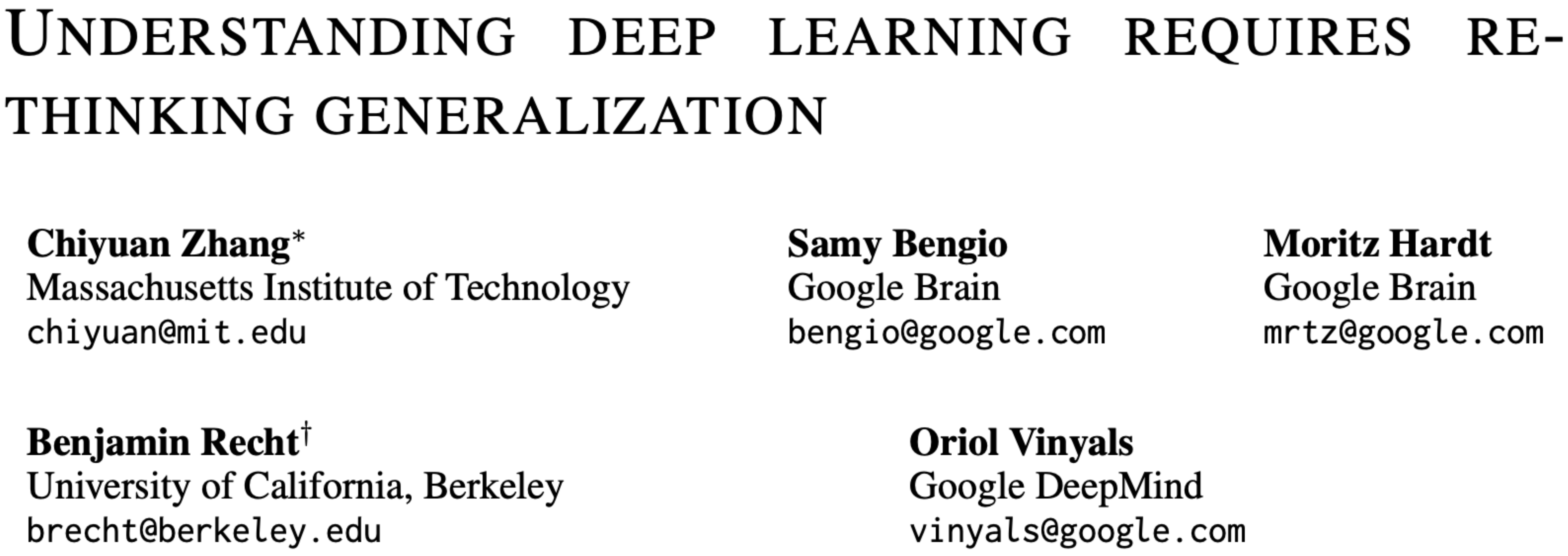
Convergence and Generalization of Neural Networks:
Neural Tangent Kernels and Beyond

fzeng 2025-02-21

The big questions of our times...

- You wake up one Saturday morning,
- and you ask yourself...

Modern neural networks are overparameterized...



UNDERSTANDING DEEP LEARNING REQUIRES RETHINKING GENERALIZATION

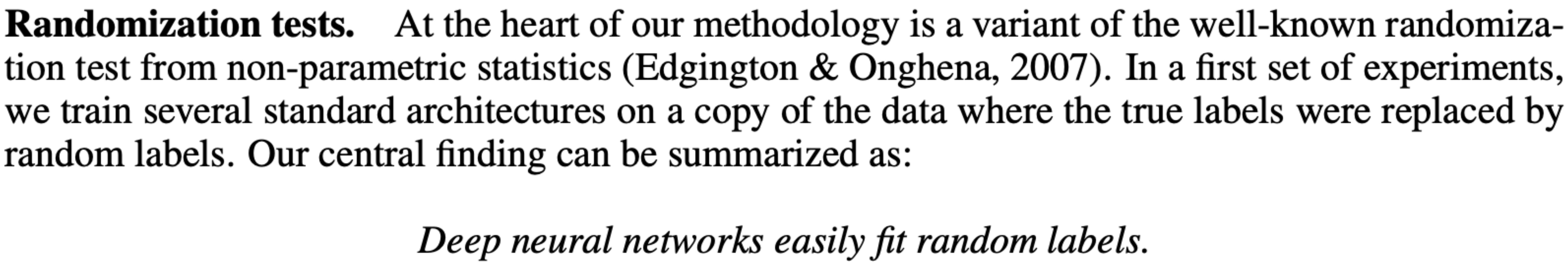
Chiyuan Zhang[*]
Massachusetts Institute of Technology
chiyuan@mit.edu

Samy Bengio
Google Brain
bengio@google.com

Moritz Hardt
Google Brain
mrtz@google.com

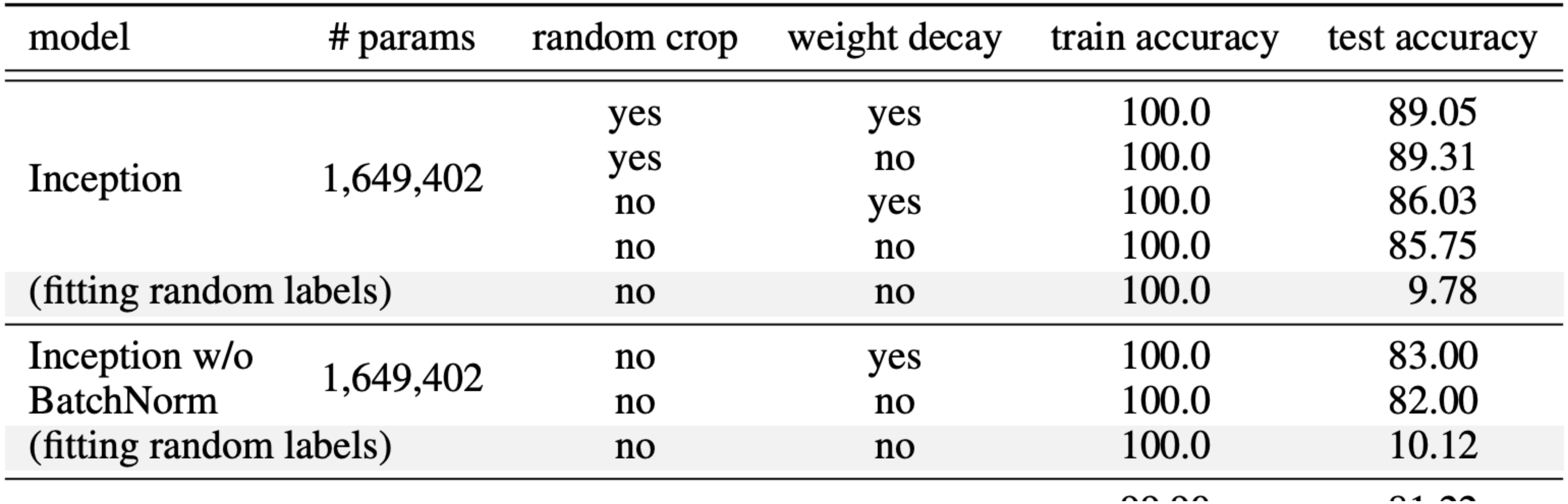
Benjamin Recht[†]
University of California, Berkeley
brecht@berkeley.edu

Oriol Vinyals
Google DeepMind
vinyals@google.com



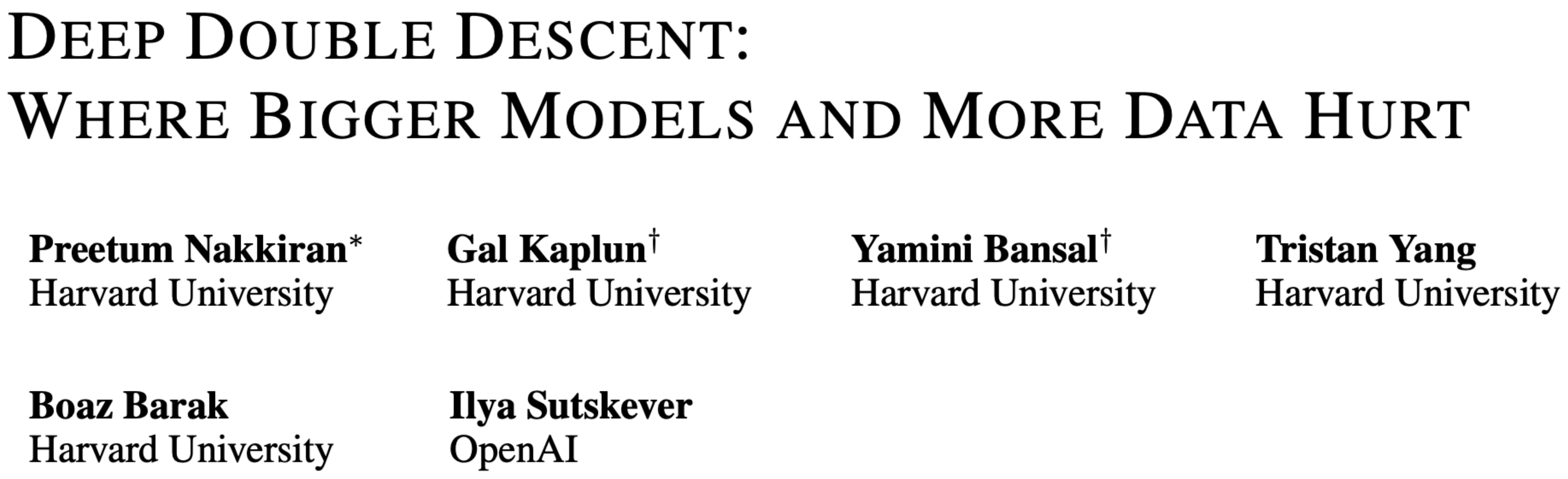
Randomization tests. At the heart of our methodology is a variant of the well-known randomization test from non-parametric statistics (Edgington & Onghena, 2007). In a first set of experiments, we train several standard architectures on a copy of the data where the true labels were replaced by random labels. Our central finding can be summarized as:

Deep neural networks easily fit random labels.



model	# params	random crop	weight decay	train accuracy	test accuracy
Inception	1,649,402	yes	yes	100.0	89.05
		yes	no	100.0	89.31
		no	yes	100.0	86.03
		no	no	100.0	85.75
(fitting random labels)		no	no	100.0	9.78
Inception w/o BatchNorm	1,649,402	no	yes	100.0	83.00
		no	no	100.0	82.00
(fitting random labels)		no	no	100.0	10.12
				99.90	81.22

