Learning Sparse Functions with Neural Networks

Theodor Misiakiewicz

Stanford University

September 16th, 2022

FLAIR – Foundation of Learning and AI Research (EPFL)

Based on a few joint works with E. Abbe (EPFL), E. Boix-Adsera (MIT), M. Celentano (Berkeley), H. Hu (Harvard), Y. M. Lu (Harvard) and A. Montanari (Stanford).

The three components of statistical learning

Statistical learning has to balance three competing goals:

- **Approximation:** rich enough model class to approximate the target function.
- **Generalization:** the trained model needs to generalize to unseen data.
- **Computation:** the learning algorithm must be computationally efficient.

Classical approach:

- Balance approximation and generalization errors using an explicit regularization.
- Learning stated as a convex optimization problem (convex ERM).

Deep learning:

- No explicit regularization (often trained until interpolation).
- Highly non-convex optimization problem.

DL success suggests radically different ways of balancing these 3 goals:

- **Implicit regularization:** the learning procedure selects a particular interpolating solution among many possible ones.
- **Tractability via overparametrization:** optimization becomes easier as the number of parameters increases.

In DL, there is a complex interplay between these three goals!

Curse of dimensionality

Empirical observation: DL is successfully applied to massive high-dimensional datasets.

Why does DL seemingly not suffer from the curse of dimensionality?

Consider a 2-layer NN with M hidden units $\boldsymbol{\Theta} = (a_j, \boldsymbol{w}_j, b_j)_{j \in [M]} \in \mathbb{R}^{M(d+2)}$,

$$\hat{f}_{\mathrm{NN}}(\boldsymbol{x}; \boldsymbol{\Theta}) = \frac{1}{M} \sum_{j \in [M]} a_j \sigma(\langle \boldsymbol{w}_j, \boldsymbol{x} \rangle + b_j) , \qquad \boldsymbol{x} \in \mathbb{R}^d .$$

Curse of dimensionality: e.g., $\mathcal{H} =$ 1-bounded and 1-Lipschitz functions on $[0,1]^d$

- *In approximation:* with M neurons [Maiorov,'99]

$$\sup_{h_* \in \mathcal{H}} \inf_{\boldsymbol{\Theta}} \| \hat{f}_{\mathrm{NN}}(\cdot; \boldsymbol{\theta}) - h_* \|_{L^2} \asymp M^{-\frac{1}{d-1}} .$$

- *In statistical complexity:* with n samples and any estimators $\hat{f}_n$ [Schmidt-Hieber,'20]

$$\inf_{\hat{f}_n} \sup_{h_* \in \mathcal{H}} \| \hat{f}_n - h_* \|_{L^2} \asymp n^{-\frac{2}{d+2}} .$$

Breaking the CoD on sparse functions

Conjecture:

Real data has low-dimensional structure. NNs can adapt to it and break the CoD.

Simplest example: *P-sparse functions* that depend on a latent low-dimensional subspace, i.e., there exists a projection $\boldsymbol{P} \in \mathbb{R}^{P\times d}$ such that $h_*(\boldsymbol{x}) = h_*(\boldsymbol{P}\boldsymbol{x})$, with $P \ll d$.

NNs break CoD on sparse functions: $\mathcal{H}_P =$ functions in $\mathcal{H}$ that are P-sparse

- *Approximation:* take $\boldsymbol{w}_j$'s aligned with the P-dim support, i.e., $\boldsymbol{w}_j \in \text{span}(\boldsymbol{P}^\mathsf{T})$

$$\sup_{h_* \in \mathcal{H}_P} \inf_{\boldsymbol{\Theta}} \|\hat{f}_{\text{NN}}(\cdot;\boldsymbol{\theta}) - h_*\|_{L^2} \asymp M^{-\frac{1}{P-1}}\,.$$

- *Stat. complexity:* $M = \infty$ + sparsity inducing norm [Bach,'17], [Schmidt-Hieber,'20]

$$\sup_{h_* \in \mathcal{H}_P} \|\hat{f}_{\text{NN}} - h_*\|_{L^2} \asymp n^{-\frac{2}{P+2}}\,. \qquad \text{(minmax optimal)}$$

Outline of the talk

In previous slide: NNs break the CoD in approximation and generalization on sparse functions, but no efficient algorithm is provided to construct these NNs.

How are these results modified when we add the computational aspect?

This talk: I will consider three scenarios which correspond to learning sparse functions with 2-layer NNs in three optimization regimes

A. **Lazy training regime.**

B. **Convex Neural Networks.**

C. **Online-SGD in the mean-field scaling.**

Goal:

Study approximation, generalization and computational aspects in each of these scenarios.

