Neural Redshift: Random Networks are not Random Functions

Damien Teney	Idiap Research Institute
Armand Nicolicioiu	ETH Zurich
Valentin Hartmann	EPFL
Ehsan Abbasnejad	University of Adelaide

Why do neural networks generalize so well?

Good performance
on test data
without overfitting

Better than most other
machine learning models
on most tasks.

Why do neural networks generalize so well?

It's not SGD.

Full-batch gradient descent **without stochasticity** works too.

Stochastic training is not necessary for generalization, Geiping et al. 2021

Models from **gradient-free** methods also generalize. E.g. rejection sampling in:

Loss landscapes are all you need: NN generalization can be explained without the implicit bias of grad. descent, Chiang et al. 2022

The **simplicity bias** was observed even **before training**. E.g. in language models in:

The no free lunch theorem, Kolmogorov complexity, and the role of inductive biases in machine learning, Goldblum et al. 2023

"SGD finds simple functions"

Not all neural networks generalize well.

Some tasks work well only with **special architectures.** E.g. sine activations in INRs (NeRFs).

Implicit neural representations with periodic activation functions, Sitzmann et al. 2020

Tabular datasets often work better with **decision trees.**

Why do tree-based models still outperform deep learning on tabular data, Grinsztajn et al. 2022

Why do neural networks generalize so well?

Inductive biases

Common architectures have properties such that they're well suited to most real-world data.

What are these properties?

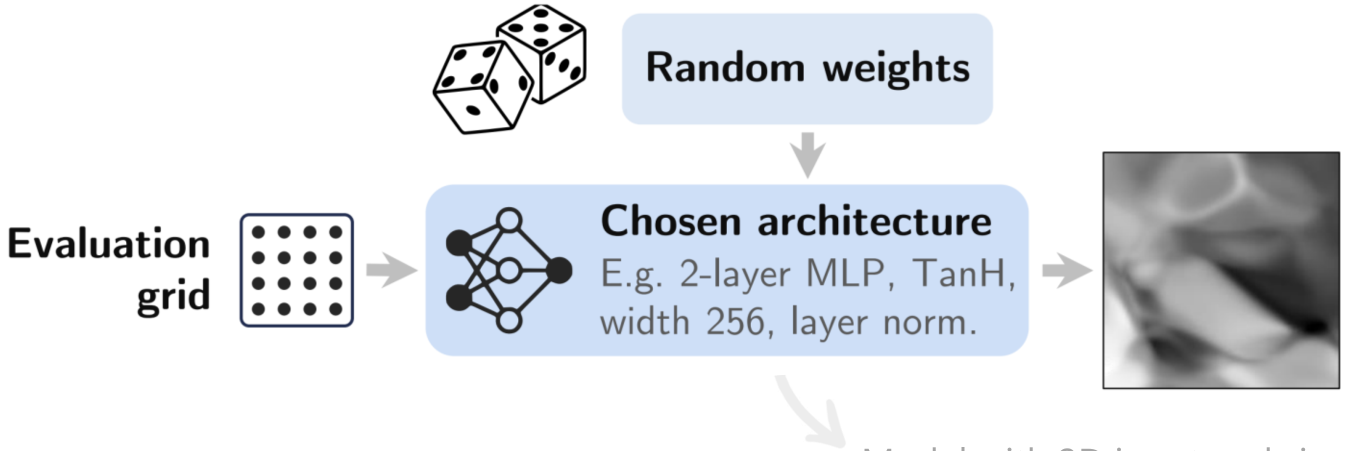
Which architecture choices provide these properties?

We measure properties in untrained networks

Existing work looks at models during/after training. (Work on the ‘simplicity bias‘, ‘spectral bias‘, etc.)

This confounds the effects of architectures/optimization.

We examine untrained (random-weight) MLPs to remove the effects of SGD.



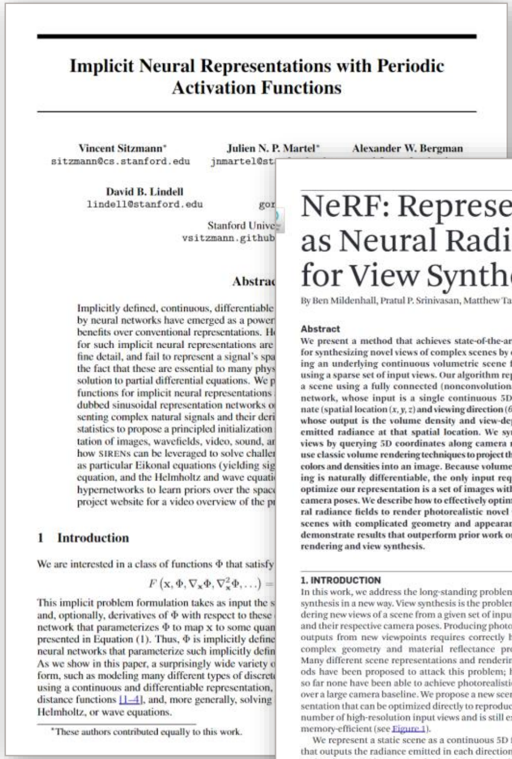
Model with 2D input and single (scalar) output (as for regression or binary classification)

Setup similar to

implicit neural representations,

a.k.a. neural fields,

a.k.a. coordinate networks.



