Neural network learning and complexity measures

Math-700

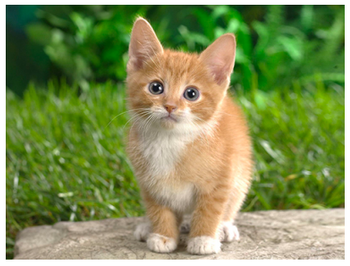
trained by
descent algorithms:
SGD/GD/…

What functions can neural networks learn well?

sample / time
complexity

Supervised learning

$$x \overset{f}{\longmapsto} y$$

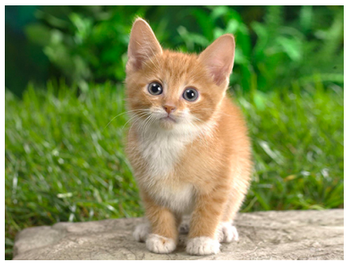


$$\hat{f}_{NN}$$

cat

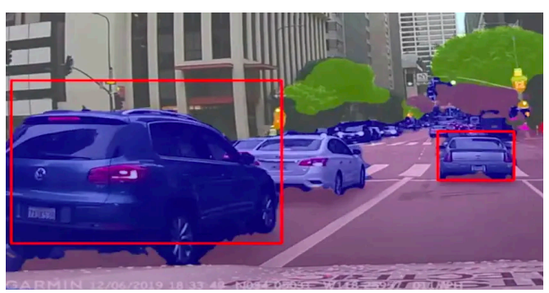
Image recognition

6



cat

Vanessa
atalanta
butterfly

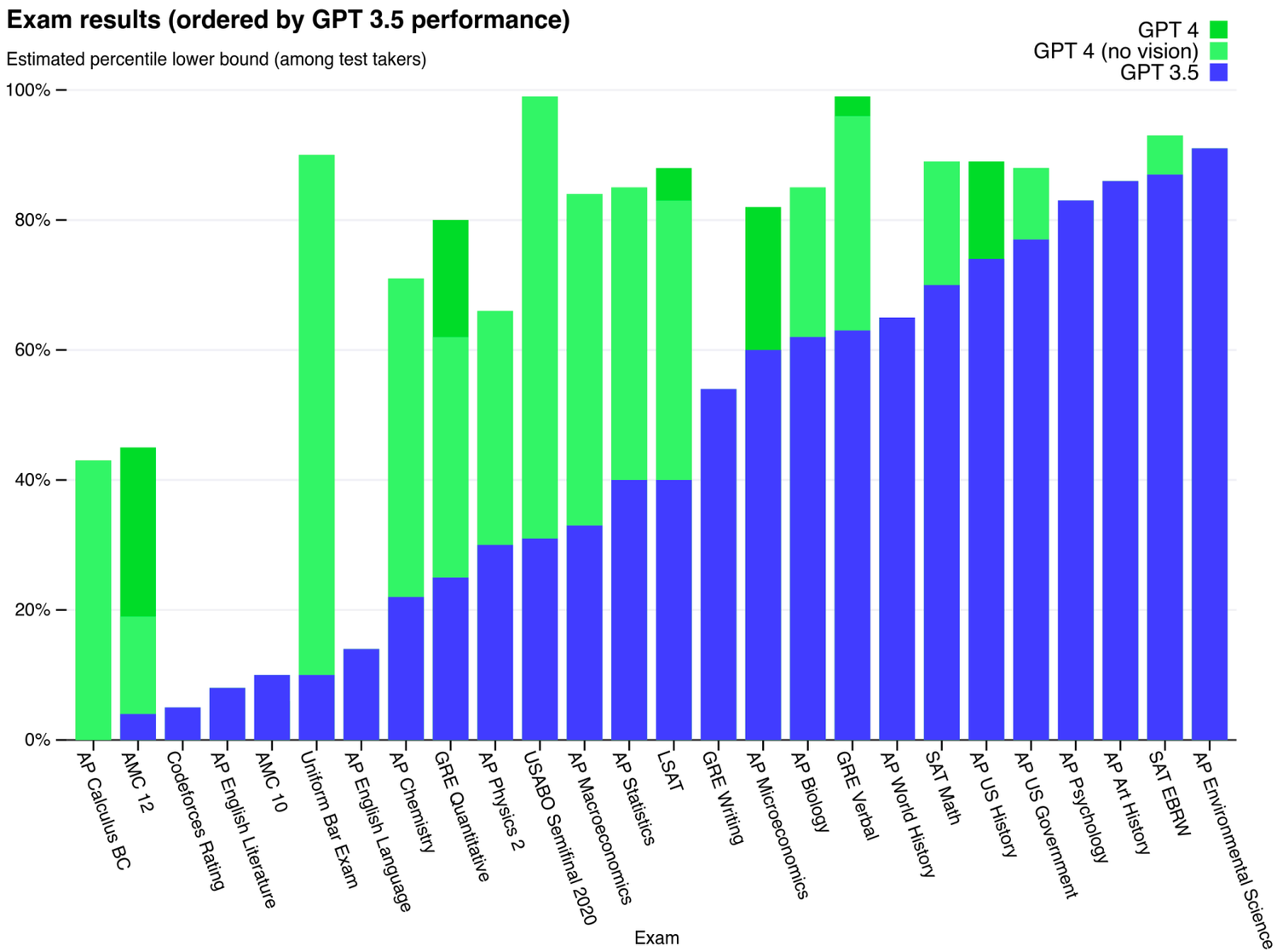


break

[MNIST, CIFAR, IMAGENET, etc]

Text completion

Milan is a great… → city



https://openai.com/research/gpt-4

More complex reasoning?

Mary has twice the age of John. Two years ago she had three times the age of John. How old is Mary?

→ 8

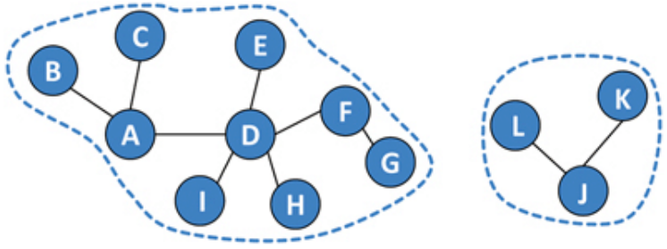
$$(1, -1, -1, 1, 1, -1, 1, -1, 1, 1, -1)$$

→ -1

$f : \{0,1\}^d \to \mathcal{Y} \subseteq \mathbb{R}$

$f : \{-1,1\}^d \to \mathcal{Y} \subseteq \mathbb{R}$

$f : [q]^d \to \mathcal{Y} \subseteq \mathbb{R}$



→ 2

AVIS GERIATRIQUE (0 = NEG, 1 = POS)	Âge	Euroscore	STS risk
1	65	6.98	6.125
1	71	4.03	4.678
0	72	4.23	6.718
1	74	1.64	1.238
1	75	0.73	1.79
1	75	1.89	1.406
0	76	9.27	12.613
0	76	2.35	2.678
1	76	5.25	5.675
1	77	5.36	5.895

→

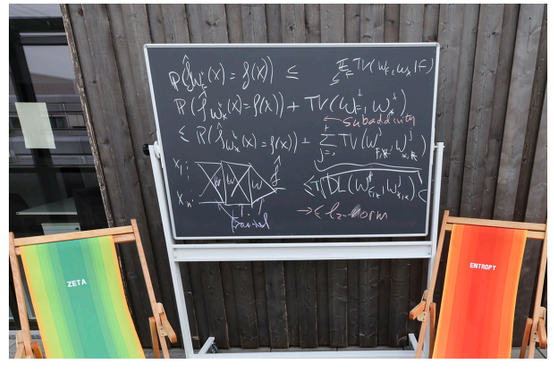
HEART TEAM (0 = CHIRURGIE; 1= TAVI, 2 =Conservateur)
1
1
1
1
1
1
1
1
1
1

```
theorem PFR_conjecture {G : Type*} [AddCommGroup G] [ElementaryAddCommGroup G 2] [Fintype G]
    [DecidableEq G] {A : Set G} {K : ℝ} (h₀A : A.Nonempty)
    (hA : Nat.card (A + A) ≤ K * Nat.card A) : ∃ (H : AddSubgroup G) (c : Set G),
    Nat.card c ≤ 2 * K ^ 12 ∧ Nat.card H ≤ Nat.card A ∧ A ⊆ c + H := by
```

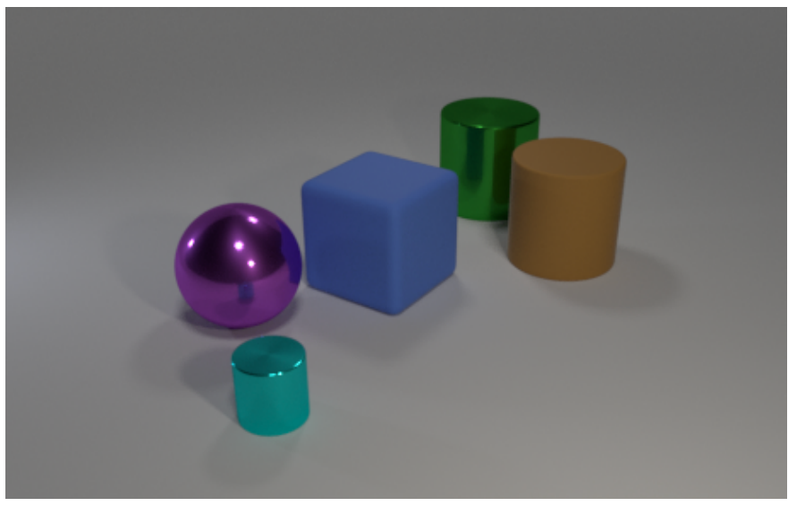
→ true

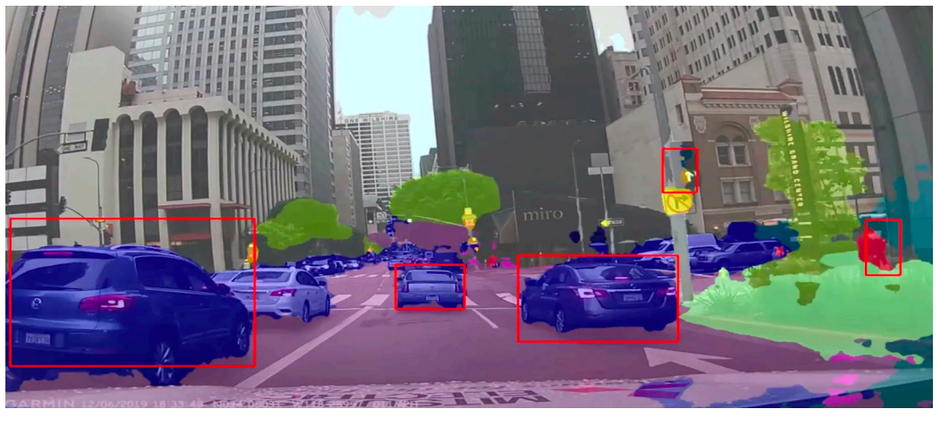
[Saxton et al. 19, Lewkowycz et al. 22,PVR,Zheng et al. 21,Zhang et al 22,Algorithmic CLRS, Mahdavi et al. 22,GSM8K, LogiQA]

