Feature Learning in Infinite-Width Neural Networks

Greg Yang

Microsoft Research AI

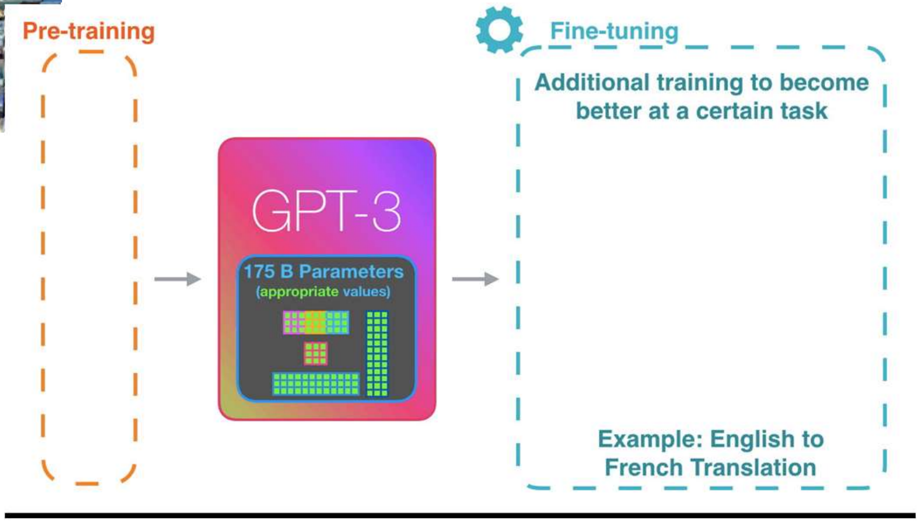
Presenting the 4th Paper of the *Tensor Programs* Series

Joint work with ex-Microsoft AI Resident Edward Hu

Feature Learning is Crucial in Deep Learning

Imagenet and Resnet

BERT and GPT3



Feature Learning is Crucial in Deep Learning

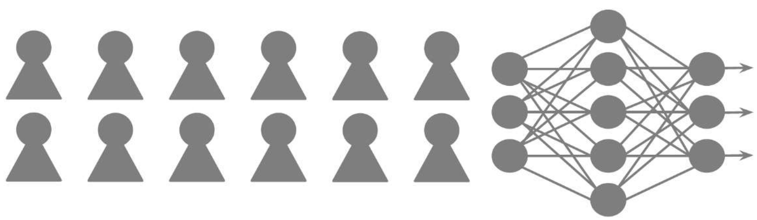
Imagenet and Resnet

BERT and GPT3

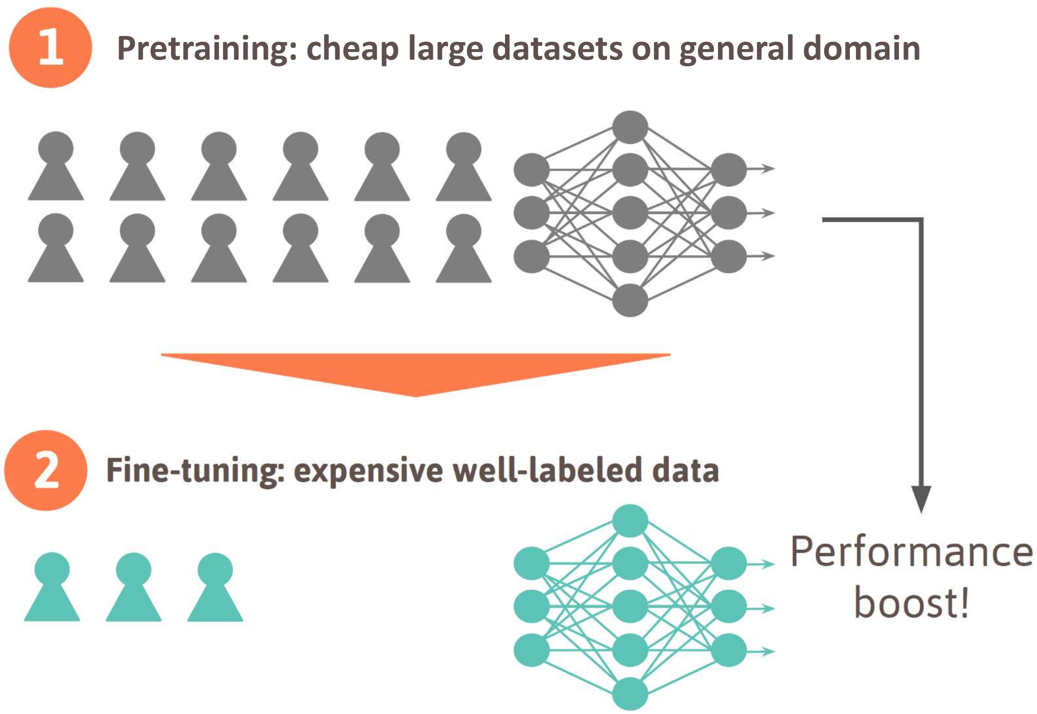
Pretraining and Transfer Learning
Cannot Happen without Feature Learning!

Pretraining and Transfer via Finetuning

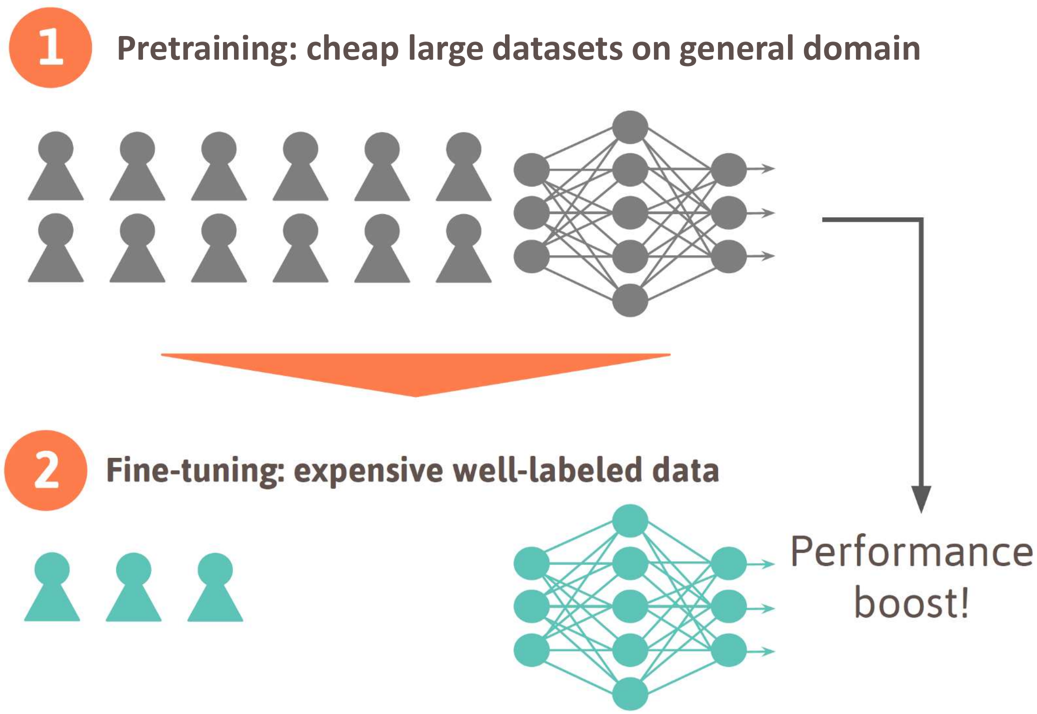
1 **Pretraining: cheap large datasets on general domain**