Implicit Neural Representations with Periodic Activation Functions

Vincent Sitzmann* sitzmann@cs.stanford.edu
Julien N. P. Martel* jnmartel@st
Alexander W. Bergman

David B. Lindell lindell@stanford.edu
gor

Stanford Unive
vsitzmann.github

Abstrac

Implicitly defined, continuous, differentiable
by neural networks have emerged as a power
benefits over conventional representations. H
for such implicit neural representations are
fine detail, and fail to represent a signal's spa
the fact that these are essential to many phys
solution to partial differential equations. We p
functions for implicit neural representations
dubbed sinusoidal representation networks o
senting complex natural signals and their deri
statistics to propose a principled initialization
tation of images, wavefields, video, sound, ar
how SIRENs can be leveraged to solve challe
as particular Eikonal equations (yielding sig
equation, and the Helmholtz and wave equati
hypernetworks to learn priors over the spac
project website for a video overview of the p

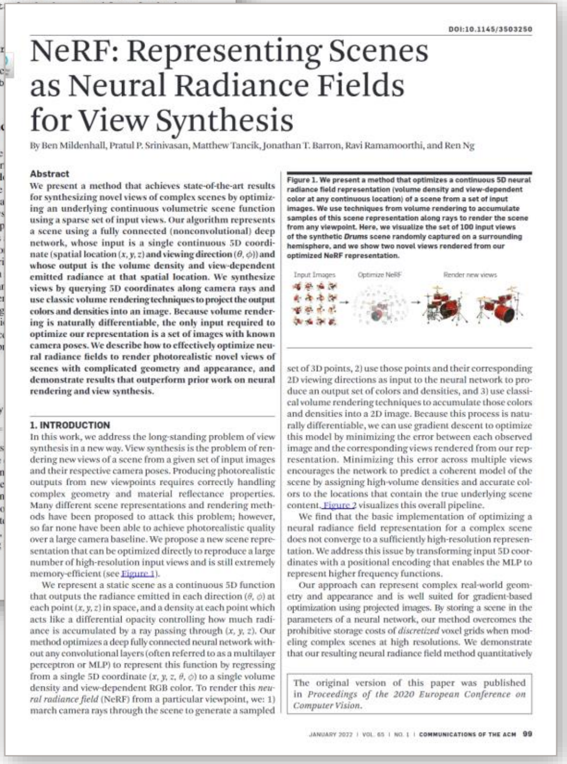
1 Introduction

We are interested in a class of functions Φ that satisfy

$$F\left(\mathbf{x}, \Phi, \nabla_{\mathbf{x}}\Phi, \nabla^2_{\mathbf{x}}\Phi, \ldots\right) =$$

This implicit problem formulation takes as input the s
and, optionally, derivatives of Φ with respect to these
network that parameterizes Φ to map x to some quan
presented in Equation (1). Thus, Φ is implicitly define
neural networks that parameterize such implicitly defin
As we show in this paper, a surprisingly wide variety o
form, such as modeling many different types of discret
using a continuous and differentiable representation,
distance functions [1–4], and, more generally, solving
Helmholtz, or wave equations.

*These authors contributed equally to this work.



DOI:10.1145/3503250

NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis

By Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng

Abstract

We present a method that achieves state-of-the-art results for synthesizing novel views of complex scenes by optimizing an underlying continuous volumetric scene function using a sparse set of input views. Our algorithm represents a scene using a fully connected (nonconvolutional) deep network, whose input is a single continuous 5D coordinate (spatial location (x, y, z) and viewing direction (θ, ϕ)) and whose output is the volume density and view-dependent emitted radiance at that spatial location. We synthesize views by querying 5D coordinates along camera rays and use classic volume rendering techniques to project the output colors and densities into an image. Because volume rendering is naturally differentiable, the only input required to optimize our representation is a set of images with known camera poses. We describe how to effectively optimize neural radiance fields to render photorealistic novel views of scenes with complicated geometry and appearance, and demonstrate results that outperform prior work on neural rendering and view synthesis.

1. INTRODUCTION

In this work, we address the long-standing problem of view synthesis in a new way. View synthesis is the problem of rendering new views of a scene from a given set of input images and their respective camera poses. Producing photorealistic outputs from new viewpoints requires correctly handling complex geometry and material reflectance properties. Many different scene representations and rendering methods have been proposed to attack this problem; however, so far none have been able to achieve photorealistic quality over a large camera baseline. We propose a new scene representation that can be optimized directly to reproduce a large number of high-resolution input views and is still extremely memory-efficient (see Figure 1).

