Simplicity Bias in Transformers and their Ability to Learn Sparse Boolean Functions

Satwik Bhattamishra[▲] Arkil Patel[■] Varun Kanade[▲] Phil Blunsom[▲¶]

[▲]University of Oxford [■]Mila and McGill [¶]Cohere

Premise and Motivation

- On NLP tasks (large) Transformers generalize well and are the dominant architecture
- Transformers are limited in expressing certain formal languages compared to RNNs
- Transformers perform worse than RNNs on certain formal languages [1, 2]

Why do Transformers with large capacity generalize well in practice?

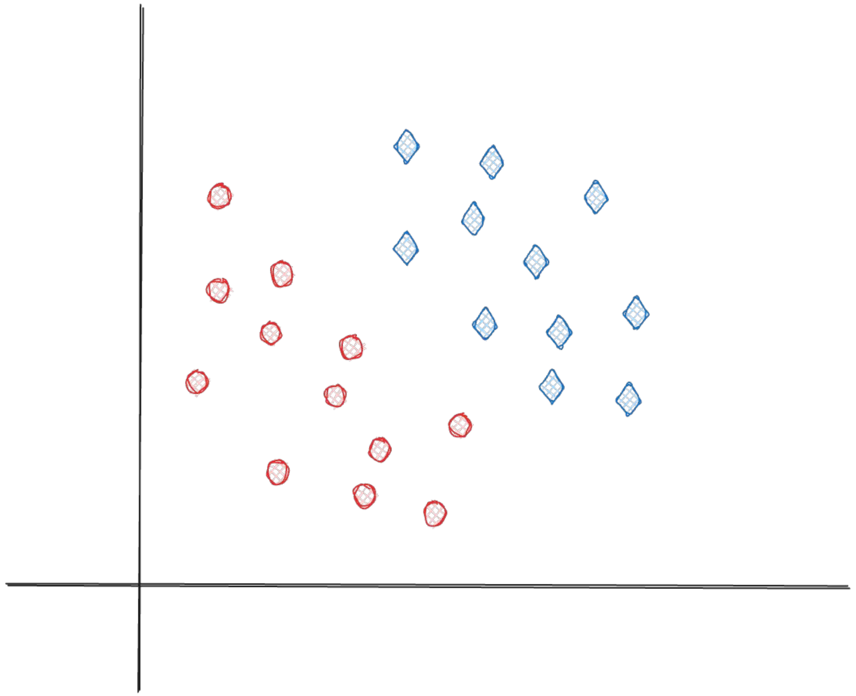
Why do Transformers perform well given their limitations in modelling formal languages?

[1] On the Ability and Limitations of Transformers to Recognize Formal Languages. EMNLP 2020. Satwik Bhattamishra, Kabir Ahuja, Navin Goyal
[2]Neural networks and the chomsky hierarchy. ICLR 2023. Delétang, Grégoire, Anian Ruoss, … , Pedro A. Ortega.

Why deep learning generalizes well?

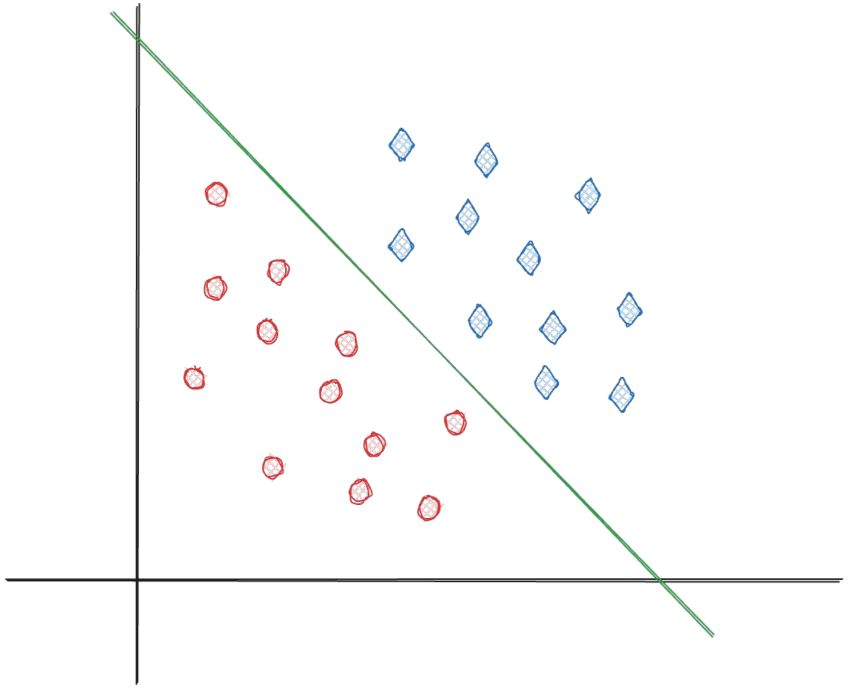
- Deep learning models can express a large class of functions
  - Have large capacity measure: VC dimension, Rademacher complexity, etc
- Given a set of examples S, there are many different parameterizations of a network that can achieve 0% training error but arbitrary test error
- However, large neural networks typically learn functions which generalize well