Pretraining and Transfer via Finetuning

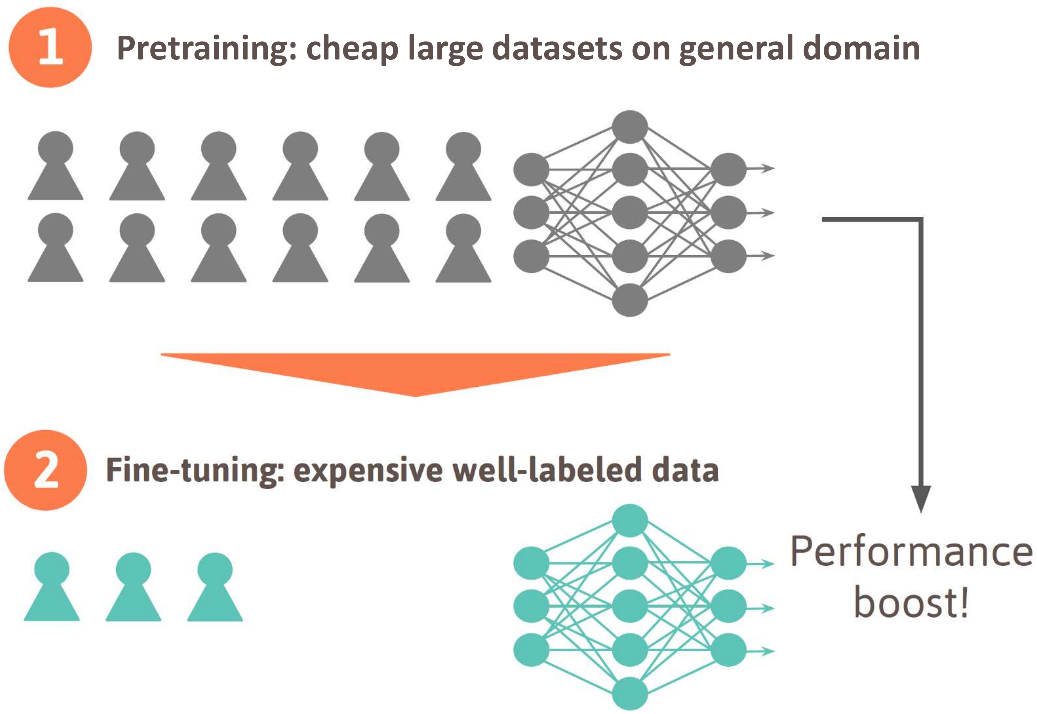


Pretraining and Transfer via Finetuning



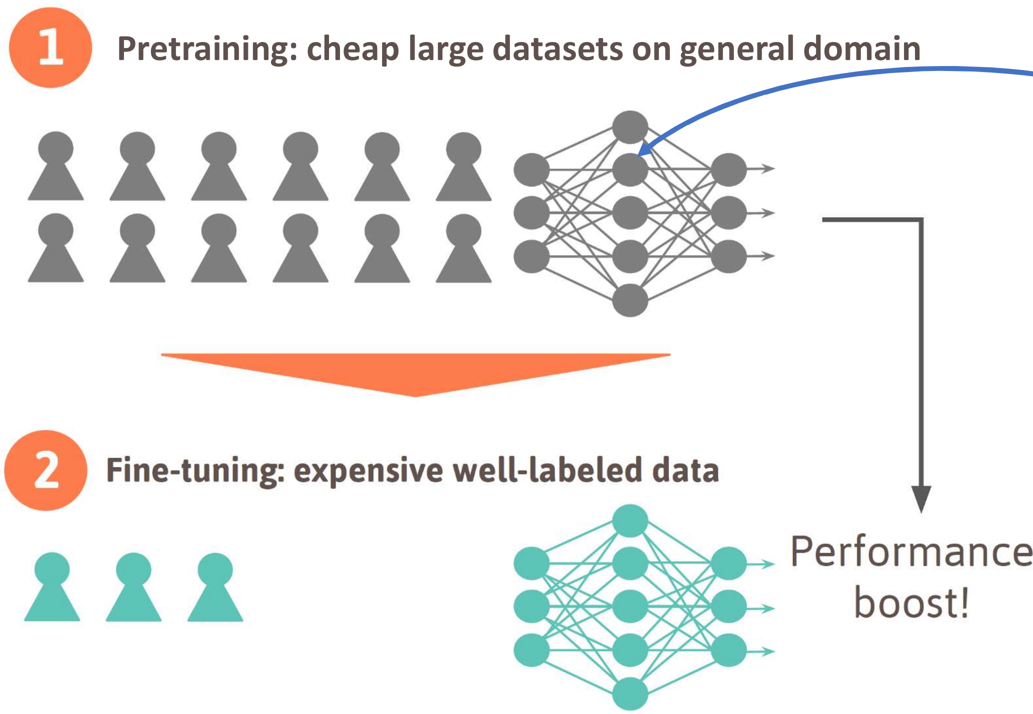
- Discard readout layer from pretraining

Pretraining and Transfer via Finetuning



- Discard readout layer from pretraining
- *Linear Finetuning*: Train new readout layer only

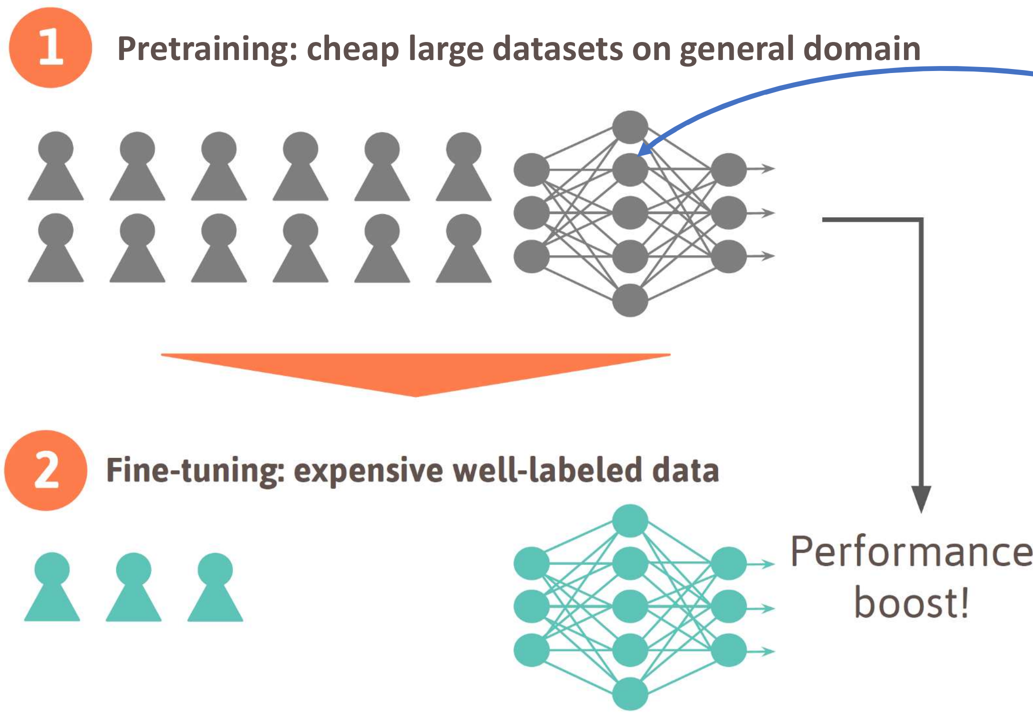
Pretraining and Transfer via Finetuning



If pretraining improves linear finetuning, then the embeddings (i.e. features) of inputs must change during pretraining!

- Discard readout layer from pretraining
- *Linear Finetuning*: Train new readout layer only

Pretraining and Transfer via Finetuning



If pretraining improves linear finetuning, then the embeddings (i.e. features) of inputs must change during pretraining!