Recognition + logic



PVR: output the label of the MNIST digit indexed by the first MNIST digit -> 1





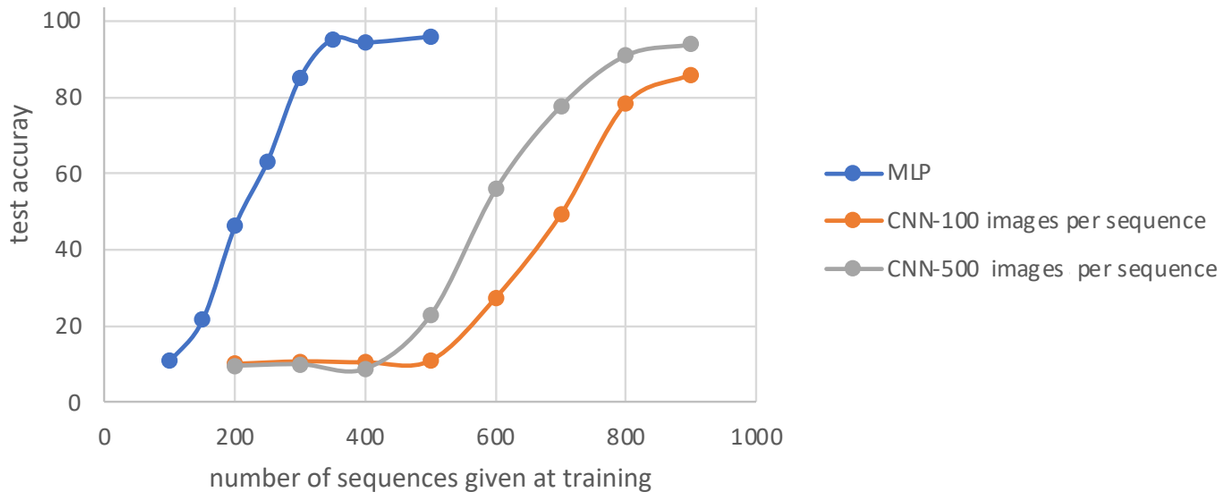
CLEVR: is the sphere on the left of the cube -> yes

Recognition + logic

$$f = \text{logic} \circ \text{recog}$$

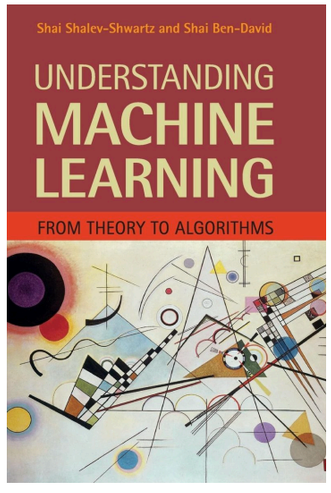
Learning on the “signal” domain seems at least as hard
(e.g., in sample complexity) as learning on the “symbolic” domain
if the “signal” domain is not *label-leaky*

target = sum mod 10



4-MLP (512/256/128/64). Adam l.r.=1e-4 / batch=64 / Epochs: 1500 (MLP), 2200 (CNN)

Basic generalization notions



Basic Supervised Learning: Mathematical Formulation

Data Distribution 📊

Supervised learning assumes the existence of an unknown, underlying **joint probability distribution** $P(X, Y)$ from which all data is sampled.

- X is the vector of input features.
- Y is the output variable or label.
- We have a labeled training dataset $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, where each (x_i, y_i) is an independent and identically distributed (i.i.d.) sample from $P(X, Y)$.

The goal is to learn a function $h : X \to Y$ (our model) that approximates the true, but unknown, mapping function $f : X \to Y$. The ideal function, called the **Bayes classifier** or **Bayes predictor**, minimizes the expected error, but is unattainable since we don't know $P(X, Y)$.

Training Error (Empirical Risk) 📈

The training error, also known as **empirical risk**, measures how well our model h performs on the **training data** D. It's the average loss over the training set.

- **Loss Function** $L(y, \hat{y})$: A function that quantifies the penalty for a predicted value $\hat{y}$ when the true value is y.
    - For regression (continuous output): Mean Squared Error (MSE), $L(y, \hat{y}) = (y - \hat{y})^2$.
    - For classification (discrete output): 0-1 loss, $L(y, \hat{y}) = 1$ if $y \neq \hat{y}$ and 0 otherwise.
- **Training Error Formula:**

$$E_{train}(h) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, h(x_i))$$

The learning algorithm's objective is to find the function h that **minimizes this training error**.

Testing Error (Generalization Error) 📉

The testing error, or **generalization error**, measures how well our model h will perform on **new, unseen data** sampled from the same data distribution $P(X, Y)$. It's the expected loss over the entire data distribution.

- **Testing Error Formula:**

$$E_{test}(h) = E_{X,Y}[L(Y, h(X))] = \int L(y, h(x))\, dP(x, y)$$

Since we don't know $P(X, Y)$, we can't calculate this exactly. We **estimate** it using a separate **test dataset** that the model has not seen during training.

- **Key Concept:** The ultimate goal of supervised learning is to minimize the **testing error**, not just the training error. A low training error combined with a high testing error indicates **overfitting**, where the model has memorized the training data rather than learned the underlying patterns.

- **Approximation Error**

$$\min_{h \in \mathcal{H}} R(h) - R(h^*)$$

This is the gap between the best possible performance of a model from the hypothesis space $\mathcal{H}$ and the true Bayes-optimal performance. It is a measure of the bias of the model class. A more complex model class (larger $\mathcal{H}$) will have a smaller approximation error.

- **Estimation Error**

$$\min_{h \in \mathcal{H}} R_D(h) - \min_{h \in \mathcal{H}} R(h)$$

This is the gap between the minimum empirical risk on the training data D and the minimum true risk achievable within the hypothesis space $\mathcal{H}$. It arises because we only have a finite training set, which is a noisy proxy for the true data distribution. This is a measure of the variance of the learning process. As the size of the training data $|D|$ increases, this error term tends to decrease.

- **Optimization Error**

$$E_D[R(\hat{h})] - \min_{h \in \mathcal{H}} R_D(h)$$

This is the gap between the performance of the model $\hat{h}$ found by our learning algorithm and the best possible model that could have been found by perfectly minimizing the empirical risk on the training data. It exists because many optimization problems are computationally difficult, and algorithms may not find the true global minimum of the training error.

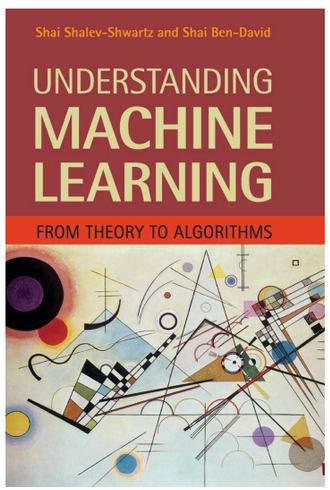
$$= \underbrace{(E_D[R(\hat{h})] - R(h_{ERM}))}_{\text{Optimization Error}} + \underbrace{(R(h_{ERM}) - R(h_{\mathcal{H}}))}_{\text{Estimation Error}} + \underbrace{(R(h_{\mathcal{H}}) - R(h^*))}_{\text{Approximation Error}}$$

What Boolean/logic functions can neural networks learn efficiently?