Overall setting for this talk

- **2-layer NN:** M hidden units and parameters $\boldsymbol{\Theta} = (a_j, \boldsymbol{w}_j, b_j)_{j\in[M]} \in \mathbb{R}^{M(d+2)}$,

$$\hat{f}_{\mathrm{NN}}(\boldsymbol{x};\boldsymbol{\Theta}) = \frac{\alpha}{M}\sum_{j\in[M]} a_j \sigma(\langle \boldsymbol{w}_j, \boldsymbol{x}\rangle + b_j)\,, \qquad \boldsymbol{x}\in\mathbb{R}^d\,.$$

  with $\alpha \in \{1, \sqrt{M}\}$. Sometimes ignore the biases $b_j = 0$.

- **Supervised learning setting:** given n data points $\{(y^{(i)}, \boldsymbol{x}^{(i)})\}_{i\in[n]}$,

$$y^{(i)} = f_*(\boldsymbol{x}^{(i)}) + \varepsilon^{(i)}\,, \qquad \varepsilon^{(i)} \text{ independent noise } \mathbb{E}[\varepsilon^{(i)}] = 0, \mathbb{E}[(\varepsilon^{(i)})^2] = \tau^2,$$
$$\boldsymbol{x}^{(i)} \sim_{iid} \mathrm{Unif}(\mathbb{S}^{d-1}(\sqrt{d})) \qquad \text{or} \qquad \mathrm{Unif}(\{+1,-1\}^d)\,.$$

- **Target function:** $f_* \in L^2$ and P-sparse, i.e.,

$$f_*(\boldsymbol{x}) = h_*(\boldsymbol{z})\,, \qquad \boldsymbol{z} = \boldsymbol{P}\boldsymbol{x} \in \mathbb{R}^P\,,$$

  for some latent (unknown) $\boldsymbol{P} \in \mathbb{R}^{P\times d}$, with P fixed and d large.
  ($\boldsymbol{P}$: P-dim subspace on the sphere or subset of P-coordinates on hypercube.)

A. Lazy training regime.

The ‘lazy training’ or ‘linear’ regime

- In some optimization regime, we can effectively replace NNs by its linarization:

$$\hat{f}_{\mathrm{NN}}(\boldsymbol{x}; \boldsymbol{\Theta}^t) \approx \hat{f}_{\mathrm{NN}}(\boldsymbol{x}; \boldsymbol{\Theta}^0) + \langle \boldsymbol{\beta}^t, \nabla_{\boldsymbol{\Theta}} \hat{f}_{\mathrm{NN}}(\boldsymbol{x}; \boldsymbol{\Theta}^0) \rangle ,$$

  where $\boldsymbol{\beta}^t := \boldsymbol{\Theta}^t - \boldsymbol{\Theta}^0$. (Here, corresponds to $\alpha = \sqrt{M}$ and M sufficiently large.)

- Take $b_j = 0$ and $\hat{f}_{\mathrm{NN}}(\boldsymbol{x}; \boldsymbol{\Theta}^0) = 0$. The *neural tangent* (NT) model is given by

$$\mathrm{NT}_M(\boldsymbol{x}; \boldsymbol{\beta}) = \frac{1}{\sqrt{M}} \sum_{j \in [M]} a_i \sigma(\langle \boldsymbol{w}_j^0, \boldsymbol{x} \rangle) + \frac{1}{\sqrt{M}} \sum_{j \in [M]} \langle \boldsymbol{q}_j, \boldsymbol{x} \rangle a_j^0 \sigma'(\langle \boldsymbol{w}_j^0, \boldsymbol{x} \rangle) .$$

- When $M = \infty$, kernel method with kernel (NTK):

$$K_{\mathrm{NT}}(\boldsymbol{x}_1, \boldsymbol{x}_2) = \mathbb{E}_{\boldsymbol{w}^0} \Big[\sigma(\langle \boldsymbol{w}^0, \boldsymbol{x}_1 \rangle) \sigma(\langle \boldsymbol{w}^0, \boldsymbol{x}_2 \rangle) + \langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle \sigma'(\langle \boldsymbol{w}^0, \boldsymbol{x}_1 \rangle) \sigma'(\langle \boldsymbol{w}^0, \boldsymbol{x}_2 \rangle) \Big] .$$

- For $M < \infty$, stylized NT model with $\boldsymbol{q}_j = 0$: the *Random Feature model*

$$\mathrm{RF}_M(\boldsymbol{x}; \boldsymbol{a}) = \frac{1}{\sqrt{M}} \sum_{j \in [M]} a_i \sigma(\langle \boldsymbol{w}_j^0, \boldsymbol{x} \rangle) .$$

  [Rahimi,Recht,’08]