Example



A linearly separable set of labelled examples

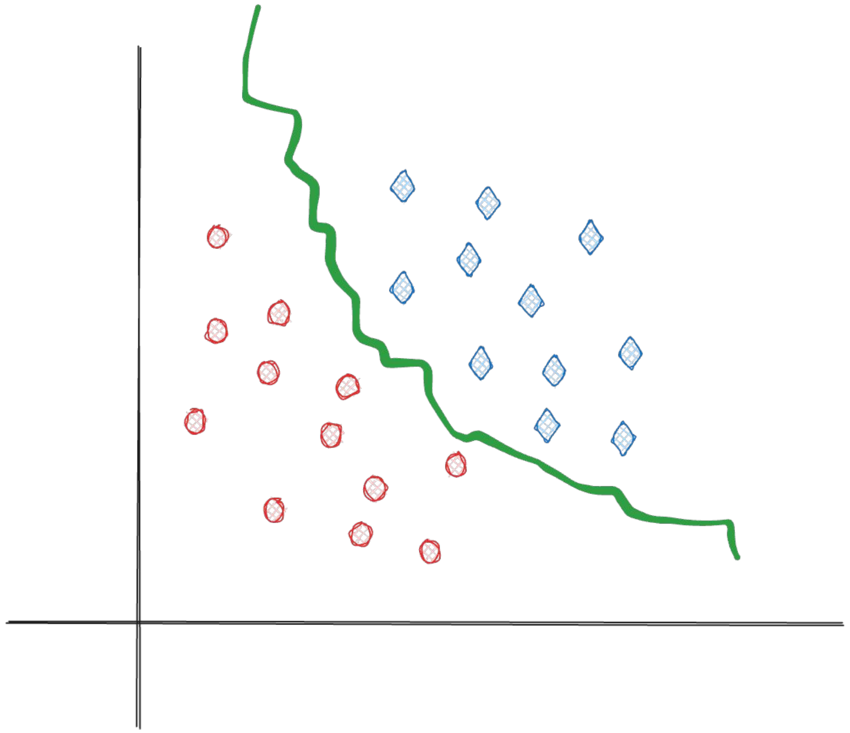
Example



A linearly separable set of labelled examples

Can be labelled perfectly by a linear classifier

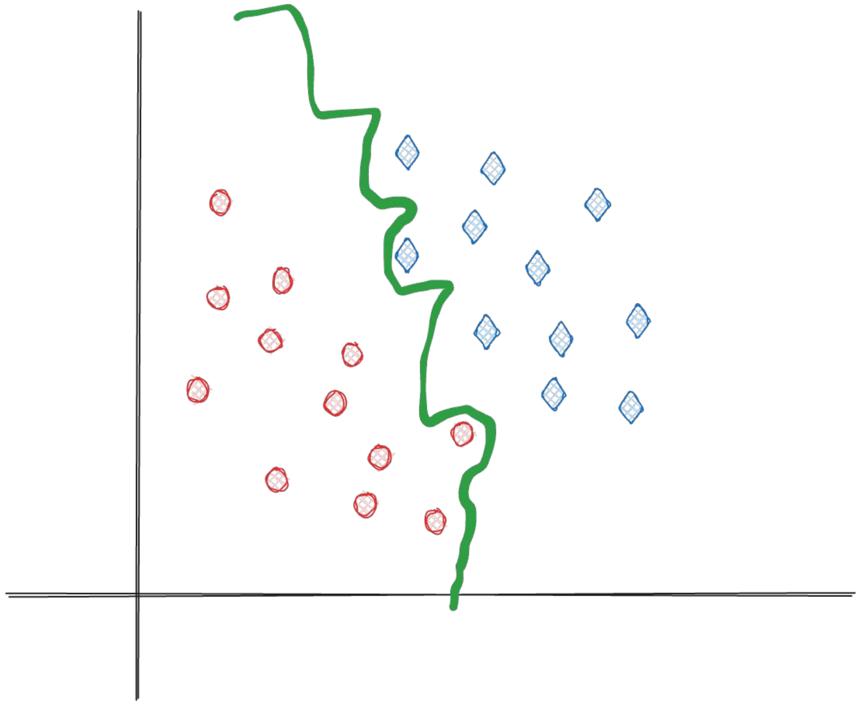
Example



A linearly separable set of labelled examples

Neural Networks are also capable of achieving 0% training error by converging to other types of functions

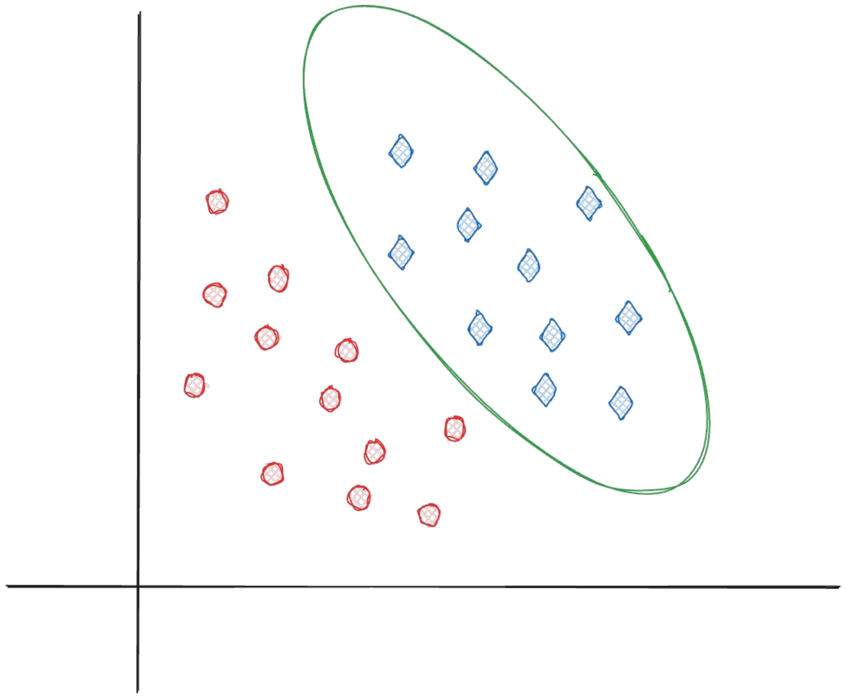
Example



A linearly separable set of labelled examples

Neural Networks are also capable of achieving 0% training error by converging to other types of functions