$$f : \{0,1\}^d \to \mathcal{Y} \subseteq \mathbb{R}$$

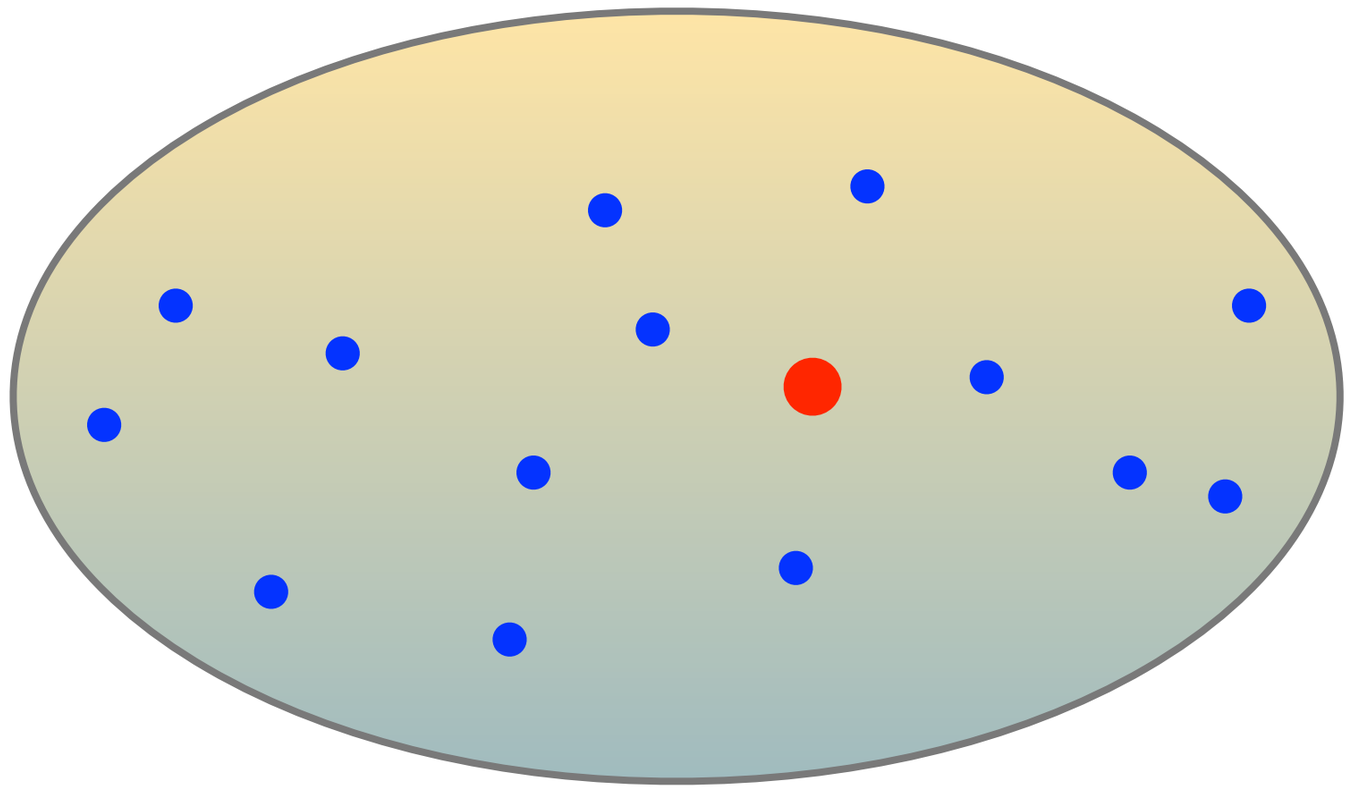
$$f : \{-1,1\}^d \to \mathcal{Y} \subseteq \mathbb{R}$$

$$f : \{0,1,\ldots,q-1\}^d \to \mathcal{Y} \subseteq \mathbb{R}$$



Lots in PAC/SQ setting -> Our interest: NN+SGD setting

Classic generalization



Classic generalization: $GEN(f, \tilde{f}, \mu) = \mathbb{E}_{X \sim \mu}[\ell(\tilde{f}_{X^m \sim \mu^m}(X), f(X))]$

-> what is the network model?

What is the network model?

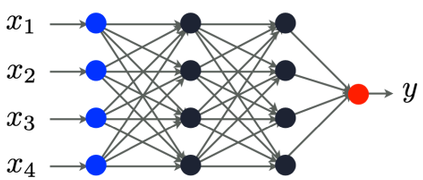
Regular-networks

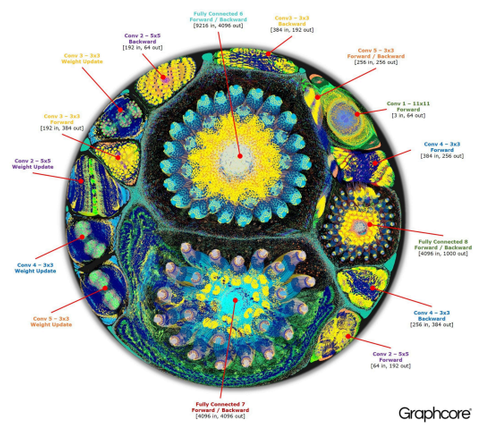
-> fully connected (iid) layers (2)

general-networks

-> polynomial-size

MLP CNN/ResNN/RecNN/GNN.. Transformers





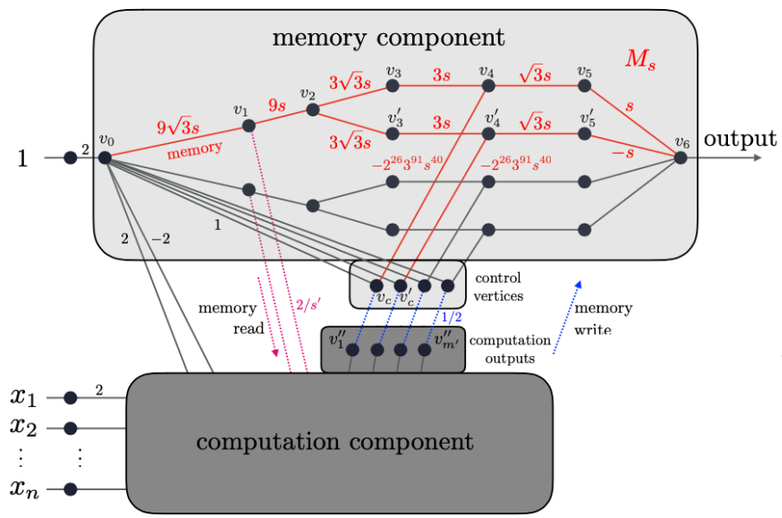
General networks

Full freedom on the NN architecture besides having polynomial size in input dimension

$$\texttt{Free\_NN\_SGD} = \{\text{Data dist. class learnable by some poly-size NN trained by SGD with all param. polynomial}\}$$

$$\texttt{PAC} = \{\text{Data dist. class learnable by some poly-time algo.}\}$$

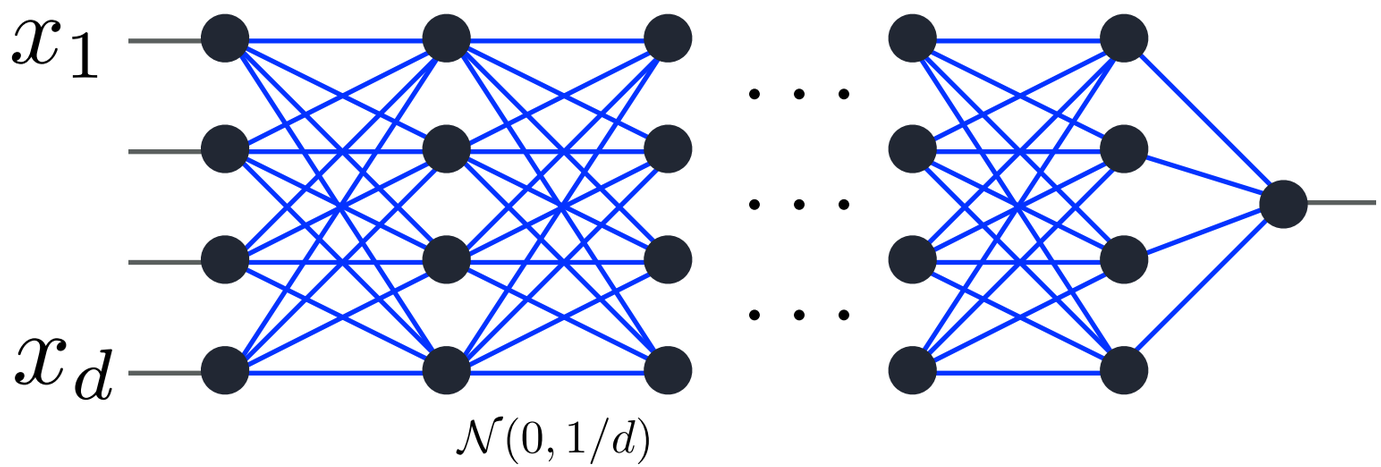
Theorem [Abbe-Sandon CPAM'22]. $\texttt{Free\_NN\_SGD} = \texttt{PAC}$



Transformers and alike:
[Wei, Chen, Ma'22]: approximation
[Pérez, Marinković,Barceló '23]: approximation
[Malach '23]: existence of data with CoT

Regular networks

The NN is constrained to be more ‘regular’: E.g., MLP with iid layers



Consider: $f : \{+1, -1\}^d \to \mathcal{Y} \subseteq \mathbb{R}$ $X_1, \ldots, X_d \overset{iid}{\sim} \mathcal{U}\{+1, -1\}$

Examples: $f_1(x) = x_1$ $f_2(x) = x_1 x_2$ $f_3(x) = x_1 + x_1 x_2$

[JGH18; LL18; Du+18; Du+19; ALS19; ALL19; Aro+19; Zou+20; OS20] [Bac17; Gho+21b; DM20; Ref+21; AL19; Gho+19; YS19; AL20a; LMZ20]

Lazy Regime (NTK Parametrization) 😴

This regime is characterized by **small learning rates** and an initialization that ensures the network's dynamics are linearizable. The key is the scaling of the output layer weights.

- **Neural Network Function Formula:**
  The network's output is scaled by $1/\sqrt{m}$ on the output layer.

$$f(x) = \frac{1}{\sqrt{m}} W_2^T \sigma(W_1 x)$$

- **Weight Initialization:**
  Both weight matrices are initialized from a standard normal distribution, typically scaled by a factor related to their input size, e.g., $\mathcal{N}(0, 1)$.
- **Learning Rates:**
  The learning rates for both layers are small. Specifically, the learning rate for the hidden layer weights, η_{W_1}, is often scaled by $1/m$, and the learning rate for the output layer, η_{W_2}, is of order $O(1)$.

$$\eta_{W_1} \sim O(1/m), \quad \eta_{W_2} \sim O(1)$$

This ensures that weight updates are small and the network behaves like a kernel method.

Feature Learning Regime (μP Parametrization) 🧠

This regime allows for significant weight updates, particularly in the hidden layer, enabling the network to learn new features. The key is to prevent the gradient from vanishing.

