, and logarithmic depth. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, 2024. URL https://arxiv.org/abs/2402.09268.

A. Templeton, T. Conerly, J. Marcus, J. Lindsey, T. Bricken, B. Chen, A. Pearce, C. Citro, E. Ameisen, A. Jones, H. Cunningham, N. L. Turner, C. McDougall, M. MacDiarmid, A. Tamkin, E. Durmus, T. Hume, F. Mosconi, C. D. Freeman, T. R. Sumers, E. Rees, J. Batson, A. Jermyn, S. Carter, C. Olah, and T. Henighan. Scaling monosemanticity: Extracting interpretable features from Claude 3 Sonnet. Transformer Circuits, 2024. URL https://transformer-circuits.pub/2024/scaling-monosemanticity/index.html.

G. Valiant. Finding correlations in subquadratic time, with applications to learning parities and the closest pair problem. *Journal of the ACM*, 62(2):13:1–13:45, 2015. doi: 10.1145/2728167. URL https://theory.stanford.edu/~virgi/cs367/papers/corrFull.pdf.

Z. Wang, E. Nichani, A. Bietti, A. Damian, D. Hsu, J. D. Lee, and D. Wu. Learning compositional functions with transformers from easy-to-hard data. In *Proceedings of the 38th Conference on Learning Theory*, 2025. URL https://arxiv.org/abs/2505.23683.