But yet it seems to help instead of hurt



Deep Double Descent:
Where Bigger Models and More Data Hurt

Preetum Nakkiran[*]
Harvard University

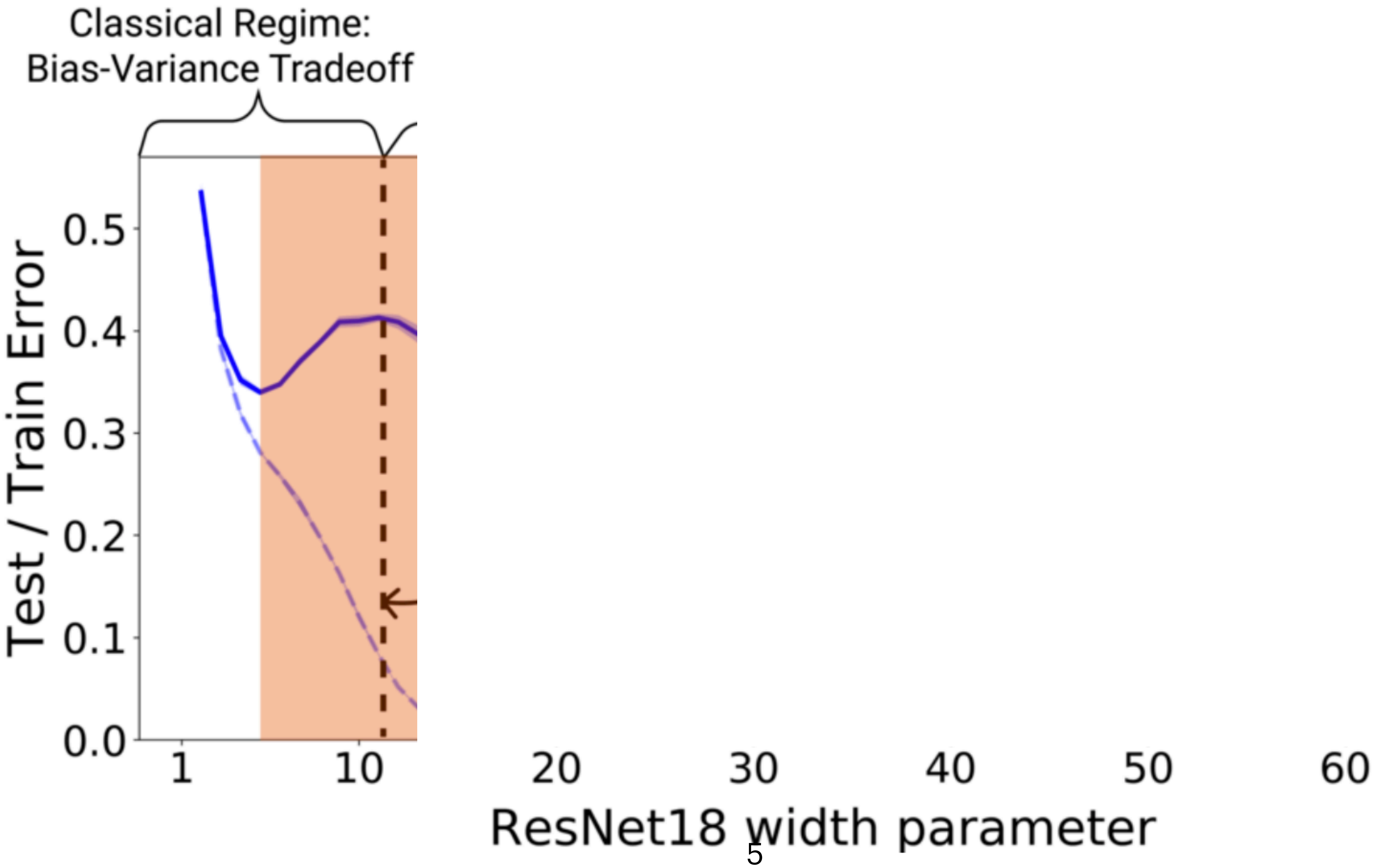
Gal Kaplun[†]
Harvard University

Yamini Bansal[†]
Harvard University

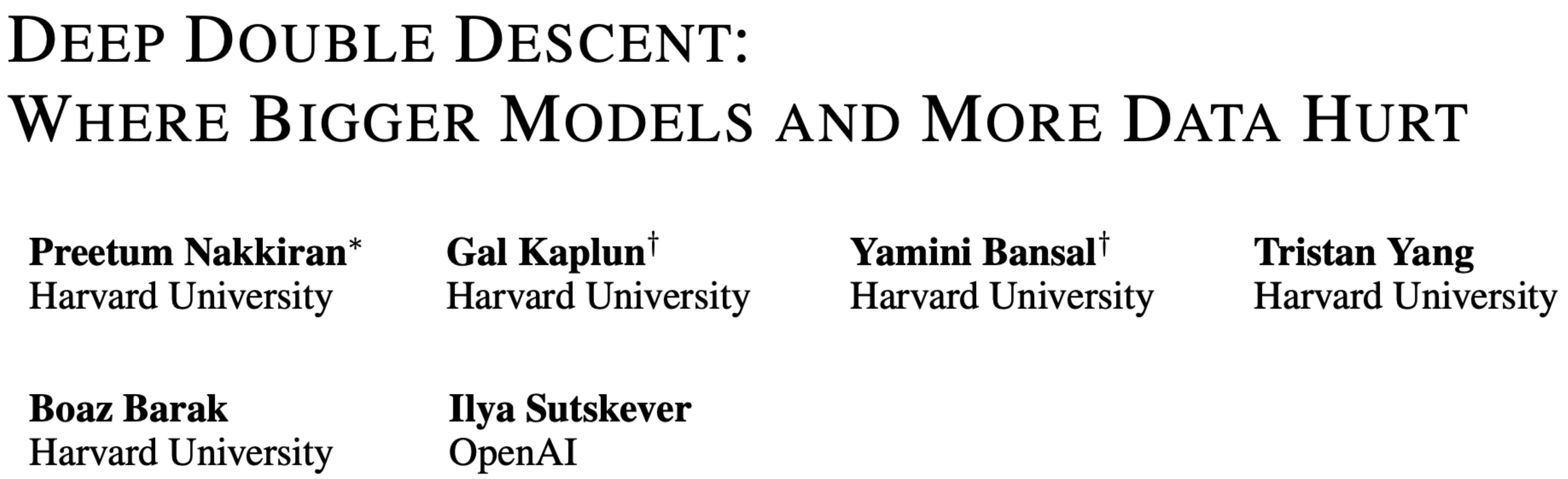
Tristan Yang
Harvard University

Boaz Barak
Harvard University

Ilya Sutskever
OpenAI



But yet it seems to help instead of hurt



DEEP DOUBLE DESCENT:
WHERE BIGGER MODELS AND MORE DATA HURT

Preetum Nakkiran*
Harvard University

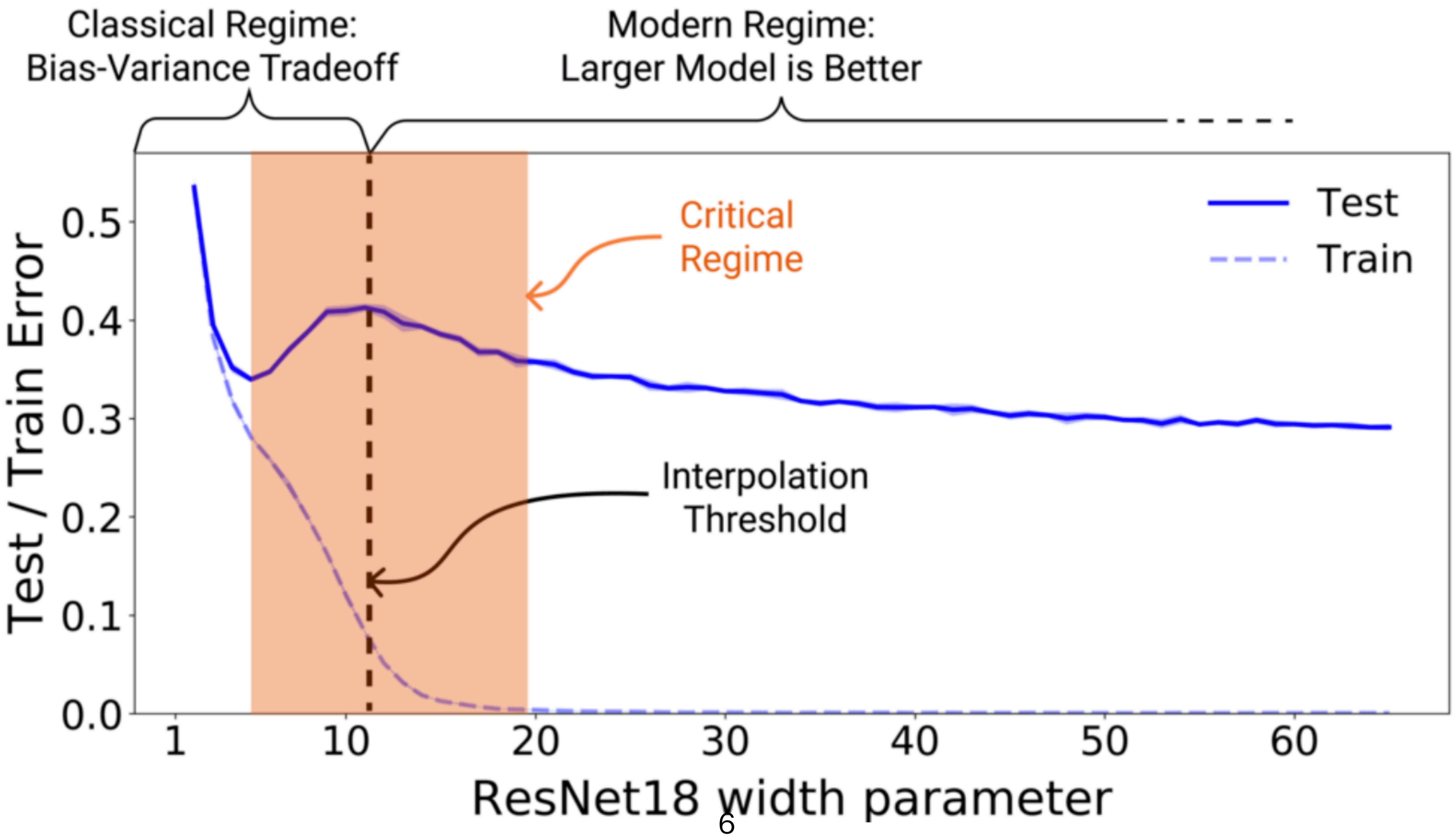
Gal Kaplun†
Harvard University

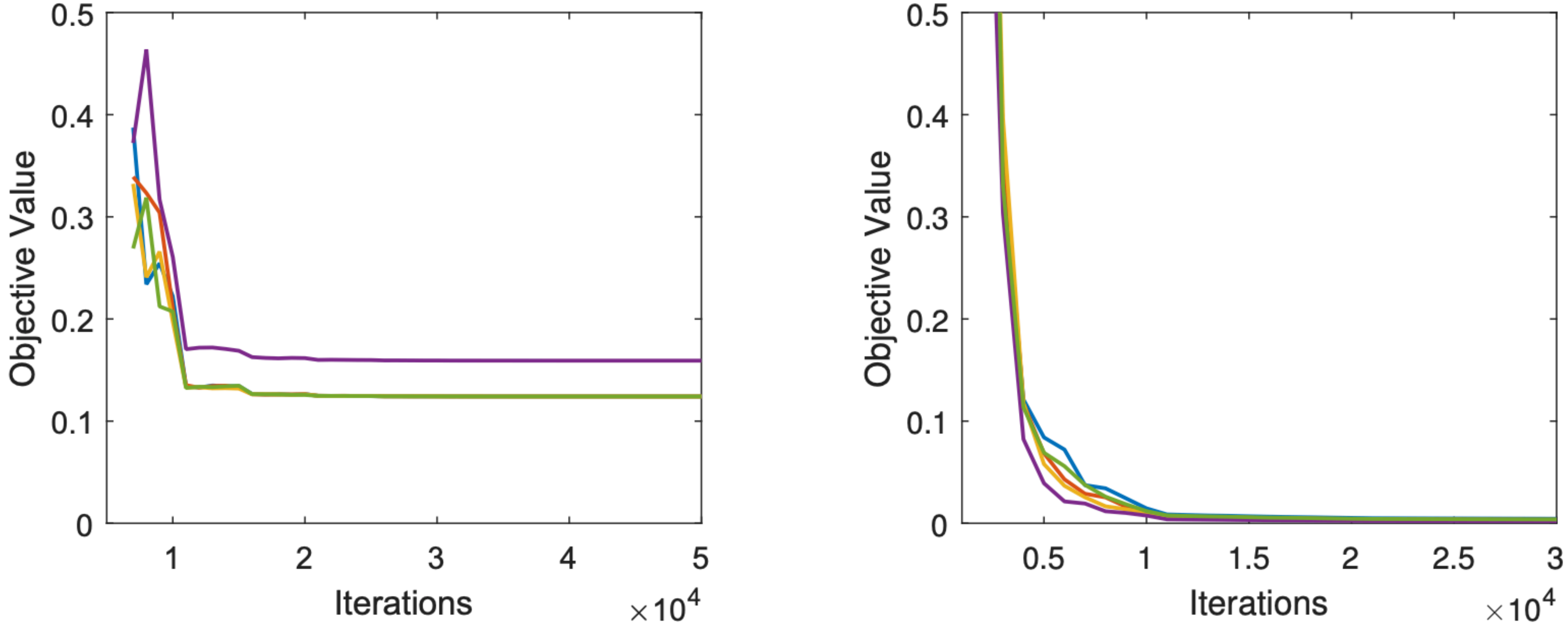
Yamini Bansal†
Harvard University

Tristan Yang
Harvard University

Boaz Barak
Harvard University

Ilya Sutskever
OpenAI



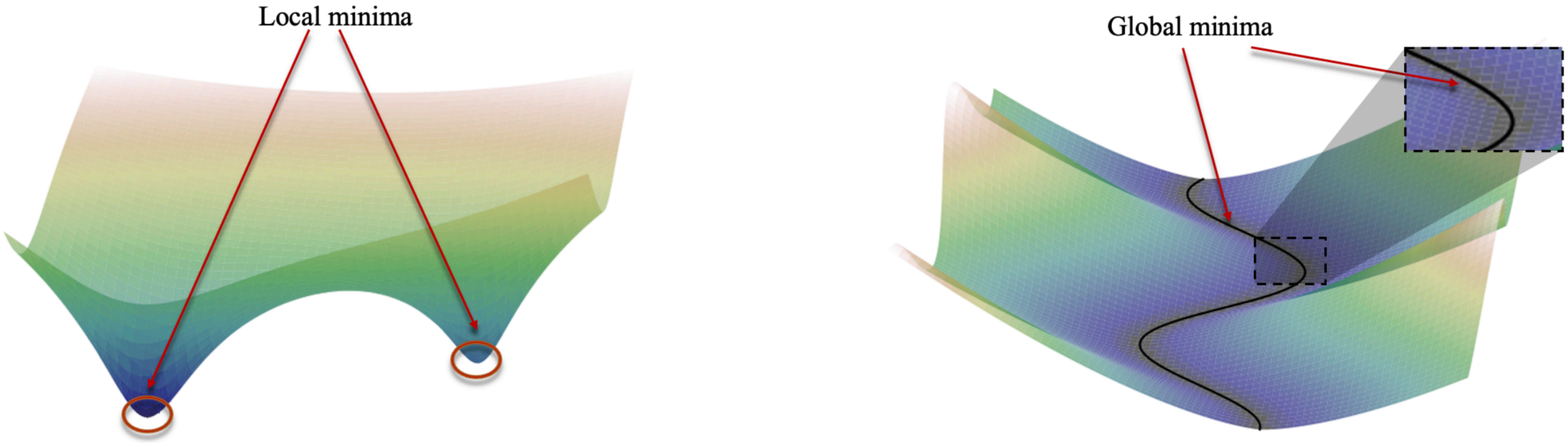


(a) Original Landscape (b) Overparametrized Landscape

Figure: Data is generated from network with $k_0 = 50$ neurons. Overparametrized network has $k = 100$ neurons[1].

Loss landscapes and optimization in over-parameterized non-linear systems and neural networks

Chaoyue Liu[a], Libin Zhu[b,c], and Mikhail Belkin[c]

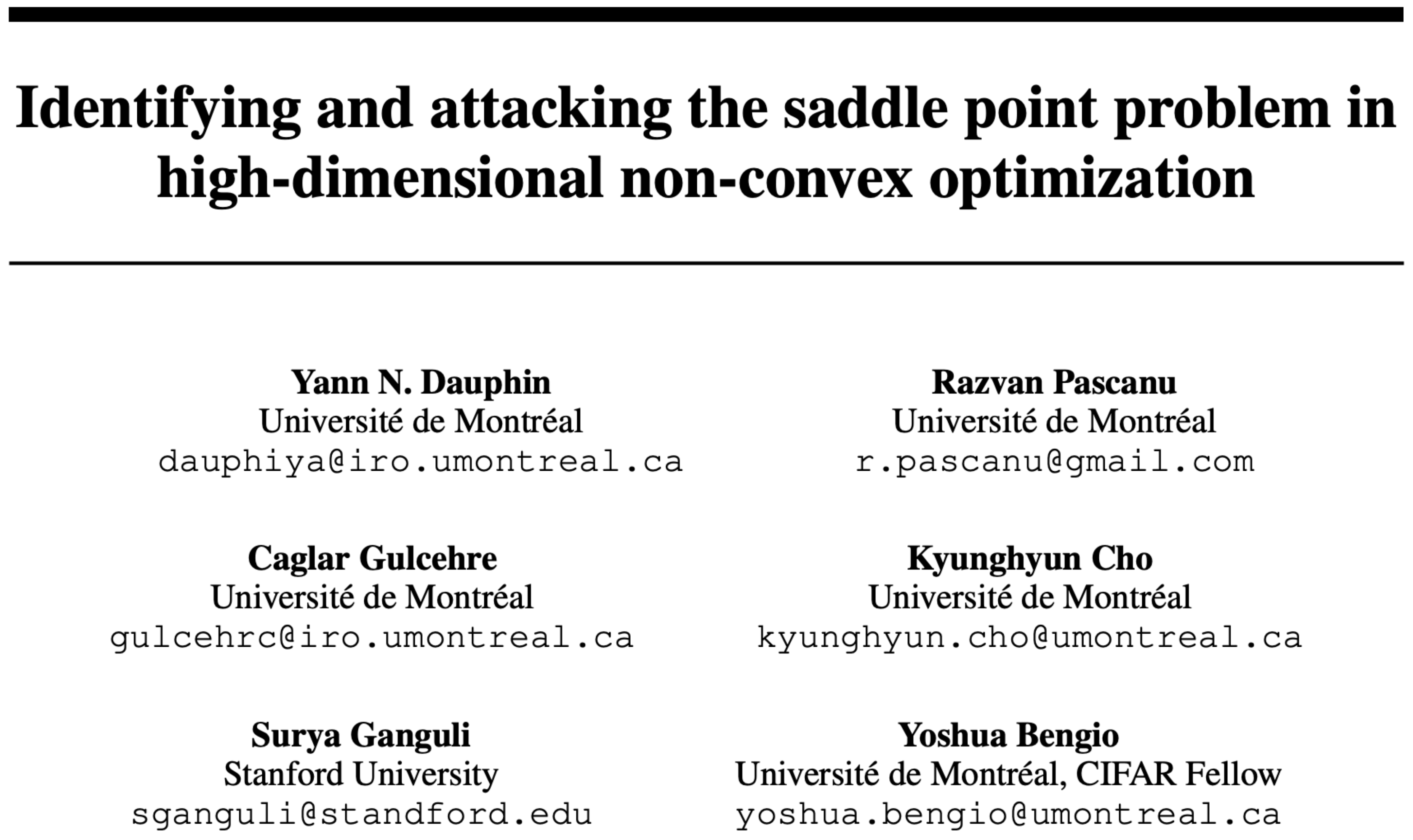


(a) Loss landscape of under-parameterized models

(b) Loss landscape of over-parameterized models

Why does SGD work so well in practice...

- Even though it could get stuck in exponentially many saddle points?



Identifying and attacking the saddle point problem in high-dimensional non-convex optimization

Yann N. Dauphin
Université de Montréal
dauphiya@iro.umontreal.ca

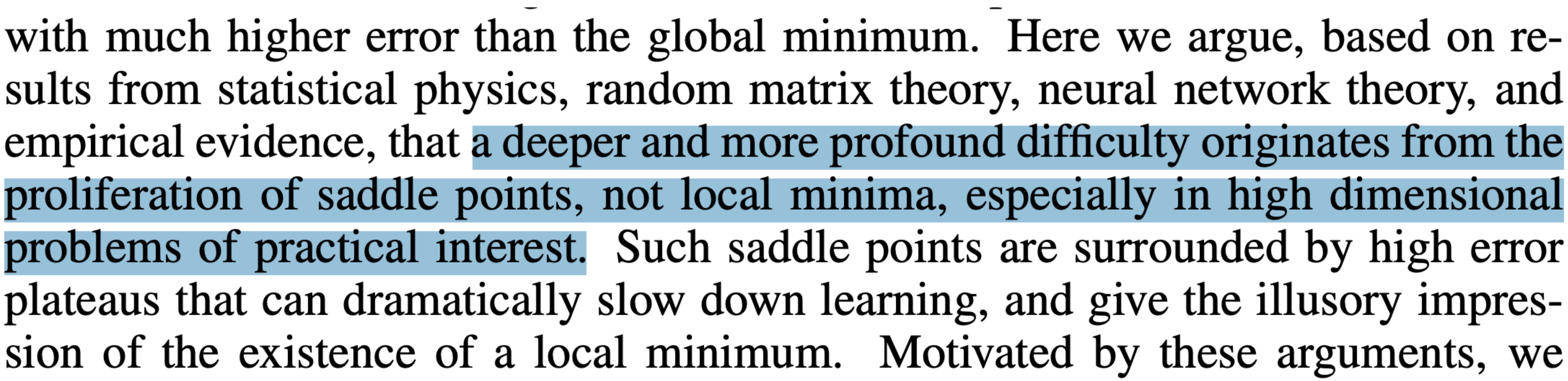
Razvan Pascanu
Université de Montréal
r.pascanu@gmail.com

Caglar Gulcehre
Université de Montréal
gulcehrc@iro.umontreal.ca

Kyunghyun Cho
Université de Montréal
kyunghyun.cho@umontreal.ca

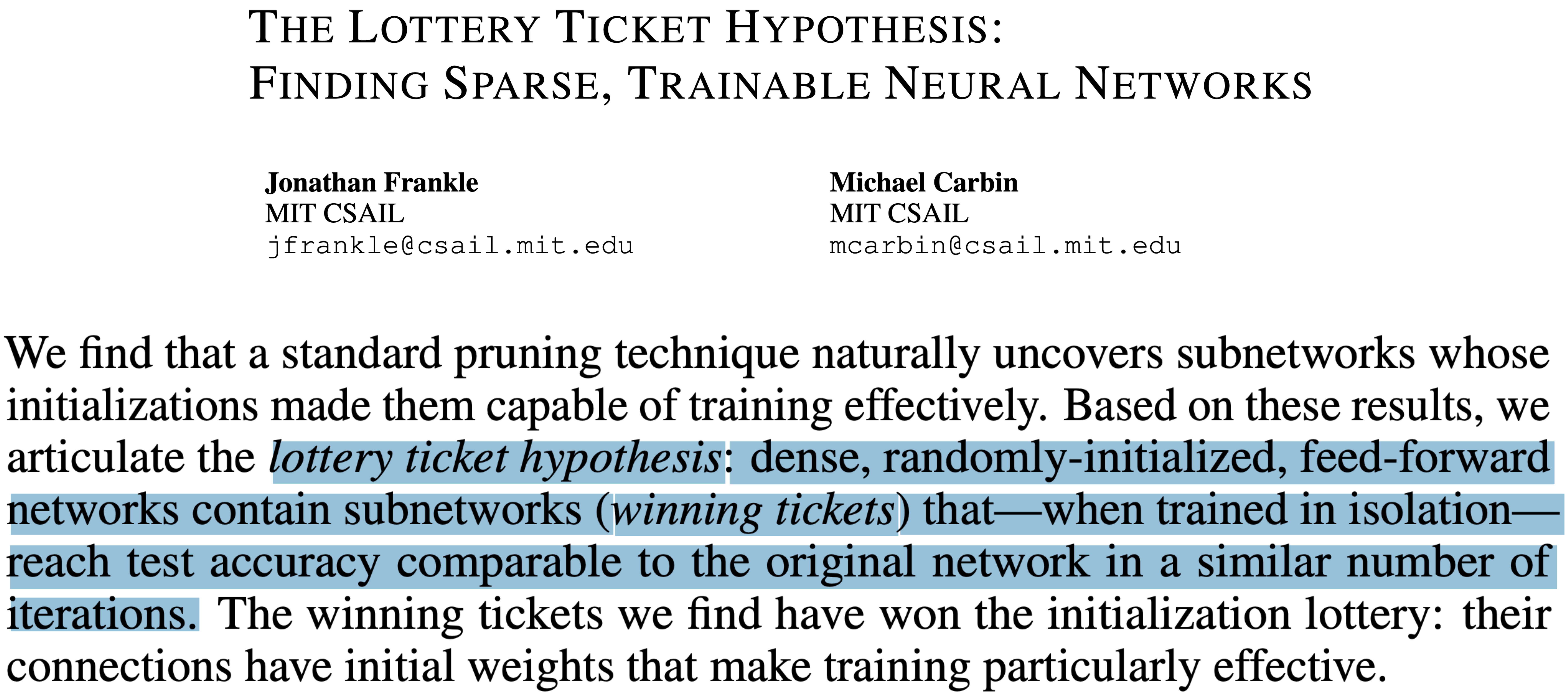
Surya Ganguli
Stanford University
sganguli@standford.edu

Yoshua Bengio
Université de Montréal, CIFAR Fellow
yoshua.bengio@umontreal.ca



with much higher error than the global minimum. Here we argue, based on results from statistical physics, random matrix theory, neural network theory, and empirical evidence, that a deeper and more profound difficulty originates from the proliferation of saddle points, not local minima, especially in high dimensional problems of practical interest. Such saddle points are surrounded by high error plateaus that can dramatically slow down learning, and give the illusory impression of the existence of a local minimum. Motivated by these arguments, we

And yet they are so sensitive to initialization?



THE LOTTERY TICKET HYPOTHESIS:
FINDING SPARSE, TRAINABLE NEURAL NETWORKS

Jonathan Frankle
MIT CSAIL
jfrankle@csail.mit.edu

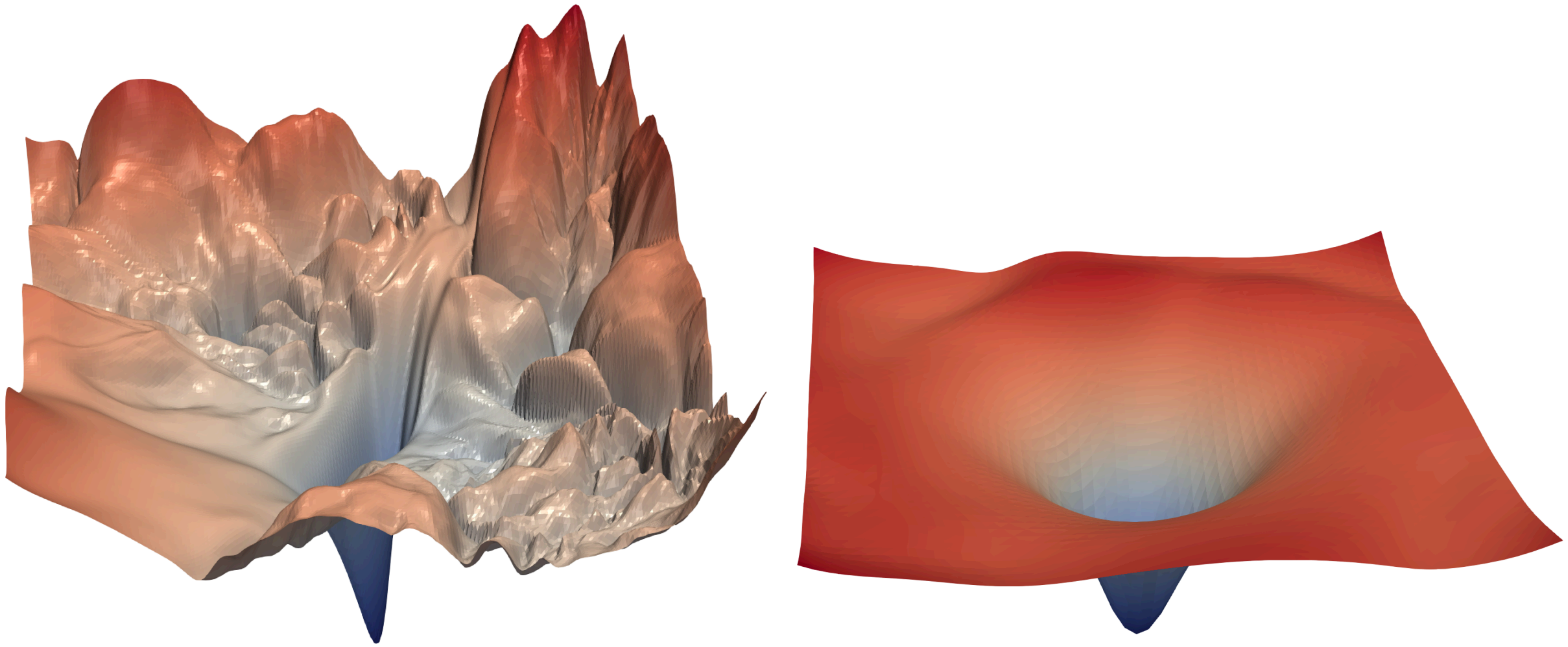
Michael Carbin
MIT CSAIL
mcarbin@csail.mit.edu

We find that a standard pruning technique naturally uncovers subnetworks whose initializations made them capable of training effectively. Based on these results, we articulate the *lottery ticket hypothesis*: dense, randomly-initialized, feed-forward networks contain subnetworks (*winning tickets*) that—when trained in isolation—reach test accuracy comparable to the original network in a similar number of iterations. The winning tickets we find have won the initialization lottery: their connections have initial weights that make training particularly effective.

Are residual connections strictly necessary to train deep networks?