- **Neural Network Function Formula:**
  The network's output is not scaled by the hidden layer width.

$$f(x) = W_2^T \sigma(W_1 x)$$

- **Weight Initialization:**
  W_1 is initialized from $\mathcal{N}(0, 1)$, while the output weights W_2 are initialized from a smaller distribution, typically $\mathcal{N}(0, 1/m)$.
- **Learning Rates:**
  The learning rate for the hidden layer weights, η_{W_1}, is kept at a constant order, while the learning rate for the output layer is scaled.

$$\eta_{W_1} \sim O(1), \quad \eta_{W_2} \sim O(1)$$

This allows the updates to W_1 to be substantial, allowing the network to learn new features.

Misiakiewicz-Montanari lecture notes

Boolean Fourier analysis

Boolean Fourier analysis. $f : \{+1, -1\}^d \to \mathbb{R}$ can be decomposed as

$$f = \sum_{S \subseteq [d]} \hat{f}_S \chi_S \quad \text{where} \quad \hat{f}_S = \langle f, \chi_S \rangle,\ \chi_S(x) = \prod_{i \in S} x_i$$

E.g.: $f(x) = 0.2x_1 - 0.5x_1x_2 + 0.9x_2x_7x_8x_9$

Assume f has sparse support: involves $P = O_d(1)$ variables (e.g., $P = 5$ above).

Many useful complexity measures of f rely on the Fourier transform.

$$\text{E.g., Stability}_\epsilon(f) = \mathbb{E} f(X) f(X_\epsilon) \sim \sum_{S \subseteq [d]} (1 - 2\epsilon)^{|S|} \hat{f}_S^2$$

Leap complexity

Let $S_1^{(f)}, \ldots, S_L^{(f)}$ be the non-zero basis elements of f

E.g., $f(x) = 0.2x_1 - 0.5x_1x_2 + 0.9x_2x_7x_8x_9 \quad \rightarrow \; S_1^{(f)} = \{1\}, S_2^{(f)} = \{1,2\}, S_3^{(f)} = \{2,7,8,9\}$

$$\text{Leap}(f) := \min_{\pi \in \Pi_L} \max_{k \in [L]} |S^{(f)}_{\pi(k)} \setminus \cup_{j=0}^{k-1} S^{(f)}_{\pi(j)}|$$

$\text{Leap}(x_1) = 1$ $\quad$ $\text{Leap}(x_1x_2) = 2$ $\quad$ $\text{Leap}(x_1 + x_1x_2) = 1$

$\text{Leap}(x_1 + x_2x_3) = 2$

same stability but
different leap

Main result: leap control and staircases

'Theorem' (Abbe-Boix-Misiakiewicz NeurIPS'22,COLT'23).
Let f be a sparse Boolean function with inputs drawn uniformly in $\{\pm 1\}^d$.
Let $\tilde{f}^{(t)}_{\mathrm{NN}}$ be the output of training a 2-layer neural network with poly(d)-edges and isotropic layers at initialization with t steps of online-SGD on the square loss. Then, for all but a measure-0 set of functions,

$$\mathbb{E}\ell(\tilde{f}^{(t)}_{\mathrm{NN}}, f) \le \epsilon \quad \text{if and only if} \quad t = \tilde{\Omega}_d(d^{\mathrm{Leap}(f)-1\vee 1})\mathrm{poly}(1/\epsilon)\,.$$

Let $\tilde{f}^{(t)}_{\mathrm{Ker}}$ be the output of training a Kernel with t samples/features. Then,

$$\mathbb{E}\ell(\tilde{f}^{(t)}_{\mathrm{Ker}}, f) \le \epsilon \quad \text{if and only if} \quad t = \Omega_d(d^{\mathrm{Deg}(f)\vee 1})\mathrm{poly}(1/\epsilon)\,.$$

$f(x) =$	$x_1x_2\cdot\ldots\cdot x_k$ (parity)	$x_1 + x_1x_2 + \ldots + x_1x_2\cdot\ldots\cdot x_k$ (staircase)
Kernels	d^k	d^k
RegNN+SGD	d^k	d

Conclusion:
Regular-NN can learn efficiently - more than kernels -
where targets have a hierarchical Fourier structure (low leap),
but otherwise not as efficiently as free-NN/PAC (e.g., parities)

Technical innovation:

1) “If and only if” statement with new leap complexity measure

spectral bias [Rahaman et al ‘19]

2) Control of the full sample+time complexity

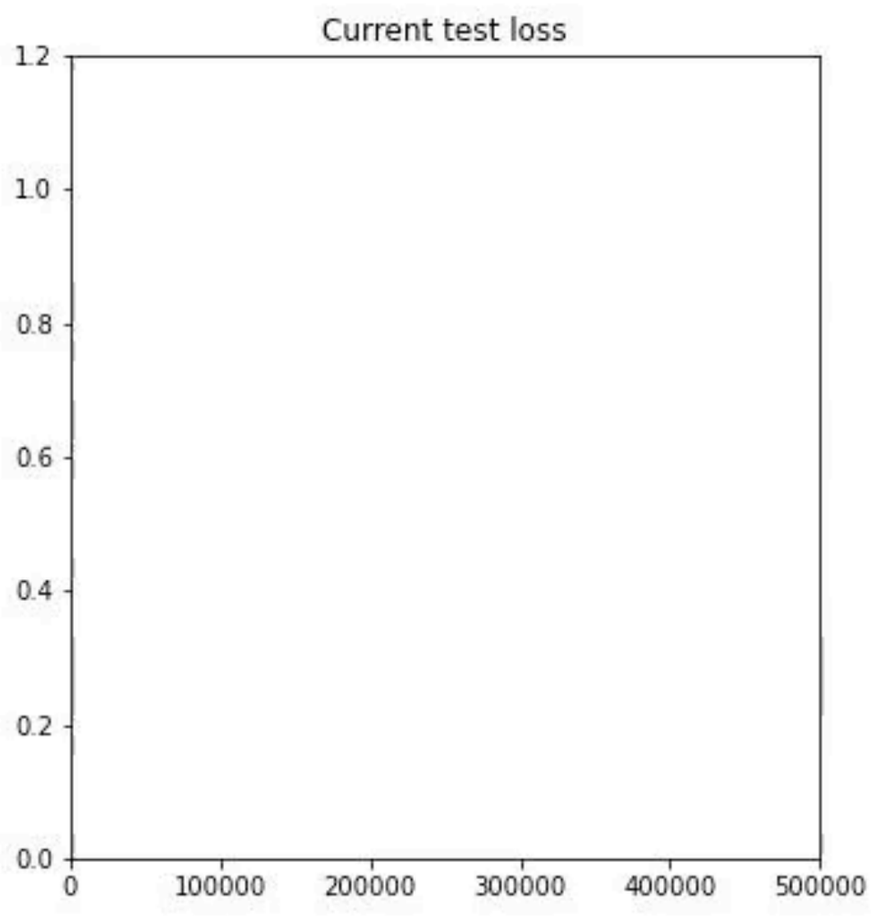
no infinite width no continuous time [Mei et al. 18, Chizat et al. 20]

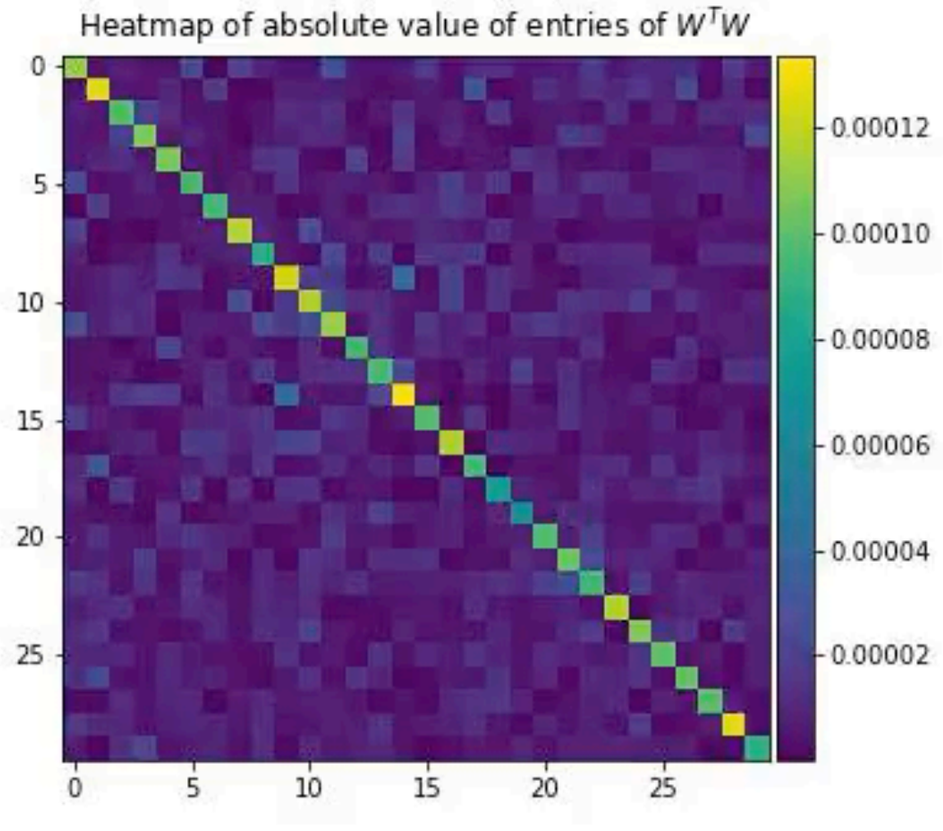
3) Control of the entire NN dynamic (beyond first GD-step and beyond single-index models)

[Bietti et al 22, Barak et al. 22, Daniely et al. 20, Ben Arous et al. 21, Damian et al. 22]

Saddles and descents

$$(x_1 + x_1x_2x_3x_4 + x_1x_2x_3x_4x_5x_6x_7x_8)/\sqrt{3} \qquad d = 30, P = 8$$