- Discard readout layer from pretraining
- *Linear Finetuning*: Train new readout layer only
- *Total Finetuning*: Train the entire network
  - Our conclusions will apply here as well

Current Theory: Neural Tangent Kernel

Naïve first order Taylor expansion

$$f(x;\theta) - f(x;\theta_0) \approx \langle \nabla_\theta f(x;\theta_0), \theta - \theta_0 \rangle$$

so

$$f_t - f_{t-1} \approx -\eta K \mathcal{L}'(f_t, y)$$

Where $\mathcal{L}$ is loss, y is label, and K is the kernel

$$K(x,z) = \langle \nabla_\theta f(x;\theta_0), \nabla_\theta f(z;\theta_0) \rangle$$

Current Theory: Neural Tangent Kernel

Naïve first order Taylor expansion

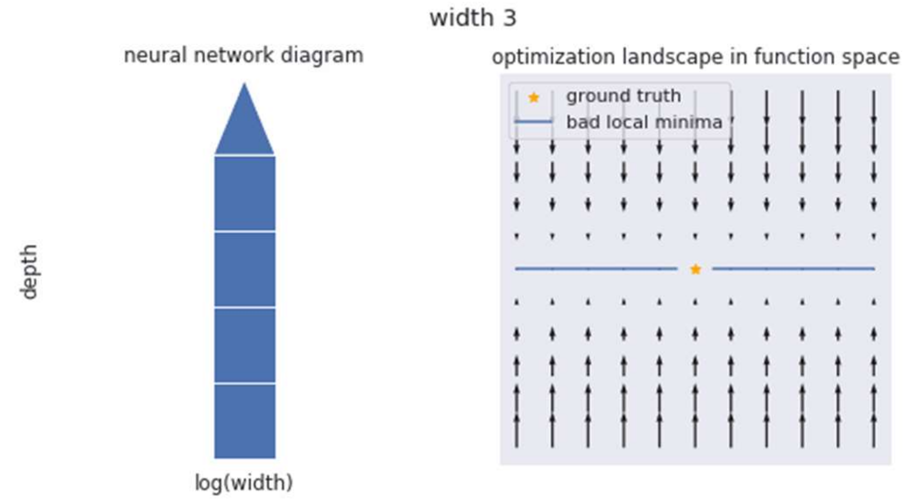
$$f(x;\theta) - f(x;\theta_0) \approx \langle \nabla_\theta f(x;\theta_0), \theta - \theta_0 \rangle$$

so

$$f_t - f_{t-1} \approx -\eta K \mathcal{L}'(f_t, y)$$

Where $\mathcal{L}$ is loss, y is label, and K is the kernel

$$K(x,z) = \langle \nabla_\theta f(x;\theta_0), \nabla_\theta f(z;\theta_0) \rangle$$



Current Theory: Neural Tangent Kernel

- Linear evolution – easy to analyze
- Yields optimization and generalization results

Naïve first order Taylor expansion

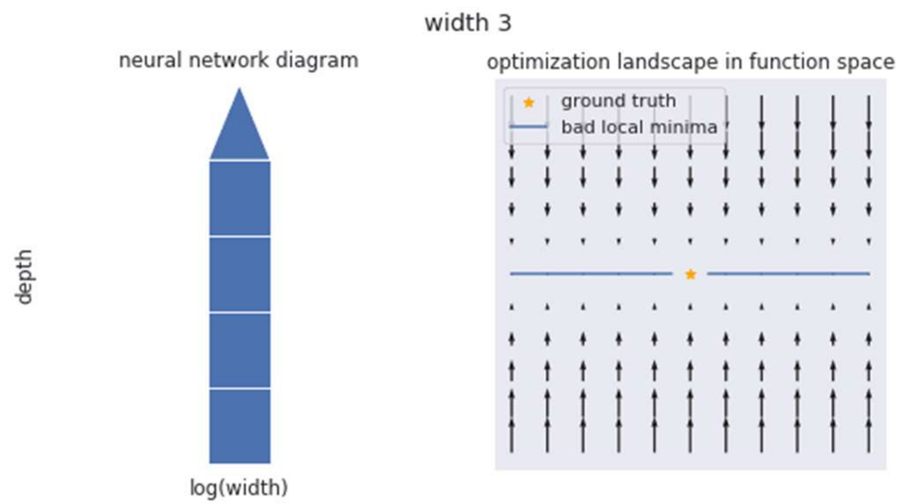
$$f(x;\theta) - f(x;\theta_0) \approx \langle \nabla_\theta f(x;\theta_0), \theta - \theta_0 \rangle$$

so

$$f_t - f_{t-1} \approx -\eta K \mathcal{L}'(f_t, y)$$

Where $\mathcal{L}$ is loss, y is label, and K is the kernel

$$K(x,z) = \langle \nabla_\theta f(x;\theta_0), \nabla_\theta f(z;\theta_0) \rangle$$



Current Theory: Neural Tangent Kernel

- Linear evolution – easy to analyze
- Yields optimization and generalization results

Naïve first order Taylor expansion

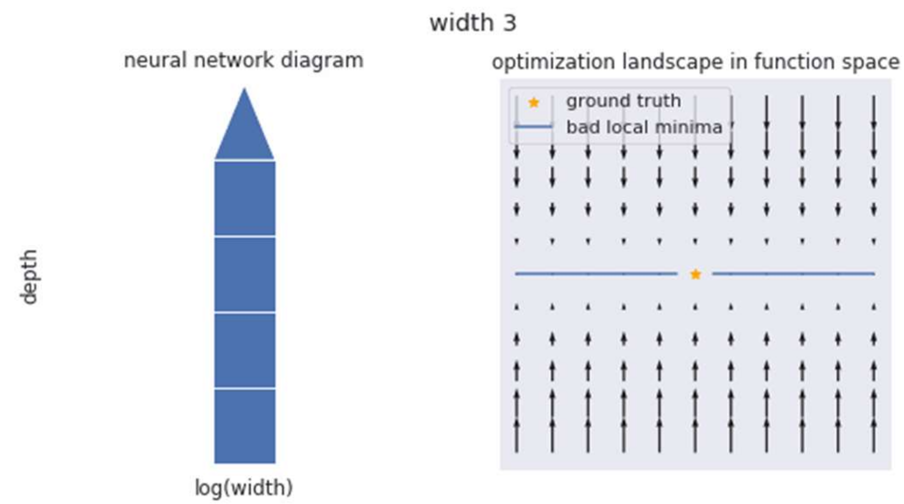
$$f(x;\theta) - f(x;\theta_0) \approx \langle \nabla_\theta f(x;\theta_0), \theta - \theta_0 \rangle$$

so

$$f_t - f_{t-1} \approx -\eta K \mathcal{L}'(f_t, y)$$

Where $\mathcal{L}$ is loss, y is label, and K is the kernel

$$K(x,z) = \langle \nabla_\theta f(x;\theta_0), \nabla_\theta f(z;\theta_0) \rangle$$



But NTK Limit Does Not Learn Features!