Deep Residual Learning for Image Recognition

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

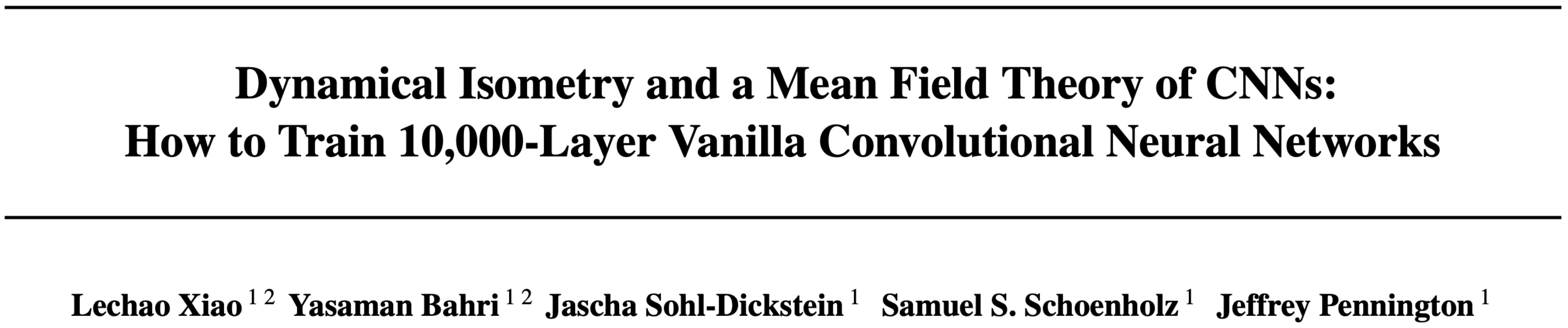


(a) without skip connections

(b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

Or is there something deeper at play?



Dynamical Isometry and a Mean Field Theory of CNNs: How to Train 10,000-Layer Vanilla Convolutional Neural Networks

Lechao Xiao[1 2] Yasaman Bahri[1 2] Jascha Sohl-Dickstein[1] Samuel S. Schoenholz[1] Jeffrey Pennington[1]

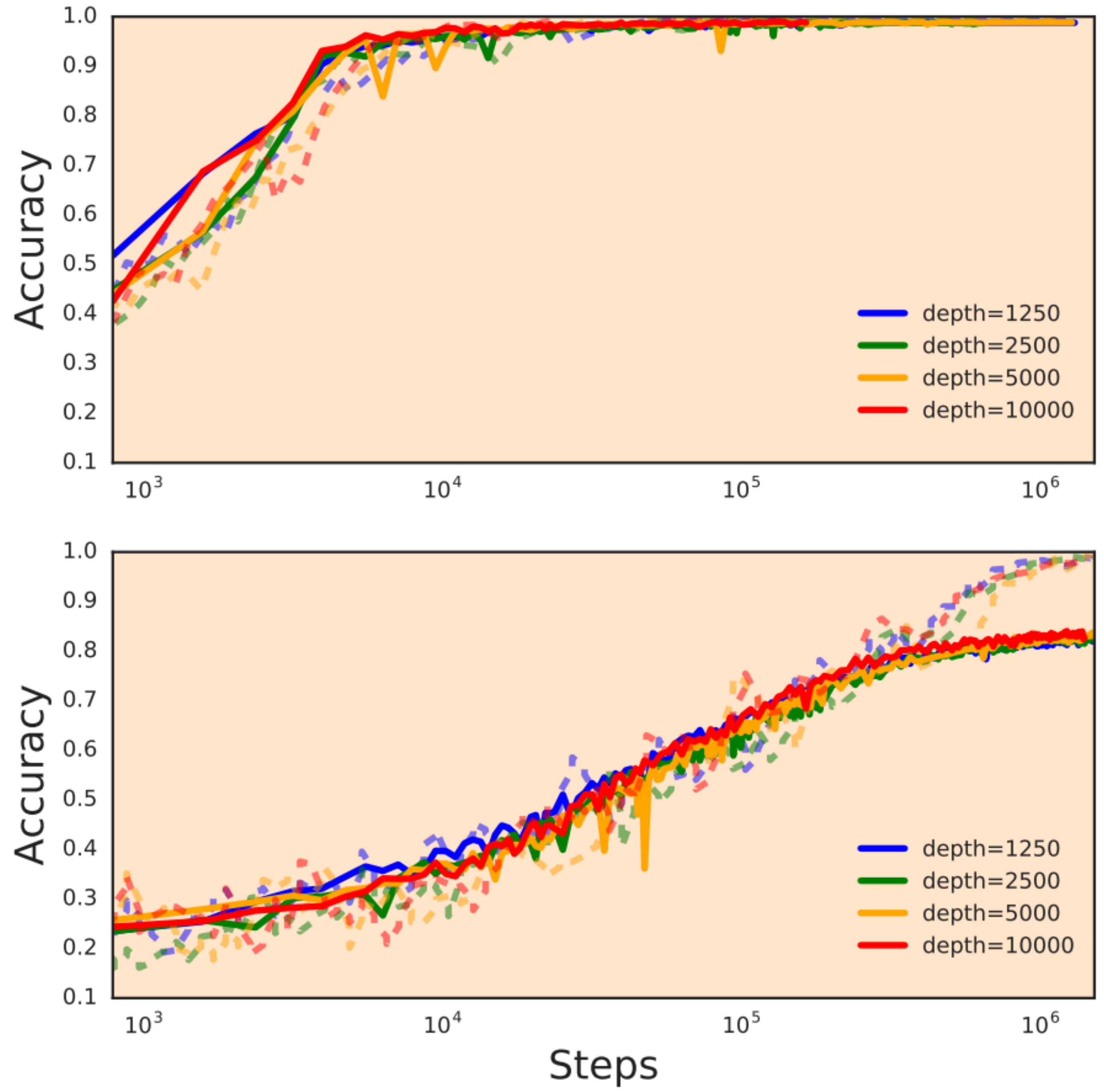


Figure 1. Extremely deep CNNs can be trained without the use of batch normalization or residual connections simply by using a Delta-Orthogonal initialization with critical weight and bias variance and appropriate (in this case, tanh) nonlinearity. Test (solid) and training (dashed) curves on MNIST (top) and CIFAR-10 (bottom) for depths 1,250, 2,500, 5,000, and 10, 000.

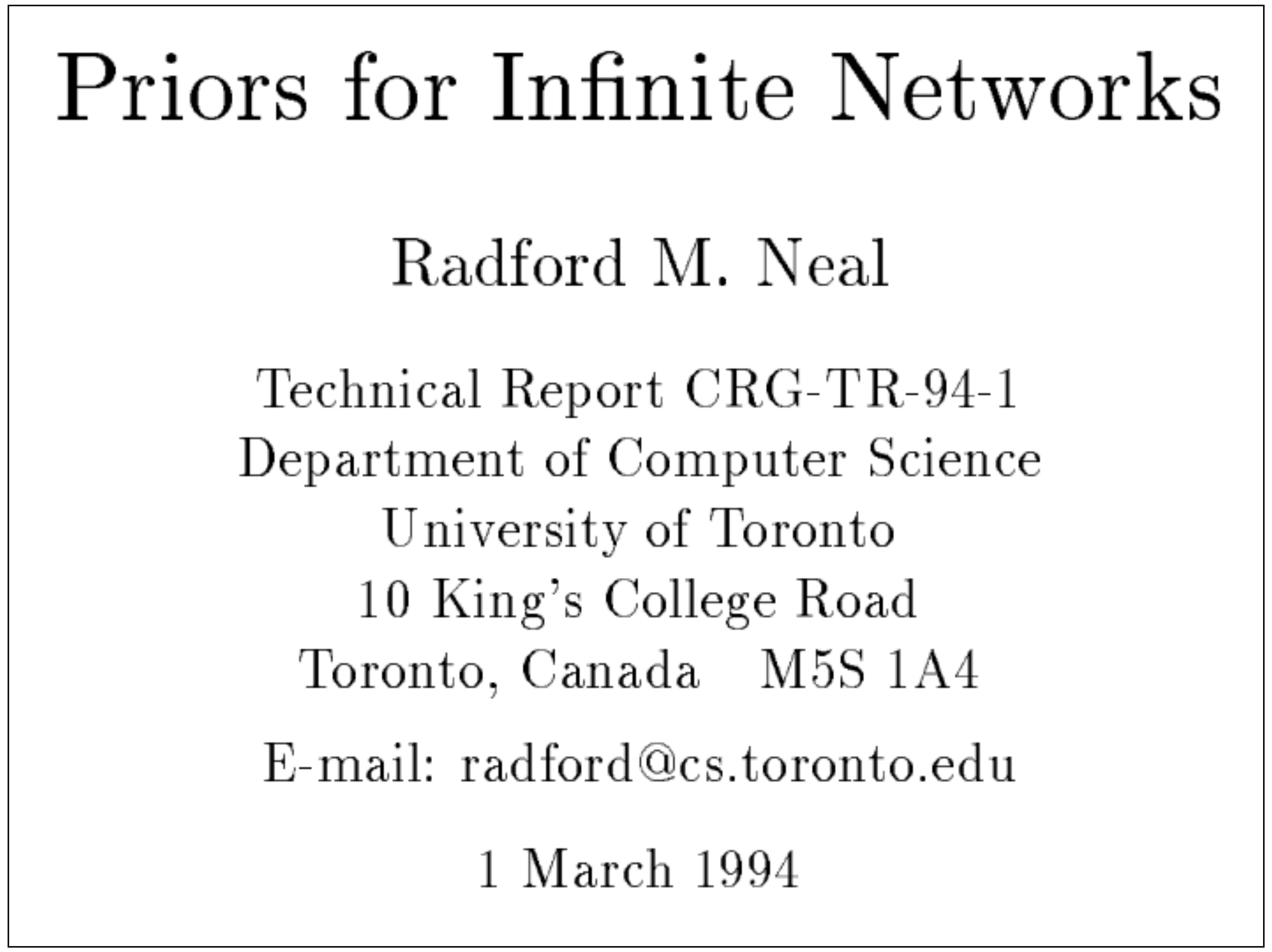
Overview

- [Neal ’94] Infinite width single-layer neural networks are Gaussian processes (NNGP)
- [Jacot et al. ’18] NTK paper
    - Extended NNGP to multi-layer MLPs
    - Introduced NTK to analyze backward pass
- [Yang et al. ’19-23] Tensor Program series
    - Architecture universality of NNGP and NTK
    - Feature learning for NTK via Maximal Update Parametrization (μP)
    - Zero-shot Hyper-parameter transfer

Part 1: Infinite-width Single-Layer Neural Networks are Gaussian Processes

Forward Pass in the Infinite-Width Limit

- If we adopt a Bayesian view towards neural networks, then the choice of initialization reflect our priors
- What does a Gaussian prior imply for what the (untrained) network computes?



Priors for Infinite Networks

Radford M. Neal

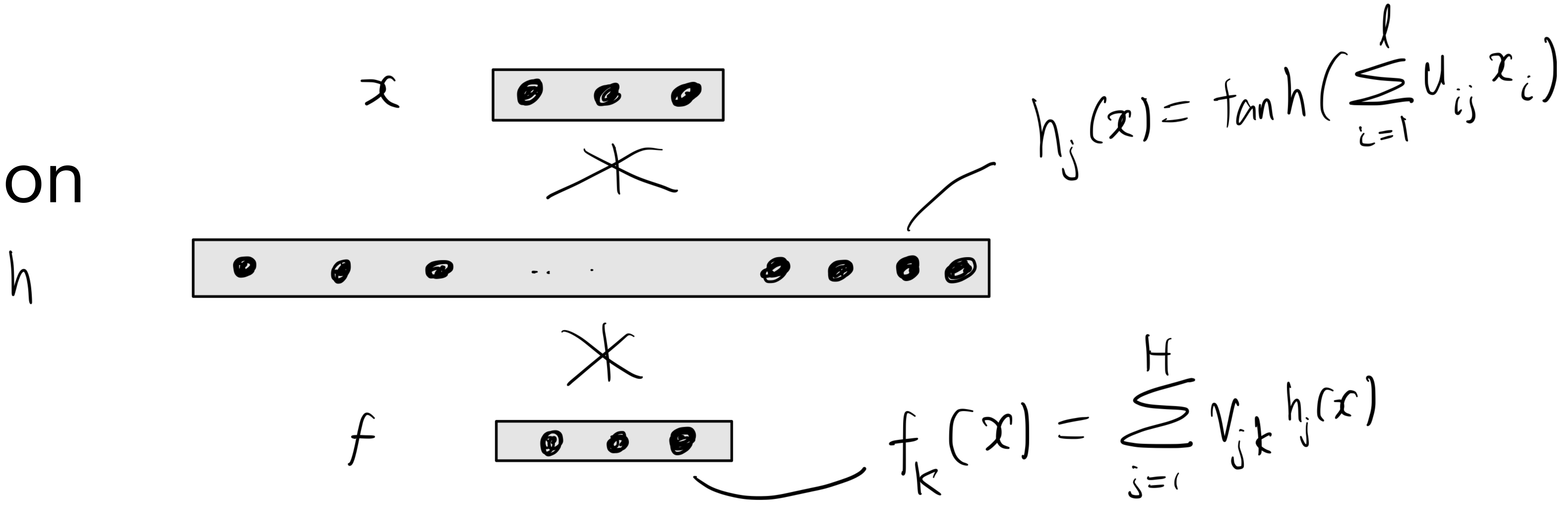
Technical Report CRG-TR-94-1
Department of Computer Science
University of Toronto
10 King's College Road
Toronto, Canada M5S 1A4

E-mail: radford@cs.toronto.edu

1 March 1994

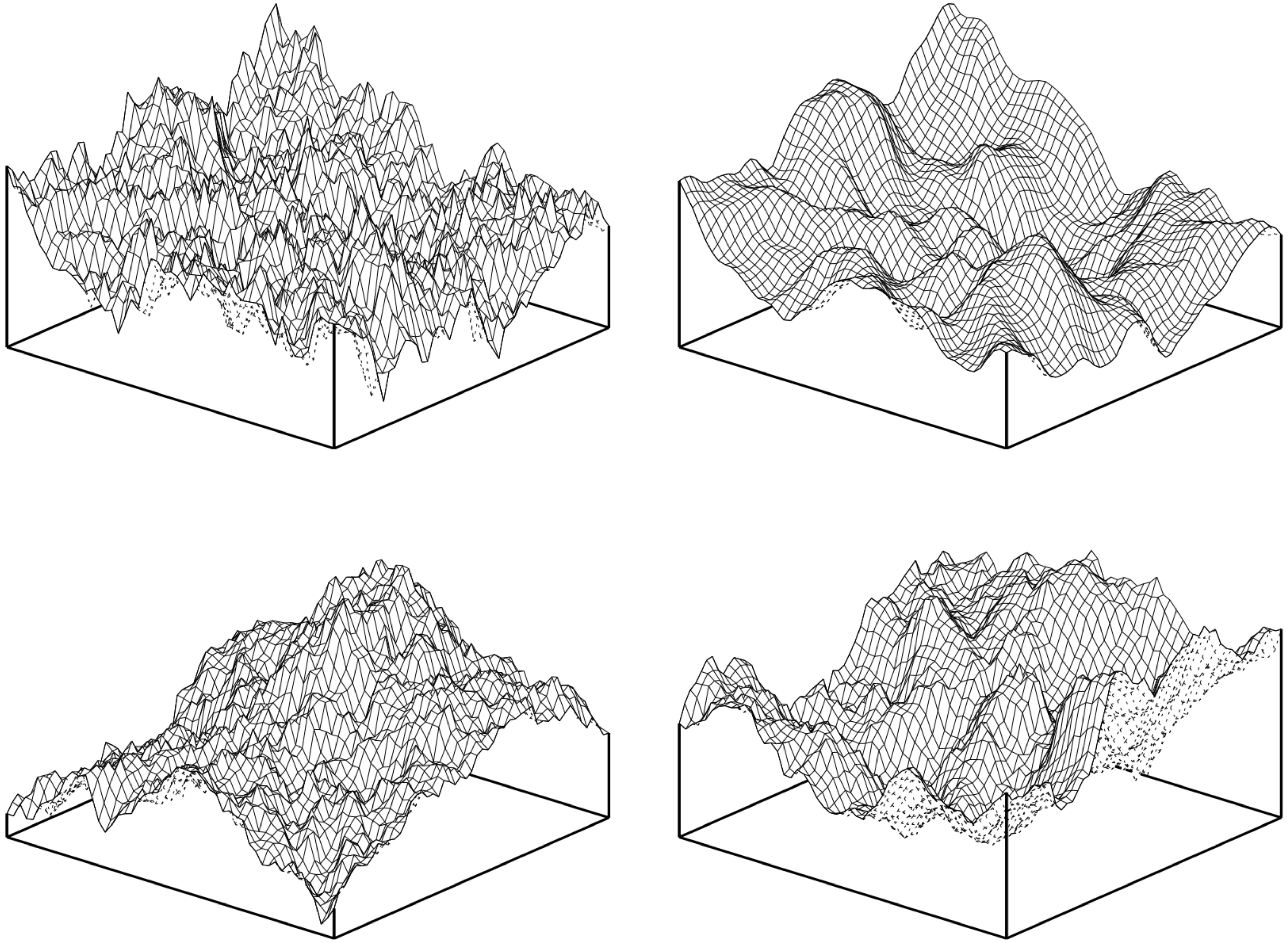
Forward Pass in the Infinite-Width Limit

- Consider NN with 1 hidden layer with H hidden units and tanh activation, 0 mean Gaussian initialization
- Can show that if we scale initialization by $1/H$, each $f_k(x)$ converges to $\mathcal{N}(0, \sigma_v)$ by CLT
- If we consider joint distribution of $f_k(x^{(1)}), f_k(x^{(2)}), \cdots, f_k(x^{(n)})$, turns out they converge to a multivariate Gaussian

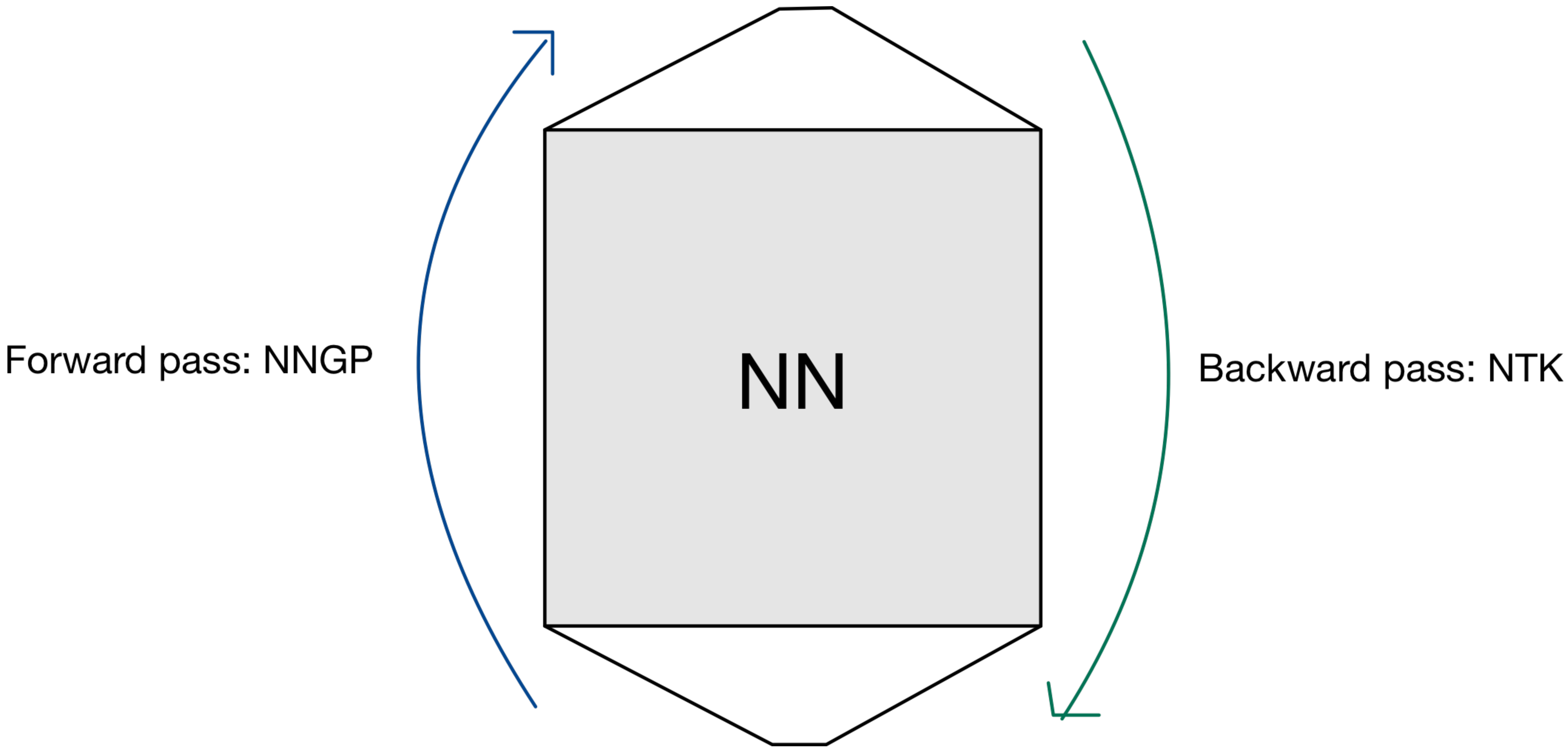

x
h
f
$h_j(x) = \tanh(\sum_{i=1}^{l} u_{ij} x_i)$
$f_k(x) = \sum_{j=1}^{H} v_{jk} h_j(x)$