Random Feature Ridge Regression

- **RF ridge regression:** take $\boldsymbol{x} \sim \mathrm{Unif}(\mathbb{S}^{d-1}(\sqrt{d}))$, $\boldsymbol{w}_j^0 \sim_{iid} \mathrm{Unif}(\mathbb{S}^{d-1}(1))$ and $\lambda \geq 0^+$,

$$\hat{\boldsymbol{a}}_\lambda = \arg\min_{\boldsymbol{a} \in \mathbb{R}^M} \left\{ \sum_{i=1}^{n} \Big(y_i - \frac{1}{\sqrt{M}} \sum_{j=1}^{M} a_j \sigma(\langle \boldsymbol{w}_j^0, \boldsymbol{x}_i \rangle) \Big)^2 + \lambda \|\boldsymbol{a}\|_2^2 \right\}.$$

- **Test error:** $R_{\mathrm{RF}}(f_*; n, M) = \mathbb{E}_{\boldsymbol{x}} \Big[\big(f_*(\boldsymbol{x}) - \mathrm{RF}_M(\boldsymbol{x}; \hat{\boldsymbol{a}}_\lambda) \big)^2 \Big]$.

Theorem (informal) [Hu, Lu, Misiakiewicz, '22]

For $f_* \in l^2$ P-sparse and σ 'generic', we get the asymptotic test error as $d, n, M \to \infty$ with $n \asymp d^{\kappa_1}$, $M \asymp d^{\kappa_2}$, for any $\kappa_1, \kappa_2 > 0$. In particular, if $d^\ell \leq \min(n, M) < d^{\ell+1}$, then

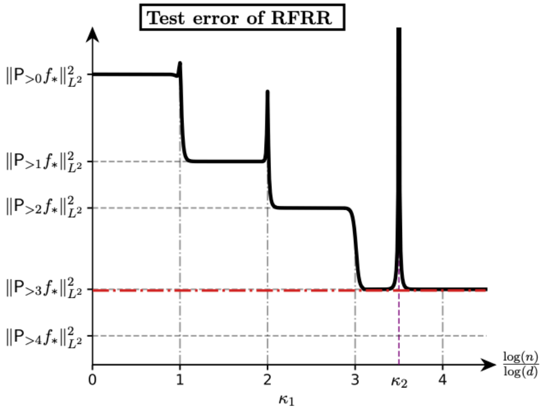
$$R_{\mathrm{RF}}(f_*; n, M) = \|\mathrm{P}_{>\ell} f_*\|_{L^2}^2 + o_{d,\mathbb{P}}(1)\,,$$

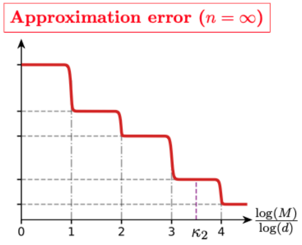
where $\mathrm{P}_{>\ell}$ is the projection orthogonal to the space of degree-ℓ polynomials.

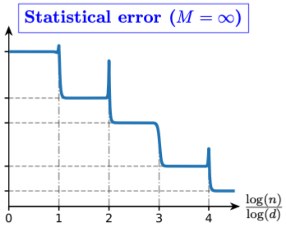
- If $\min(n, M) = \Theta_d(d^\ell)$, RFRR fits the best degree-ℓ polynomial approximation.
- **Statistical error** $= R_{\mathrm{RF}}(f_*; n, M = \infty)$ (kernel ridge regression with NTK).

  Approximation error $= R_{\mathrm{RF}}(f_*; n = \infty, M) = \min_{\boldsymbol{a}} \|f_* - \mathrm{RF}_M(\cdot; \boldsymbol{a})\|_{L^2}^2$.

Number of samples $n = d^{\kappa_1}$ and number of features $M = d^{\kappa_2}$:







Test error$(n, M) \approx \max\{$approximation error(M), statistical error$(n)\}$

Discussion: learning in the lazy training regime

- Linear method (convex): **efficient to solve.**
- **Simple approximation/statistical complexity trade-off:**
  stat error dominates for $M \geq n^{1+\varepsilon}$, while approx error dominates for $M \leq n^{1-\varepsilon}$.
- NNs in the lazy training regime:
  - Adaptive to smoothness (lower degree polynomials are easier to learn).
  - Not adaptive to latent low-dimensional subspace (bounds independent of sparsity).
- In this regime, NNs **do not break the CoD** on sparse functions: **no feature learning** (no alignment of the weights with the sparse support).

We need to go beyond this linear regime to break the CoD!

B. Convex Neural Networks

Intuition: why do kernel methods not adapt to sparsity?