Word2Vec example

Will explain later

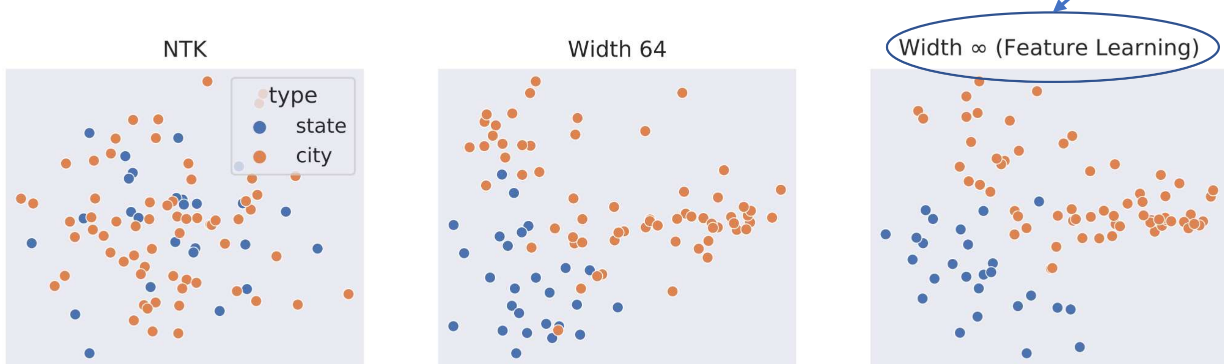
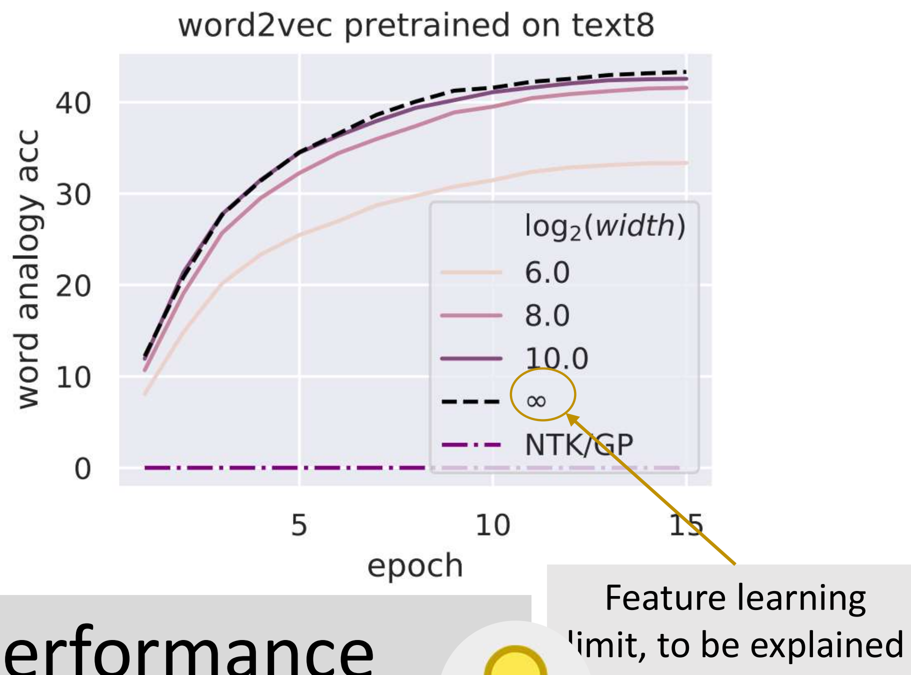


Figure 1: PCA of Word2Vec embeddings of common US cities and states, for NTK, width-64, and width-∞ feature learning neural networks. NTK embeddings are essentially random, while cities and states get naturally separated in embedding space as width increases in the feature learning regime.

Feature Learning ∞-Width NN on Real Tasks

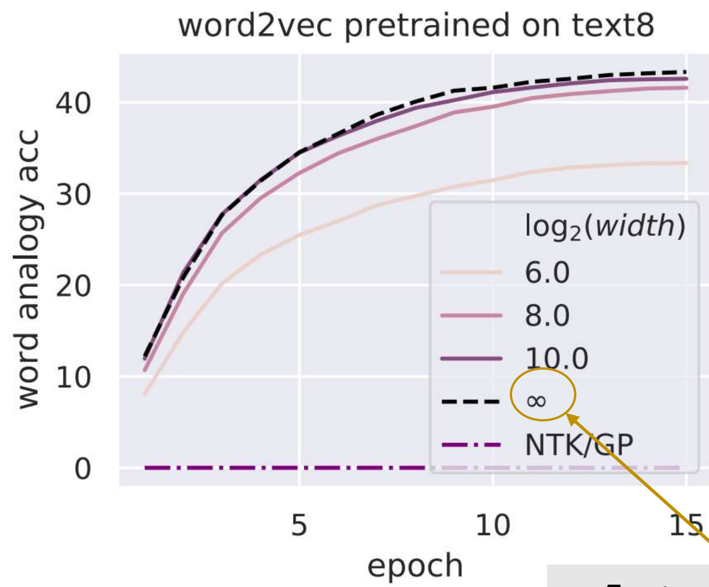
- Word2Vec
  - Pretraining: learn word embeddings so that similar words have close embeddings, in an unsupervised manner.
  - Transfer: word analogy task, e.g. “what to a queen is as a man to a woman?”



- NTK Limit has trivial performance
- Performance is monotonic in width

Feature Learning ∞-Width NN on Real Tasks

- Word2Vec
  - Pretraining: learn word embeddings so that similar words have close embeddings, in an unsupervised manner.
  - Transfer: word analogy task, e.g. “what to a queen is as a man to a woman?”



Feature learning limit, to be explained

- NTK Limit has trivial performance
- Performance is monotonic in width

Similar Results on Metalearning (MAML)

Overview

Our experiments verify this is the right limit

Our Theoretical Contributions

- Classify all "viable" ∞-width limits into feature learning and kernel limits
- Identify "*the maximal*" feature learning limit
- Propose the *Tensor Programs* technique for deriving its equations, and more generally, the limit of any neural computation

Significance

- Framework for studying feature learning in overparametrized NN
- Formulas for training feature-learning ∞-width NN in variety of settings (e.g. pretraining, metalearning, reinforcement learning, GANs, etc)
- Mostly solves the problem of taking ∞-width limits

Outline of This Talk

- Parametrizations of Neural Networks
- Dichotomy of Parametrizations
- The “Maximal” Feature Learning Limit
- The *Tensor Programs* technique for deriving the limit

Outline of This Talk

- Parametrizations of Neural Networks
- Dichotomy of Parametrizations
- The “Maximal” Feature Learning Limit
- The *Tensor Programs* technique for deriving the limit

abc-Parametrizations

- Consider L-hidden-layer perceptron
  - Input dim d, output dim 1, width n, nonlinearity ϕ, input $\xi \in \mathbb{R}^d$
  - Weights $W^1 \in \mathbb{R}^{n\times d}$ and $W^2, \dots, W^L \in \mathbb{R}^{n\times n}$ and $W^{L+1} \in \mathbb{R}^{1\times n}$
  - $h^1(\xi) = W^1\xi \in \mathbb{R}^n$
  - $x^l(\xi) = \phi\left(h^l(\xi)\right) \in \mathbb{R}^n, h^{l+1}(\xi) = W^{l+1}x^l(\xi) \in \mathbb{R}^n$ for $l = 1, \dots, L-1$
  - Network output (i.e. logits) $f(\xi) = W^{L+1}x^L(\xi)$

abc-Parametrizations

- Consider L-hidden-layer perceptron
  - Input dim d, output dim 1, width n, nonlinearity ϕ, input $\xi \in \mathbb{R}^d$
  - Weights $W^1 \in \mathbb{R}^{n \times d}$ and $W^2, \dots, W^L \in \mathbb{R}^{n \times n}$ and $W^{L+1} \in \mathbb{R}^{1 \times n}$
  - $h^1(\xi) = W^1 \xi \in \mathbb{R}^n$
  - $x^l(\xi) = \phi\left(h^l(\xi)\right) \in \mathbb{R}^n, h^{l+1}(\xi) = W^{l+1} x^l(\xi) \in \mathbb{R}^n$ for $l = 1, \dots, L-1$
  - Network output (i.e. logits) $f(\xi) = W^{L+1} x^L(\xi)$

An *abc-parametrization* is given by a set of numbers $\{a_l, b_l\}_l \cup \{c\}$ s.t.

a) Parametrize each $W^l = n^{-a_l} w^l$ where w^l is trained instead of W^l

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

c) SGD learning rate is ηn^{-c} for some width-independent η.