Forward Pass in the Infinite-Width Limit

- Distributions over functions like this is a Gaussian process
- Implies that we can use perform exact Bayesian inference to "train on" infinite-width NNs



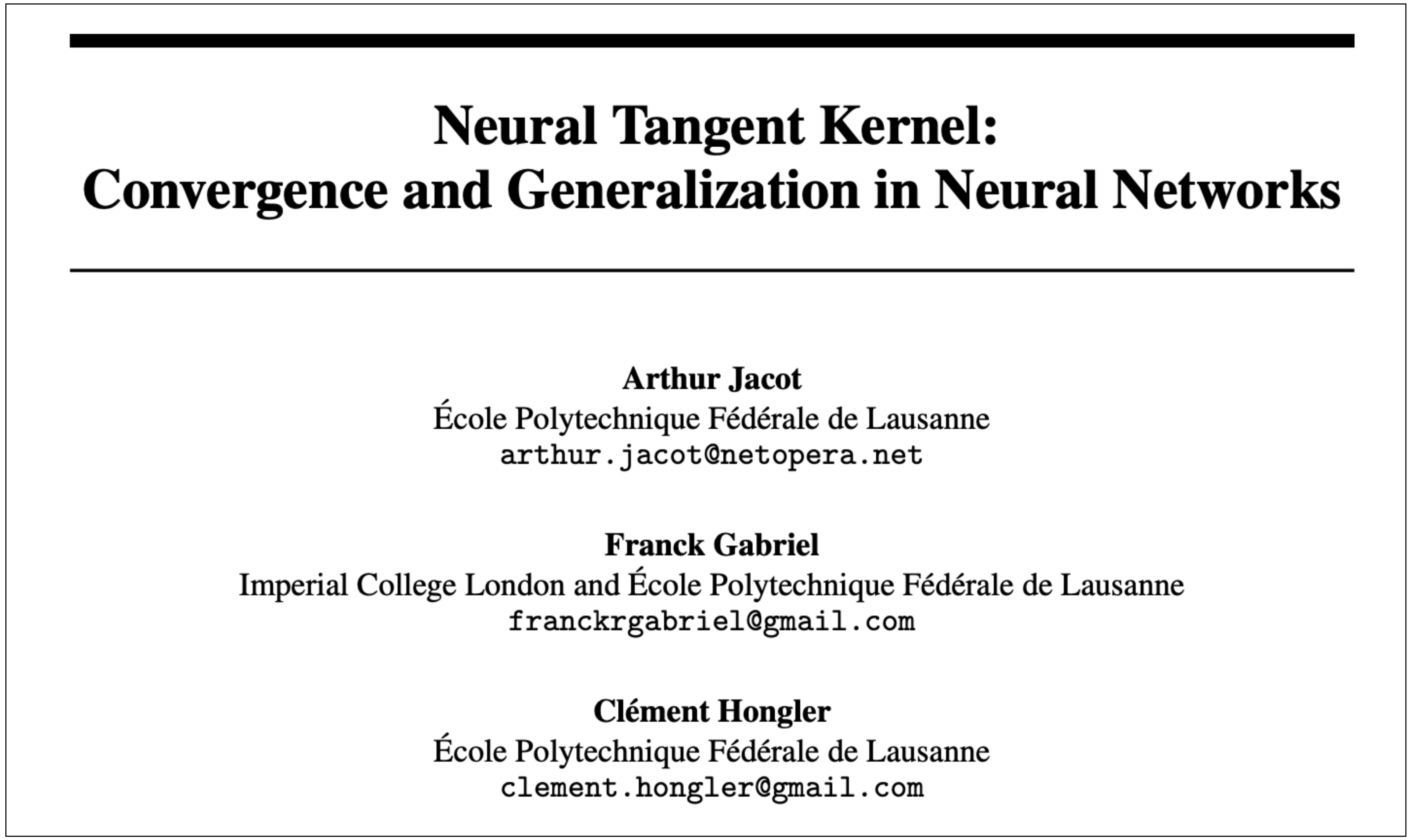
The Intellectual Predecessor to NTK



Part 2: Neural Tangent Kernel

Neural Tangent Kernel

- Showed that in the limit of large width, a NN optimized via gradient descent evolves like a linear model
- 2 steps in analysis:
  - Initialization: if a neural network is parametrized and initialized appropriately, then its NTK converges to a deterministic kernel Θ_∞
  - Training: as the network undergoes training, its NTK remains frozen in its initial step, and the network evolves like kernel gradient descent via Θ_∞



**Neural Tangent Kernel:
Convergence and Generalization in Neural Networks**

Arthur Jacot
École Polytechnique Fédérale de Lausanne
arthur.jacot@netopera.net

Franck Gabriel
Imperial College London and École Polytechnique Fédérale de Lausanne
franckrgabriel@gmail.com

Clément Hongler
École Polytechnique Fédérale de Lausanne
clement.hongler@gmail.com

NTK Initialization

Standard multi-layer MLP:

$$\alpha^{(0)}(x;\theta) = x$$

$$\tilde{\alpha}^{(\ell+1)}(x;\theta) = \frac{1}{\sqrt{n_\ell}} W^{(\ell)} \alpha^{(\ell)}(x;\theta) + \beta b^{(\ell)}$$

$$\alpha^{(\ell)}(x;\theta) = \sigma\left(\tilde{\alpha}^{(\ell)}(x;\theta)\right),$$

- All parameters initialized by $\mathcal{N}(0,1)$
- Note: they moved the $1/n_l$ factor outside, an analysis for this choice will be explained with μP later

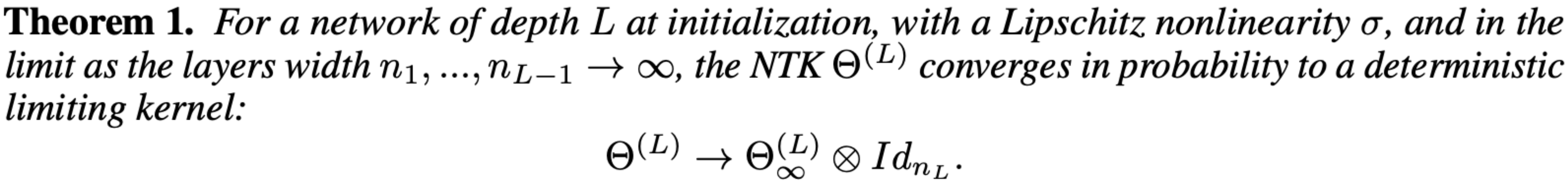
NTK Initialization

- NTK defined as inner product of Jacobian of network output wrt its parameters θ:

$$\Theta(x, x') = \left\langle \nabla_\theta f(x; \theta), \nabla_\theta f(x'; \theta) \right\rangle$$

- This is a kernel wrt x and x'
- What is a kernel?
  - $K(x, x')$ is a similarity measure between its inputs
  - Associated with a feature map ϕ: $K(x, x') = \langle \phi(x), \phi(x') \rangle$

NTK Initialization



Theorem 1. *For a network of depth L at initialization, with a Lipschitz nonlinearity σ, and in the limit as the layers width $n_1, ..., n_{L-1} \to \infty$, the NTK $\Theta^{(L)}$ converges in probability to a deterministic limiting kernel:*

$$\Theta^{(L)} \to \Theta_\infty^{(L)} \otimes Id_{n_L}.$$

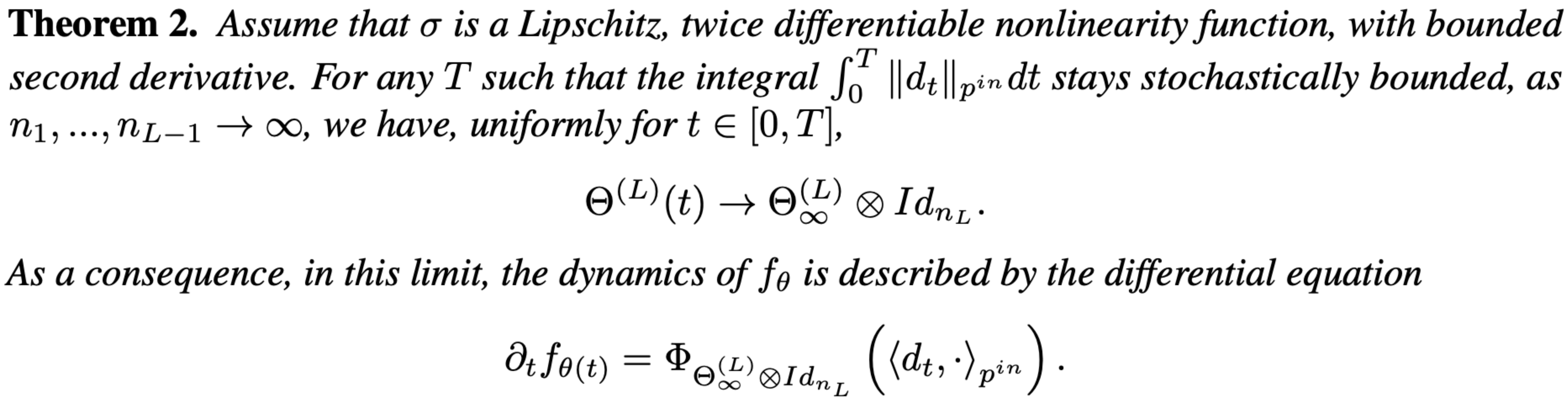
- Inner product between every pair of features $\nabla_\theta f(x;\theta), \nabla_\theta f(x';\theta)$ converges even though θ is random!
- “Strange” condition: requires taking $n_1, \cdots, n_L \to \infty$ in that sequence, nobody does that

NTK Training

- We can perform a first-order Taylor approximation of the neural network around some base point θ_0:
$$f(x;\theta) - f\left(x;\theta_0\right) \approx \left\langle \nabla_\theta f\left(x;\theta_0\right), \theta - \theta_0 \right\rangle$$
- Good approximation as long as θ close to θ_0
- Prima facie this seems useless - feels like f can't change if θ doesn't change much. But in high dimensions this is actually possible
- And in fact they showed this is an accurate description of training dynamics in the infinite width limit
- So like doing linear regression against input featurizer, model evolves linearly

NTK Training

- Previous attempts to analyze optimization trajectories quickly became hopelessly complicated precisely because of all the changing quantities
- But they showed that the limiting NTK stays constant over time:



Theorem 2. *Assume that σ is a Lipschitz, twice differentiable nonlinearity function, with bounded second derivative. For any T such that the integral $\int_0^T \|d_t\|_{p^{in}} dt$ stays stochastically bounded, as $n_1, ..., n_{L-1} \to \infty$, we have, uniformly for $t \in [0, T]$,*

$$\Theta^{(L)}(t) \to \Theta_\infty^{(L)} \otimes Id_{n_L}.$$

As a consequence, in this limit, the dynamics of f_θ is described by the differential equation

$$\partial_t f_{\theta(t)} = \Phi_{\Theta_\infty^{(L)} \otimes Id_{n_L}} \left(\langle d_t, \cdot \rangle_{p^{in}} \right).$$

NTK Training

- Then it can be shown that the network under gradient flow follows the following ODE: $\partial_t f_t = -\eta\Theta_\infty \cdot \nabla_f L(f_t)$
- This becomes just kernel gradient descent with fixed kernel Θ_∞
    - Convex problem
    - PD kernel converges to global min
- If L is least-squares loss, we can solve this explicitly for all t: $f_t - f^\star = e^{-\eta t \Theta_\infty}(f_0 - f^\star)$

Generalization Properties

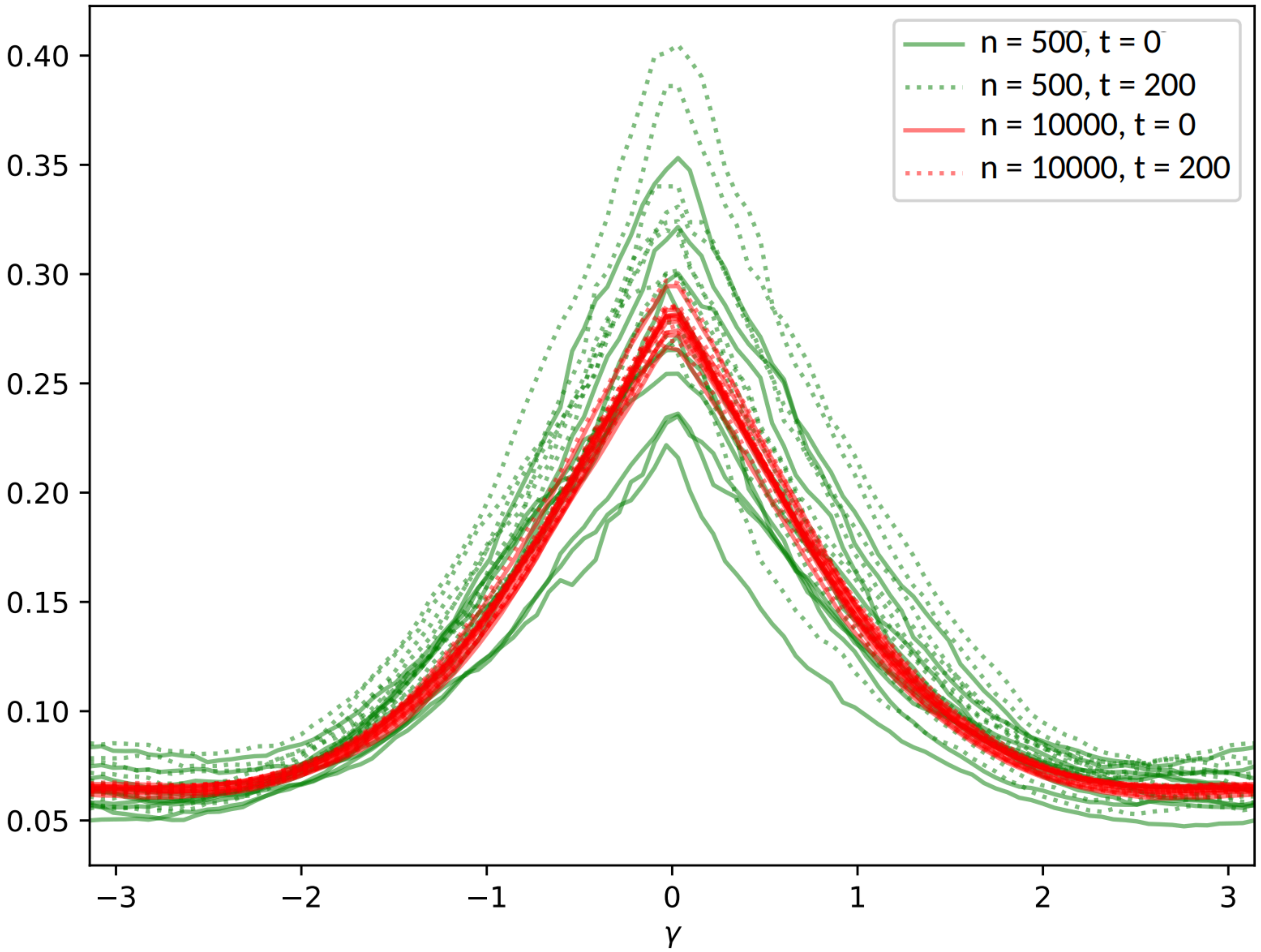
- Kernel gradient descent has implicit bias towards minimizing the RKHS norm
- Results in choosing simpler solutions that generalizes better

Summary

- When the widths of a network tend to infinity with appropriately initialized weights, its NTK converges deterministically to a limiting kernel
- During training, this kernel remains constant and follows kernel gradient descent with a convex trajectory
- So all we need to understand gradient descent is its NTK
- Solves age-old question of why NNs can be trained to do so well even though landscape is non-convex

“Results” 🤔

- Some toy experiments to give empirical evidence NTK converges to its limiting kernel as width $n \to \infty$
- Plot: $\Theta(x_0, x)$ for fixed $x_0 = (1{,}0), x = (\cos(\gamma), \sin(\gamma))$ to visualize values of kernel



Limitations

- No feature learning: in the NTK limit, layer features learned during training essentially same as that from random initialization, aka “lazy training”
- But we know empirically (i.e Imagenet, BERT) that feature learning is important in DL

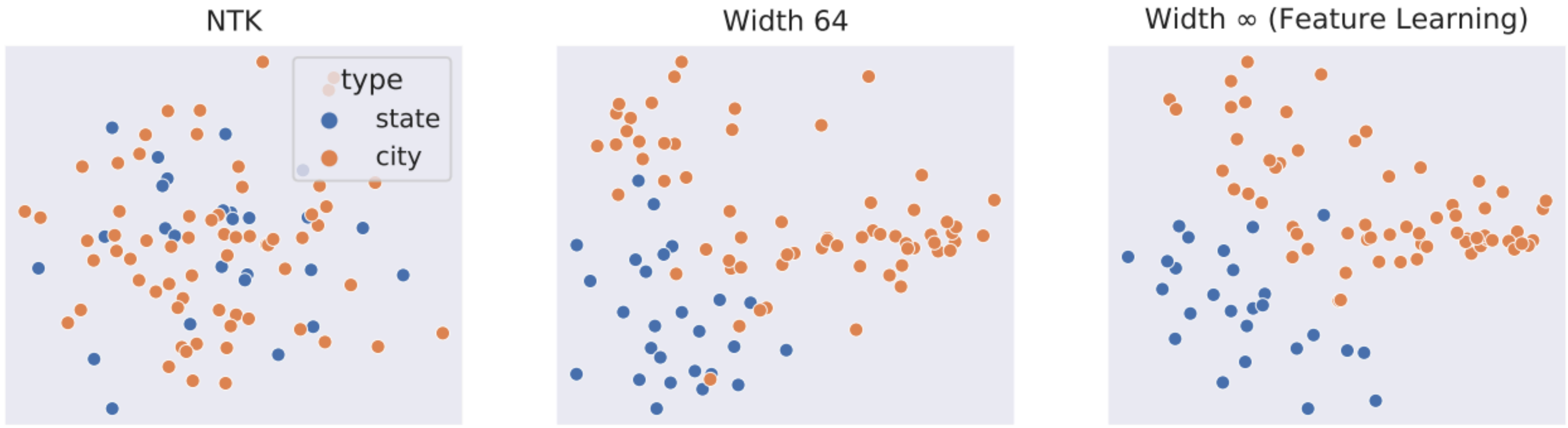


Figure 1: PCA of Word2Vec embeddings of top US cities and states, for NTK, width-64, and width-∞ feature learning networks (Definition 5.1). NTK embeddings are essentially random, while cities and states get naturally separated in embedding space as width increases in the feature learning regime.