- Define the set of **infinite-width 2-layer NNs**:

$$\mathcal{F} := \left\{ \hat{f}_{\mathrm{NN}}(\boldsymbol{x}; a) = \int_\Omega a(\boldsymbol{w})\sigma(\langle \boldsymbol{w}, \boldsymbol{x}\rangle)\mu(\mathrm{d}\boldsymbol{w}) \right\}, \qquad \|a\|_{L^2}^2 = \int_\Omega |a(\boldsymbol{w})|^2 \mu(\mathrm{d}\boldsymbol{w})$$

  - $\overline{\mathcal{F}}$ contain all finite-with NNs.
  - $\mathcal{F}_2 = \{f \in \mathcal{F} : \|a\|_{L^2} < \infty\}$ is a RKHS with norm $\|f\|_{\mathcal{F}_2} = \inf\{\|a\|_{L^2} : f = \hat{f}_{\mathrm{NN}}(\cdot; a)\}$.

- **Kernel ridge regression** with kernel $K(\boldsymbol{x}_1, \boldsymbol{x}_2) = \int_\Omega \sigma(\langle \boldsymbol{w}, \boldsymbol{x}_1\rangle)\sigma(\langle \boldsymbol{w}, \boldsymbol{x}_2\rangle)\mu(\mathrm{d}\boldsymbol{w})$:

$$\hat{f}_{2,n,\delta} = \arg\min_{f \in \mathcal{F}} \left\{ \sum_{i \in [n]} (y_i - f(\boldsymbol{x}_i))^2 + \lambda \|f\|_{\mathcal{F}_2}^2 \right\} = \arg\min_{\|f\|_{\mathcal{F}_2} \le \delta} \left\{ \sum_{i \in [n]} (y_i - f(\boldsymbol{x}_i))^2 \right\}.$$

"KRR = fitting the data with a low-RKHS norm neural network."

- **Learning** $f_*(\boldsymbol{x}) = \sigma(\langle \boldsymbol{w}_*, \boldsymbol{x}\rangle)$: intuitively we want to fit it with $a_*(\boldsymbol{w}) \propto \delta_{\boldsymbol{w}, \boldsymbol{w}_*}$
  However, for any $\|\hat{f}_{\mathrm{NN}}(a) - f_*\|_{L^2}^2 \le \varepsilon$, $\|a\|_{L^2} \asymp \left(\frac{1}{\varepsilon}\right)^d$ and we need to take $\delta \asymp \left(\frac{1}{\varepsilon}\right)^d$

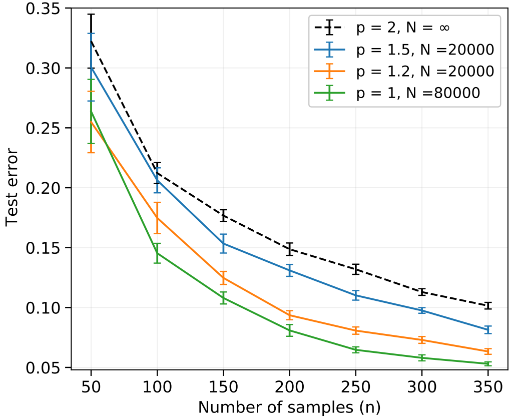
$$\text{Test error} \approx \frac{\delta^2}{\sqrt{n}} \text{ and therefore } n \asymp \left(\frac{1}{\varepsilon}\right)^d.$$

$\mathcal{F}_2$**-norm is not adapted to sparsity.**

What if instead of $\|a\|_{L^2}^2$, we regularize with $\|a\|_{L^p}^p$, $p < 2$?

Intuition: as p decreases, learning with $\|a\|_{L^p}$-regularization is better at capturing functions that are highly dependent on a low-dimensional subspace.

Take $\boldsymbol{x} \sim \text{Unif}(\mathbb{S}^{d-1}(\sqrt{d}))$, $\boldsymbol{w} \sim \text{Unif}(\mathbb{S}^{d-1}(1))$, ReLu, and $f_*(\boldsymbol{x}) = \sigma(x_1)$ $(d = 30)$:



Convex neural networks

- For $p \in [1,2]$, denote the $\mathcal{F}_p$-norm:

$$\|f\|_{\mathcal{F}_p} := \inf \left\{ \|a\|_{L^p} : f = \hat{f}_{\mathrm{NN}}(\cdot; a) \right\}, \qquad \|a\|_{L^p}^p = \int_\Omega |a(\boldsymbol{w})|^p \mu(\mathrm{d}\boldsymbol{w}) .$$

- Define the $\mathcal{F}_p$-regularized problems:

$$\hat{f}_{p,n,\delta} = \arg\min_f \left\{ \sum_{i \in [n]} (y_i - f(\boldsymbol{x}_i))^2 + \lambda \|f\|_{\mathcal{F}_p}^p \right\} = \arg\min_{\|f\|_{\mathcal{F}_p} \leq \delta} \left\{ \sum_{i \in [n]} (y_i - f(\boldsymbol{x}_i))^2 \right\} .$$

- 'Convex neural networks' [Bengio et al.,'06], [Bach,'17].