We represent a static scene as a continuous 5D function that outputs the radiance emitted in each direction (θ, ϕ) at each point (x, y, z) in space, and a density at each point which acts like a differential opacity controlling how much radiance is accumulated by a ray passing through (x, y, z). Our method optimizes a deep fully connected neural network without any convolutional layers (often referred to as a multilayer perceptron or MLP) to represent this function by regressing from a single 5D coordinate (x, y, z, θ, ϕ) to a single volume density and view-dependent RGB color. To render this *neural radiance field* (NeRF) from a particular viewpoint, we: 1) march camera rays through the scene to generate a sampled set of 3D points, 2) use those points and their corresponding 2D viewing directions as input to the neural network to produce an output set of colors and densities, and 3) use classical volume rendering techniques to accumulate those colors and densities into a 2D image. Because this process is naturally differentiable, we can use gradient descent to optimize this model by minimizing the error between each observed image and the corresponding views rendered from our representation. Minimizing this error across multiple views encourages the network to predict a coherent model of the scene by assigning high-volume densities and accurate colors to the locations that contain the true underlying scene content. Figure 2 visualizes this overall pipeline.

We find that the basic implementation of optimizing a neural radiance field representation for a complex scene does not converge to a sufficiently high-resolution representation. We address this issue by transforming input 5D coordinates with a positional encoding that enables the MLP to represent higher frequency functions.

Our approach can represent complex real-world geometry and appearance and is well suited for gradient-based optimization using projected images. By storing a scene in the parameters of a neural network, our method overcomes the prohibitive storage costs of *discretized* voxel grids when modeling complex scenes at high resolutions. We demonstrate that our resulting neural radiance field method quantitatively

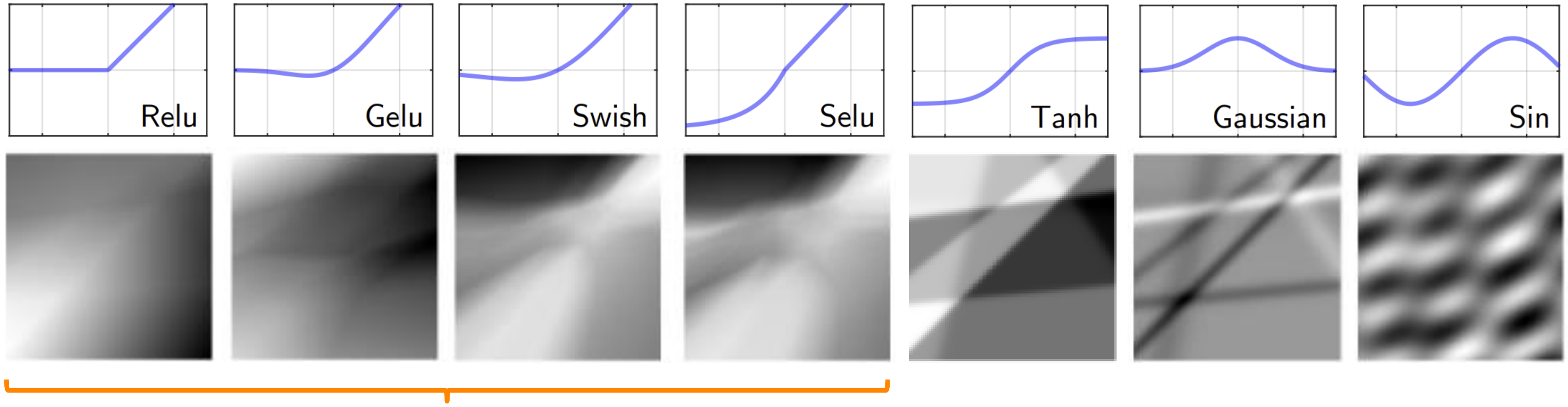
Figure 1. We present a method that optimizes a continuous 5D neural radiance field representation (volume density and view-dependent color at any continuous location) of a scene from a set of input images. We use techniques from volume rendering to accumulate samples of this scene representation along rays to render the scene from any viewpoint. Here, we visualize the set of 100 input views of the synthetic *Drums* scene randomly captured on a surrounding hemisphere, and we show two novel views rendered from our optimized NeRF representation.

The original version of this paper was published in *Proceedings of the 2020 European Conference on Computer Vision.*

JANUARY 2022 | VOL. 65 | NO. 1 | COMMUNICATIONS OF THE ACM 99

Different activation $\implies$ different function 'shape'

Examples of functions implemented by random-weight, 2D-input networks:



Popular activations:

simplicity bias,
smooth functions.

Why do we care about random-weight networks?

📖 They reflect a prior distribution over the space of functions.

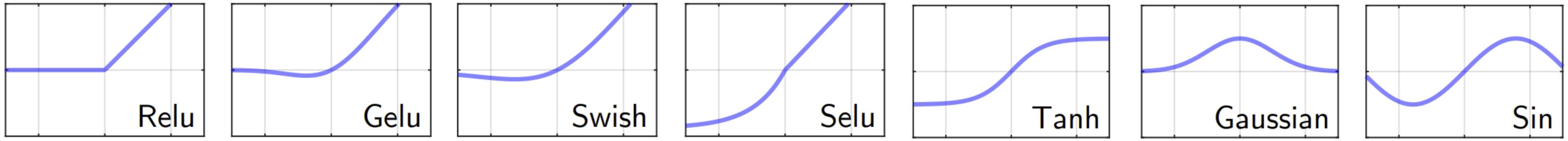
Earlier work showed that (S)GD acts like Bayesian inference with this prior.