2-MLP / isotropic layers / SGD l.r.=1e-3

Dimension-free mean-field PDE

$$\hat{f}_{\mathsf{NN}}(\boldsymbol{x};\rho) = \int a\sigma(\langle \boldsymbol{w},\boldsymbol{x}\rangle)\rho(\mathrm{d}a\,\mathrm{d}\boldsymbol{w}) \qquad (\text{2NN-MFL})$$

[Mei, Misiakiewicz, Montanari '20]

$$\partial_t \rho_t = \nabla_{\boldsymbol{\theta}} \cdot (\rho_t \boldsymbol{H}(t) \nabla_{\boldsymbol{\theta}} \psi(\boldsymbol{\theta};\rho_t)) \qquad (\text{Wasserstein GF})$$

$$\psi(\boldsymbol{\theta};\rho_t) = a\mathbb{E}_{\boldsymbol{x}}\Big[\{\hat{f}_{\mathsf{NN}}(\boldsymbol{x};\rho_t) - f_*(\boldsymbol{x})\}\sigma(\langle \boldsymbol{w},\boldsymbol{x}\rangle)\Big] + \frac{1}{2}\boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{\Lambda}\boldsymbol{\theta}$$

3 limits:
- **width**
- **time**
- **samples**

Dimension-free formulation:

$$\hat{f}_{\mathsf{NN}}(\boldsymbol{x};\rho_t) = \int a^t \sigma(\langle \boldsymbol{x},\boldsymbol{w}^t\rangle)\rho_t(\mathrm{d}\boldsymbol{\theta}^t) = \int a^t \mathbb{E}_{\boldsymbol{r}}\big[\sigma(\langle \boldsymbol{u}^t,\boldsymbol{z}\rangle + \langle \boldsymbol{v}^t,\boldsymbol{r}\rangle)\big]\rho_t(\mathrm{d}\boldsymbol{\theta}^t)$$

$$\overline{\rho}_t \in \mathcal{P}(\mathbb{R}^{P+2}) \qquad \overline{a}^0 \sim \mu_a,\ \overline{\boldsymbol{u}}^0 = \mathbf{0},\ \overline{s}^0 = m_2^w$$

Theorem [ABM'22]. If $N \gg 1$, $\eta \ll 1/d$, $T = O(1)$ and $P = O(1)$,

$$\sup_{k\in[T/\eta]\cap\mathbb{N}} \big\|\hat{f}_{\mathsf{NN}}(\cdot;\boldsymbol{\Theta}^k) - \hat{f}_{\mathsf{NN}}(\cdot;\overline{\rho}_{k\eta})\big\|_{L^2} = o_d(1) \quad \text{with prob. } 1 - o_N(1)$$

Dynamic of support correlation

$$w^{(t+1)} = w^{(t)} + \eta A^{(t)} = w^{(0)} + \eta \sum_{s=0}^{t-1} A^{(s)}$$

$$= w^{(0)} + \eta \sum_{s=0}^{t-1} \bar{A}^{(s)} + \eta \sum_{s=0}^{t-1} (A^{(s)} - \bar{A}^{(s)})$$

drift

martingale

$$\leq \eta\sqrt{t}$$

Weights incidents to variables in the support of the target grow with a stronger drift -> the support gets detected

[Ben Arous et al. 21]

Customized SGD: Layerwise, projected, low l.r., early stopped

Control interactions: $a_i \sim_U [-A, A]$, $A = \text{poly}^{-O(1)}(d)$

$w_1^{(t)} = a^{(t)}, w_d^{(t)} = b^{(t)}$ $\qquad a^{(0)} = b^{(0)} \sim 1/\sqrt{d}$

$$a^{(t+1)} = a^{(t)} + \eta A^{(t)} = a^{(0)} + \eta \sum_{s=0}^{t-1} A^{(s)}$$

$$= a^{(0)} + \eta \sum_{s=0}^{t-1} \bar{A}^{(s)} + \eta \sum_{s=0}^{t-1} (A^{(s)} - \bar{A}^{(s)})$$

drift martingale [BGJ'21]

Control martingale: $\eta\sqrt{T} \le 1/\sqrt{d}$

Control drift: $\eta T \ge d^{L/2-1}$

Choose: $\eta \propto d^{-L/2}$, $T \propto d^{L-1}$

Project on sphere moderate components and stop when fittable

Length generalization problem

Anil, C., Wu, Y., Andreassen, A., Lewkowycz, A., Misra, V., Ramasesh, V., Slone, A., Gur-Ari, G., Dyer, E., and Neyshabur, B. (2022). Exploring length generalization in large language models.

Learning a k-parity on inputs of Hamming weight $\leq r$; $\quad r \leq k$.

Theorem. Consider a Boolean function $f : \{\pm 1\}^d \to \mathbb{R}$. Then (i) there exists a unique function $f_r : \{\pm 1\}^d \to \mathbb{R}$ such that $\forall x \in B_r, f_r(x) = f(x)$ and $\deg(f_r) \leq r$; (ii) when f is a parity function (monomial) of degree $k \leq d$, the GOTU_{ℓ_2} of the MDI is larger than $\binom{k-1}{r}^2$.

$$1 + \sum_{T \subseteq [k]:|T|=1} \prod_{i \in T} (x_i - 1) + \cdots + \sum_{T \subseteq [k]:|T|=r} \prod_{i \in T} (x_i - 1)$$

Corollary. $\mathrm{GOTU}_{0/1} = 1/2 + o_d(1)$ if $r \ll k$.

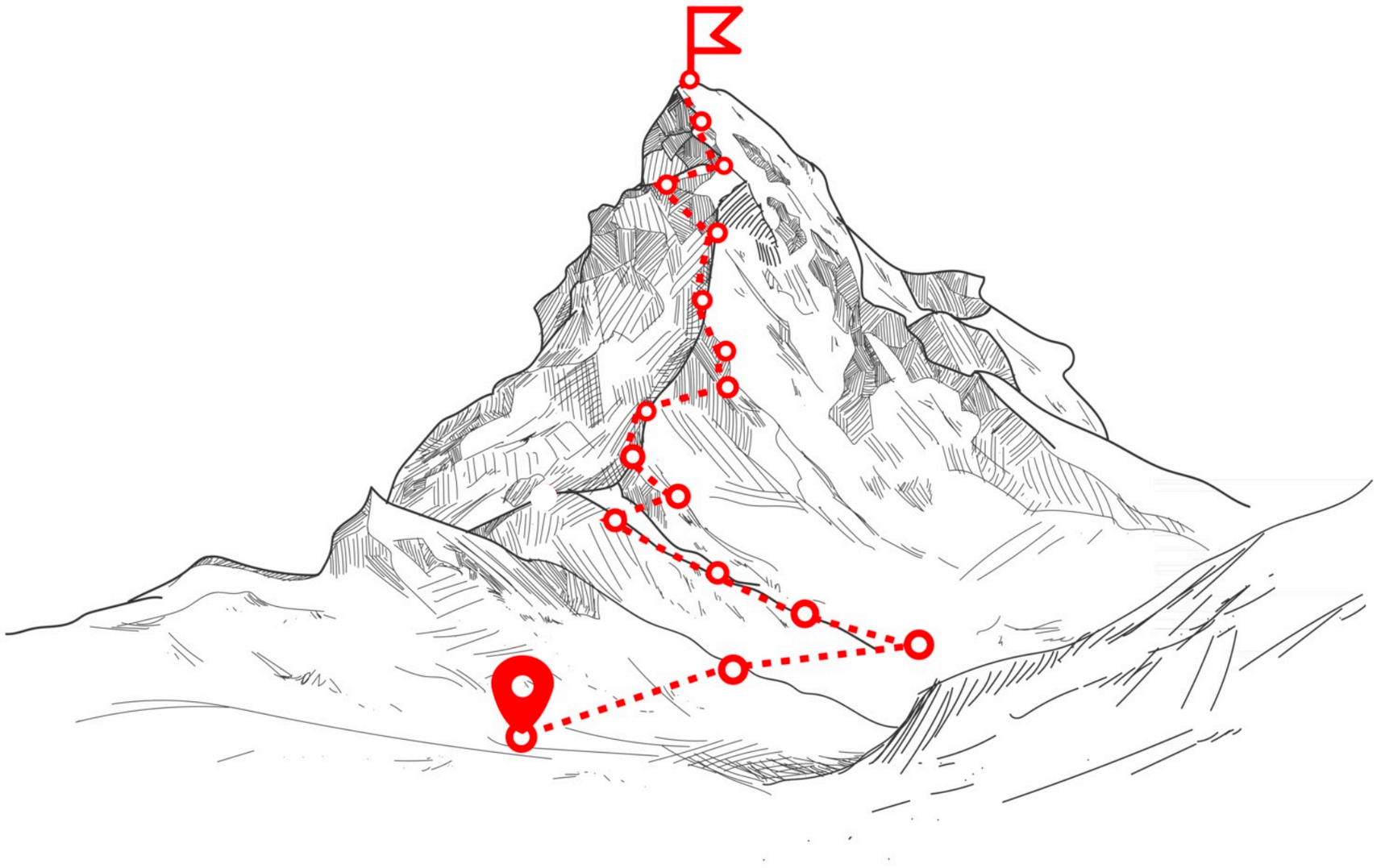
$\to$ explains length generalization problem [Anil et al.'22]

Can we break the leap and accelerate learning?