Examples

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

	Definition	NTK	Standard (Pytorch default)	Mean Field ($L = 1$)
a_l	$W^l = n^{-a_l} w^l$	$\begin{cases} 0 & \text{if } l = 1 \\ \frac{1}{2} & \text{if } l > 1 \end{cases}$	0	$\begin{cases} 0 & \text{if } l = 1 \\ 1 & \text{if } l = 2 \end{cases}$
b_l	$w^l_{\alpha\beta} \sim \mathcal{N}(0, n^{-2b_l})$	0	$\begin{cases} 0 & \text{if } l = 1 \\ \frac{1}{2} & \text{if } l > 1 \end{cases}$	0
c	$LR = \eta n^{-c}$	0	0	-1

Examples

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

	Definition	NTK	Standard (Pytorch default)	Mean Field ($L = 1$)
a_l	$W^l = n^{-a_l}w^l$	$\begin{cases} 0 & \text{if } l = 1 \\ \frac{1}{2} & \text{if } l > 1 \end{cases}$	0	$\begin{cases} 0 & \text{if } l = 1 \\ 1 & \text{if } l = 2 \end{cases}$
b_l	$w^l_{\alpha\beta} \sim \mathcal{N}(0, n^{-2b_l})$	0	$\begin{cases} 0 & \text{if } l = 1 \\ \frac{1}{2} & \text{if } l > 1 \end{cases}$	0
c	$LR = \eta n^{-c}$	0	0	-1

We are ignoring dependences on input dim d (which should be thought of as a constant)

Examples

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

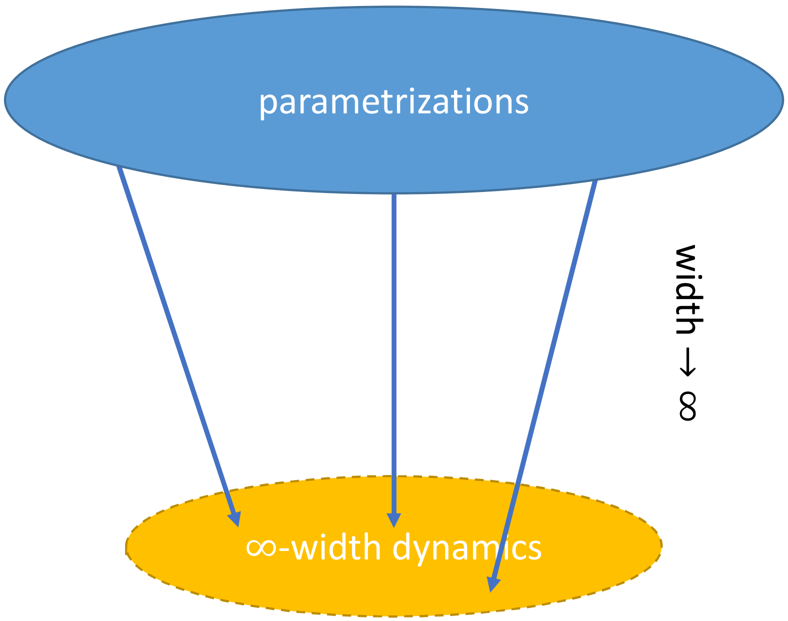
	Definition	NTK	Standard (Pytorch default)	Mean Field ($L = 1$)
a_l	$W^l = n^{-a_l}w^l$	$\begin{cases} 0 & \text{if } l = 1 \\ \frac{1}{2} & \text{if } l > 1 \end{cases}$	0	$\begin{cases} 0 & \text{if } l = 1 \\ 1 & \text{if } l = 2 \end{cases}$
b_l	$w^l_{\alpha\beta} \sim \mathcal{N}(0, n^{-2b_l})$	0	$\begin{cases} 0 & \text{if } l = 1 \\ \frac{1}{2} & \text{if } l > 1 \end{cases}$	0
c	$LR = \eta n^{-c}$	0	0	-1

We are ignoring dependences on input dim d (which should be thought of as a constant)

By changing a_l, b_l while fixing $a_l + b_l$, we effectively give each layer l its own learning rate.

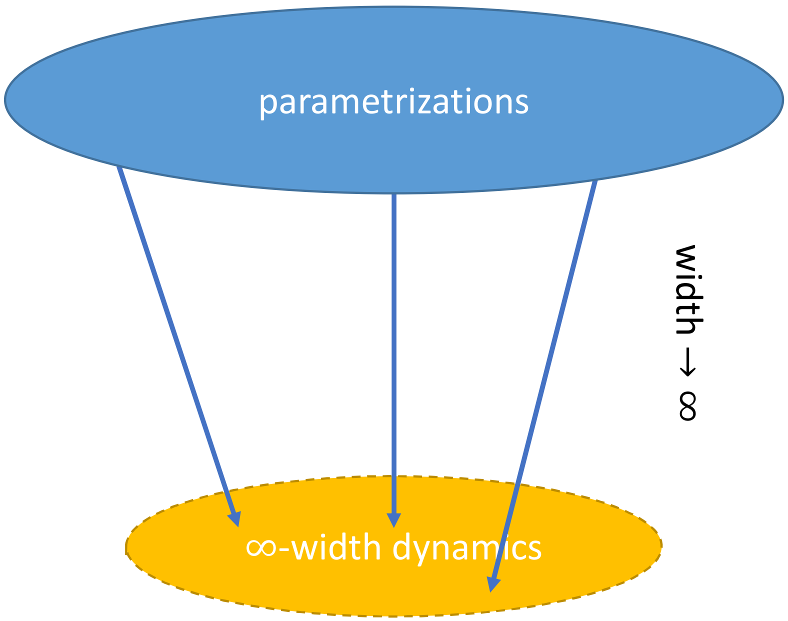
Why Do We Care About Parametrizations?

- -width limit
  - abc-parametrizations correspond to natural class of limits, including NTK, GP, Mean Field, etc
  - abc-parametrizations correspond to natural class of limits, including NTK, GP, Mean Field, etc



Why Do We Care About Parametrizations?

- -width limit
  - abc-parametrizations correspond to natural class of limits, including NTK, GP, Mean Field, etc
- Parametrizations are important practically
  - E.g. Glorot, He, Fixup



Outline of This Talk

- Parametrizations of Neural Networks
- Dichotomy of Parametrizations
- The “Maximal” Feature Learning Limit
- The *Tensor Programs* technique for deriving the limit

Outline of This Talk

- Parametrizations of Neural Networks
- Dichotomy of Parametrizations
- The “Maximal” Feature Learning Limit
- The *Tensor Programs* technique for deriving the limit

Instability and Triviality

$LR = \eta n^{-c}$

- If LR too large w.r.t width (i.e. c too small), then logits or preactivations blow up with width during training
  - We say this abc-parametrization is $unstable$
- If LR too small w.r.t width (i.e. c too big), then the NN function doesn't evolve in the ∞-width limit
  - We say this abc-parametrization is *trivial*
- Only *nontrivial stable* abc-parametrizations are meaningful

Dynamical Dichotomy Theorem

- Any *nontrivial stable* abc-parametrization yields a (discrete-time) ∞-width limit
- When trained for any finite time, this limit either
  1. (Feature learning limit) allows the embedding $x^L(\xi)$ to evolve nontrivially or
  2. (Kernel limit) is described by kernel gradient descent in function space, but not both.