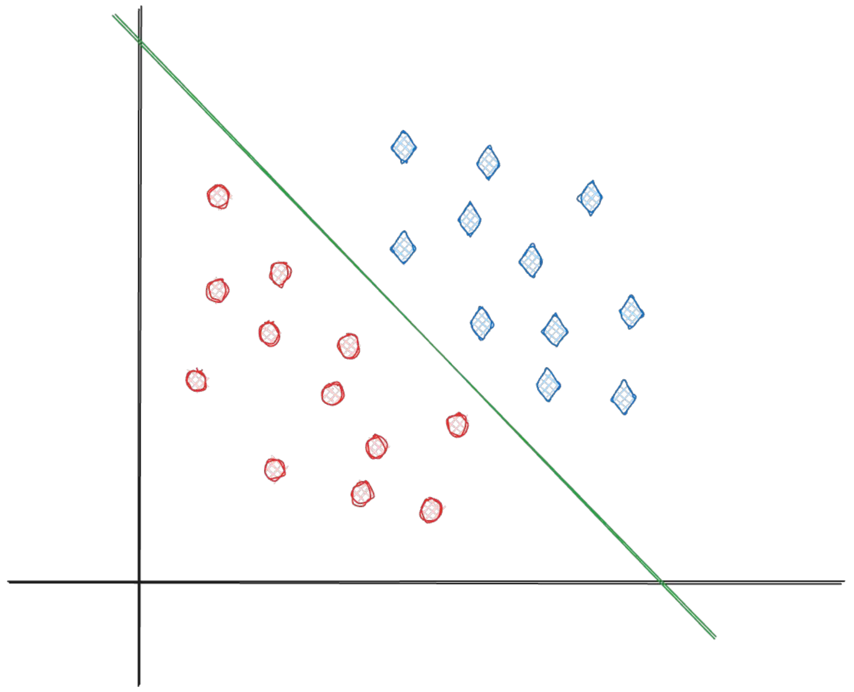
Example



A linearly separable set of labelled examples

Neural Networks are also capable of achieving 0% training error by converging to other types of functions

Example



A linearly separable set of labelled examples

But they typically learn a simple function more similar to the target

Why Deep learning generalizes well?

- There is some form of implicit regularization in Neural Networks (Simplicity bias)
- Hypotheses
  - SGD has some beneficial properties [1, 2]
  - Volume hypothesis: Parameter space of NNs are dominated by low complexity functions [3, 4]
- If most of the functions in the parameter space of a model are low complexity functions, then any local search based optimizer is more likely to find a low complexity function

[1] Implicit Regularization in Deep Matrix Factorization. Sanjeev Arora, Nadav Cohen, Wei Hu, Yuping Luo. Neurips 2019.
[2] SGD Noise and Implicit Low-Rank Bias in Deep Neural Networks. Galanti, T, Poggio, T. 2022
[3] Is SGD a Bayesian sampler? Well, almost. Chris Mingard, Guillermo Valle-Pérez, Joar Skalse, Ard A. Louis. JMLR 2021
[4] Loss Landscapes are All You Need: Neural Network Generalization Can Be Explained Without the Implicit Bias of Gradient Descent. ICLR 2023

Preliminaries

- We work with Boolean functions f: $\{0, 1\}^n$ -> {0, 1}
- Focus on a complexity measure called **sensitivity**: indicates how likely it is that a function value will change due to some small change in input

$$s(f, x) = \sum_{i=1}^{n} \mathbb{I}[f(x) \neq f(x^{\oplus i})],$$

$$\mathcal{S}(f) = \Pr_{x \sim \{0,1\}^n, i \sim [n]} [f(x) \neq f(x^{\oplus i})]$$

Parity-n : Input:= 100001001 - label : 1
Parity over length n has sensitivity 1
k-sparse parity over length n has sensitivity k/n

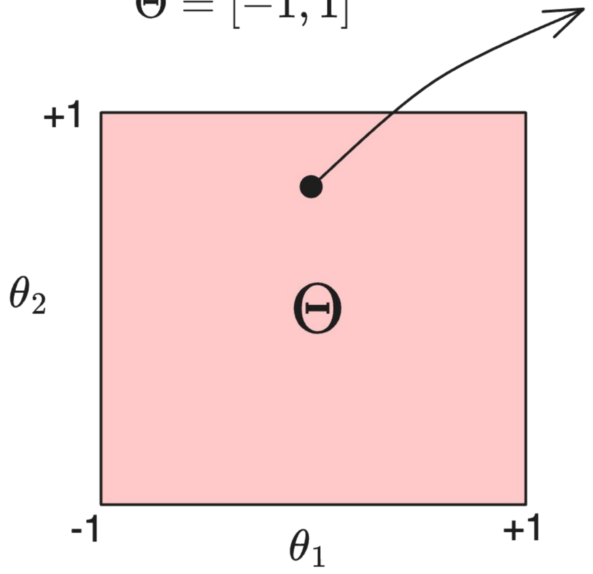
Why sensitivity?

- Function of Lower sensitivity => Simpler Function
    - Simpler functions are more likely to generalize
- Lower sensitivity function => Lower Kolmogorov Complexity (+ other measures)
- Lower sensitivity function => Lower generalization error [1] (?)
- Practical NLP tasks have low* sensitivity [2]
- Can be approximated by sampling

[1] Sensitivity and Generalization in Neural Networks: an Empirical Study. ICLR 2019. Roman Novak, Yasaman Bahri, …
[2] Sensitivity as a Complexity Measure for Sequence Classification Tasks. TACL 2021. Michael Hahn, Dan Jurafsky, Richard Futrell

Parameter function map

$$\Theta = [-1, 1]^2$$

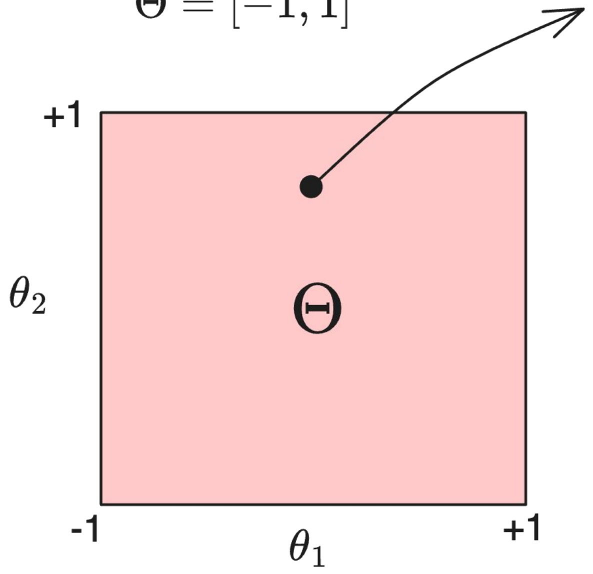


$$f_\theta : \{0,1\}^n \to \{0,1\}$$

$$x_1 = 000 : f_\theta(x_1)$$
$$x_2 = 001 : f_\theta(x_2)$$
$$x_3 = 010 : f_\theta(x_3)$$
$$\vdots$$
$$x_{2^3} = 111 : f_\theta(x_{2^3})$$