Is SGD a Bayesian sampler? Well, almost. Mingard et al. 2022

➡️ Among the many solutions that fit the training data, those closest to the prior will be favored.

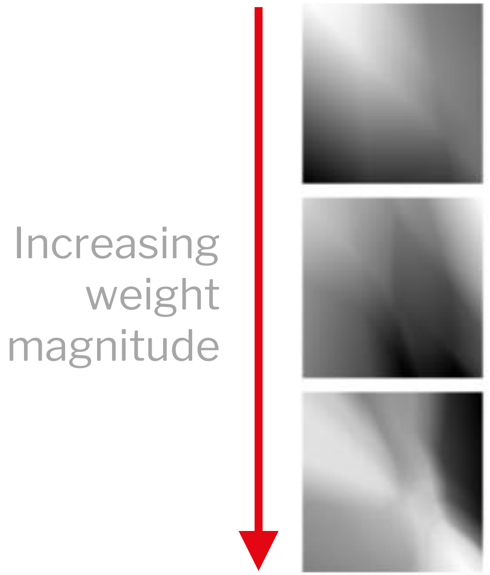
💡 Intuitively, random networks gives an idea how the trained model ‘fills in the gaps’ between training points

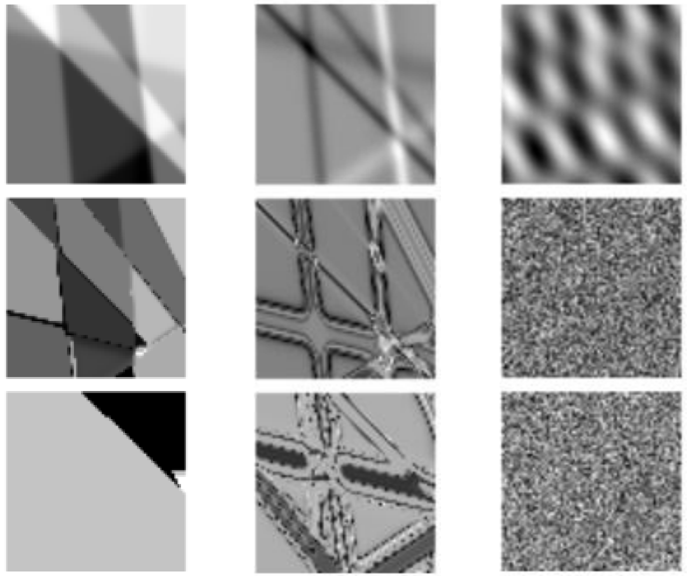
Larger weights/activations $\Longrightarrow$ higher complexity



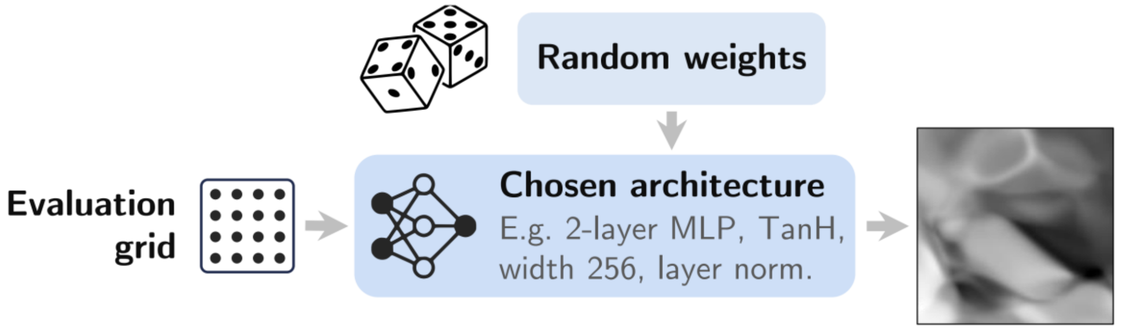
ReLU-like activations:
no/weak sensitivity to weight magnitude