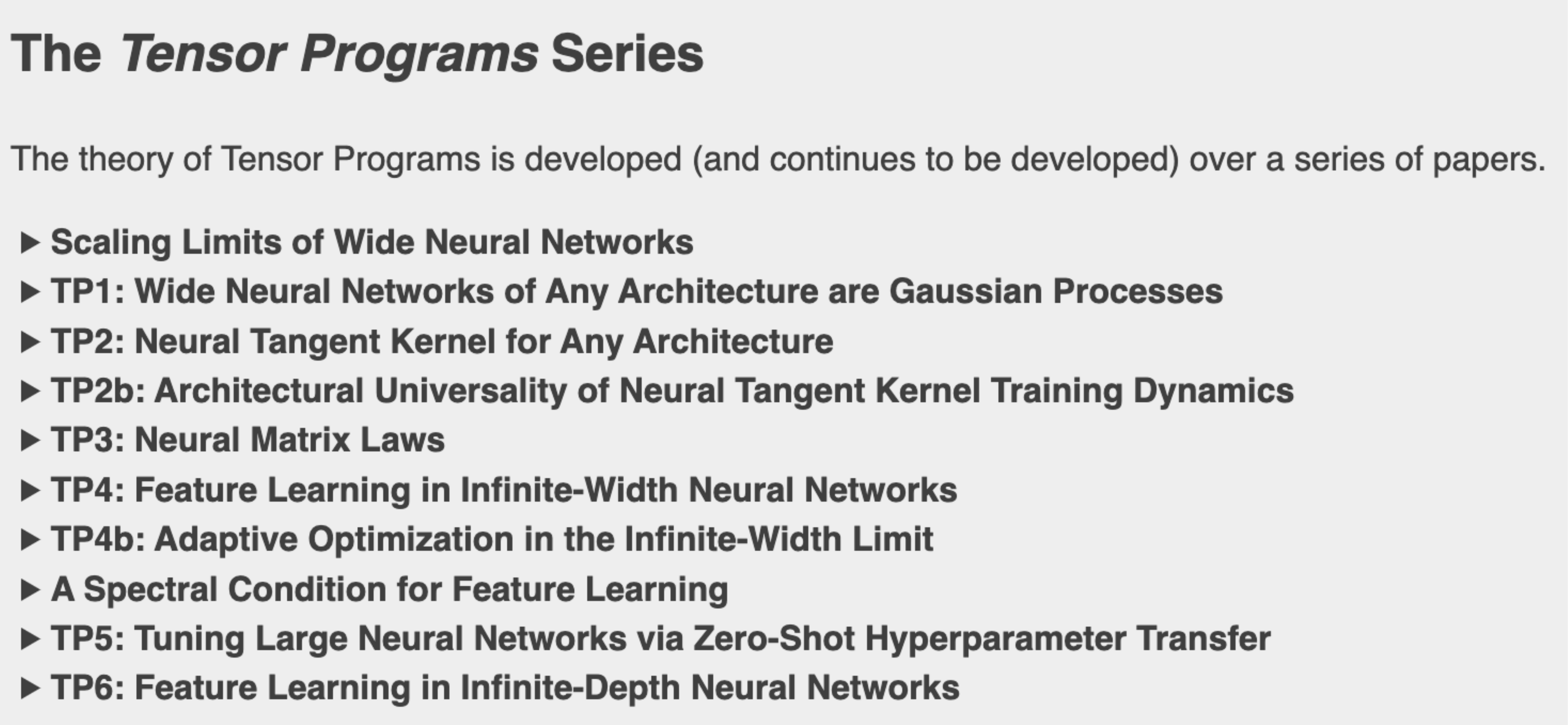
Extending NTK

- Subsequent flurry of work to try to extend NTK to other architectures:
- [Arora, Du et al. ’19] CNNs (CNTK)
- [Du, Hou et al. ’19] Graph Neural Tangent Kernel
- [Alemohammad, Wang et al. ’20] RNNs: Recurrent NTK
- [Hron, Bahri et al. ’20] Attention: Infinite Attention

Part 3: Tensor Programs

Tensor Programs

- To derive a unified Theory of Everything for deep learning



The *Tensor Programs* Series

The theory of Tensor Programs is developed (and continues to be developed) over a series of papers.

▶ **Scaling Limits of Wide Neural Networks**

▶ **TP1: Wide Neural Networks of Any Architecture are Gaussian Processes**

▶ **TP2: Neural Tangent Kernel for Any Architecture**

▶ **TP2b: Architectural Universality of Neural Tangent Kernel Training Dynamics**

▶ **TP3: Neural Matrix Laws**

▶ **TP4: Feature Learning in Infinite-Width Neural Networks**

▶ **TP4b: Adaptive Optimization in the Infinite-Width Limit**

▶ **A Spectral Condition for Feature Learning**

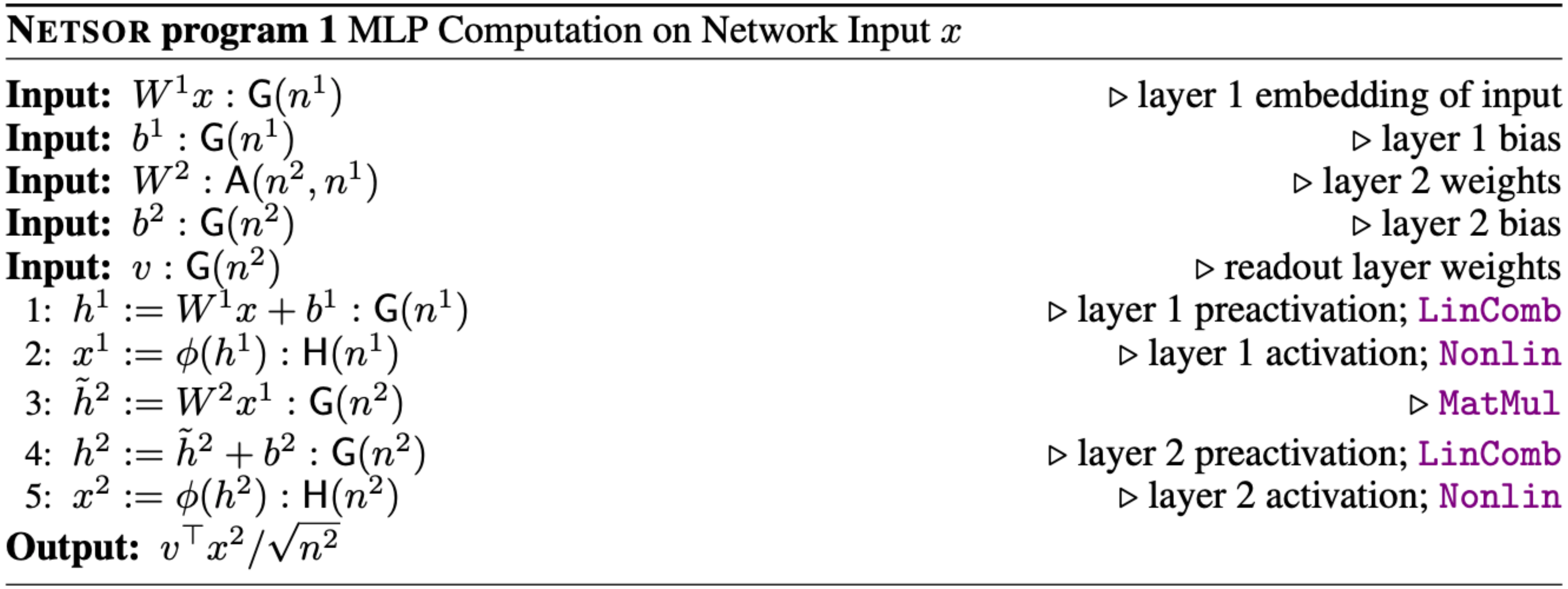
▶ **TP5: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer**

▶ **TP6: Feature Learning in Infinite-Depth Neural Networks**

Greg Yang, behind the Tensor Programs series and currently working on Grok

Tensor Programs

- Observation: (almost) all NNs are defined in terms of:
  - Matrix multiplication (MatMul)
  - Taking a linear combination (LinComb)
  - Non-linearity (NonLin)
- For different architectures, we just have to translate them to a Tensor Program and mechanically turn the crank to track correlations btw vectors & apply the Master Theorem

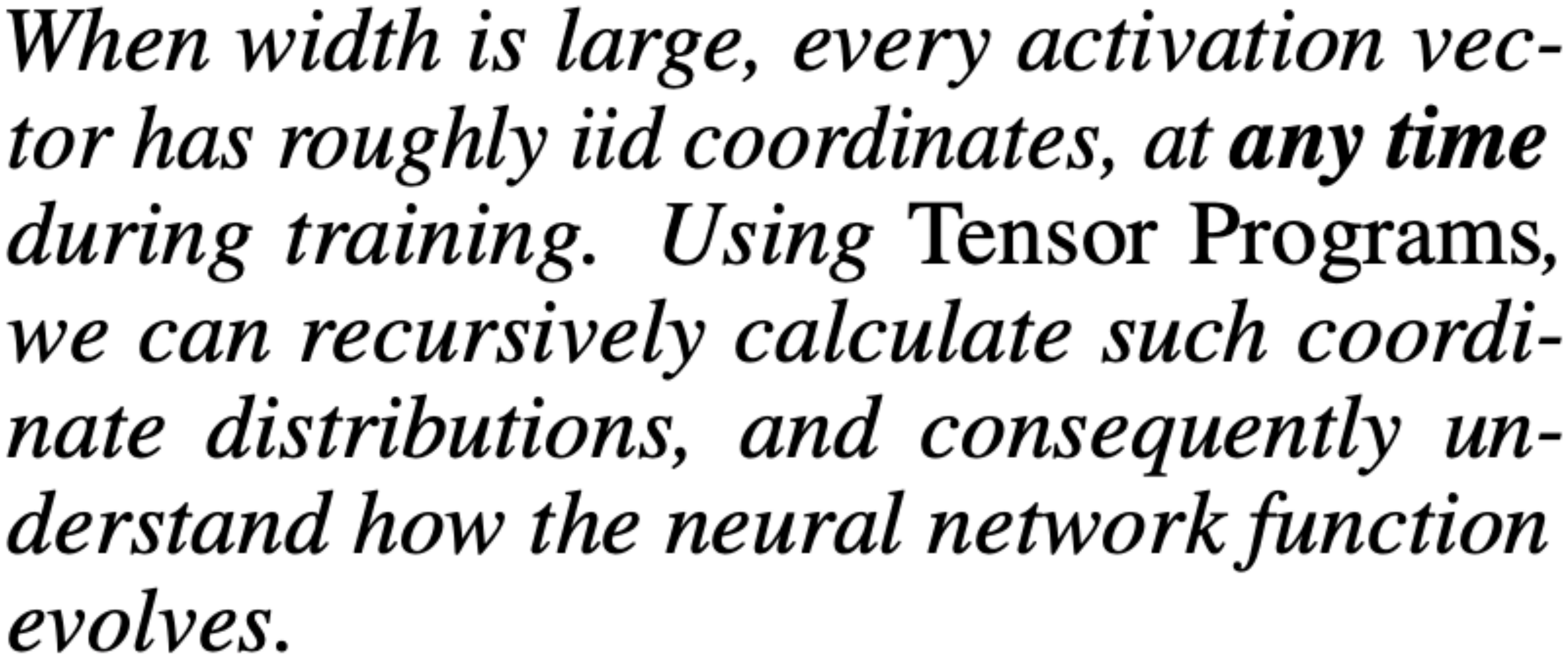


NETSOR program 1 MLP Computation on Network Input x

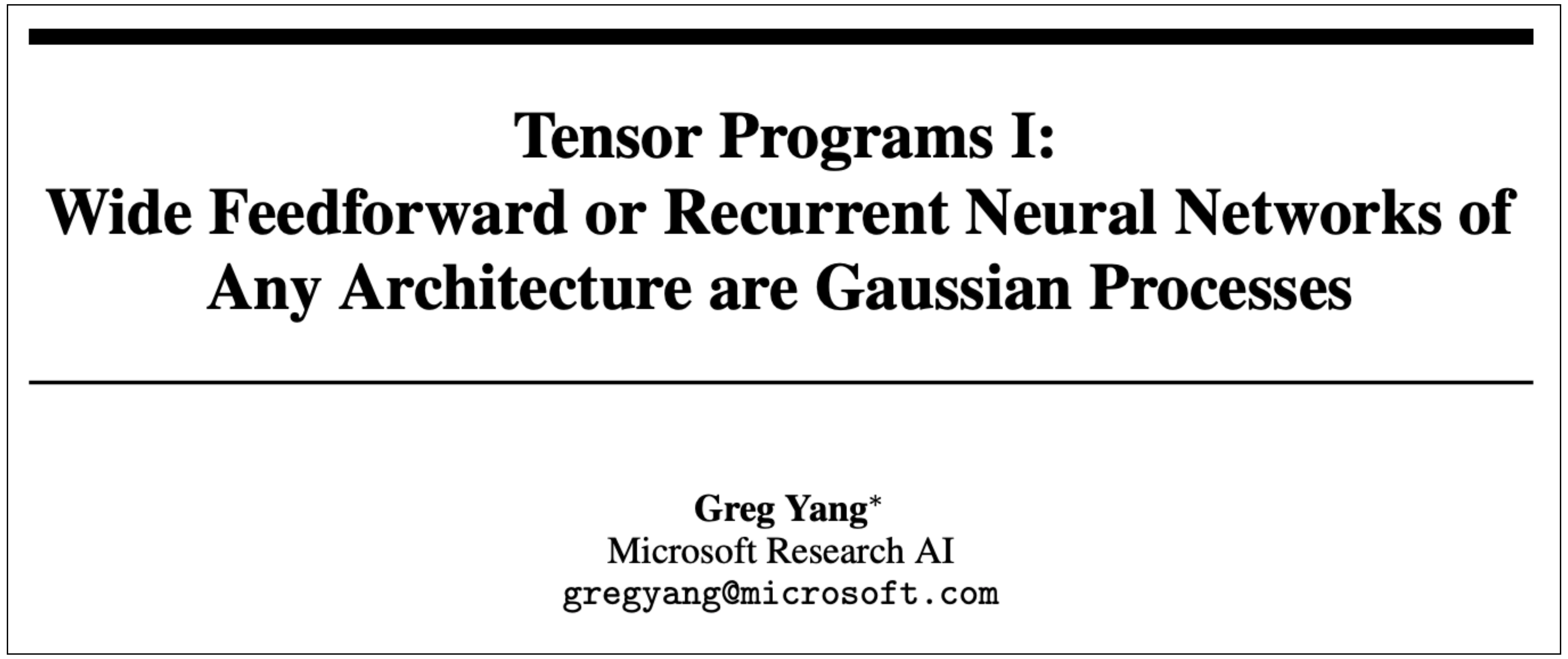
```
Input: $W^1 x : \mathsf{G}(n^1)$    ▷ layer 1 embedding of input
Input: $b^1 : \mathsf{G}(n^1)$    ▷ layer 1 bias
Input: $W^2 : \mathsf{A}(n^2, n^1)$    ▷ layer 2 weights
Input: $b^2 : \mathsf{G}(n^2)$    ▷ layer 2 bias
Input: $v : \mathsf{G}(n^2)$    ▷ readout layer weights
$h^1 := W^1 x + b^1 : \mathsf{G}(n^1)$    ▷ layer 1 preactivation; LinComb
$x^1 := \phi(h^1) : \mathsf{H}(n^1)$    ▷ layer 1 activation; Nonlin
$\tilde{h}^2 := W^2 x^1 : \mathsf{G}(n^2)$    ▷ MatMul
$h^2 := \tilde{h}^2 + b^2 : \mathsf{G}(n^2)$    ▷ layer 2 preactivation; LinComb
$x^2 := \phi(h^2) : \mathsf{H}(n^2)$    ▷ layer 2 activation; Nonlin
Output: $v^\top x^2 / \sqrt{n^2}$
```

Tensor Programs

- Why does it work?

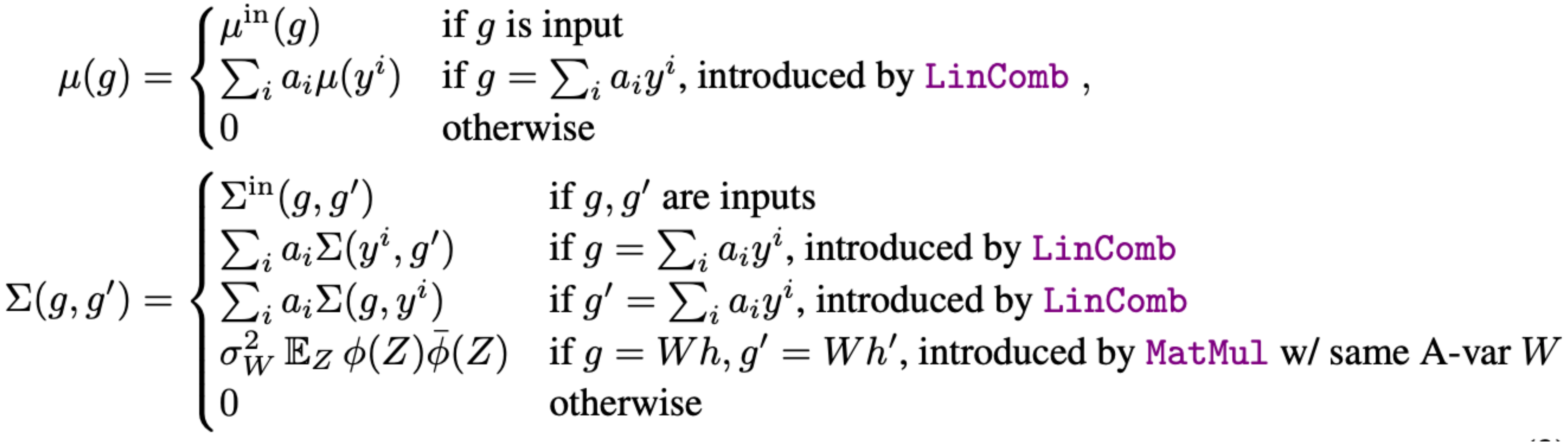


When width is large, every activation vector has roughly iid coordinates, at ***any time*** *during training. Using* Tensor Programs, *we can recursively calculate such coordinate distributions, and consequently understand how the neural network function evolves.*



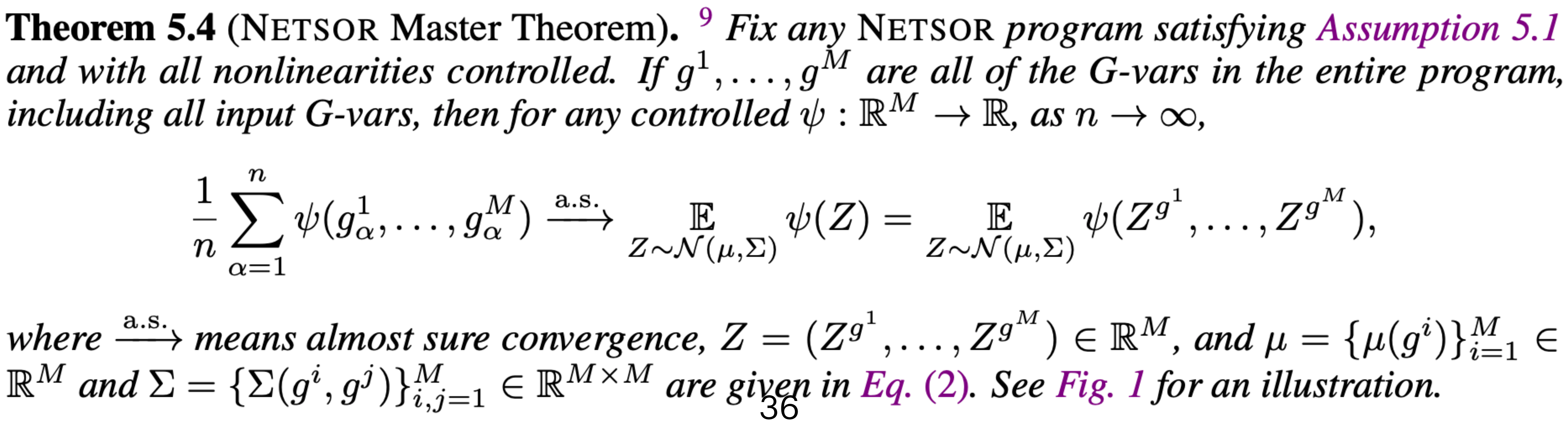
Tensor Programs I: Wide Feedforward or Recurrent Neural Networks of Any Architecture are Gaussian Processes

Greg Yang*
Microsoft Research AI
gregyang@microsoft.com



$$\mu(g) = \begin{cases} \mu^{\text{in}}(g) & \text{if } g \text{ is input} \\ \sum_i a_i \mu(y^i) & \text{if } g = \sum_i a_i y^i, \text{ introduced by } \texttt{LinComb}, \\ 0 & \text{otherwise} \end{cases}$$