Parameter function map

$$\Theta = [-1, 1]^2$$



$$f_\theta : \{0,1\}^n \rightarrow \{0,1\}$$

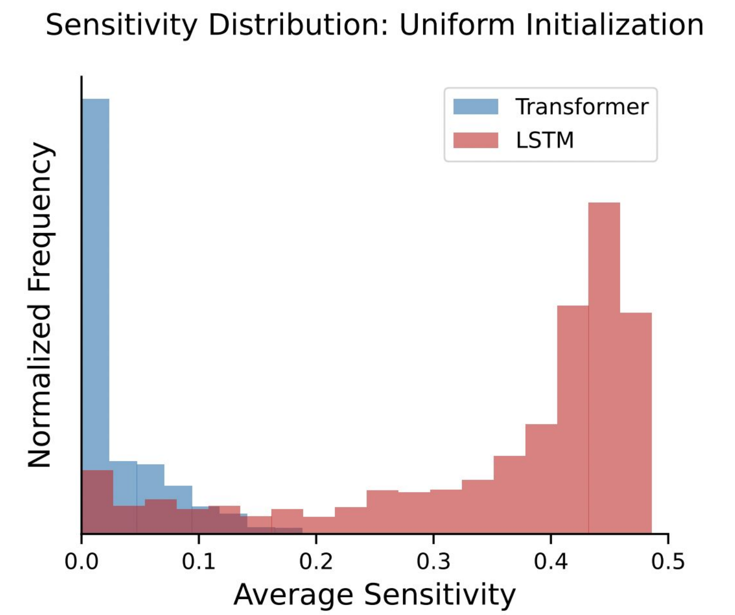
$$\begin{aligned} x_1 &= 000 : f_\theta(x_1) \\ x_2 &= 001 : f_\theta(x_2) \\ x_3 &= 010 : f_\theta(x_3) \\ &\vdots \\ x_{2^3} &= 111 : f_\theta(x_{2^3}) \end{aligned}$$

Monte Carlo Method

Sample N Transformers or LSTMs using an initialization method (Uniform, Xavier normal, etc).

Estimate the sensitivity of each of the sampled models to obtain the distribution.

Sensitivity distribution of Random Transformers and LSTMs



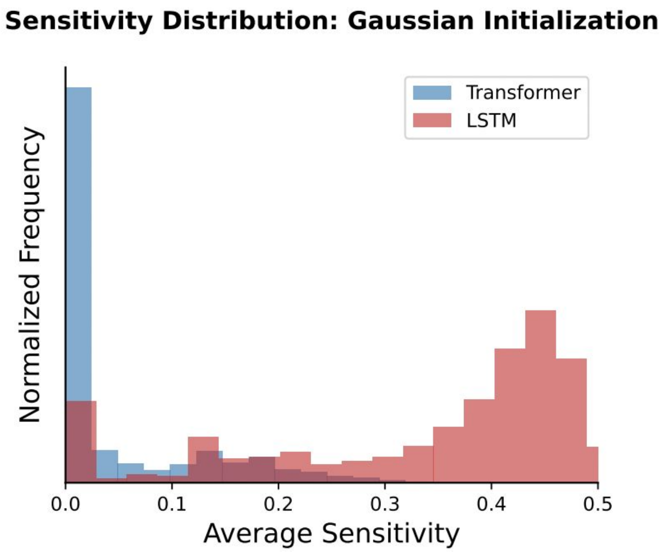
Sample N Transformers or LSTMs using an initialization method (Uniform, Xavier normal, etc).

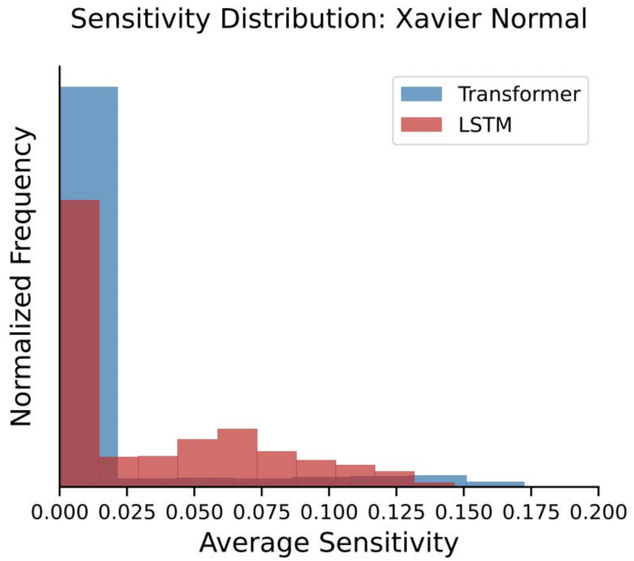
Estimate the sensitivity of each of the sampled models to obtain the distribution.

Input lengths: 20
Models: 2 Layers, 256 Width
Parameters uniformly in $[-10, +10]^p$

Sensitivity distribution of Random Transformers and LSTMs

- Functions of low sensitivity are overrepresented in the parameter space of Transformers





Finding Parity

$$\mathcal{F} = \{f \mid f : \{0,1\}^5 \rightarrow \{0,1\}\}$$

$\mathrm{Parity}\text{-}5$ outputs 1 if number of 1s in the input is odd and 0 otherwise.

There are 2^{2^5} functions over $\{0,1\}^5$

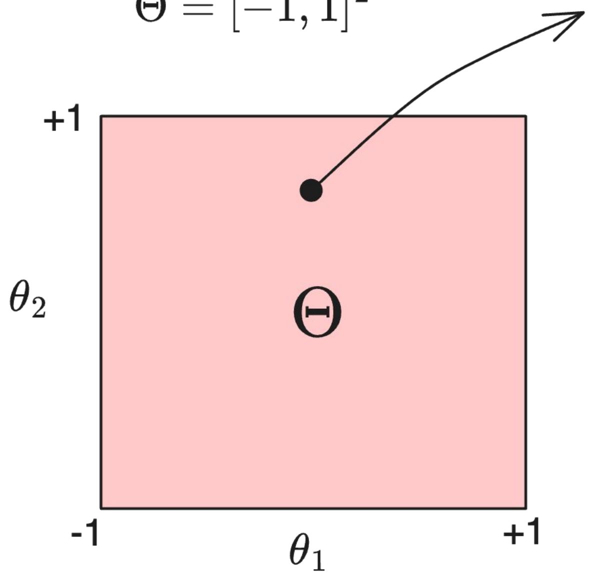
If we sample uniformly over all 2^{2^5} functions, the probably of obtaining a $\mathrm{Parity}\text{-}5$ function is less than 1 in a Billion

On uniformly sampling 2-layer LSTMs 10 million times, we find 1 in 30k of them represent a $\mathrm{Parity}\text{-}5$ function.