- Denote the $\mathcal{F}_p$-ball $\mathcal{F}_p(\delta) = \{ f : \|f\|_{\mathcal{F}_p} \leq \delta \}$. By Jensen's inequality,

$$\mathcal{F}_2(\delta) \subset \mathcal{F}_p(\delta) \subset \mathcal{F}_1(\delta) .$$

$\mathcal{F}_1(\delta)$ **contains** $\mathcal{F}_2(\delta)$ + **sparse functions:** e.g., $\sigma(\langle \boldsymbol{w}_*, \cdot \rangle) \in \overline{\mathcal{F}_1(1)} \setminus \overline{\mathcal{F}_2(1)}$.

Approximation/Generalization decomposition of $\mathcal{F}_p$-problems

Consider $\boldsymbol{x} \sim \mathrm{Unif}(\mathbb{S}^{d-1}(\sqrt{d}))$, $\boldsymbol{w} \sim \mathrm{Unif}(\mathbb{S}^{d-1}(1))$ and $\sigma(x) = (x)_+$.

- Denote $R(f, f_*) = \mathbb{E}[(f(\boldsymbol{x}) - f_*(\boldsymbol{x}))^2]$ and $\hat{R}_n(f, f_*) = \frac{1}{n}\sum_{i\in[n]}(f(\boldsymbol{x}_i) - f_*(\boldsymbol{x}_i))^2$.
- Decomposition of the test error for $\mathcal{F}_p$-problem with δ-regularization

$$R(\hat{f}_{p,n,\delta}, f_*) \leq \underbrace{\left[\inf_{f\in\mathcal{F}_p(\delta)} R(f, f_*)\right]}_{\text{Approx. error}} + \underbrace{2 \sup_{f\in\mathcal{F}_p(\delta)} \left|\hat{R}_n(f, f_*) - R(f, f_*)\right|}_{\text{Generalization error}} .$$

- Uniform deviation bound on generalization error [Bach,'17]:

$$\mathbb{E}_{\boldsymbol{X}}\left[\sup_{f\in\mathcal{F}_p(\delta)} \left|\hat{R}_n(f, f_*) - R(f, f_*)\right|\right] \leq C(\|f_*\|_\infty + \delta)\frac{\delta}{\sqrt{n}}.$$

Breaking the CoD on sparse functions with $\mathcal{F}_1$-problem [Bach,'17]

Take f_* P-sparse such that $f_*(\boldsymbol{x}) = h_*(\boldsymbol{Px})$ that is 1-Lipschitz:

- Bound on the approximation error [Bach,'17]:

$$\inf_{f \in \mathcal{F}_{\mathbf{1}}(\delta)} \|f - f_*\|_\infty \leq C\delta^{-\Theta(1/P)} .$$

- **Breaking CoD in statistical complexity:**

$$R(\hat{f}_{1,n,\delta}, f_*) \leq C\delta^{-\Theta(1/P)} + C\frac{\delta^2}{\sqrt{n}} \asymp C\frac{\log(n)}{n^{\Theta(1/P)}} ,$$

  by taking $\delta \asymp n^{\Theta(1)}$.

- **Breaking CoD in approximation:** $M = \left(\frac{1}{\varepsilon}\right)^{\Theta(P)}$ are sufficient to get ε-approximation of f_* with M neurons [Matousek,'96],[Bach,'17].

Can we solve efficiently the $\mathcal{F}_1$-problem?

Random feature approximation to $\mathcal{F}_p$-problems

- $\mathcal{F}_p$-problems are convex problems but on infinite-dimensional space.
  For $p = 2$, we can use the kernel trick, but for $p < 2$, tractability is unclear.

- Approximate μ by finitely supported $\hat{\mu}_M$: sample M weights $\boldsymbol{w}_j \sim_{iid} \mu$,

$$\hat{f}_{\mathrm{NN}}(\boldsymbol{x}; a) = \int_\Omega a(\boldsymbol{w})\sigma(\langle \boldsymbol{x}, \boldsymbol{w}\rangle)\mu(\mathrm{d}\boldsymbol{w}) \longrightarrow \hat{f}_{\mathrm{NN}}^{(M)}(\boldsymbol{x}; \boldsymbol{a}) = \frac{1}{M}\sum_{j\in[M]} a_j\sigma(\langle \boldsymbol{x}, \boldsymbol{w}_j\rangle)\,.$$

- $\mathcal{F}_p$-RF-problem: finite-width problem

$$\hat{\boldsymbol{a}}_\lambda = \arg\min_{\boldsymbol{a}\in\mathbb{R}^M} \Big\{ \sum_{i\in[n]} \big(y_i - \hat{f}_{\mathrm{NN}}^{(M)}(\boldsymbol{x}_i; \boldsymbol{a})\big)^2 + \frac{\lambda}{M}\sum_{j\in[M]} |a_j|^p \Big\}\,.$$

Theorem (informal) [Celentano, Misiakiewicz, Montanari, '21]

For $p > 1$, $M = n^{\Theta(1/(p-1))}$ are sufficient and necessary to approximate the $\mathcal{F}_p$-problem.