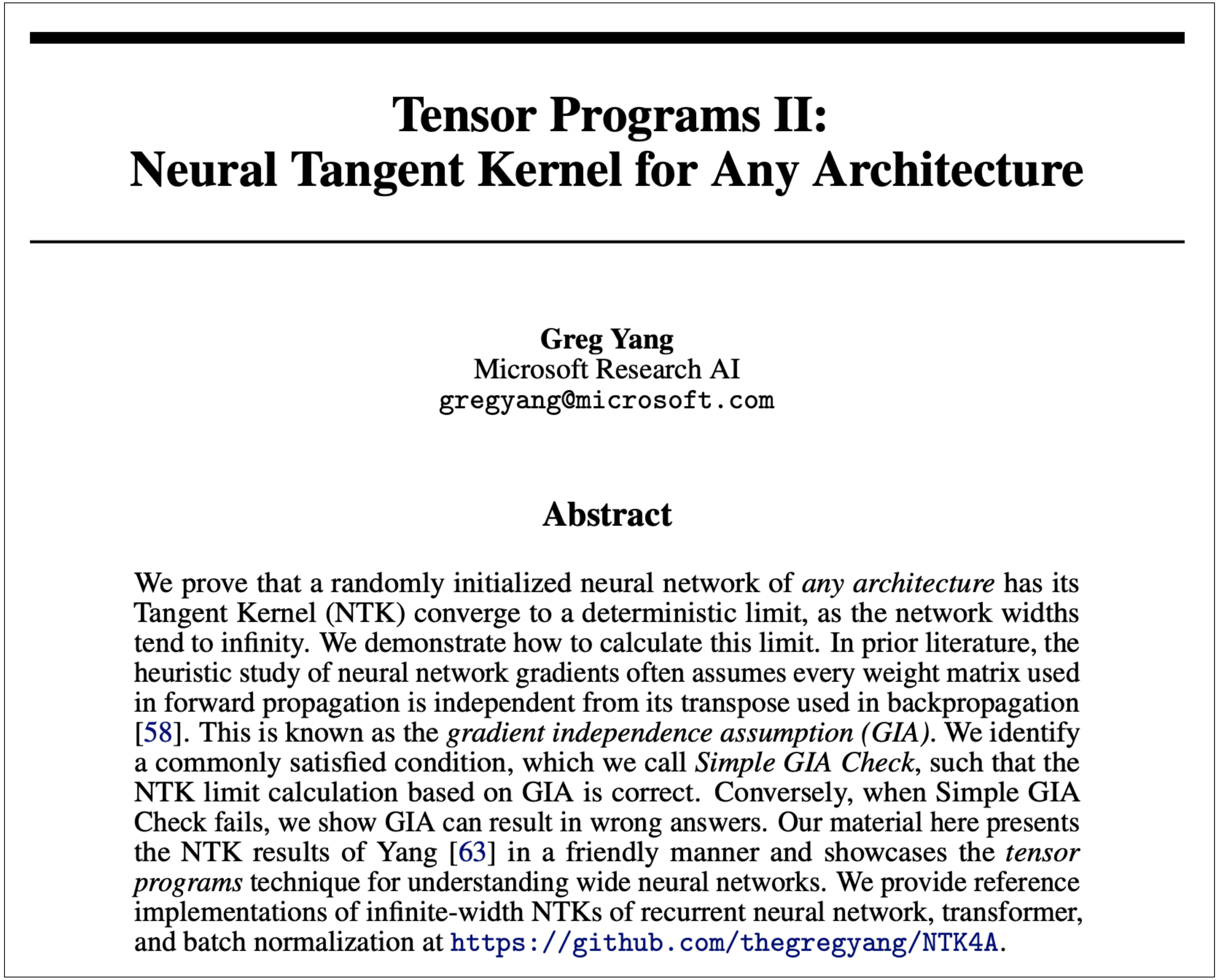
$$\Sigma(g, g') = \begin{cases} \Sigma^{\text{in}}(g, g') & \text{if } g, g' \text{ are inputs} \\ \sum_i a_i \Sigma(y^i, g') & \text{if } g = \sum_i a_i y^i, \text{ introduced by } \texttt{LinComb} \\ \sum_i a_i \Sigma(g, y^i) & \text{if } g' = \sum_i a_i y^i, \text{ introduced by } \texttt{LinComb} \\ \sigma_W^2 \mathbb{E}_Z \phi(Z) \bar{\phi}(Z) & \text{if } g = Wh, g' = Wh', \text{ introduced by } \texttt{MatMul} \text{ w/ same A-var } W \\ 0 & \text{otherwise} \end{cases}$$



Theorem 5.4 (NETSOR Master Theorem). [9] *Fix any* NETSOR *program satisfying Assumption 5.1 and with all nonlinearities controlled. If $g^1, \ldots, g^M$ are all of the G-vars in the entire program, including all input G-vars, then for any controlled $\psi : \mathbb{R}^M \to \mathbb{R}$, as $n \to \infty$,*

$$\frac{1}{n} \sum_{\alpha=1}^{n} \psi(g_\alpha^1, \ldots, g_\alpha^M) \xrightarrow{\text{a.s.}} \mathop{\mathbb{E}}_{Z \sim \mathcal{N}(\mu, \Sigma)} \psi(Z) = \mathop{\mathbb{E}}_{Z \sim \mathcal{N}(\mu, \Sigma)} \psi(Z^{g^1}, \ldots, Z^{g^M}),$$

where $\xrightarrow{\text{a.s.}}$ means almost sure convergence, $Z = (Z^{g^1}, \ldots, Z^{g^M}) \in \mathbb{R}^M$, and $\mu = \{\mu(g^i)\}_{i=1}^M \in \mathbb{R}^M$ and $\Sigma = \{\Sigma(g^i, g^j)\}_{i,j=1}^M \in \mathbb{R}^{M \times M}$ are given in Eq. (2). See Fig. 1 for an illustration.



Tensor Programs II: Neural Tangent Kernel for Any Architecture

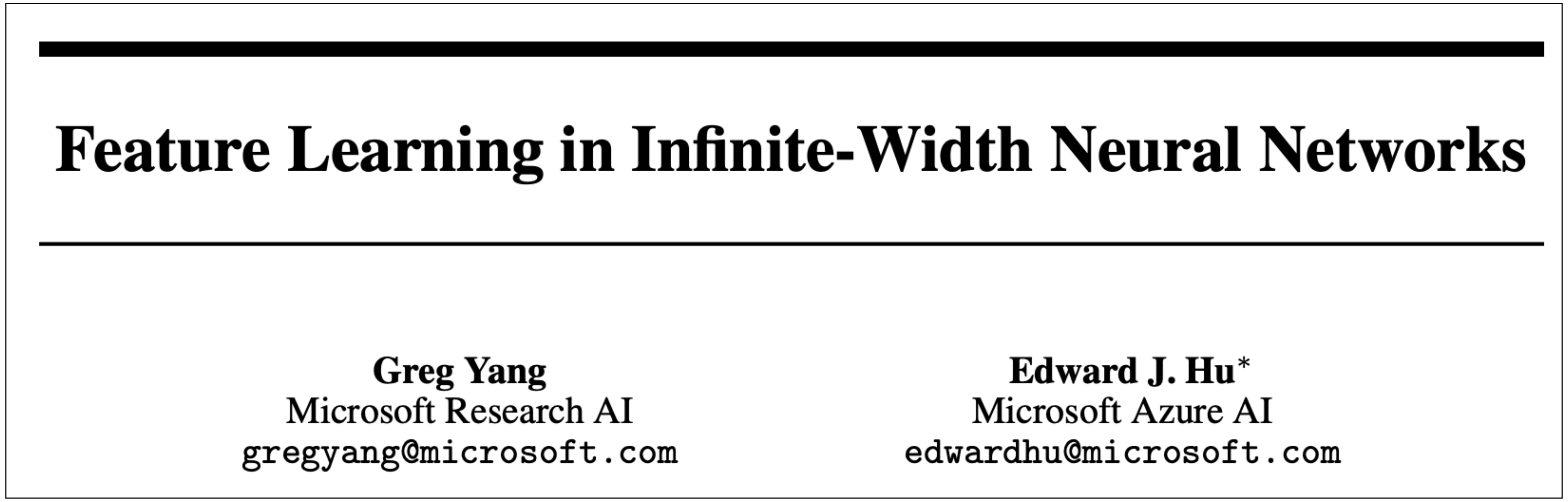
Greg Yang
Microsoft Research AI
gregyang@microsoft.com

Abstract

We prove that a randomly initialized neural network of *any architecture* has its Tangent Kernel (NTK) converge to a deterministic limit, as the network widths tend to infinity. We demonstrate how to calculate this limit. In prior literature, the heuristic study of neural network gradients often assumes every weight matrix used in forward propagation is independent from its transpose used in backpropagation [58]. This is known as the *gradient independence assumption (GIA)*. We identify a commonly satisfied condition, which we call *Simple GIA Check*, such that the NTK limit calculation based on GIA is correct. Conversely, when Simple GIA Check fails, we show GIA can result in wrong answers. Our material here presents the NTK results of Yang [63] in a friendly manner and showcases the *tensor programs* technique for understanding wide neural networks. We provide reference implementations of infinite-width NTKs of recurrent neural network, transformer, and batch normalization at `https://github.com/thegregyang/NTK4A`.

Tensor Programs IV

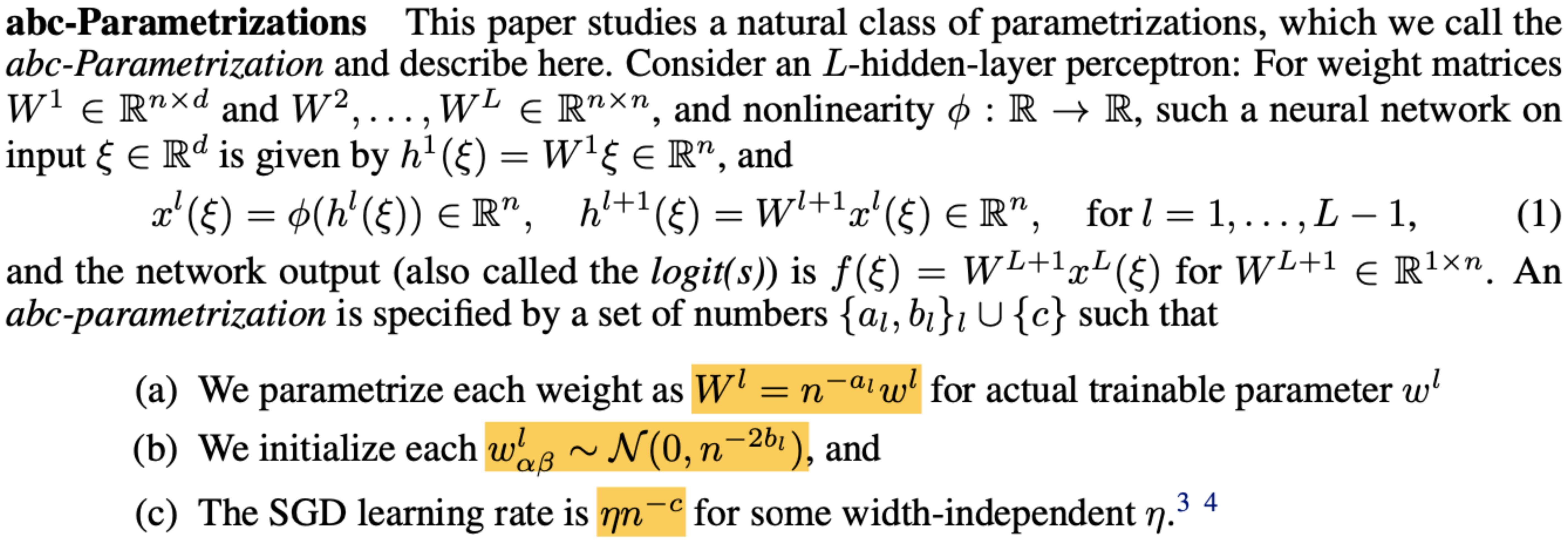
- Common parametrizations for neural networks can be defined in terms of:
  - (a) Scalings of weights
  - (b) Scaling initialization variance
  - (c) Scaling learning rate



Feature Learning in Infinite-Width Neural Networks

Greg Yang
Microsoft Research AI
gregyang@microsoft.com

Edward J. Hu*
Microsoft Azure AI
edwardhu@microsoft.com



abc-Parametrizations This paper studies a natural class of parametrizations, which we call the *abc-Parametrization* and describe here. Consider an L-hidden-layer perceptron: For weight matrices $W^1 \in \mathbb{R}^{n\times d}$ and $W^2, \ldots, W^L \in \mathbb{R}^{n\times n}$, and nonlinearity $\phi : \mathbb{R} \to \mathbb{R}$, such a neural network on input $\xi \in \mathbb{R}^d$ is given by $h^1(\xi) = W^1\xi \in \mathbb{R}^n$, and

$$x^l(\xi) = \phi(h^l(\xi)) \in \mathbb{R}^n, \quad h^{l+1}(\xi) = W^{l+1}x^l(\xi) \in \mathbb{R}^n, \quad \text{for } l = 1, \ldots, L-1, \tag{1}$$

and the network output (also called the *logit(s)*) is $f(\xi) = W^{L+1}x^L(\xi)$ for $W^{L+1} \in \mathbb{R}^{1\times n}$. An *abc-parametrization* is specified by a set of numbers $\{a_l, b_l\}_l \cup \{c\}$ such that

(a) We parametrize each weight as $W^l = n^{-a_l}w^l$ for actual trainable parameter w^l

(b) We initialize each $w^l_{\alpha\beta} \sim \mathcal{N}(0, n^{-2b_l})$, and

(c) The SGD learning rate is ηn^{-c} for some width-independent η.[3] [4]

Intuition

- Scaling weights a_l vs initialization b_l is symmetric: we can proportionately increase one while decreasing the other to preserve the same functional behavior
  - I.e $1/\sqrt{n}W, W \sim \mathcal{N}(0,1)$ functionally same as $W, W \sim \mathcal{N}(0,1/n)$
- By controlling between a_l and b_l, we can influence the per-layer learning rate as the gradients for layer l get scaled by a_l

Tensor Programs IV

- Stable: initial activations/ preactivations, changes in features, changes logits are all O(1)
- Feature learning: feature change has $\Omega(1)$ coordinates

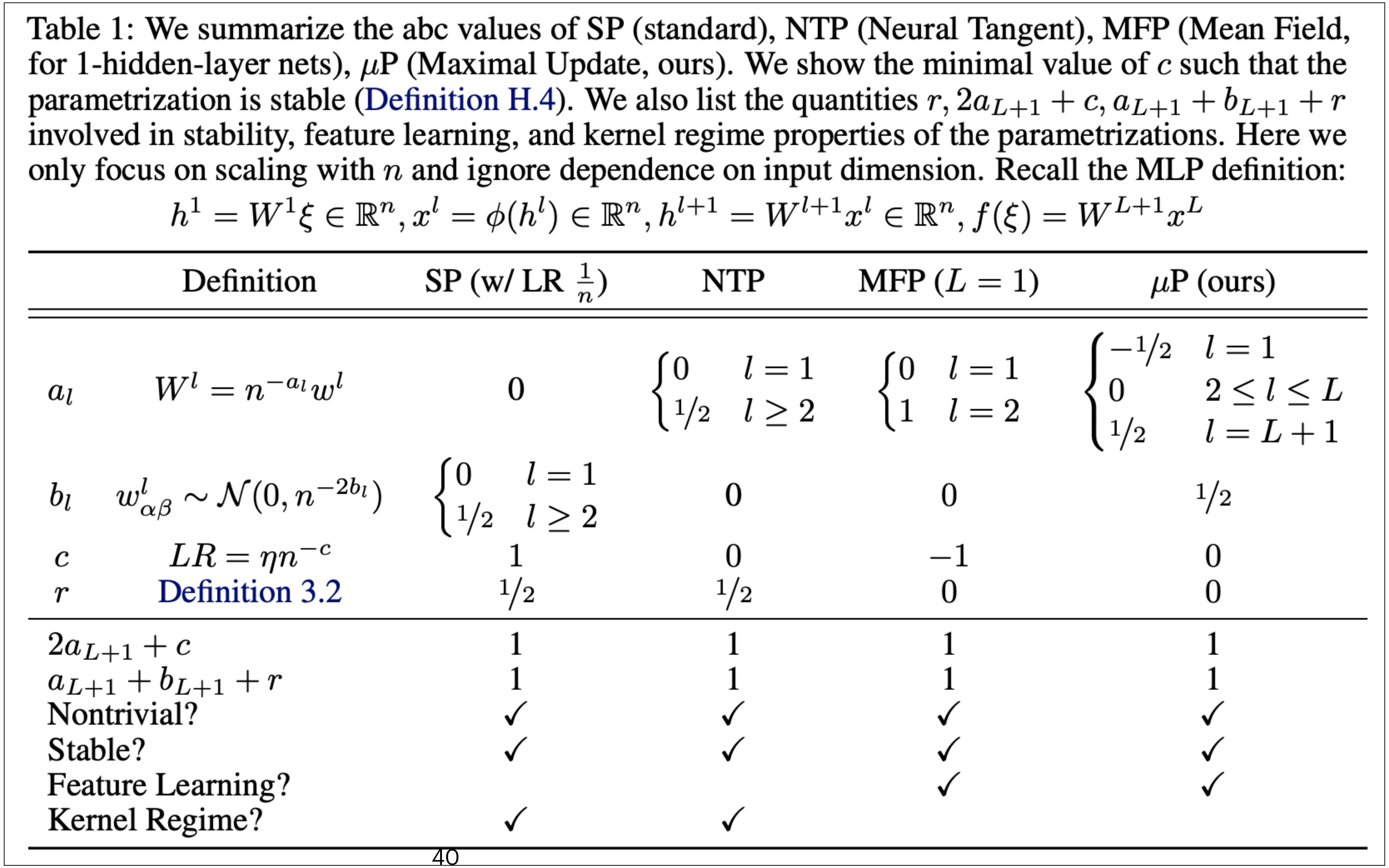


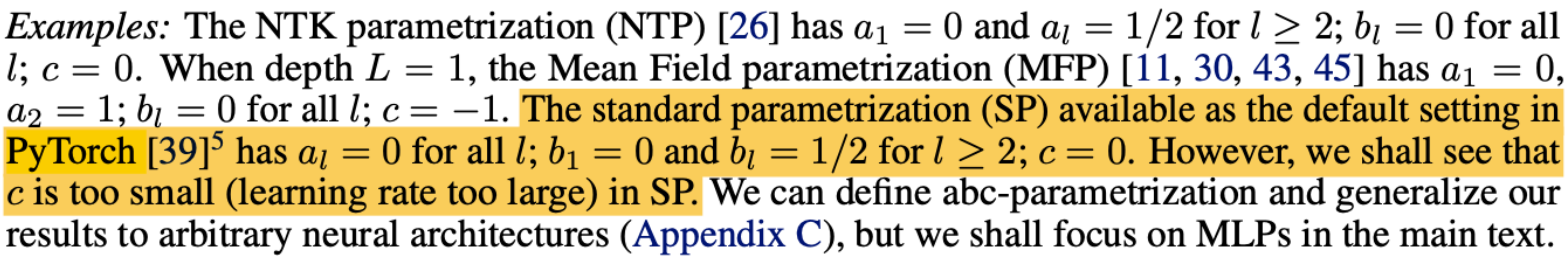
Table 1: We summarize the abc values of SP (standard), NTP (Neural Tangent), MFP (Mean Field, for 1-hidden-layer nets), μP (Maximal Update, ours). We show the minimal value of c such that the parametrization is stable (Definition H.4). We also list the quantities r, $2a_{L+1}+c$, $a_{L+1}+b_{L+1}+r$ involved in stability, feature learning, and kernel regime properties of the parametrizations. Here we only focus on scaling with n and ignore dependence on input dimension. Recall the MLP definition:

$$h^1 = W^1\xi \in \mathbb{R}^n, x^l = \phi(h^l) \in \mathbb{R}^n, h^{l+1} = W^{l+1}x^l \in \mathbb{R}^n, f(\xi) = W^{L+1}x^L$$

	Definition	SP (w/ LR $\frac{1}{n}$)	NTP	MFP ($L=1$)	μP (ours)
a_l	$W^l = n^{-a_l}w^l$	0	$\begin{cases} 0 & l=1 \\ 1/2 & l \geq 2 \end{cases}$	$\begin{cases} 0 & l=1 \\ 1 & l=2 \end{cases}$	$\begin{cases} -1/2 & l=1 \\ 0 & 2 \leq l \leq L \\ 1/2 & l=L+1 \end{cases}$
b_l	$w^l_{\alpha\beta} \sim \mathcal{N}(0, n^{-2b_l})$	$\begin{cases} 0 & l=1 \\ 1/2 & l \geq 2 \end{cases}$	0	0	$1/2$
c	$LR = \eta n^{-c}$	1	0	-1	0
r	Definition 3.2	$1/2$	$1/2$	0	0
$2a_{L+1}+c$		1	1	1	1
$a_{L+1}+b_{L+1}+r$		1	1	1	1
Nontrivial?		✓	✓	✓	✓
Stable?		✓	✓	✓	✓
Feature Learning?				✓	✓
Kernel Regime?		✓	✓		