On sampling Transformers 10 million times, none of them represented a function in $\mathrm{Parity}\text{-}5$

Parameter function map

$\Theta = [-1,1]^2$



$f_\theta : \{0,1\}^n \to \{0,1\}$

$x_1 = 000 : f_\theta(x_1)$
$x_2 = 001 : f_\theta(x_2)$
$x_3 = 010 : f_\theta(x_3)$
$\vdots$
$x_{2^3} = 111 : f_\theta(x_{2^3})$

Sampling-based Learning Algorithm

Given Samples $S = ((x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m))$

1. Sample model f_θ uniformly over $[-1, 1]^p$
2. If f_θ is consistent with S return f_θ
3. Else Go to step 1

$P(f)$:= Probability of obtaining function f on sampling a model

$P(f|S)$:= Probability of obtaining f conditioned on the event that f is consistent with S

Bayesian Sampler

$$S = ((x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m))$$

$P(f)$:= Prior

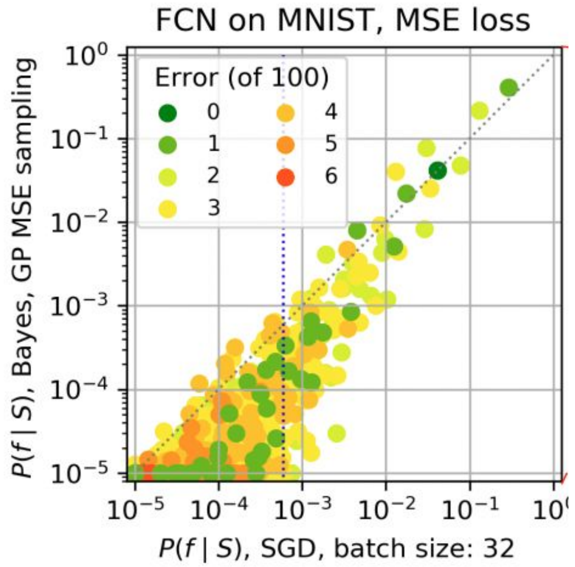
$P(f|S)$:= Posterior

$$P_B(f|S) = \frac{P(S|f)P(f)}{P(S)}$$
$$= \frac{P(f)}{\sum_{f \in U(S)} P(f)}$$

For a fixed training set, all variations in $P_B(f|S)$ comes from the prior $P(f)$

$$U(S) = \{f \mid f(x) = y \ \forall (x, y) \in S\}$$

Is SGD a Bayesian sampler?



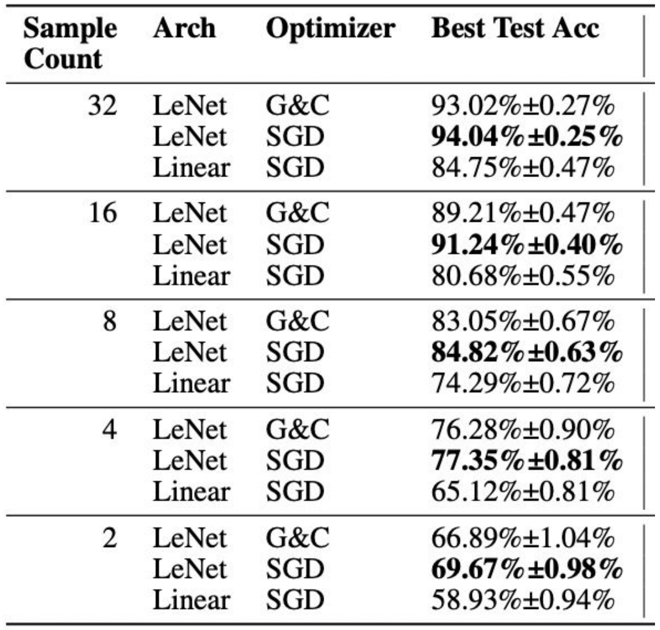
(a) $P_{\mathrm{B}}(f|S)$ v.s.$P_{\mathrm{SGD}}(f|S)$

Mingard et. al. [1] compares the distribution $P_B(f|S)$ and $P_{SGD}(f|S)$ across a range of architectures and datasets and find that they are highly correlated

Fig: Comparing the predictions of sampling based algo and SGD ≈1 million times on 100 examples.

[1] Is SGD a Bayesian sampler? Well, almost. Chris Mingard, Guillermo Valle-Pérez, Joar Skalse, Ard A. Louis. JMLR 2021

Sampling ≈ SGD performance



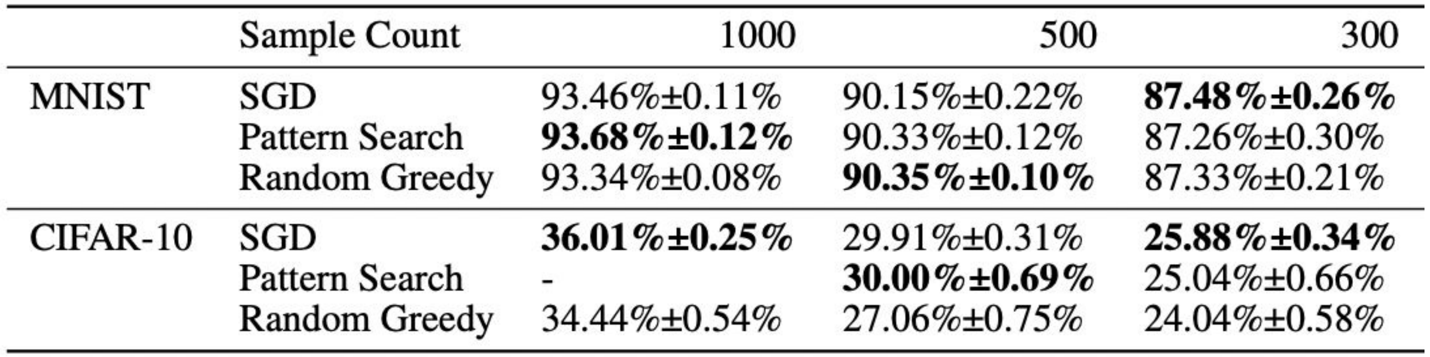
Sample Count	Arch	Optimizer	Best Test Acc
32	LeNet	G&C	93.02%±0.27%
	LeNet	SGD	**94.04%±0.25%**
	Linear	SGD	84.75%±0.47%
16	LeNet	G&C	89.21%±0.47%
	LeNet	SGD	**91.24%±0.40%**
	Linear	SGD	80.68%±0.55%
8	LeNet	G&C	83.05%±0.67%
	LeNet	SGD	**84.82%±0.63%**
	Linear	SGD	74.29%±0.72%
4	LeNet	G&C	76.28%±0.90%
	LeNet	SGD	**77.35%±0.81%**
	Linear	SGD	65.12%±0.81%
2	LeNet	G&C	66.89%±1.04%
	LeNet	SGD	**69.67%±0.98%**
	Linear	SGD	58.93%±0.94%