- **Upper bound:** concentration of the landscape of the dual problem.
  Lower bound: impossibility of fitting a single ReLu with random features.

Learning with $\mathcal{F}_p$-problem

- Learning $\mathcal{F}_p(1)$-functions for $p > 1$: to get a $O(\varepsilon)$-test error
  - With $\mathcal{F}_p$-regularization:

$$\text{Approx. error: } \delta = 1\,, \qquad \text{Gen. error: } n = \Theta\left(\frac{1}{\varepsilon}\right)^2, \qquad \text{Comp. time: } T \asymp \left(\frac{1}{\varepsilon}\right)^{\Theta\left(\frac{1}{p-1}\right)}.$$

  - But with $\mathcal{F}_q$-regularization, $q > p$: $\delta \geq \left(\frac{1}{\varepsilon}\right)^{\Omega((q-p)d)}$ and $n \geq \left(\frac{1}{\varepsilon}\right)^{\Omega((q-p)d)}$.

  $\mathcal{F}_p$-problems are **tractable** methods that **break the CoD on an increasing set of functions** $\mathcal{F}_p(1)$ as $p \searrow 1$.

- Learning $\mathcal{F}_1(1)$-functions (that contains sparse functions) with $\mathcal{F}_p$-regularization:

$$\text{Approx: } \delta \geq \left(\frac{1}{\varepsilon}\right)^{\Omega((p-1)d)}, \qquad \text{Gen: } n \geq \left(\frac{1}{\varepsilon}\right)^{\Omega((p-1)d)}, \qquad \text{Comp: } T \geq \left(\frac{1}{\varepsilon}\right)^{\Omega(d)}.$$

  Taking $p = 1 + \frac{1}{d}$ **breaks CoD in stat. complexity**. But **computational bottleneck.**

- **Evidence that this bottleneck is fundamental:**
  - $\mathcal{F}_1(1)$: NP-hard to learn (dist. free) [Neyshabur et al.,'15], [Celentano,**M.**,Montanari,'21].
  - $\mathcal{F}_1(d)$ has superpolynomial statistical dimension (CSQ algos will fail).

Discussion: learning with convex neural networks

- For $p > 1$, convex NNs are tractable methods that break the CoD on an increasing set of functions as $p \searrow 1$ (they can perform 'feature learning').
- For sparse functions, taking $p = 1 + \frac{1}{d}$ breaks the CoD in statistical complexity.

  However, $\mathcal{F}_p$-RF methods need $\left(\frac{1}{\varepsilon}\right)^d$ computational time for any $p \in (1, 2]$.
- This "computational bottleneck" is fundamental:
  - The class of sparse functions is hard to learn in worst case.
  - From SQ/CSQ-type bounds, some sparse functions will be much harder to learn than others.
- In this sense, while sparsity is a good complexity measure for approximation and generalization, it is **not a tight measure for computational complexity.**

  Some sparse functions are easier to learn than others. Sparsity is too coarse a measure.

Question:

Which sparse functions are efficiently learned by SGD-trained NNs?

C. Online-SGD in the mean-field scaling

Setting

- **Data on the hypercube:** $\boldsymbol{x} \sim \mathrm{Unif}(\{+1,-1\}^d)$ and $f_*(\boldsymbol{x}) = h_*(\boldsymbol{z})$ where $\boldsymbol{z} \in \{\pm 1\}^P$ is the latent support.

- **2-layer NN:** M hidden units and $\boldsymbol{\Theta} = (\boldsymbol{\theta}_j)_{j\in[M]} = (a_j, \boldsymbol{w}_j)_{j\in[M]} \in \mathbb{R}^{M(d+1)}$

$$\hat{f}_{\mathrm{NN}}(\boldsymbol{x};\boldsymbol{\Theta}) = \frac{1}{M}\sum_{j\in[M]} a_j\sigma(\langle \boldsymbol{w}_j, \boldsymbol{x}\rangle)\,.$$

- Fit the target function f_* by minimizing

$$\min_{\boldsymbol{\Theta}} R(f_*,\boldsymbol{\Theta}) = \mathbb{E}_{\boldsymbol{x}}\left[\left(f_*(\boldsymbol{x}) - \hat{f}_{\mathrm{NN}}(\boldsymbol{x};\boldsymbol{\Theta})\right)^2\right]\,.$$