Tensor Programs IV

- Note: they modified SP to have LR 1/n to avoid blowup, this forces it into uninteresting kernel regime

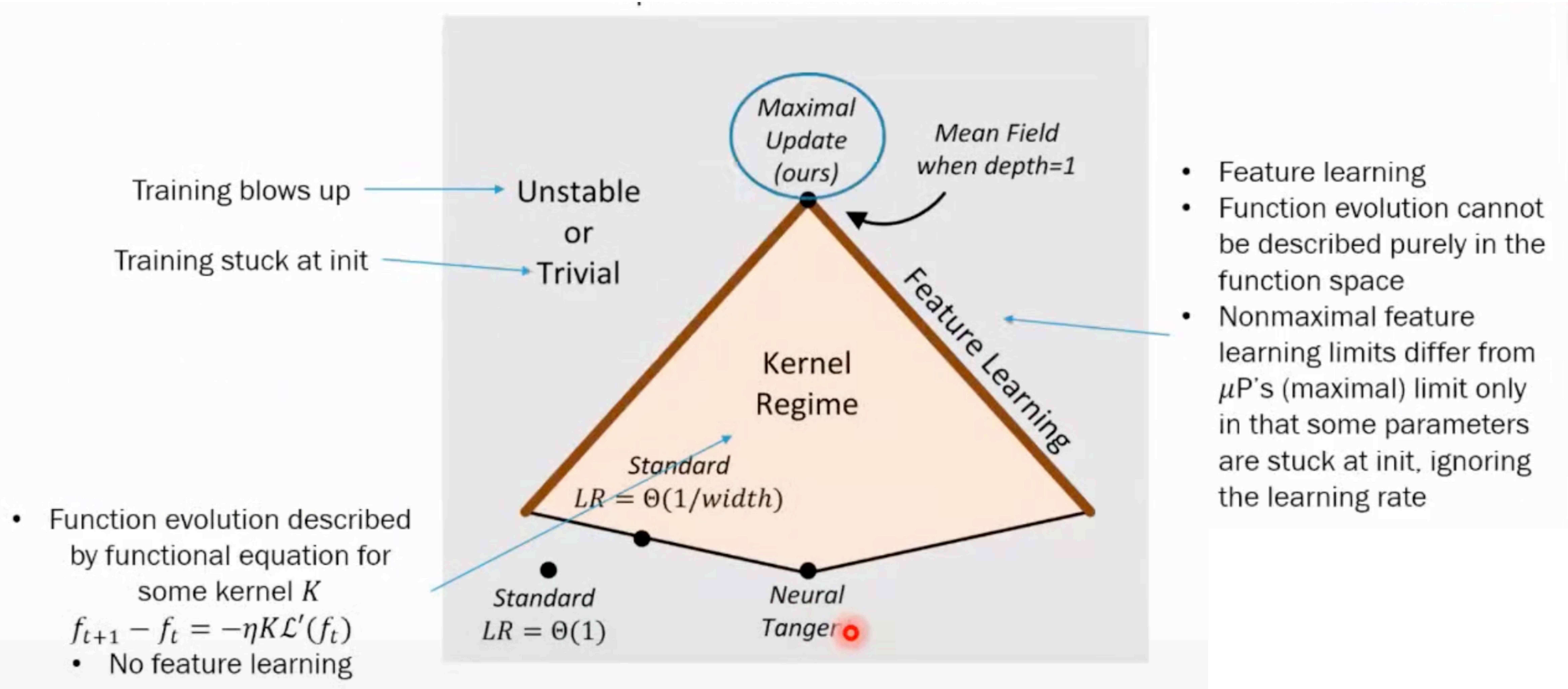


Examples: The NTK parametrization (NTP) [26] has $a_1 = 0$ and $a_l = 1/2$ for $l \geq 2$; $b_l = 0$ for all l; $c = 0$. When depth $L = 1$, the Mean Field parametrization (MFP) [11, 30, 43, 45] has $a_1 = 0$, $a_2 = 1$; $b_l = 0$ for all l; $c = -1$. The standard parametrization (SP) available as the default setting in PyTorch [39][5] has $a_l = 0$ for all l; $b_1 = 0$ and $b_l = 1/2$ for $l \geq 2$; $c = 0$. However, we shall see that c is too small (learning rate too large) in SP. We can define abc-parametrization and generalize our results to arbitrary neural architectures (Appendix C), but we shall focus on MLPs in the main text.

Comparison for 3 layer NN

- Standard parametrization:
  - $f(x) = W^{3T}\phi\left(W^{2T}\phi\left(W^{1T}x\right)\right)$
  - $W^1_{ij} \sim \mathcal{N}(0,1), \quad W^2_{ij}, W^3_{ij} \sim \mathcal{N}(0,1/n)$
- NTK Parametrization:
  - $f(x) = \frac{1}{\sqrt{n}}W^{3T}\phi\left(\frac{1}{\sqrt{n}}W^{2T}\phi\left(W^{1T}x\right)\right)$
  - $W^1_{ij}, W^2_{ij}, W^3_{ij} \sim \mathcal{N}(0,1)$
- μP:
  - $f(x) = \frac{1}{\sqrt{n}}W^{3T}\phi\left(W^{2T}\phi\left(\frac{1}{\sqrt{n}}W^{1T}x\right)\right)$
  - $W^1_{ij}, W^2_{ij}, W^3_{ij} \sim \mathcal{N}(0,1/n)$
- All uses LR = η

Tensor Programs IV



- Feature learning
- Function evolution cannot be described purely in the function space
- Nonmaximal feature learning limits differ from μP's (maximal) limit only in that some parameters are stuck at init, ignoring the learning rate

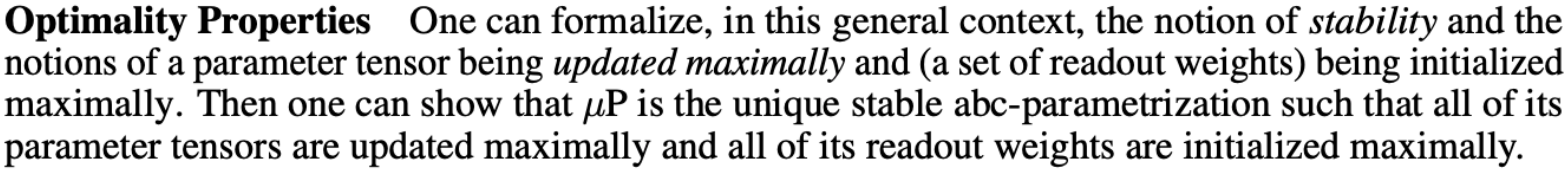
- Function evolution described by functional equation for some kernel K

$$f_{t+1} - f_t = -\eta K \mathcal{L}'(f_t)$$

- No feature learning

Tensor Programs IV

- Their Maximal Update Parametrization (μP) allows for “maximal” updates in the sense of $\Theta(1)$ changes in each coordinate during training while being stable
- This is unique:



Optimality Properties One can formalize, in this general context, the notion of *stability* and the notions of a parameter tensor being *updated maximally* and (a set of readout weights) being initialized maximally. Then one can show that μP is the unique stable abc-parametrization such that all of its parameter tensors are updated maximally and all of its readout weights are initialized maximally.

- This gives rise to feature learning

Experiments

- As predicted by theory, optimal LR with μP doesn't change much across width
- In SP going wider can lead to worse performance

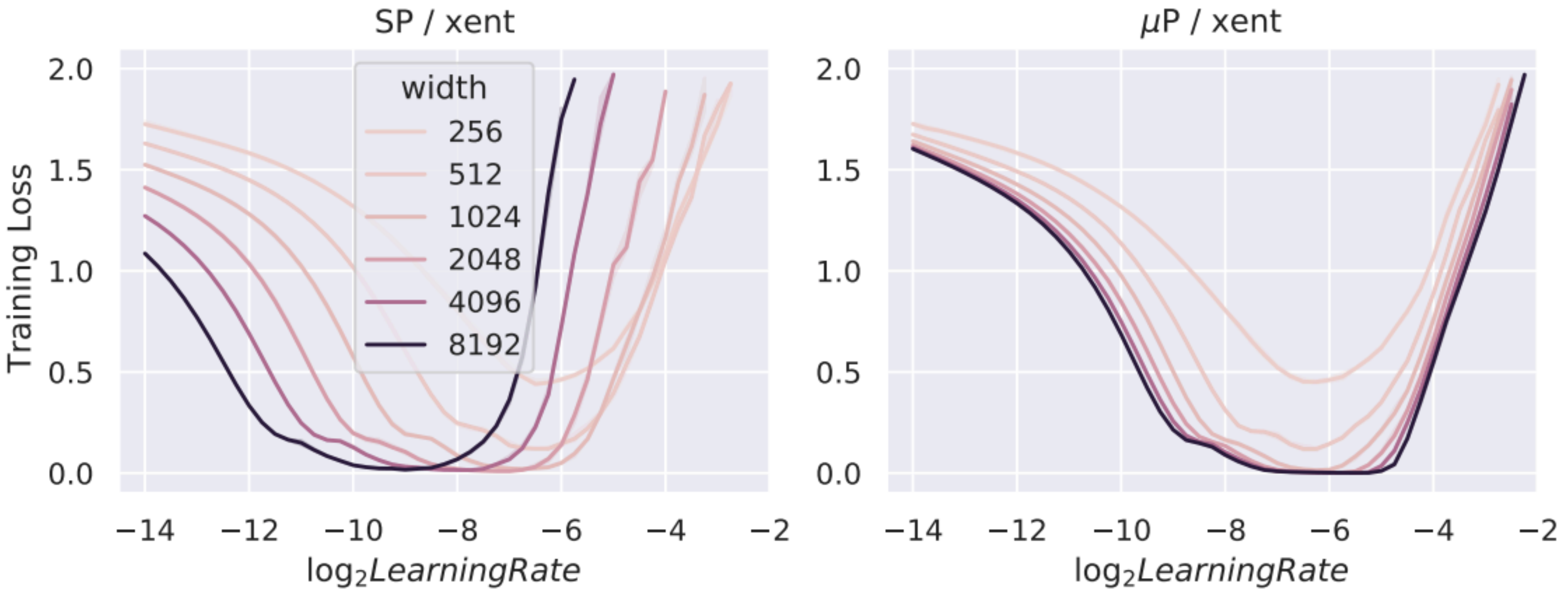
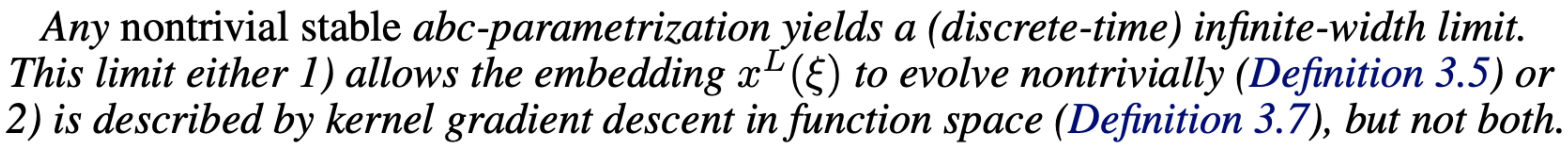


Figure 3: MLP width different hidden sizes trained for 20 epoch on CIFAR-10 using SGD. **Left** uses standard parametrization (SP); **right** uses maximal update parametrization (μP). μP networks exhibit better learning rate stability than their SP counterparts.

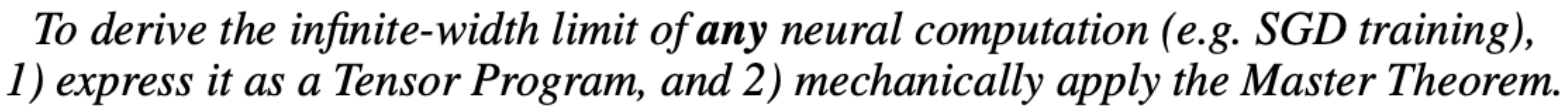
Tensor Programs IV

- Can NTK be “fixed” to have feature learning? No:



Any nontrivial stable *abc-parametrization yields a (discrete-time) infinite-width limit. This limit either 1) allows the embedding $x^L(\xi)$ to evolve nontrivially (Definition 3.5) or 2) is described by kernel gradient descent in function space (Definition 3.7), but not both.*

- Proof:

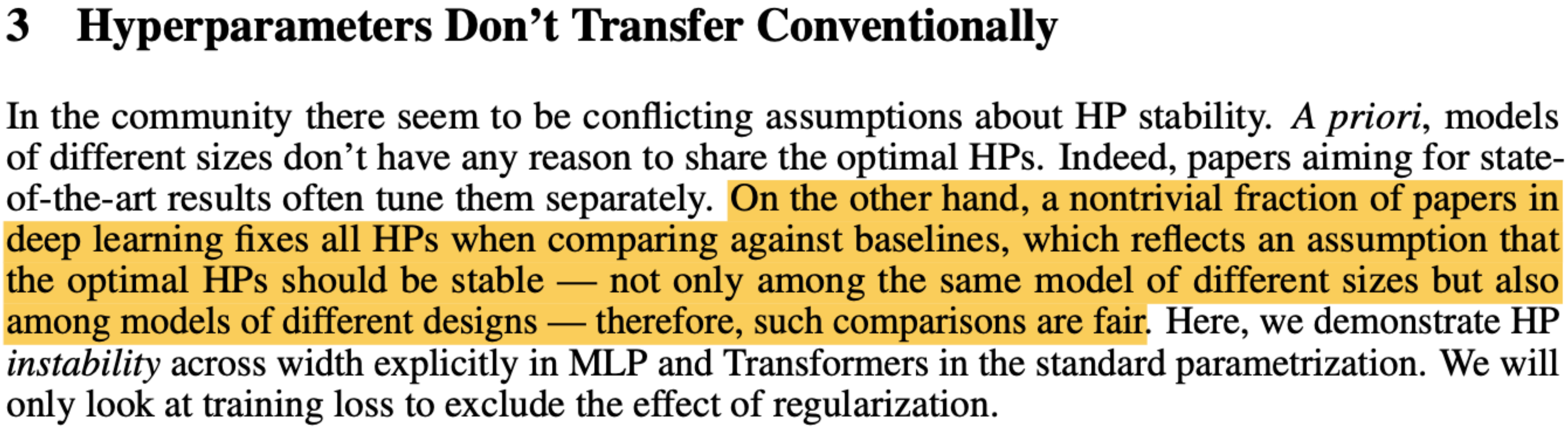


To derive the infinite-width limit of ***any*** *neural computation (e.g. SGD training), 1) express it as a Tensor Program, and 2) mechanically apply the Master Theorem.*

Zero-Shot Hyper-Parameter Transfer

Tensor Programs V

- How to select hyper-parameters across different model sizes/ architectures?

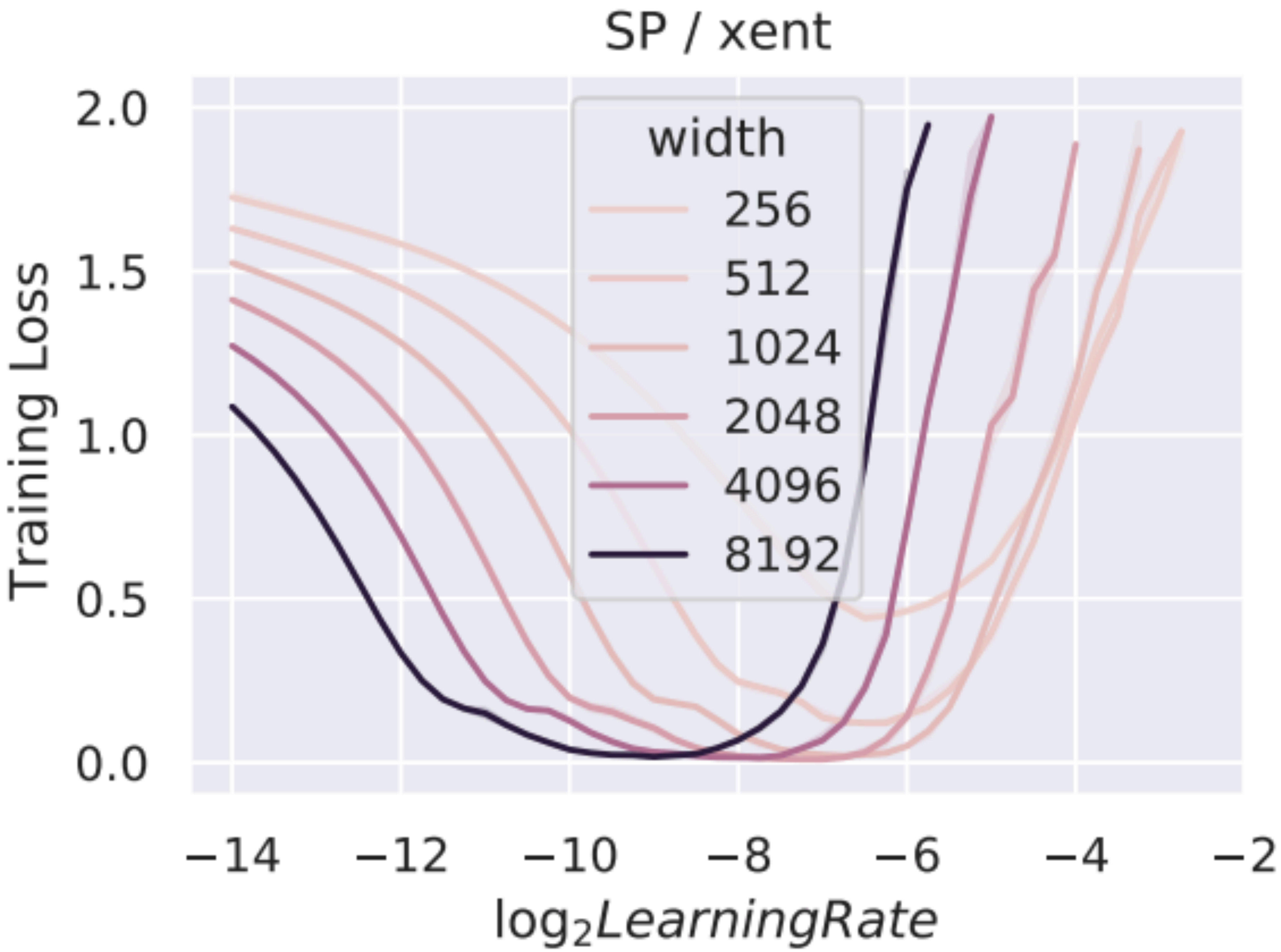


3 Hyperparameters Don't Transfer Conventionally

In the community there seem to be conflicting assumptions about HP stability. *A priori*, models of different sizes don't have any reason to share the optimal HPs. Indeed, papers aiming for state-of-the-art results often tune them separately. On the other hand, a nontrivial fraction of papers in deep learning fixes all HPs when comparing against baselines, which reflects an assumption that the optimal HPs should be stable — not only among the same model of different sizes but also among models of different designs — therefore, such comparisons are fair. Here, we demonstrate HP *instability* across width explicitly in MLP and Transformers in the standard parametrization. We will only look at training loss to exclude the effect of regularization.