Other activations:
strong sensitivity to weight magnitude

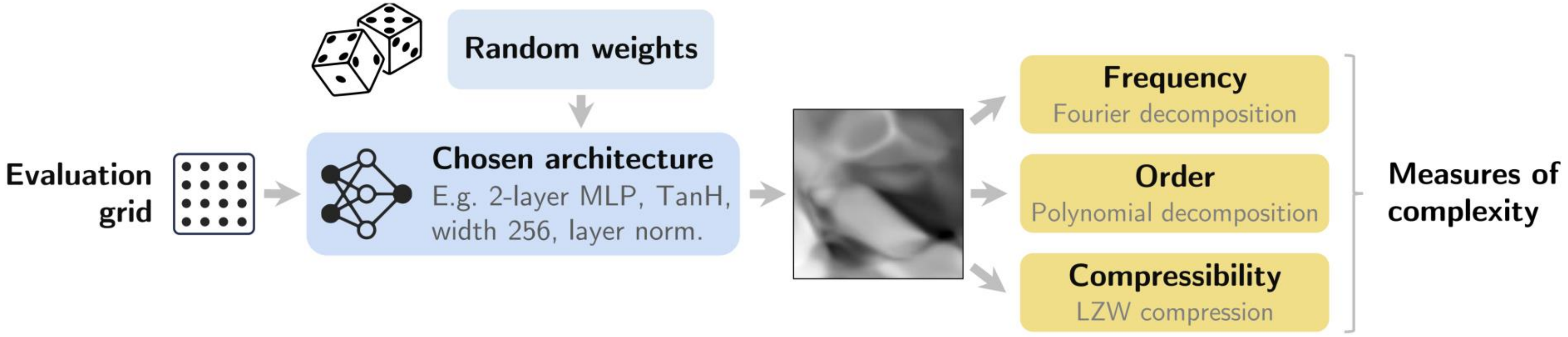




How to quantify these properties?

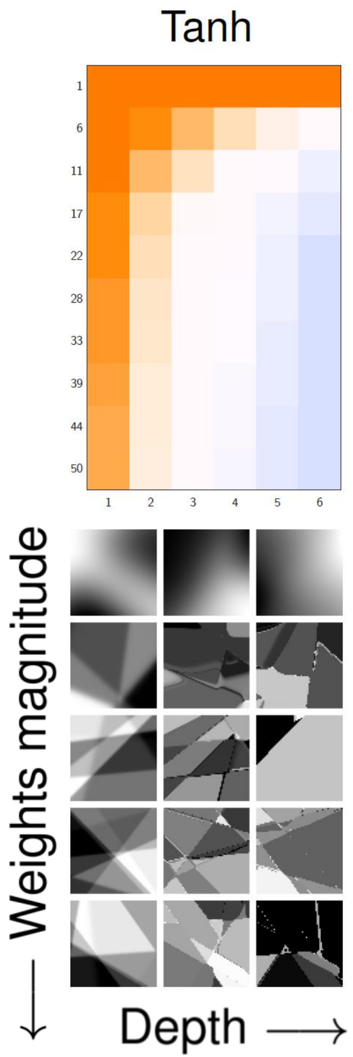


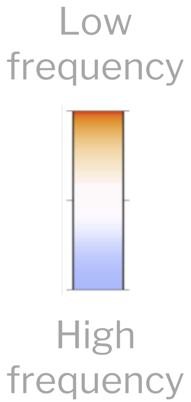
How to quantify these properties?