- **Online SGD in the 'mean-field scaling':**
  - *Initialization:* $(\boldsymbol{\theta}_j)_{j\in[M]} \sim_{iid} \rho_0$.
  - *Update:* at each step k, fresh sample $(\boldsymbol{x}_k, y_k)$ with $y_k = f_*(\boldsymbol{x}_k) + \varepsilon_k$,

$$\boldsymbol{\theta}_j^{k+1} = \boldsymbol{\theta}_j^k + \eta(y_k - \hat{f}_{\mathrm{NN}}(\boldsymbol{x}_k;\boldsymbol{\theta}^k))\cdot\nabla_{\boldsymbol{\theta}_j}\left[a_j\sigma(\langle \boldsymbol{w}_j^k, \boldsymbol{x}_k\rangle)\right]\,.$$

Examples:

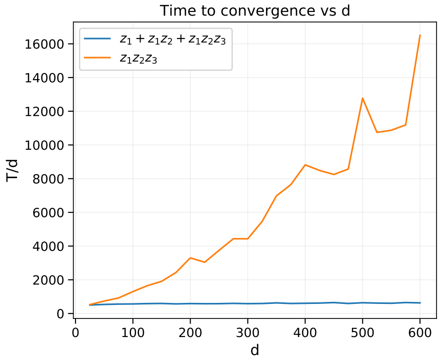
$$h_{*,1}(\boldsymbol{z}) = z_1 + z_1 z_2 + z_1 z_2 z_3 \,, \qquad h_{*,2}(\boldsymbol{z}) = z_1 z_2 z_3 \,.$$

Are these 2 functions equivalent for SGD-trained NNs? Which one is easier to learn?

Examples:

$$h_{*,1}(\boldsymbol{z}) = z_1 + z_1 z_2 + z_1 z_2 z_3\,, \qquad h_{*,2}(\boldsymbol{z}) = z_1 z_2 z_3\,.$$

Take $M = 300$, $\eta = \frac{1}{d}$ and shifted sigmoid activation.
Number of iterations to convergence vs d:



Can we characterize which sparse functions are learnable by SGD in $O(d)$ steps?

Merged staircase property

Fourier coefficient for $S \subseteq [P]$: $\hat{h}_*(S) = \mathbb{E}_{\boldsymbol{z}}\left[h_*(\boldsymbol{z})\chi_S(\boldsymbol{z})\right]$ where $\chi_S(\boldsymbol{z}) = \prod_{i \in S} z_i$.

$$h_*(\boldsymbol{z}) = \sum_{S \in \mathcal{Q}} \hat{h}_*(S)\chi_S(\boldsymbol{z}),$$

where $\mathcal{Q}$ contains all non-zero Fourier coefficients $\hat{h}_*(S) \neq 0$.

Merged-Staircase property (MSP)

$h_* : \{-1,+1\}^P \to \mathbb{R}$ has the *merged-staircase property* (MSP) if we can write elements of $\mathcal{Q}$ in order $(S_1, \ldots, S_r)$ such that for any $j \in [r]$, we have $|S_j \setminus (S_1 \cup \ldots \cup S_{j-1})| \leq 1$.

Examples of MSP functions:

$$h_*(\boldsymbol{z}) = z_1 + z_1z_2 + z_1z_2z_3 + z_1z_2z_3z_4\,,$$
$$h_*(\boldsymbol{z}) = z_1 + z_1z_2 + z_2z_3 + z_3z_4 + z_3z_4z_5\,.$$

Examples of non-MSP functions:

$$h_*(\boldsymbol{z}) = z_1 + z_1z_2z_3 + z_1z_2z_3z_4\,,$$
$$h_*(\boldsymbol{z}) = z_1 + z_1z_2 + z_3z_4 + z_3z_4z_5\,.$$

MSP is necessary and nearly sufficient

Theorem [Abbe,Boix-Adsera,Misiakiewicz,'22]

MSP is necessary and nearly sufficient* to learn h_* in $O(d)$ steps/samples of online SGD** in the mean-field regime.

*Excludes a set of MS fcts $h_*(\boldsymbol{z}) = \sum_{S \in \mathcal{Q}} h_*(S)\chi_S(\boldsymbol{z})$ with $\{h_*(S)\}_{S \in \mathcal{Q}}$ of measure 0. (This is unavoidable: some degenerate cases of MS functions are not learned)

**For the sufficiency result, we train the first layer, then the second layer.

Example

$$\underbrace{h_{*,1}(\boldsymbol{z}) = z_1 + z_1 z_2 + z_1 z_2 z_3}_{K = O_d(d) \text{ online SGD steps is enough}}, \qquad \underbrace{h_{*,2}(\boldsymbol{z}) = z_1 z_2 z_3}_{\text{needs } K \gg d \text{ steps}}.$$

MS functions are learned by online-SGD in mean-field scaling with $M = O_d(1)$ and $n = K = O_d(d)$ ($K =$ number of iterations).