Chiang et. al. [1] compare the performance of networks trained using the sampling algorithm with that of SGD on datasets like MNIST, CIFAR-10 etc.

They find both algorithms lead to similar test-accuracy indicating that gradient dynamics are not the primary source of regularization

[1] Loss Landscapes are All You Need: Neural Network Generalization Can Be Explained Without the Implicit Bias of Gradient Descent
Ping-yeh Chiang, Renkun Ni, David Yu Miller, Arpit Bansal, Jonas Geiping, Micah Goldblum, Tom Goldstein. ICLR 2023

Sampling ≈ SGD performance



	Sample Count	1000	500	300
MNIST	SGD	93.46%±0.11%	90.15%±0.22%	**87.48%±0.26%**
	Pattern Search	**93.68%±0.12%**	90.33%±0.12%	87.26%±0.30%
	Random Greedy	93.34%±0.08%	**90.35%±0.10%**	87.33%±0.21%
CIFAR-10	SGD	**36.01%±0.25%**	29.91%±0.31%	**25.88%±0.34%**
	Pattern Search	-	**30.00%±0.69%**	25.04%±0.66%
	Random Greedy	34.44%±0.54%	27.06%±0.75%	24.04%±0.58%

Chiang et. al. [1] apply a more fancy pattern-search algorithm (no gradients) to fit sample sets of larger size and they perform competitively with SGD

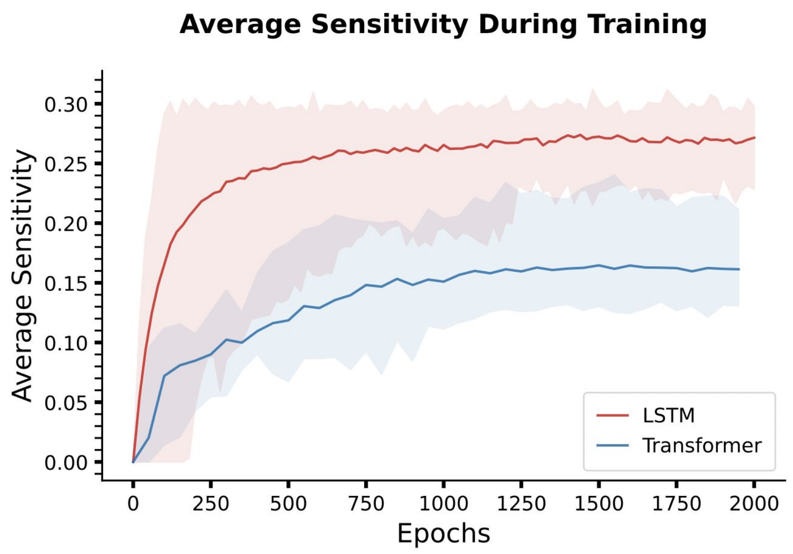
[1] Loss Landscapes are All You Need: Neural Network Generalization Can Be Explained Without the Implicit Bias of Gradient Descent
Ping-yeh Chiang, Renkun Ni, David Yu Miller, Arpit Bansal, Jonas Geiping, Micah Goldblum, Tom Goldstein. ICLR 2023

Sensitivity during Training

Training on Random Boolean functions

- What kind of functions do models converge after getting 0 error on training data
- Training data
  - 1000 Examples: Uniformly sampled over $\{0, 1\}^n$
  - Each examples is labelled 0 or 1 uniformly at random
  - Transformers and LSTMs are trained until they reach 0% training error
- Estimate the sensitivity at different iterations of training

Sensitivity during Training



Sensitivity of functions represented by Transformers and LSTMs at different stages of training when trained on data labelled by a random Boolean function.

While both achieve 0% train error, Transformer converge to functions of lower sensitivity.

Observation is reminiscent of [1, 2]

[1] SGD on Neural Networks Learns Functions of Increasing Complexity. Nakkiran et al., Neurips 2019
[2] On the Spectral Bias of Neural Networks. Rahaman et al., ICML 2019

Sensitivity during Training

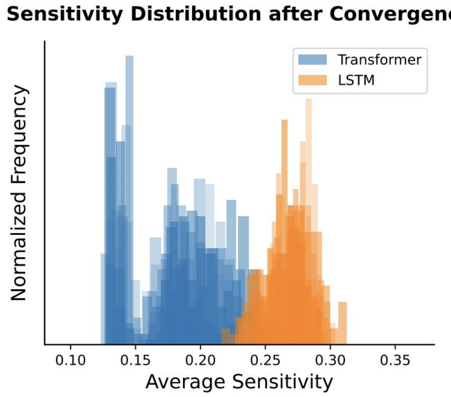
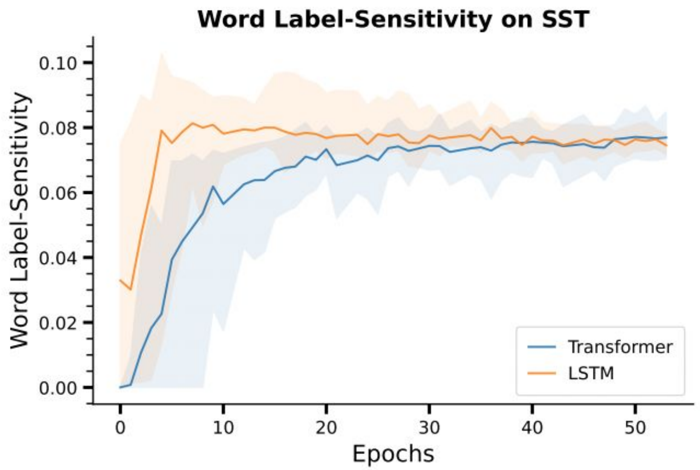


Figure 4: Distribution of sensitivity of Transformers and LSTMs trained on Boolean strings with random labels. Refer to Section 4.2 for details.