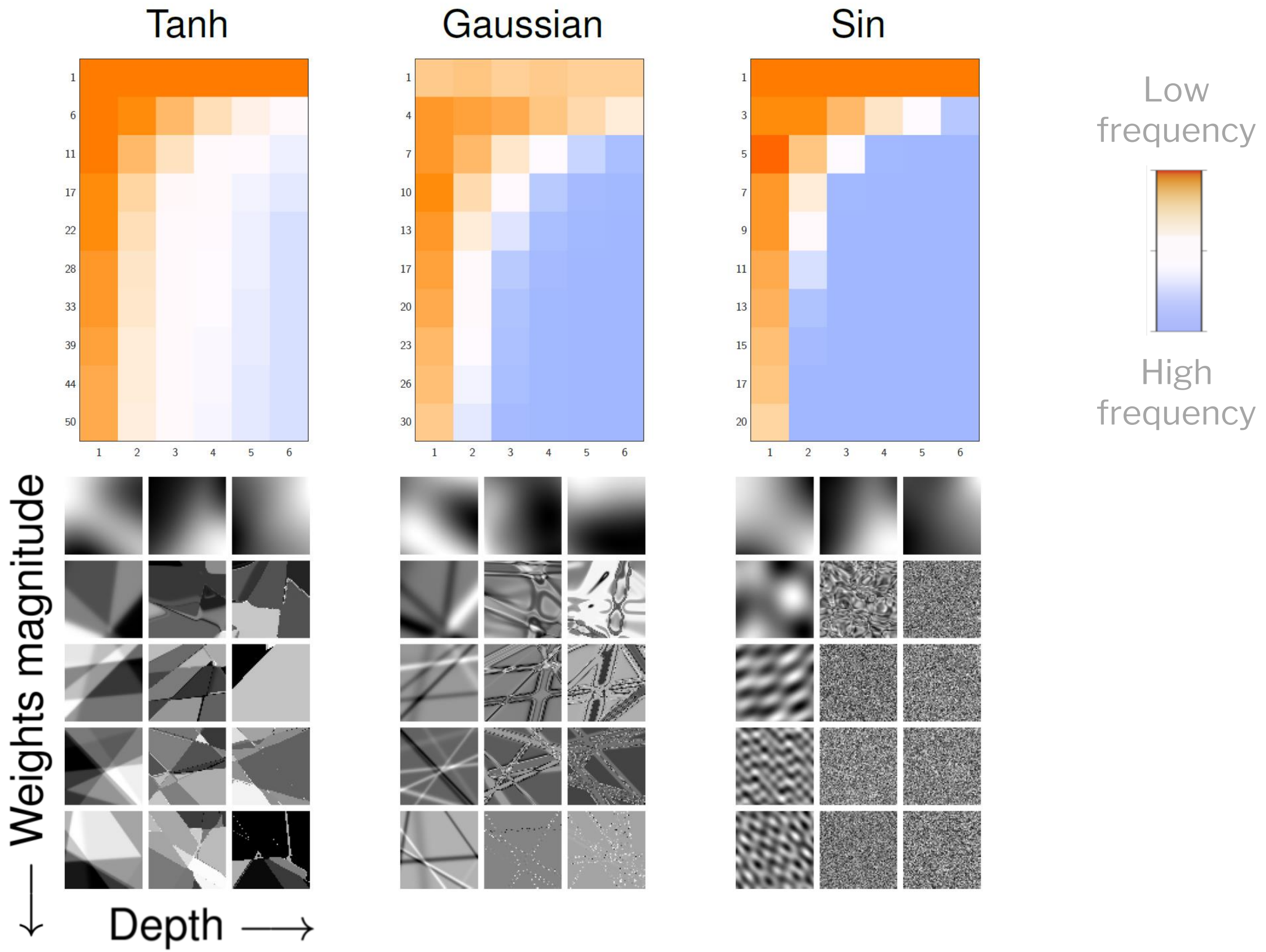


Quantifiable characterizations of inductive biases

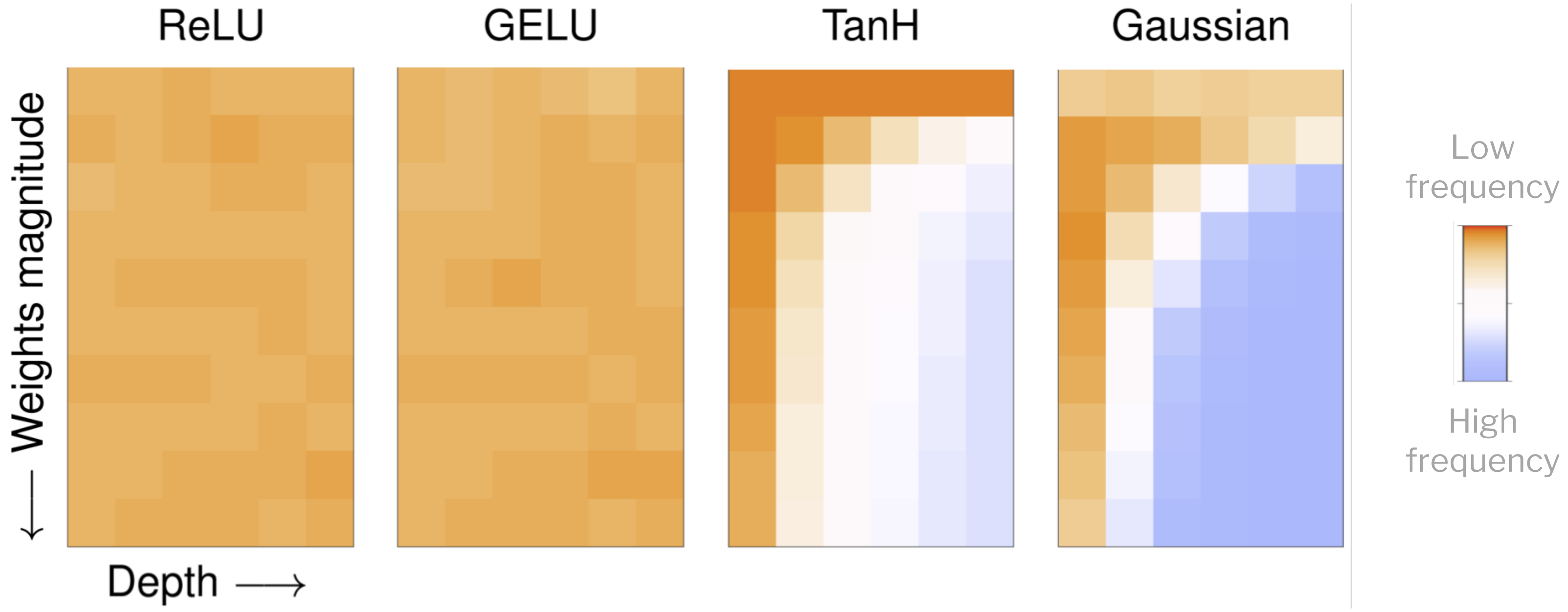
- frequency in Fourier decomposition
- order in decomposition in polynomial basis (Chebyshev, Legendre)
- compressibility (dictionary size with Lempel–Ziv compression)