Example: Mean Field when $L = 1$

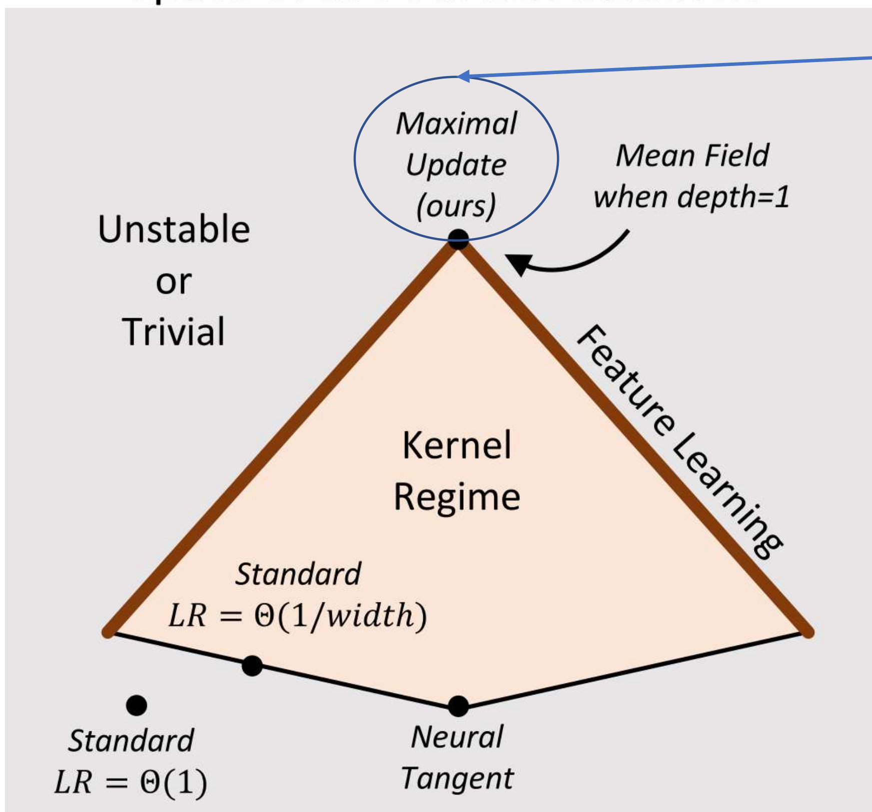
Example: NTK

for some kernel K,
$$f_{t+1} - f_t = -K\mathcal{L}'(f_t, y)$$

What is ruled out?
e.g. higher order generalizations of NTK dynamics like
$$f_{t+1} - f_t = -\langle K^{(2)}, \mathcal{L}'(f_t, y)^{\otimes 2}\rangle$$

Interesting consequence:
the NN function must be identically 0 at init in any feature learning limit

A Caricature of Space of abc-Parametrizations

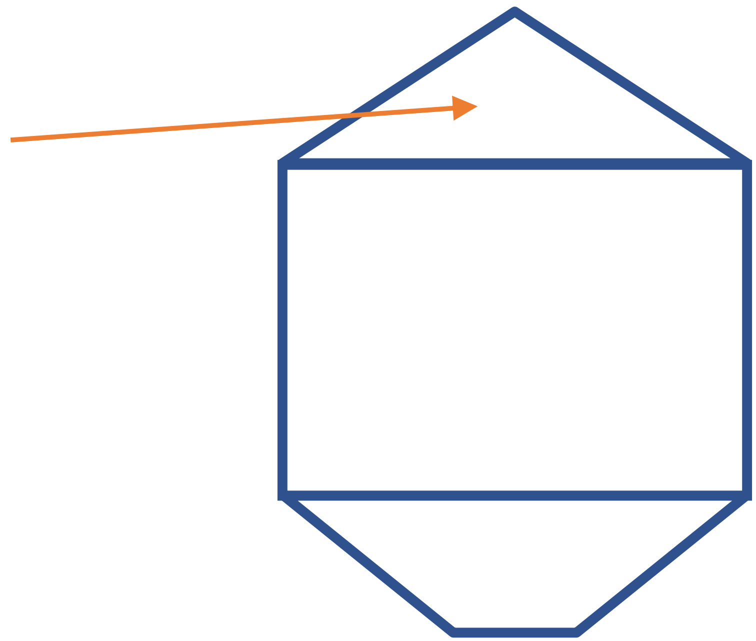


Will explain soon

- The *nontrivial stable* params form a high dimensional polytope
- Feature learning params form 2 facets
- Everything else is in kernel regime

NTK, Standard Param Don't Learn Features

- Intuition why
  - The last layer weights get too much gradient, relative to weights in the body
  - We want to use larger learning rate to enable feature learning, but then the logits would blow up.



Outline of This Talk

- Parametrizations of Neural Networks
- Dichotomy of Parametrizations
- The “Maximal” Feature Learning Limit
- The *Tensor Programs* technique for deriving the limit

Outline of This Talk

- Parametrizations of Neural Networks
- Dichotomy of Parametrizations
- The “Maximal” Feature Learning Limit
- The *Tensor Programs* technique for deriving the limit

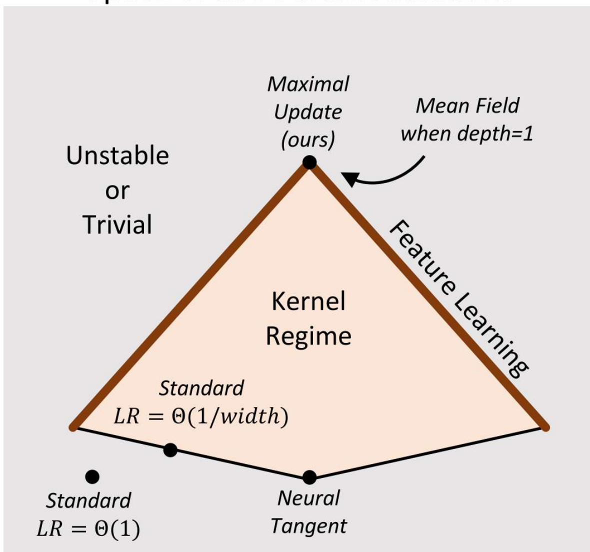
Maximal Update Parametrization (μP)

- Modify Standard Param to get Maximal Update Param
  - Last layer: divide logits by $\sqrt{n}$ and use $\Theta(1)$ learning rate
    - i.e. $a_{L+1} = \frac{1}{2}, c = 0$
    - i.e. $f(\xi) = \frac{1}{\sqrt{n}} w^{L+1} x^L(\xi)$ where $w_{\alpha\beta}^{L+1} \sim \mathcal{N}\left(0, \frac{1}{n}\right)$
    - This alone suffices to enable feature learning
  - First layer: increase the gradient by n by setting $a_1 = -\frac{1}{2}, b_1 = 1/2$
    - i.e. $h^1(\xi) = \sqrt{n} w^1 \xi$ where $w_{\alpha\beta}^1 \sim \mathcal{N}\left(0, \frac{1}{n}\right)$
    - Needed to enable feature learning in *every* layer

Maximality of Maximal Update Param

- Maximal Update Param is *Maximally Feature Learning*: *Every* layer learns features ($h^l(\xi), x^l(\xi)$ for every l will evolve during training).