(1) non-regular networks (cf. universality result)?
(2) curriculum

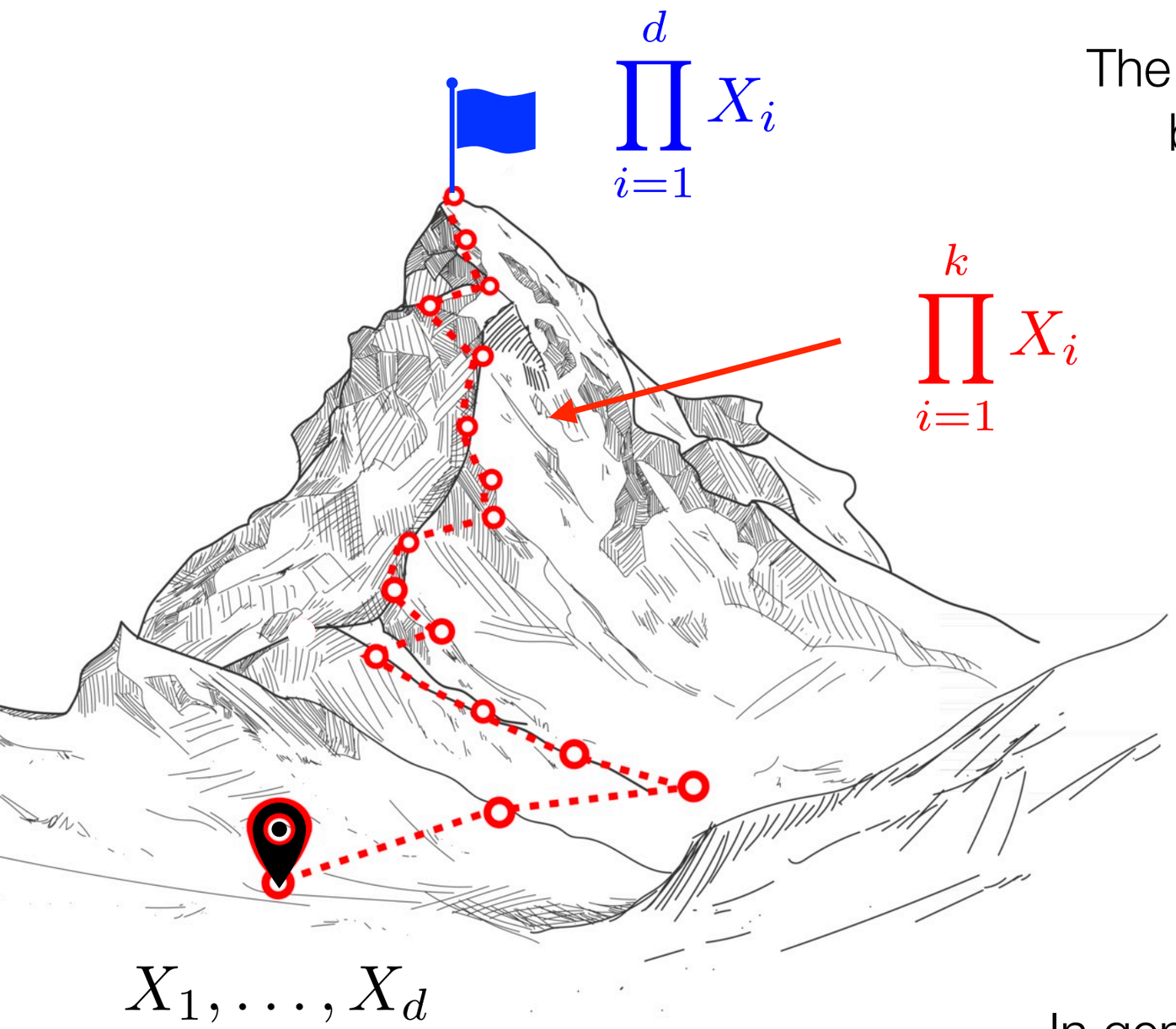
1. Target curriculum
(play with intermediate targets)

Set a climbing route to reach target

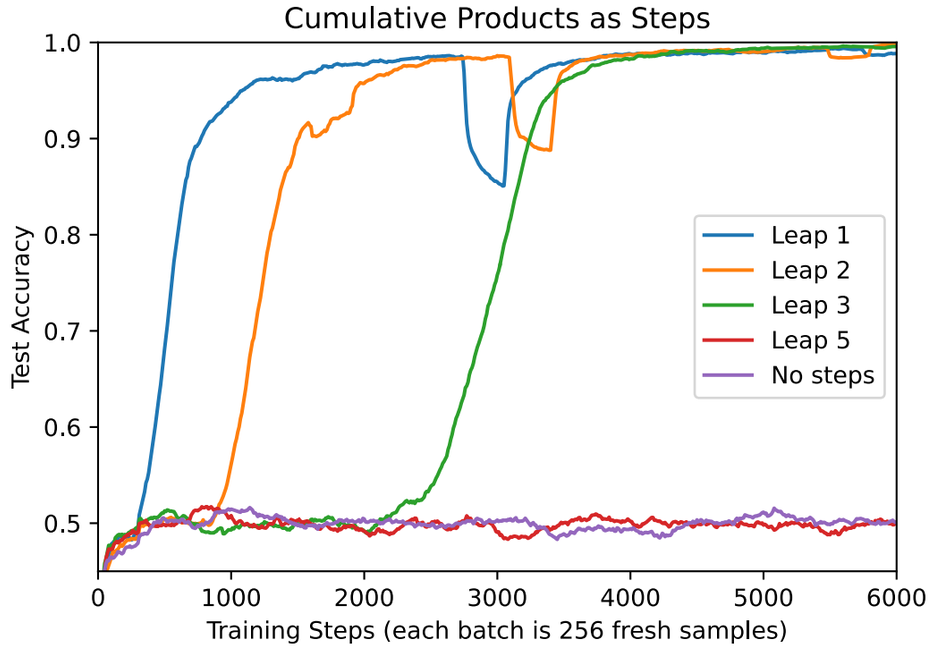


The target may not be a staircase function, but try to create intermediate steps!

Target curriculum: set a climbing route

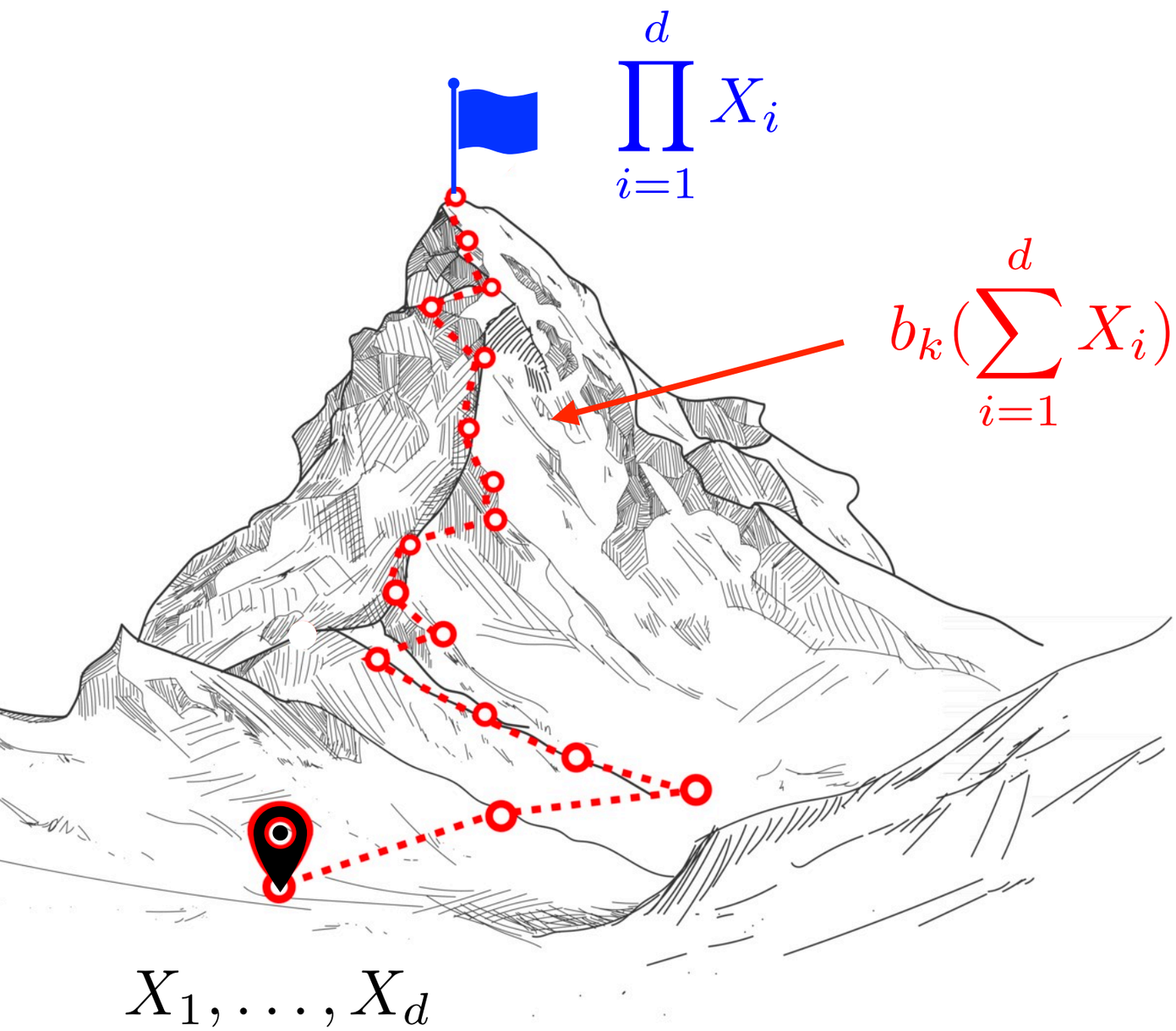


The target may not be a staircase function,
but try to create intermediate steps!