Tanh
1
6
11
17
22
28
33
39
44
50
1
2
3
4
5
6
Weights magnitude
Depth

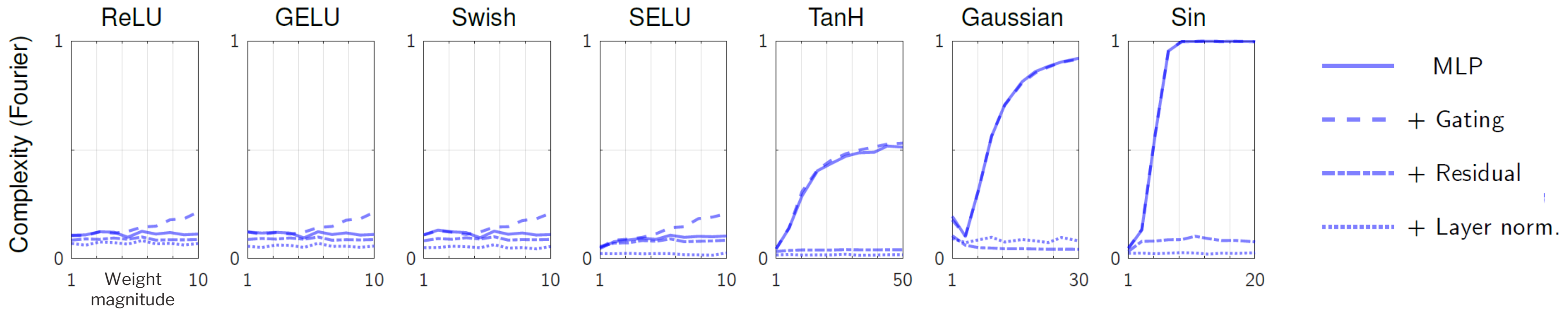

Low frequency
High frequency


Tanh
Gaussian
Sin
1 6 11 17 22 28 33 39 44 50
1 4 7 10 13 17 20 23 26 30
1 3 5 7 9 11 13 15 17 20
1 2 3 4 5 6
Low frequency
High frequency
Weights magnitude
Depth

The strong simplicity bias is unique to ReLU-like activations

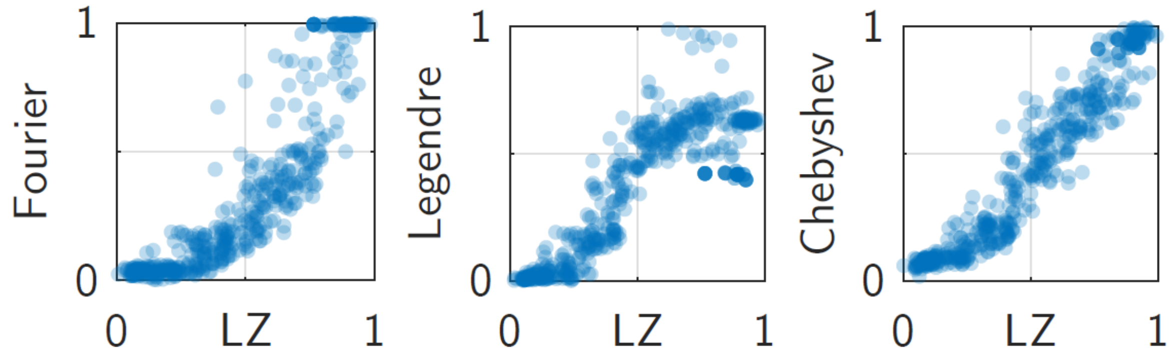


Impact of other components



Lower complexity	No impact	Higher complexity
ReLU-like activations	Width	Other activations
Layer normalization	Bias magnitudes	Depth
Residual connections		Multiplicative interactions

Different complexity measures are correlated



...Despite measuring each a different proxy for complexity:

- frequency (Fourier)
- order of polynomial decomposition (Legendre, Chebyshev)
- compressibility (LZ)

Is this still relevant after training?

📝 We correlated complexity at init. with **generalization performance** after training.