What about non-MS functions?

Intuition for MS functions: SGD sequentially aligns the weigths with the support of h_*, adding one coordinate at a time (which can be done efficiently).

What about non-MS functions? Consider learning a k-parity function:

$$h_*(\boldsymbol{z}) = z_1 z_2 \cdots z_k \, .$$

Theorem [Abbe,Boix-Adsera,Misiakiewicz,'22]

Take a shifted ReLU and $M = \varepsilon^{-C} 2^{Ck}$ neurons, online-SGD on hinge-loss in 2 phases:

1. $\tilde{O}_d(d^{k-1})$ steps on first layer $\boldsymbol{w}_j$'s with step size $\eta = \tilde{O}_d(d^{-k/2})$;
2. $O_d(1)$ steps on the second layer weights a_j's with step size $\eta = O_d(1)$;

achieves ε-error with probability at least $1 - d^{-C}$.

Intuition: it is harder for SGD to align the weights to several coordinates at once. It takes $\tilde{O}(d^{k-1})$ SGD steps for k coordinates.

Computational-lower bound for learning parities

Can we hope to do better than $K = \tilde{O}(d^{k-1})$ steps?

- Total computational time to learn parities with online-SGD from previous theorem:

$$T = O(KMd) = \tilde{O}_d(d^k)\,.$$

- Computational lower-bound for SQ algorithms is $\Omega(d^k)$.

- Online-SGD **nearly matches this computational lower-bound** and we don't expect regular training of regular neural networks to go beyond SQ-algorithms.

Open question:

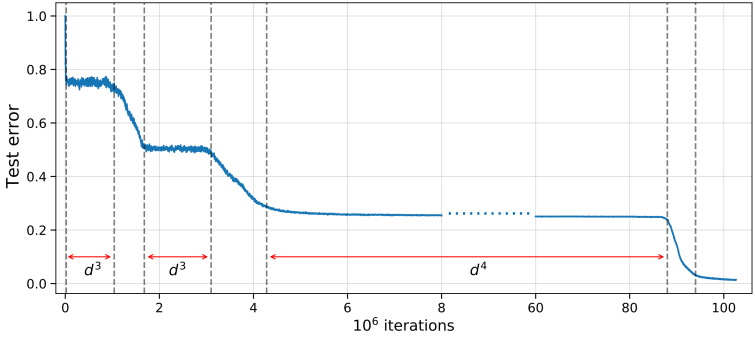
What about the optimal sample complexity?

Leap staircases: saddle-to-saddle behavior in SGD-dynamics

So far: MS-functions and parity functions. **What about more general functions?**

Staircase functions with leaps:

$$h_*(\boldsymbol{z}) = z_1 + z_1 z_2 z_3 z_4 z_5 + z_1 z_2 \cdots z_9 + z_1 z_2 \cdots z_{14} \,.$$



Picture: [Abbe,Boix,**Misiakiewicz**,'22] SGD learns sequentially the monomials and at each leap of size-k, there is a plateau and it takes $\tilde{O}_d(d^{k-1})$ iterations to escape.

Again, nearly matches the SQ-computational lower-bound for learning this class of fcts.

Discussion: learning in the mean-field scaling

For learning sparse functions on the hypercube:

- **A staircase-like structure is necessary for efficient learning.**
- **Conjectural picture:** sparse functions classified in sets $\{k\text{-leap MSP}\}_{k\geq 1}$ such that online-SGD on 2-layer NNs
  - 1-leap MSP needs $\Theta(d)$ SGD steps to learn.
  - k-leap MSP for $k \geq 2$ needs $\tilde{\Theta}(d^{k-1})$ SGD steps to learn.
- Mean-field regime has 'near-optimal' feature learning for sparse functions:
  - Only need $M = g(P)$ independent of d.
  - Nearly matches SQ-computational lower bounds on leap staircases.

Open questions:

- What about the sample complexity?
- What about less sparse functions $P = \omega_d(1)$? Using higher-depth NNs?
- What happens in real data? What are the structures in real data that enables efficient feature learning? The staircase picture might be too naive.

Summary

- Fixed-feature methods (such as NNs in the lazy training regime) are **not able to adapt to sparsity** and suffer from the CoD.

  We need non-linear training, where computatioinal efficiency is not guaranteed.

- While sparsity is a good complexity measure for approximation and generalization, it is **not a tight measure for computational complexity.**

- **Which sparse functions can be efficiently learned by SGD-trained NNs?**

- On hypercube, **staircase-like structure is necessary for efficient feature learning.**

  Intuitively, SGD sequentially aligns the weights with the support of the sparse function. **Lower degree monomials help to learn higher degree monomials.**

- **Lots of open questions!**

Thank you!*

*I am looking for a job!