Sensitivity of functions represented by Transformers and LSTMs at different stages of training when trained on data labelled by a random Boolean function.

While both achieve 0% train error, Transformer converge to functions of lower sensitivity.

Sensitivity during Training



- Apply an extension of Boolean sensitivity for models trained on natural language data
- Both Transformers and LSTMs learn functions of increasing sensitivity during training
- Both of them achieve similar test accuracy
- Similar Observations on the IMDB dataset

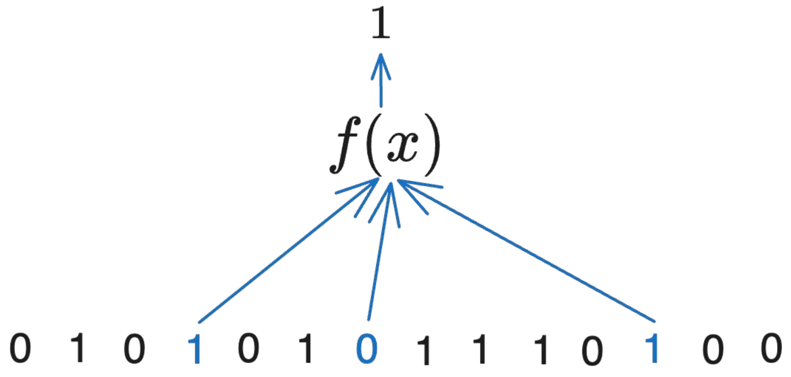
Performance on Sparse Boolean Functions

Performance on Boolean Functions

- Motivated by the differences described earlier, we compare the performance of Transformers and LSTMs on learning well-known sparse Boolean functions
- Sparse Boolean functions have low sensitivity (and low complexity based on other measures)
- We find that Transformers outperform LSTMs on several types of sparse Boolean functions.

K-Sparse functions

$$f_k : \{0,1\}^n \to \{0,1\} \qquad |\text{K-sparse}| \approx \binom{n}{k} 2^{2^k}$$



Output Depends on at most k input bits

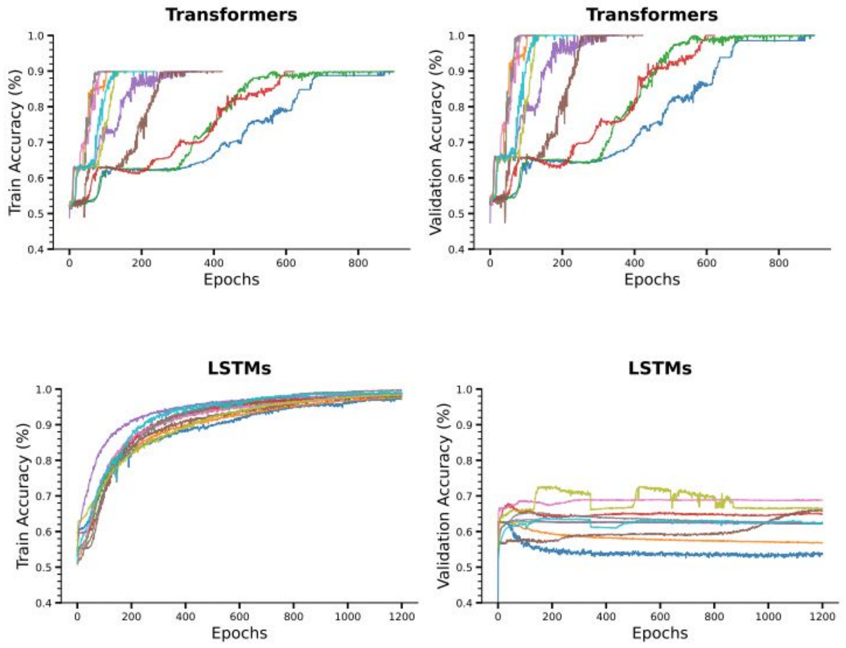
Task: Standard Binary Classification

Fixed sized Training sets: 10k - 50k

Input distribution: Uniform over $\{0, 1\}^n$
Outputs: Sample a function from a class (K-sparse, Parities, etc)

Tune hyperparameters to find best-performing models

Performance on K-Sparse Functions

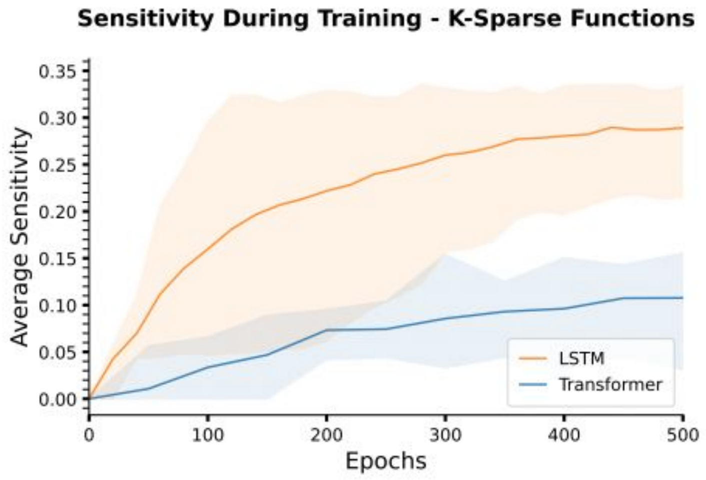


K-sparse functions: Output depends on at most k-bits of the input

Upper: Transformers generalise well even with 10% label noise during training

Lower: LSTMs seem to overfit on the training data and generalize poorly

Performance on K-Sparse Functions



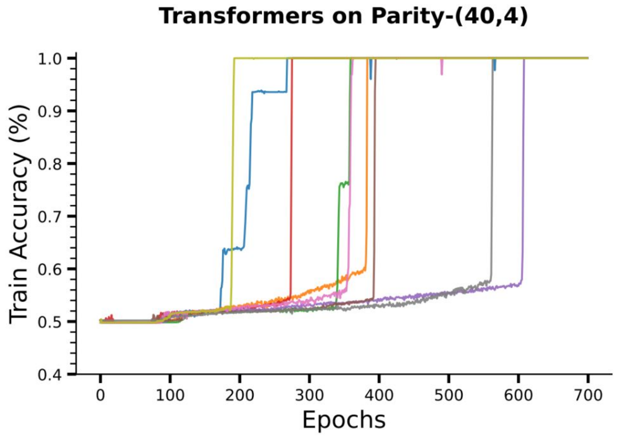
K-sparse functions: Output depends on at most k-bits of the input