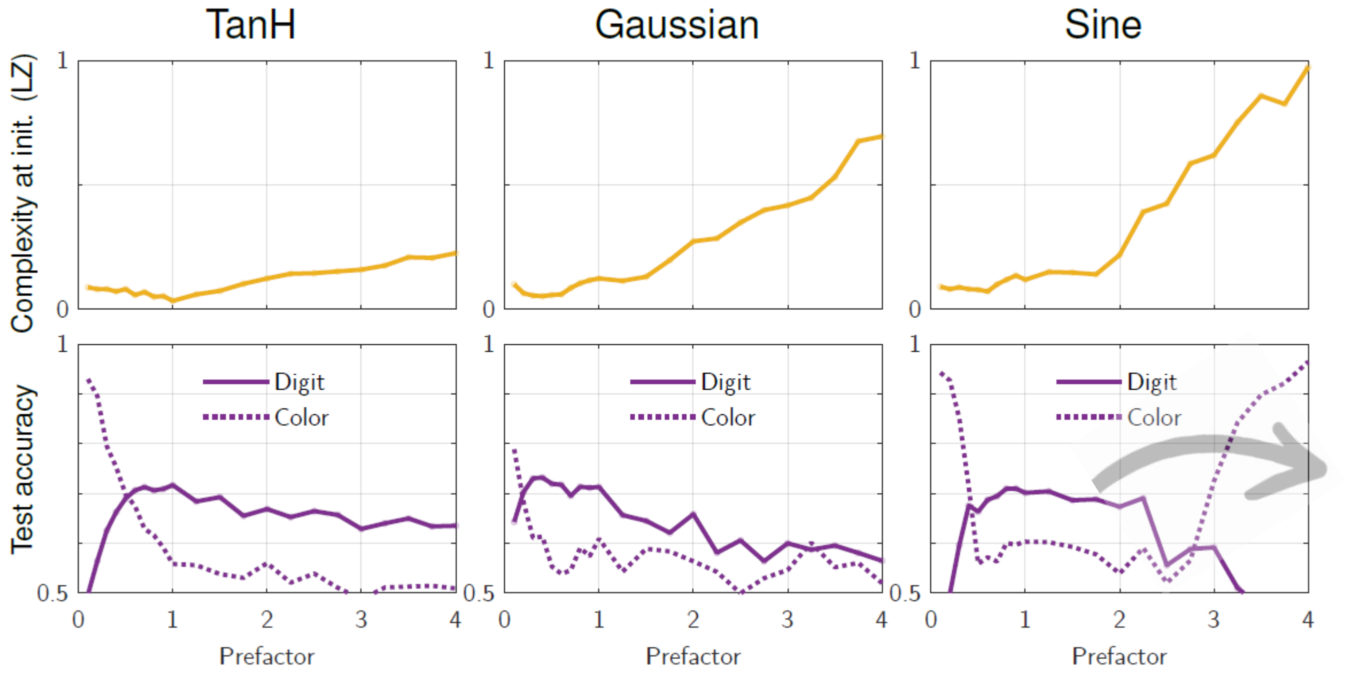
📈 Main conclusion: **generalization** occurs when the architecture's preferred complexity matches the target function's complexity.

❗ In some cases, a bias towards **higher complexity is desirable**.

Examples: learning INRs, learning the parity function, avoiding shortcut learning.

Mitigating shortcut learning (Colored-MNIST)

We tweak the preferred complexity with a fixed prefactor before the activations.

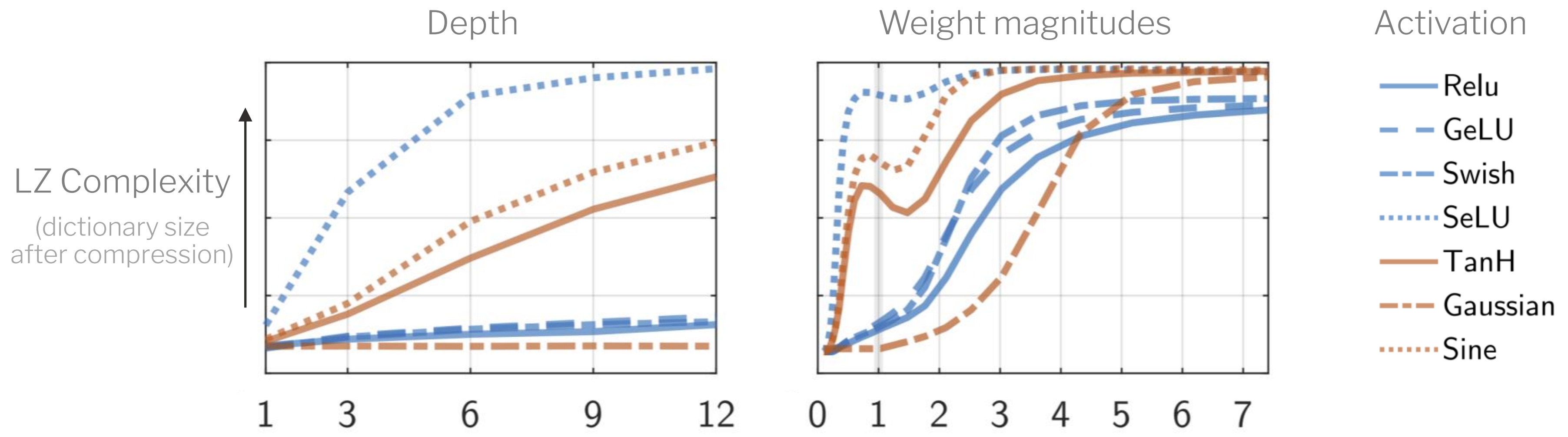


Sweet spot to learn the digit (task-specific!)

Transformers are biased towards compressible sequences

We sample sequences of tokens from an **untrained** GPT-2.

Similar interventions (to those observed with MLPs) increase the complexity of the sequences.



Transformers seem to **inherit inductive biases from their building blocks** via mechanisms similar to those in simple MLPs.

Take-aways

⭐ Fresh explanations for the **performance of neural networks** independent from gradient-based training.

🔍 The '**simplicity bias**' is not a universal property of all neural architectures.
It can be explained without gradient descent.
It is not always desirable. E.g. causing shortcut learning, preventing learning complex patterns, ...

💡 These findings suggest possibilities for nudging inductive biases and controlling the functions implemented by trained models.
E.g. via reparameterization, learning activation functions, ...