Space of abc-Parametrizations

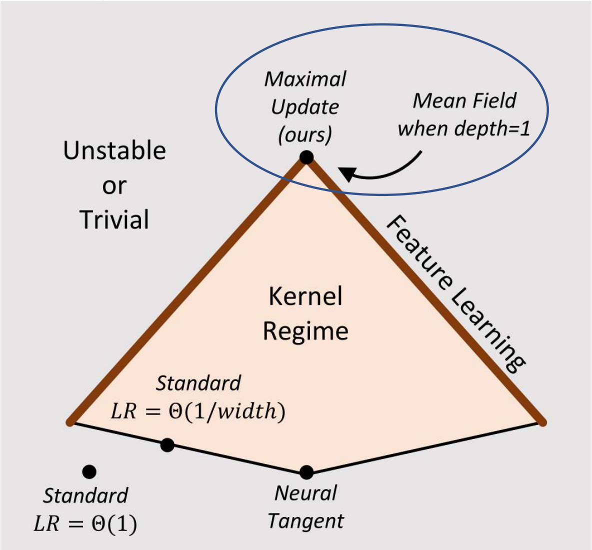


1-Dimension Degeneracy in abc

- For any $\theta \in \mathbb{R}$ and $t \geq 0$, $f(\xi)$ at time t stays fixed for all input ξ if

$$a_l \leftarrow a_l + \theta, \qquad b_l \leftarrow b_l - \theta, \qquad c \leftarrow c - 2\theta$$

Space of abc-Parametrizations



Equivalent when $L = 1$ *because, for* $\theta = 1/2$,
$a_l \leftarrow a_l + \theta, b_l \leftarrow b_l - \theta, c \leftarrow c - 2\theta$

Summary

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

	Definition	NTK	Standard	Standard ($1/n$ LR)	Mean Field ($L = 1$)	Maximal Update
a_l	$W^l = n^{-a_l}w^l$	$\begin{cases} 0 & \text{if } l = 1 \\ \frac{1}{2} & \text{if } l > 1 \end{cases}$	0	0	$\begin{cases} 0 & \text{if } l = 1 \\ 1 & \text{if } l = 2 \end{cases}$	$\begin{cases} -\frac{1}{2} & \text{if } l = 1 \\ 0 & \text{if } 2 \le l \le L \\ \frac{1}{2} & \text{if } l = L+1 \end{cases}$
b_l	$w^l_{\alpha\beta} \sim \mathcal{N}(0, n^{-2b_l})$	0	$\begin{cases} 0 & \text{if } l = 1 \\ \frac{1}{2} & \text{if } l > 1 \end{cases}$	$\begin{cases} 0 & \text{if } l = 1 \\ \frac{1}{2} & \text{if } l > 1 \end{cases}$	0	1/2
c	$LR = \eta n^{-c}$	0	0	1	-1	0
Nontrivial		✔	✔	✔	✔	✔
Stable		✔	✘	✔	✔	✔
Feature Learning		✘	-	✘	✔	✔
Any depth		✔	✔	✔	✘	✔

Outline of This Talk

- Parametrizations of Neural Networks
- Dichotomy of Parametrizations
- The “Maximal” Feature Learning Limit
- The *Tensor Programs* technique for deriving the limit

Outline of This Talk

- Parametrizations of Neural Networks
- Dichotomy of Parametrizations
- The “Maximal” Feature Learning Limit
- The *Tensor Programs* technique for deriving the limit
    - Key Idea
    - Motivating Example: Linear 1-Hidden-Layer NN
    - What is a Tensor Program?
    - Master Theorem
    - Infinite-Width Limit for All

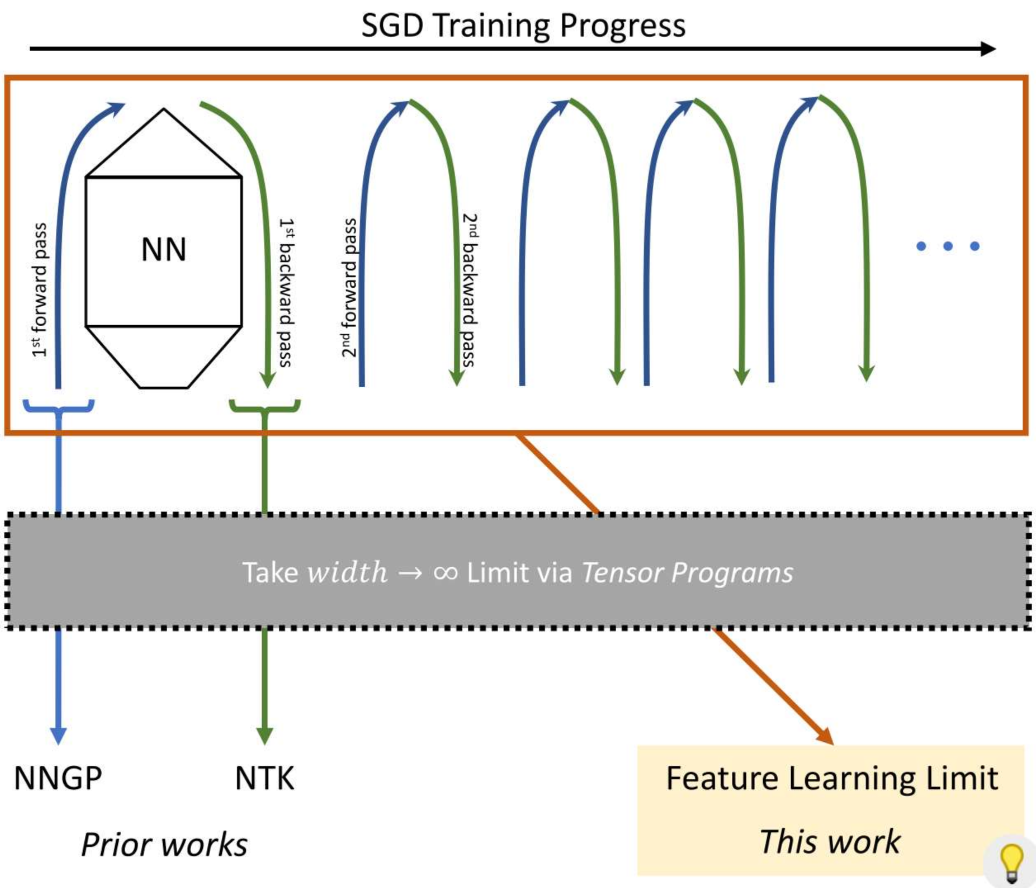
Outline of This Talk

- Parametrizations of Neural Networks
- Dichotomy of Parametrizations
- The “Maximal” Feature Learning Limit
- The *Tensor Programs* technique for deriving the limit
  - Key Idea
  - Motivating Example: Linear 1-Hidden-Layer NN
  - What is a Tensor Program?
  - Master Theorem
  - Infinite-Width Limit for All

Key Theoretical Idea: Tensor Programs

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.*

Key Theoretical Idea: Tensor Programs



Outline of This Talk

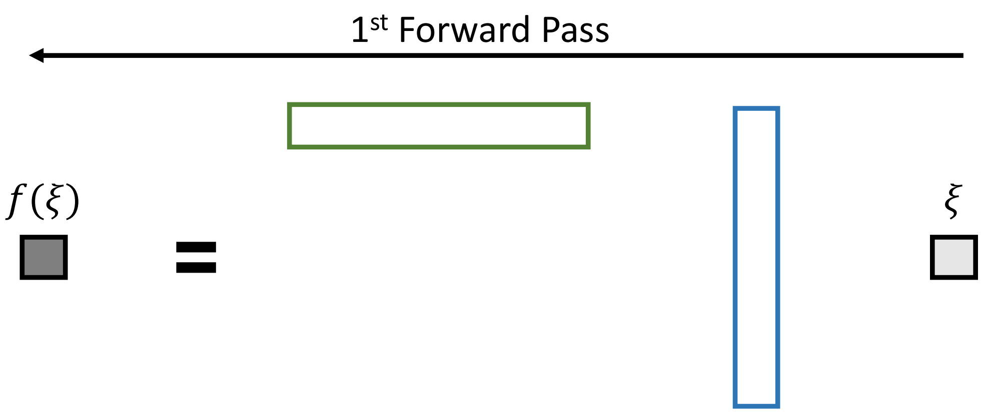
- Parametrizations of Neural Networks
- Dichotomy of Parametrizations
- The “Maximal” Feature Learning Limit
- The *Tensor Programs* technique for deriving the limit
    - Key Idea
    - Motivating Example: Linear 1-Hidden-Layer NN
    - What is a Tensor Program?
    - Master Theorem
    - Infinite-Width Limit for All