In general: try to find useful intermediate steps
(also related to scratchpad/chain-of-thoughts)

Set a climbing route to reach target



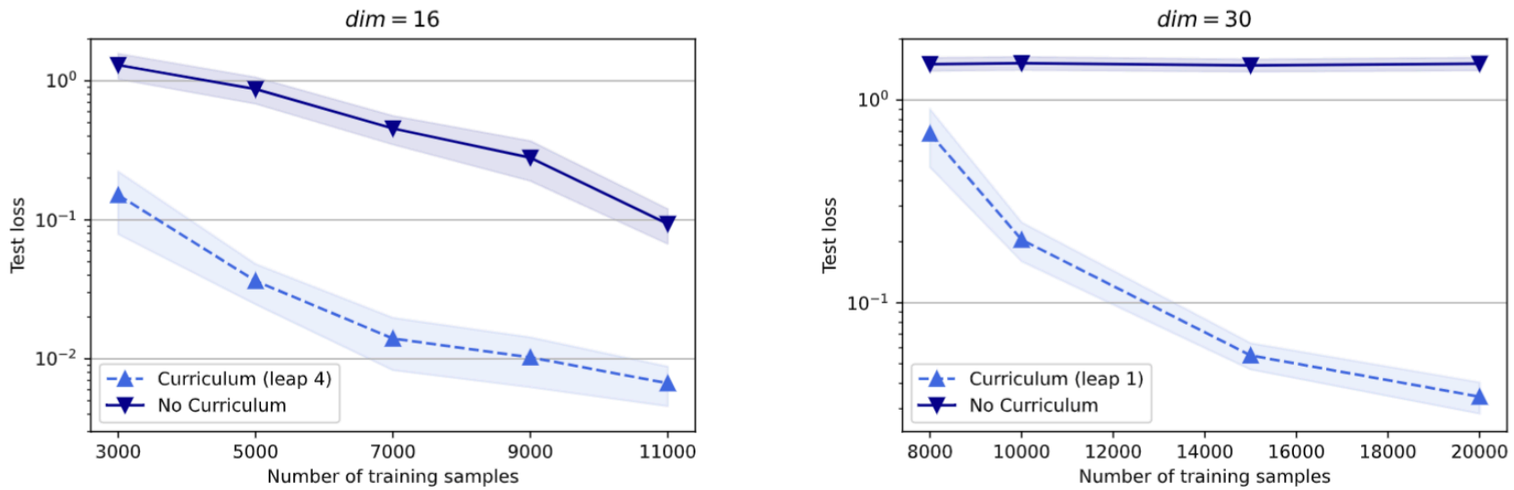
Using the sum in binary expansion

2. Input curriculum
(play with intermediate inputs)

Curriculum: feed "simpler" inputs first

Degree-Curriculum Algorithm. [Abbe-Bengio-Lotfi-Rizk ICML'22]
(1) Sort inputs by increasing Hamming weight: $B_1 \subset B_2 \subset \ldots \subset B_d$.
(2) Train sequentially on B_l, $l = 1, \ldots, d$ until vanishing train loss.

Target: $\prod_{i=1}^{d} X_i$



-> The sequence of learned functions gives a staircase!

'Theorem' [Abbe-Cornacchia-Lotfi NeurIPS'23]. Separation for a random k-parity function on a mixture data distribution: for some mixture parameters, curriculum learning can learn in $O(d)$ steps on a 2-layer $O(d)$-edge NN, while no-curriculum learning cannot learn in $O(d^2)$ steps on any $O(d)$-edge NN.

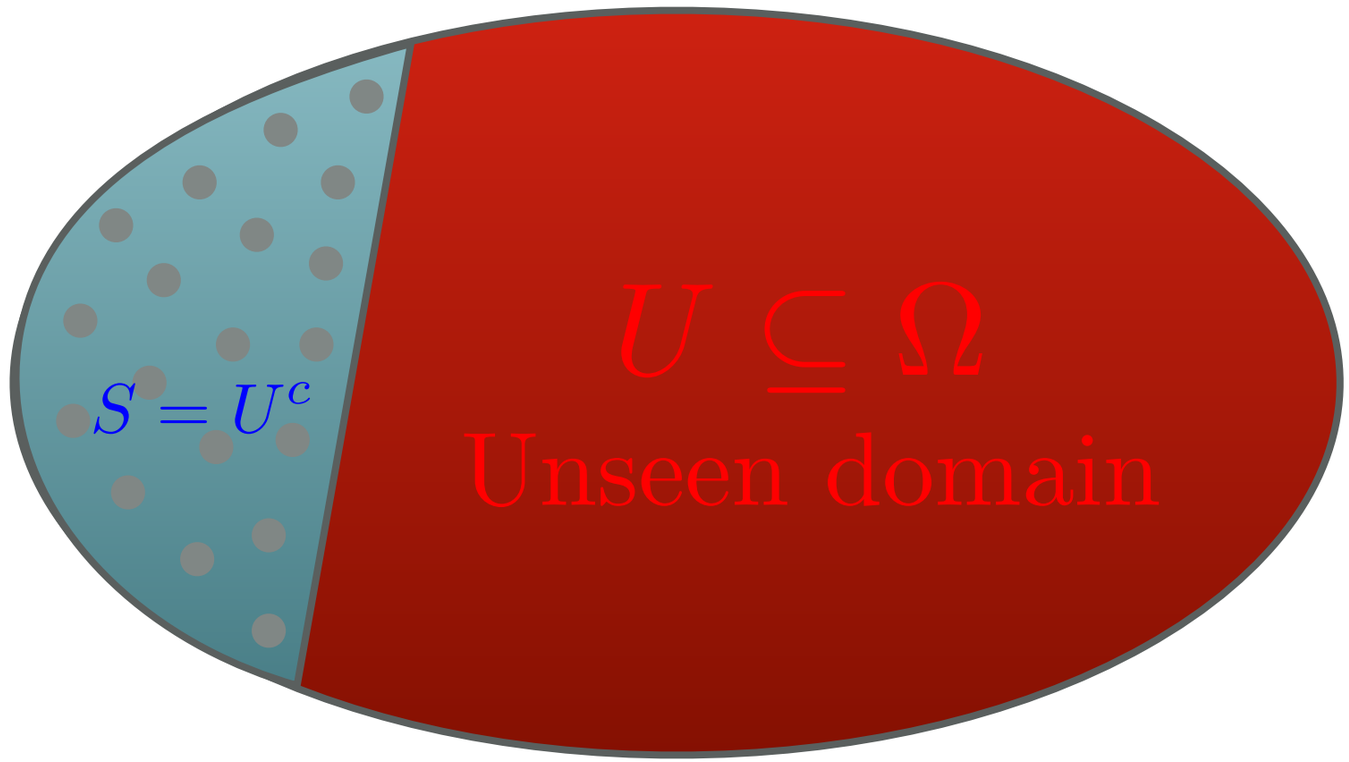
So far..

Learning with regular networks takes place in a hierarchical manner and we can now quantify in what sense with the leap.
Limitation: this is for (sparse) targets on iid inputs.

This insight leads to techniques to further boost the learning, with curriculum designs (if available).

In most reasoning tasks,
one usually cannot sample “exhaustively” a domain.
-> the test distribution often differs from the train one

Generalization on the unseen (GOTU)



Classic generalization: $GEN(f, \tilde{f}, \mu) = \mathbb{E}_{X \sim \mu}[\ell(\tilde{f}_{X^m \sim \mu^m}(X), f(X))]$

GOTU generalization: $GOTU(f, \tilde{f}, \mu_S, \mu_U) = \mathbb{E}_{X \sim \mu_U}[\ell(\tilde{f}_{X^m \sim \mu_S^m}(X), f(X))]$

$GOTU(f, \tilde{f}, S, U) = \mathbb{E}_{X \sim U}[\ell(\tilde{f}_{X^m = U^c}(X), f(X))]$

Warm-up example

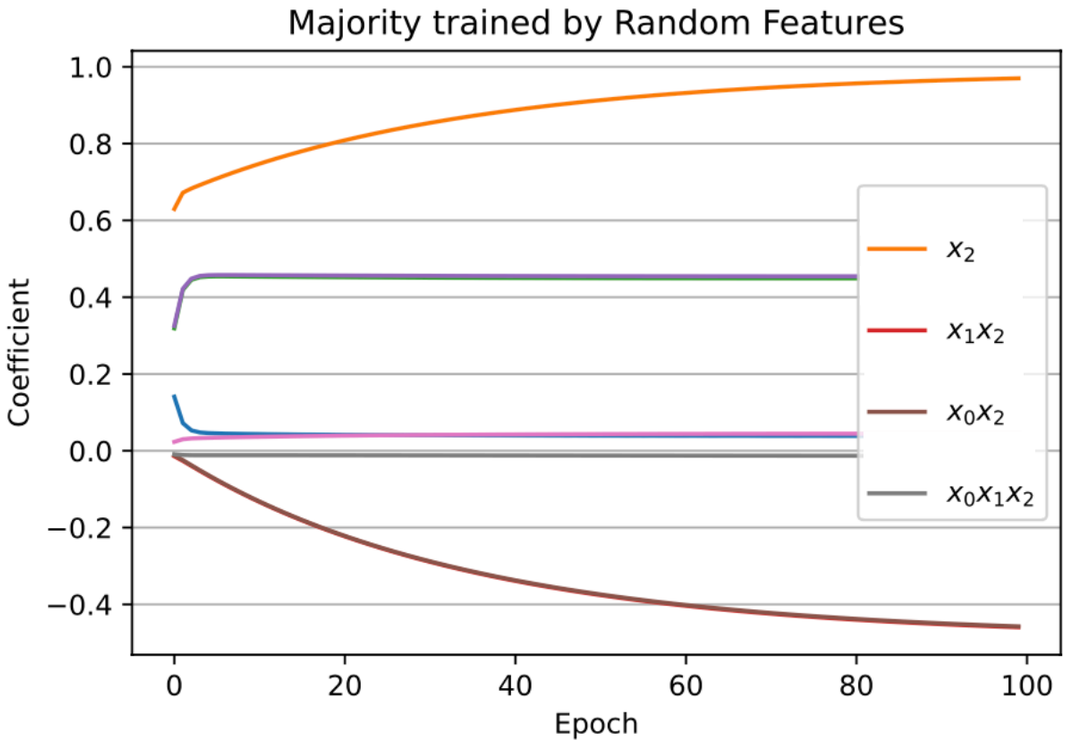
Question. Learn a target f in GOTU setting with
unseen domain $U = \{x \in \{\pm 1\}^d | (x_0, x_1) \neq (-1, -1)\}$