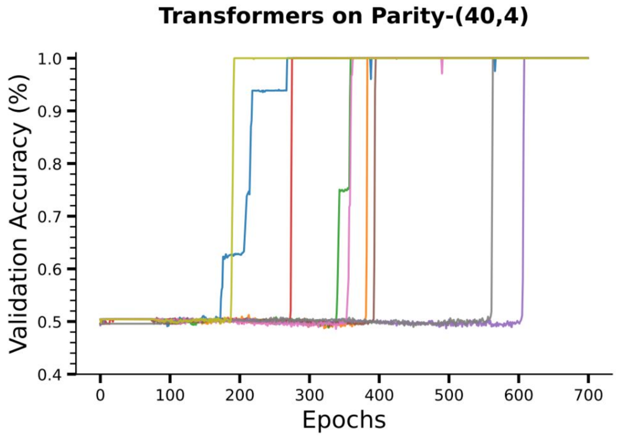
Both Transformers and LSTMs learn functions of increasing sensitivity

Transformers converge to functions with the same sensitivity as the target function

LSTMs learn functions of higher sensitivity than the target function

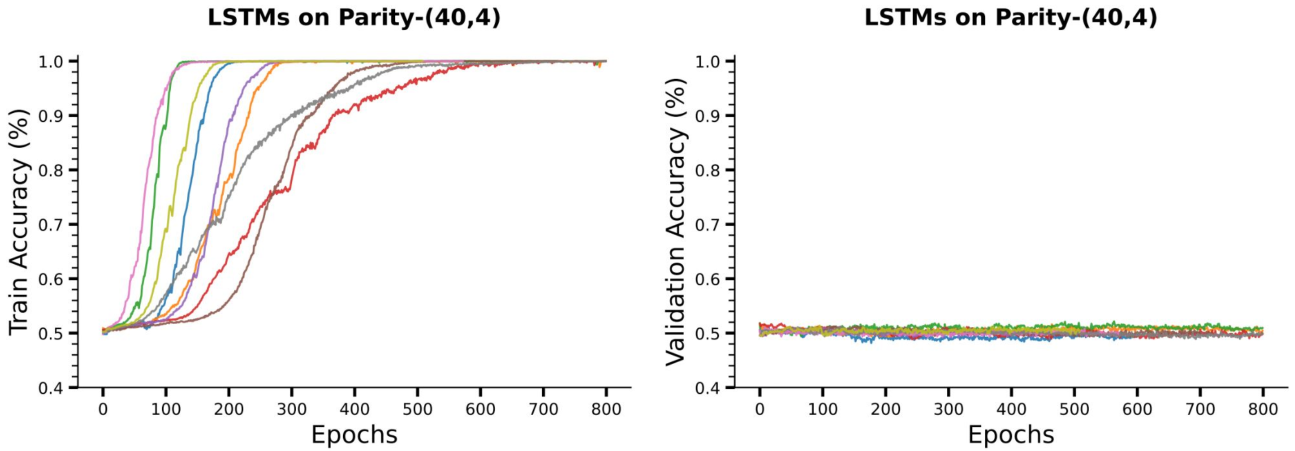
Performance on Sparse Parities





Transformers on Sparse Parities

Performance on Sparse Parities



LSTMs on Sparse Parities

Remarks on Learning Parities

- Learning Parities is known to be a hard problem
  - Efficiently learnable in the PAC model
  - Requires $n^{\Omega(k)}$ queries in a restricted model of learning
- NNs such as Transformers can only learn Sparse parities for small values of *n* and *k*
- For smaller *n* (e.g. 20) and *k*, LSTMs succeed in learning Sparse Parities as well
- See [1] which explores how FFNs learn Parities theoretically

[1] Hidden Progress in Deep Learning: SGD Learns Parities Near the Computational Limit. Boaz Barak, Benjamin L. Edelman, Surbhi Goel, Sham Kakade, Eran Malach, Cyril Zhang. Neurips 2022

Performance on Mixed Parities Dataset

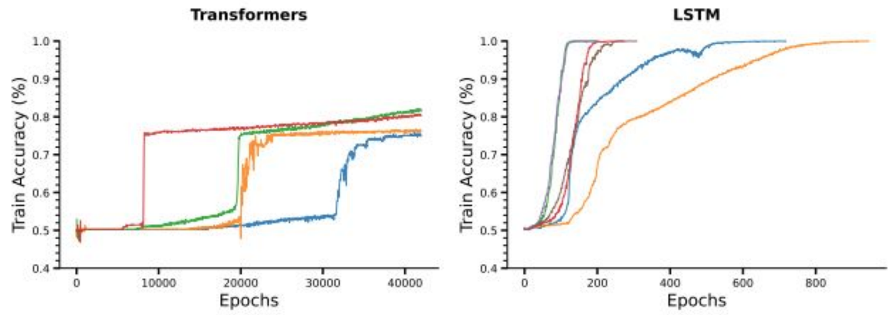


Figure 21: Training curves for Transformers and LSTMs trained on the Mixed Parity dataset (length n=30) with 15k training examples. Refer to Section E for details.

- Mixed Parity dataset:
  - 50% labelled by Sparse Parity
  - 50% labelled by Parity-n
- Evaluated separately on each Task
- Transformers
  - 100% acc. on Sparse Parity
  - 50% acc. on Parity-n
- LSTMs
  - 50% acc. on Sparse Parity
  - 100% acc. on Parity-n

Summary

- Functions of low sensitivity are overrepresented in the parameter space of Transformers
- Both Transformers and LSTMs learn functions of increasing sensitivity
- Transformers typically converge to functions of lower sensitivity than LSTMs (*on Boolean functions)
- On several K-sparse functions, Transformers generalize well whereas LSTMs overfit on the training set

Thank you!