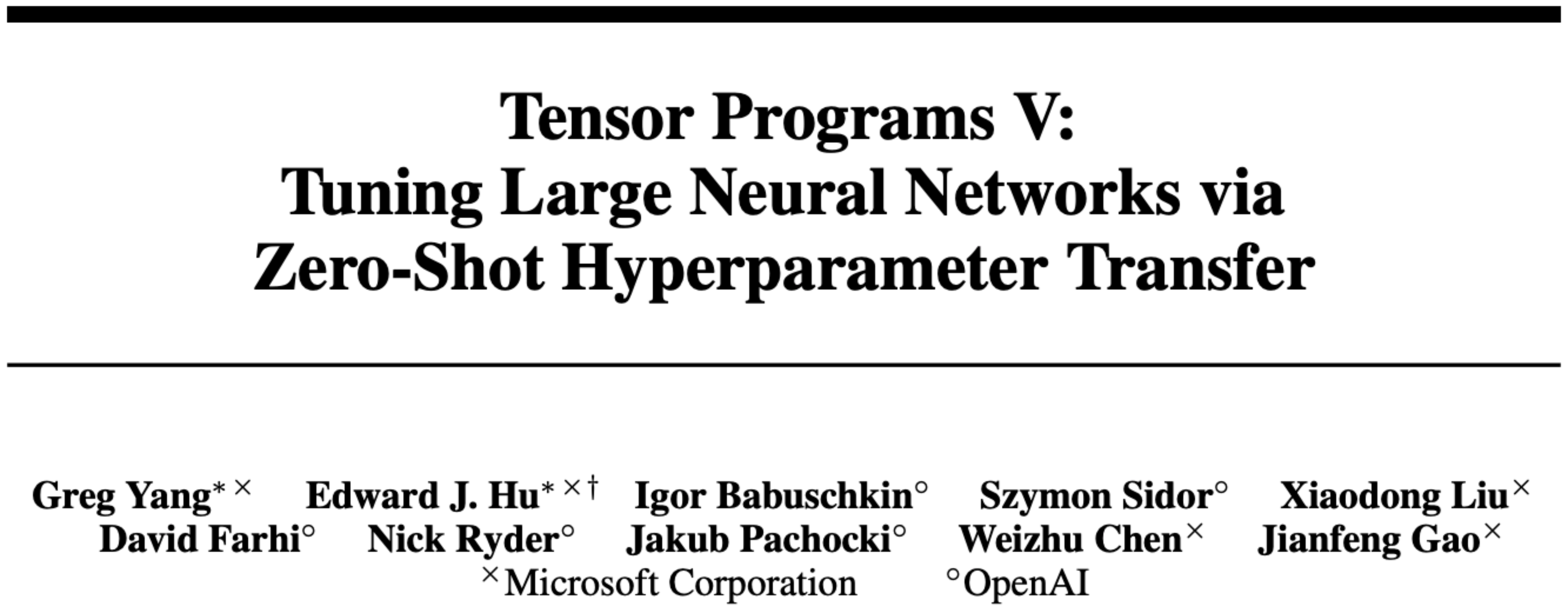
Tensor Programs V

- With standard parametrization, best LR at smaller width could be terrible for larger widths



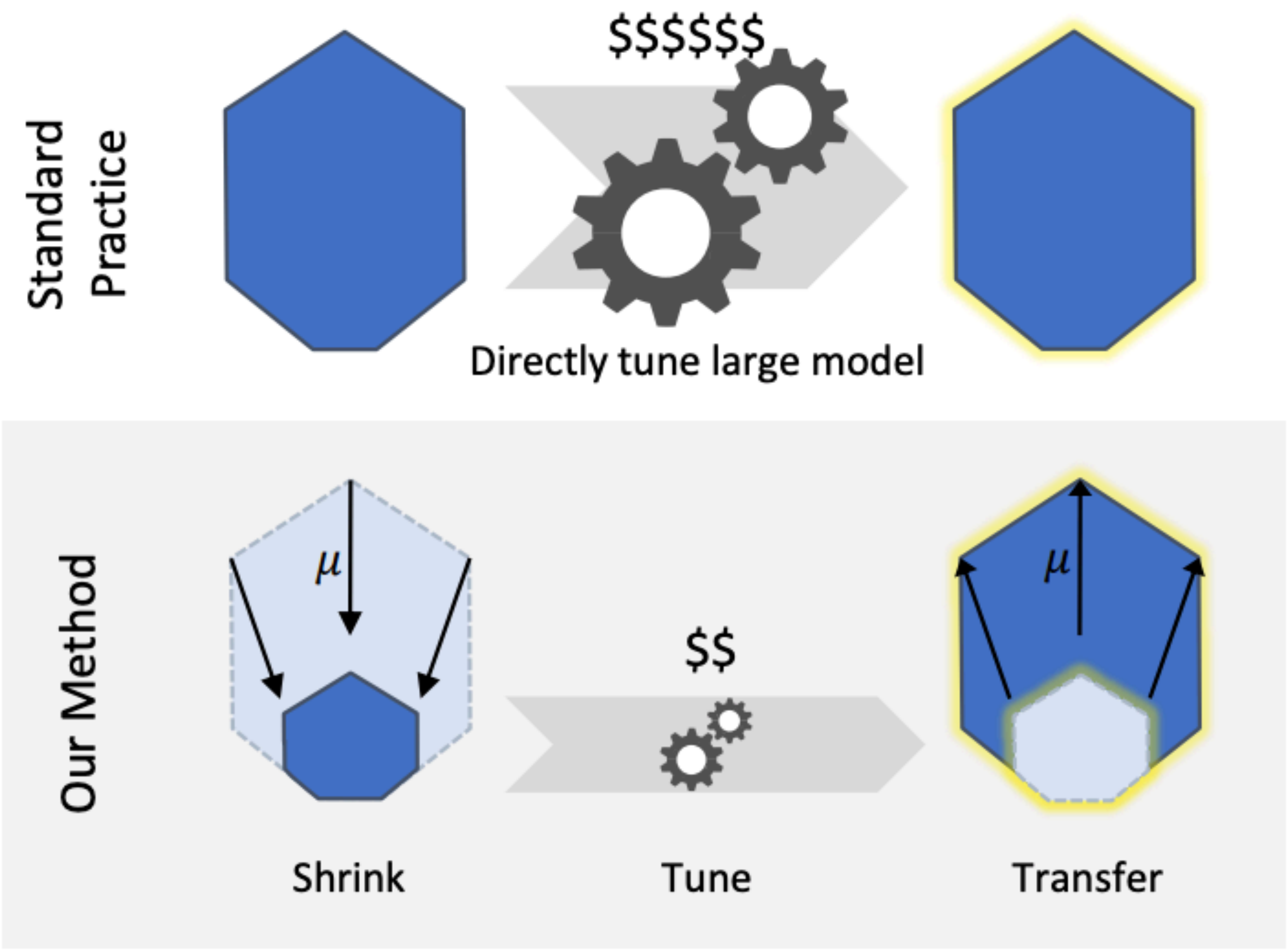
Tensor Programs V

- Significant major practical application from this line of work
- Problem: hyper-parameter tuning on large networks (i.e LLM training runs) expensive
- They show that we can tune hyper-parameters on smaller models, and transfer them to larger models using *μ*P



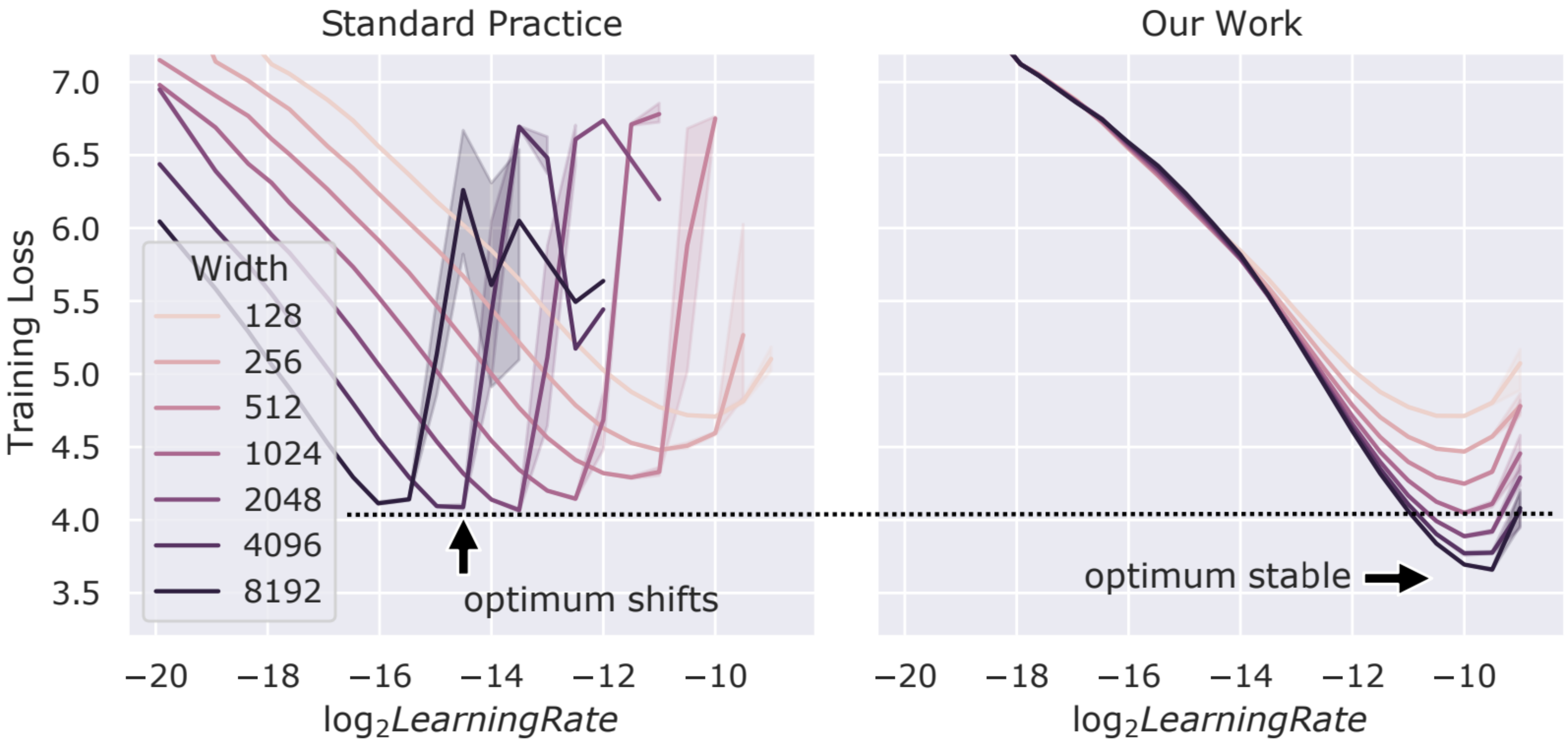
Tensor Programs V:
Tuning Large Neural Networks via
Zero-Shot Hyperparameter Transfer

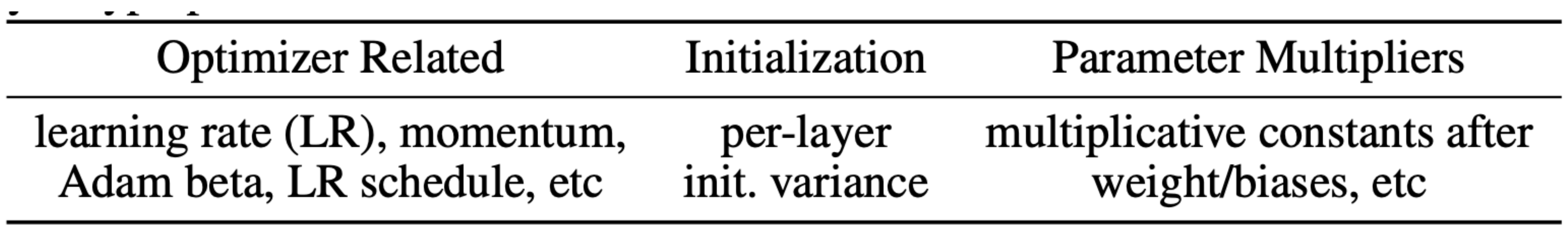
Greg Yang[*×] **Edward J. Hu**[*×†] **Igor Babuschkin**[°] **Szymon Sidor**[°] **Xiaodong Liu**[×]
David Farhi[°] **Nick Ryder**[°] **Jakub Pachocki**[°] **Weizhu Chen**[×] **Jianfeng Gao**[×]
[×]Microsoft Corporation [°]OpenAI



Tensor Programs V

- Hyper-parameter stability: tune once, transfer across any width (with theoretical justification)
- What can be transferred?





Optimizer Related	Initialization	Parameter Multipliers
learning rate (LR), momentum, Adam beta, LR schedule, etc	per-layer init. variance	multiplicative constants after weight/biases, etc

- But empirically works for other dimensions too:
  - Depth
  - Batch size
  - Sequence length
  - Training time

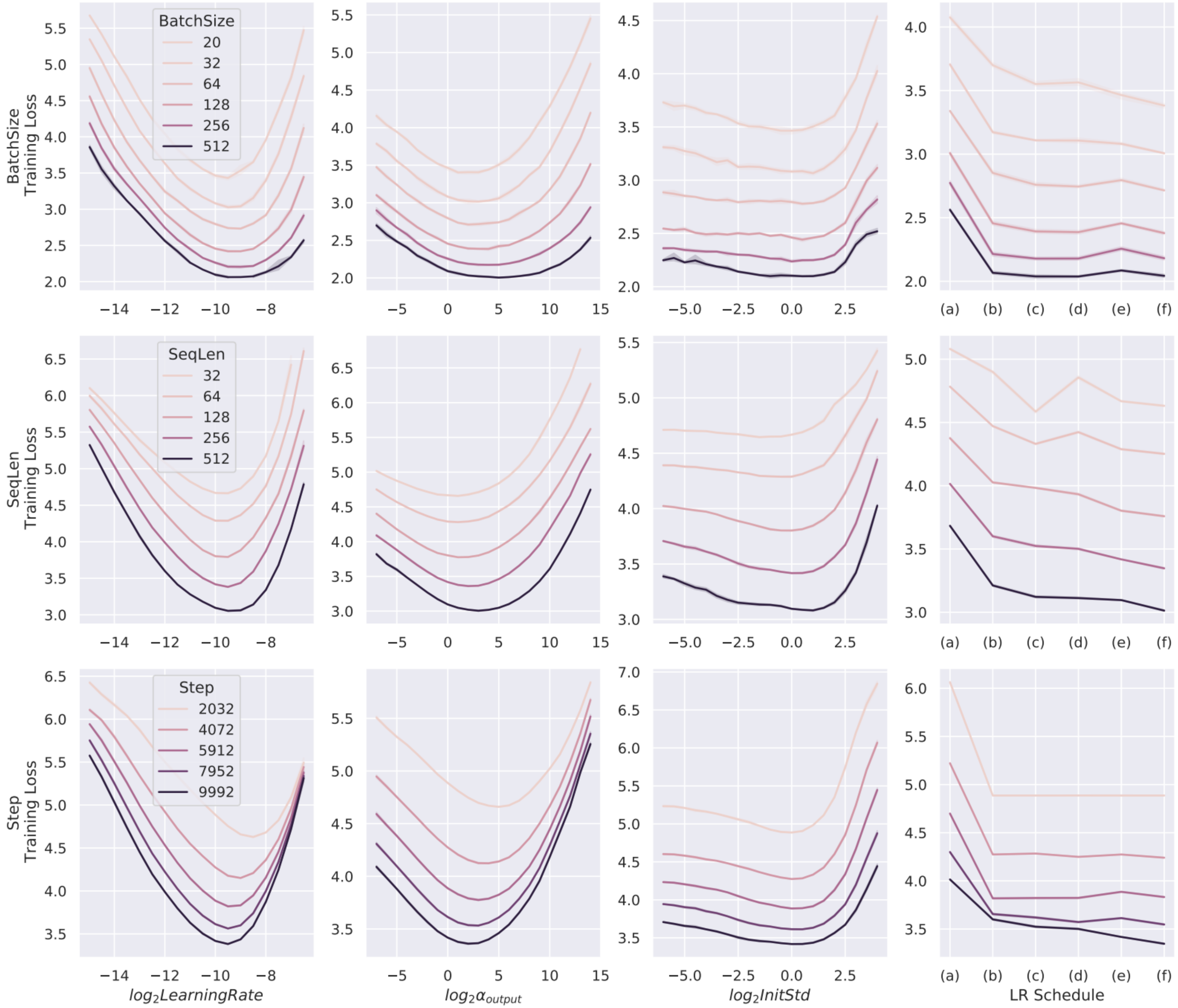


Figure 19: **Empirical validation of μTransfer across Batch Size, Sequence Length, and Training Time on pre-LN Transformers.** Same setting as Fig. 4. Despite some shift, the optimal HPs are roughly stable when transferring from batch size 32, sequence length 128, and 5000 training steps.

Tensor Programs V

- SP does not work: some layer updates too fast, others too slow
- Problems that μP was designed to avoid

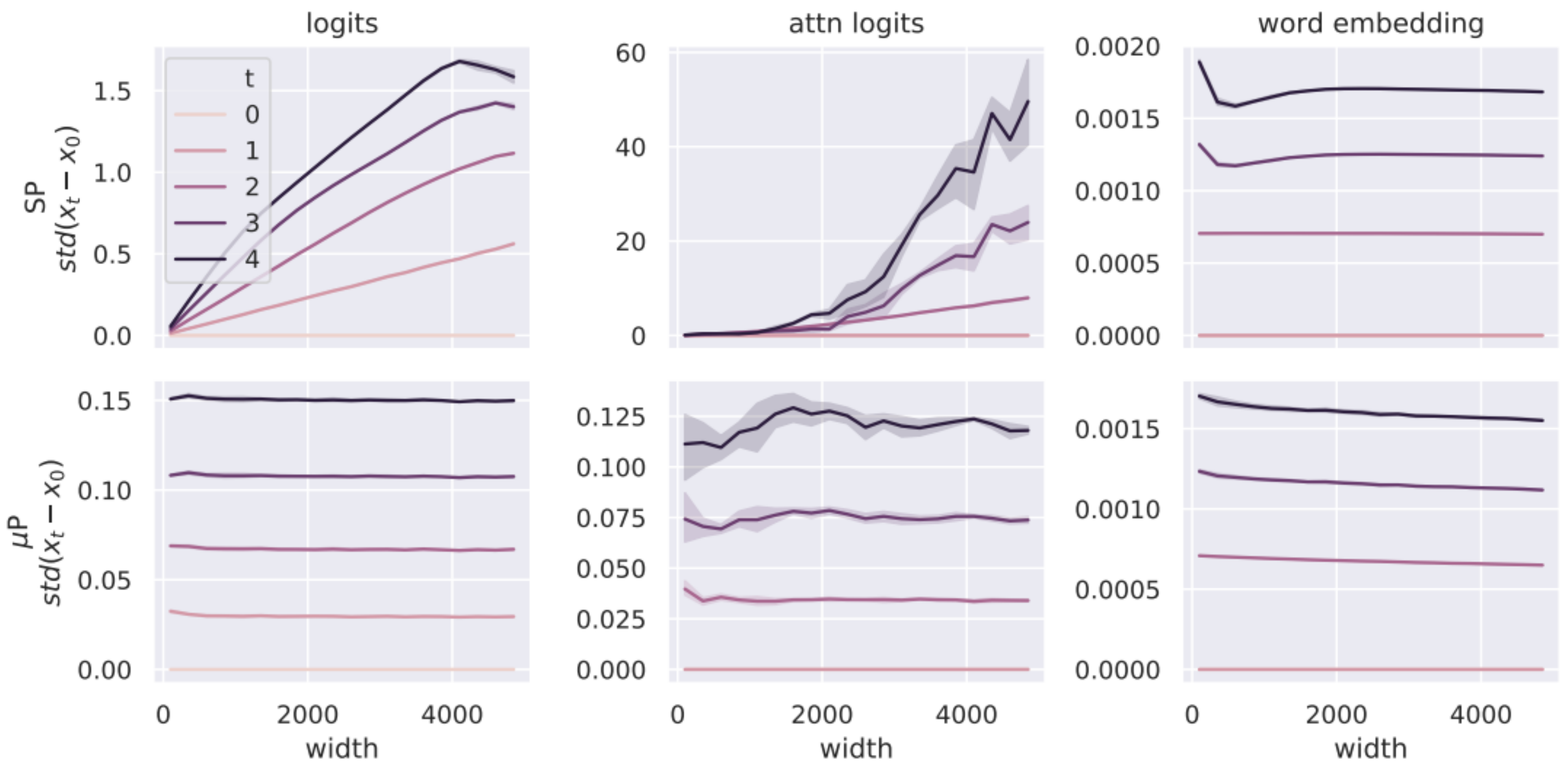


Figure 5: **Logits and attention logits, but not word embeddings, of a Transformer blow up with width in SP after 1 step of training.** In contrast, all three are well-behaved with width in μP. Here we measure how much different values change coordinatewise from initialization over 4 steps of Adam updates, as a function of width. Specifically, we plot the standard deviation of the coordinates of $x_t - x_0$, for $t = 0, \ldots, 4$, and $x \in$ {logits, attention logits, word embeddings}, where $t = 0$ indicates initialization.

Summary

- NTK was close to answering many questions in DL theory
  - In the infinite-width limit, a network’s NTK converges to a deterministic kernel, and stays constant throughout training
  - Training described by kernel gradient descent
  - Convergence, global optimality now very easy to show
- Tensor Programs developed to analyze limiting behavior of NNs, showed NNGP and NTK behavior universal
- Fatal flaw: lack of feature learning, proven impossible to have together with kernel dynamics
- μP parametrization arose from TP analysis and supports feature learning while being stable
- This is also stable for transferring hyper-parameters across widths (and empirically other dimensions)