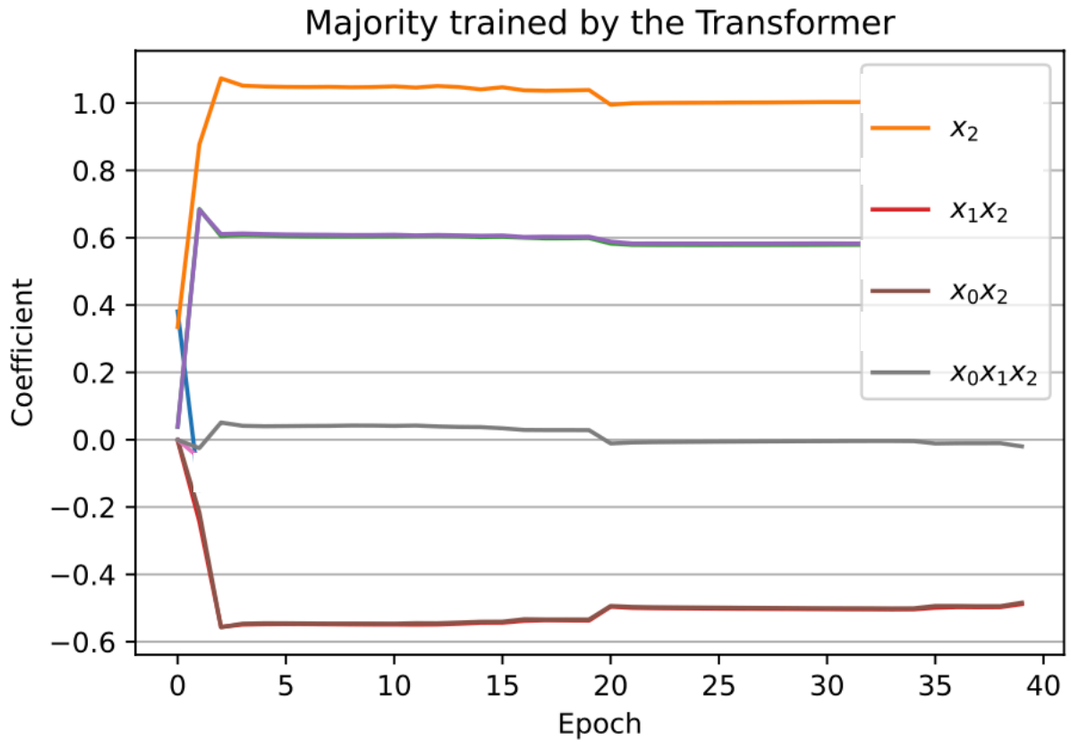
$$f(x) + \underbrace{\Delta(x)}_{\text{anything}} \underbrace{(1 - x_0)(1 - x_1)}_{\text{vanishing ideal on } U} \rightarrow \text{ all training data interpolators!}$$

Question. Which $\Delta(x)$ is learned by NN+SGD?

Experiment. $f = \text{Majority\_3}(x) = \text{sign}(\sum_{i=0}^{2} x_i)$, $d = 40$.
$\tilde{f} =$ transformer.

Warm-up example





$$x_0x_1 = x_0 + x_1 - 1$$

$$\text{Target}: \quad \text{Majority}(x) = \frac{1}{2}x_0 + \frac{1}{2}x_1 + \frac{1}{2}x_2 - \frac{1}{2}x_0x_1x_2$$

$$\text{Learned} \approx \frac{1}{2}x_0 + \frac{1}{2}x_1 + x_2 - \frac{1}{2}x_0x_2 - \frac{1}{2}x_1x_2$$

The min-degree-bias theorem

'Theorem' (Abbe-Bengio-Lotfi-Rizk ICML'23).
Let f Boolean sparse. Let $\tilde{f}^{(\infty)}_{\mathrm{NN},U}$ be the output of training a RF/DL neural network up to convergence with GD/GF and GOTU setting with Unseen set U. Then,

$$\tilde{f}^{(\infty)}_{\mathrm{NN},U} \xrightarrow[\alpha \to 0]{d \to \infty \; N \overset{\text{whp}}{\to} \infty} \mathrm{MinDegInter}(U^c, f)$$

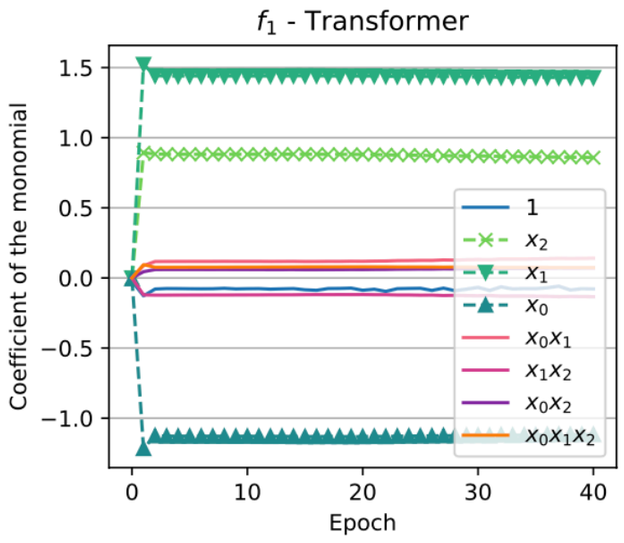
Experimental results. Transformers have a strong min-degree bias.

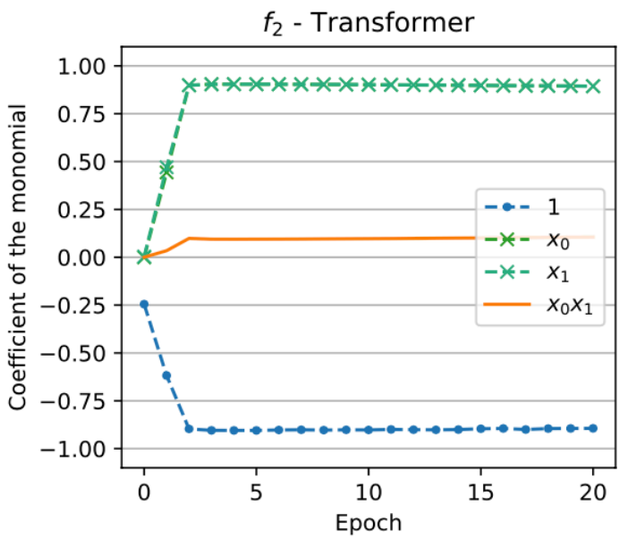
$\mathrm{MinDegInter}(U^c, f) = U^c$-vanishing ideal f-coset with minimal degree-profile $\overrightarrow{\mathrm{Deg}}$

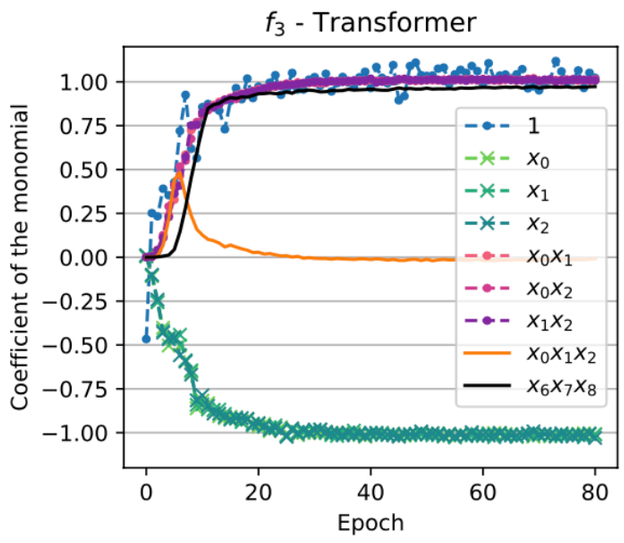
$$\overrightarrow{\mathrm{Deg}}(\tilde{f}) := (m_d, \ldots, m_1, m_0) \quad \text{where} \quad m_k := \sum_{S \subseteq [d] : |S| = k} \langle \tilde{f}, \chi_S \rangle^2$$

More experiments

1. $f_1(x) = x_0x_1 - 1.25x_1x_2 + 1.5x_2x_0$ and $\mathcal{U}_1 = \{x_0x_1x_2 = -1\}$. In this case we have $x_0x_1 = x_2$, $x_1x_2 = x_0$, and $x_2x_0 = x_1$ for the training samples, hence the min-degree interpolator is $\tilde{f}_1(x) = x_2 - 1.25x_0 + 1.5x_1$.
2. $f_2(x) = x_0x_1$ and $\mathcal{U}_2 = \{(x_0, x_1) = (-1, -1)\}$. Note that in the min-degree interpolator is $\tilde{f}_2(x) = x_1 + x_0 - 1$ for the seen domain.
3. $f_3(x) = x_0x_1x_2 + x_1x_2x_3 + \cdots + x_{13}x_{14}x_0 + x_{14}x_0x_1$ and $\mathcal{U}_3 = \{(x_0, x_1, x_2) = (-1, -1, -1)\}$. In this case the min-degree interpolator is given by $\tilde{f}_3(x) = (x_0x_1 + x_1x_2 + x_2x_0 - x_0 - x_1 - x_2 + 1) + x_1x_2x_3 + \cdots + x_{13}x_{14}x_0 + x_{14}x_0x_1$.







Transformer depth 12, 6 attention heads, embedding dim. 256. Training: Adam, l.r. 2*10^-5, batch 256

4-MLP (512,1024,512,64) / SGD, l.r. 2*10^-5, batch 64

Minimizing degree is not necessarily the most desirable occam's razor.

Thank you