Outline of This Talk

- Parametrizations of Neural Networks
- Dichotomy of Parametrizations
- The “Maximal” Feature Learning Limit
- The *Tensor Programs* technique for deriving the limit
  - Key Idea
  - Motivating Example: Linear 1-Hidden-Layer NN
  - What is a Tensor Program?
  - Master Theorem
  - Infinite-Width Limit for All

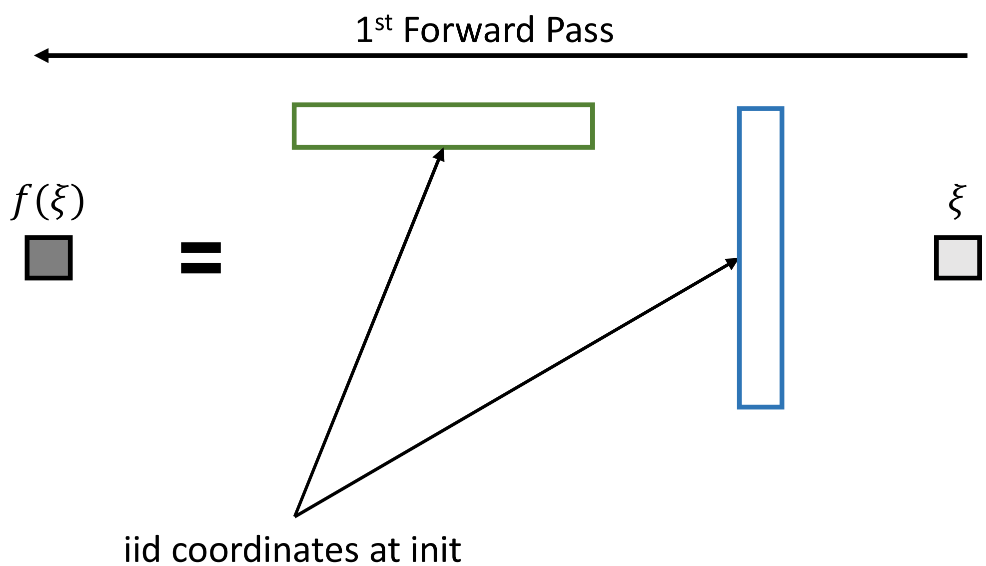
Motivating Example: Linear 1-Hidden-Layer

Assume input and output dim = 1



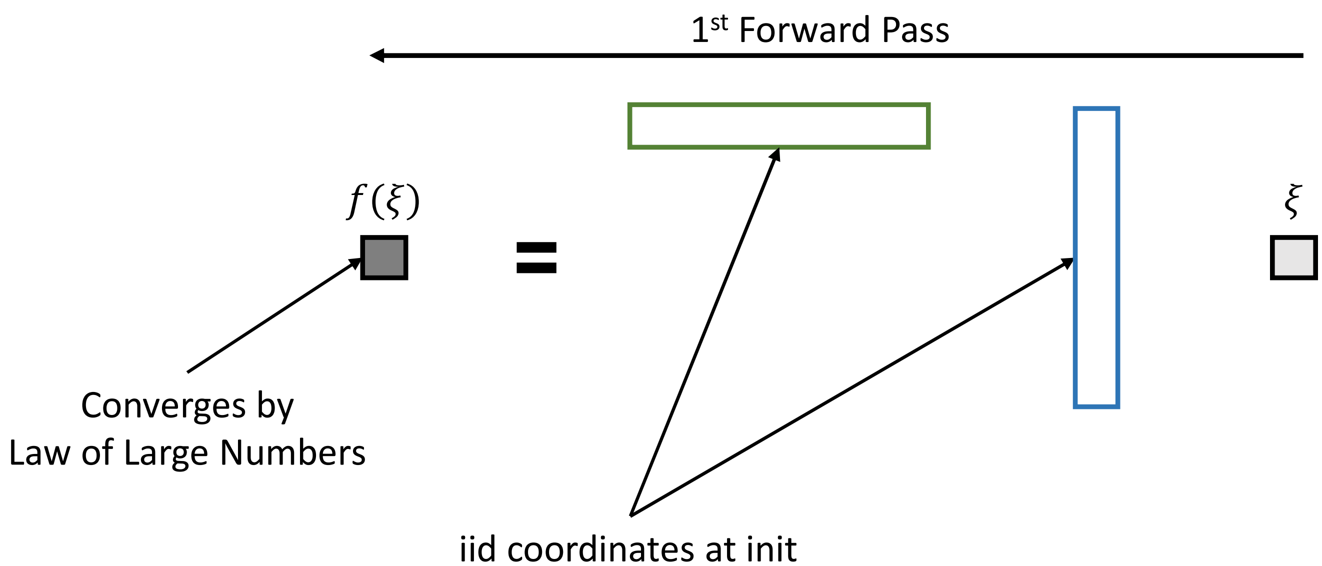
Motivating Example: Linear 1-Hidden-Layer

Assume input and output dim = 1

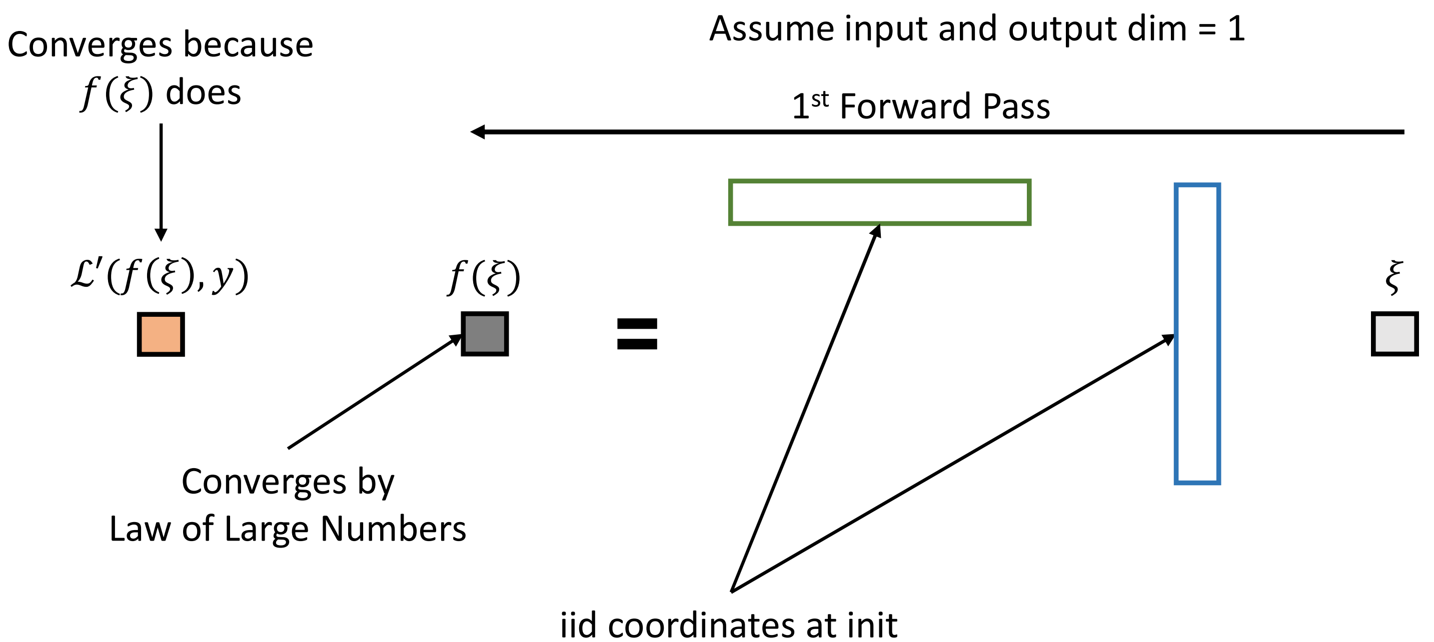


Motivating Example: Linear 1-Hidden-Layer

Assume input and output dim = 1

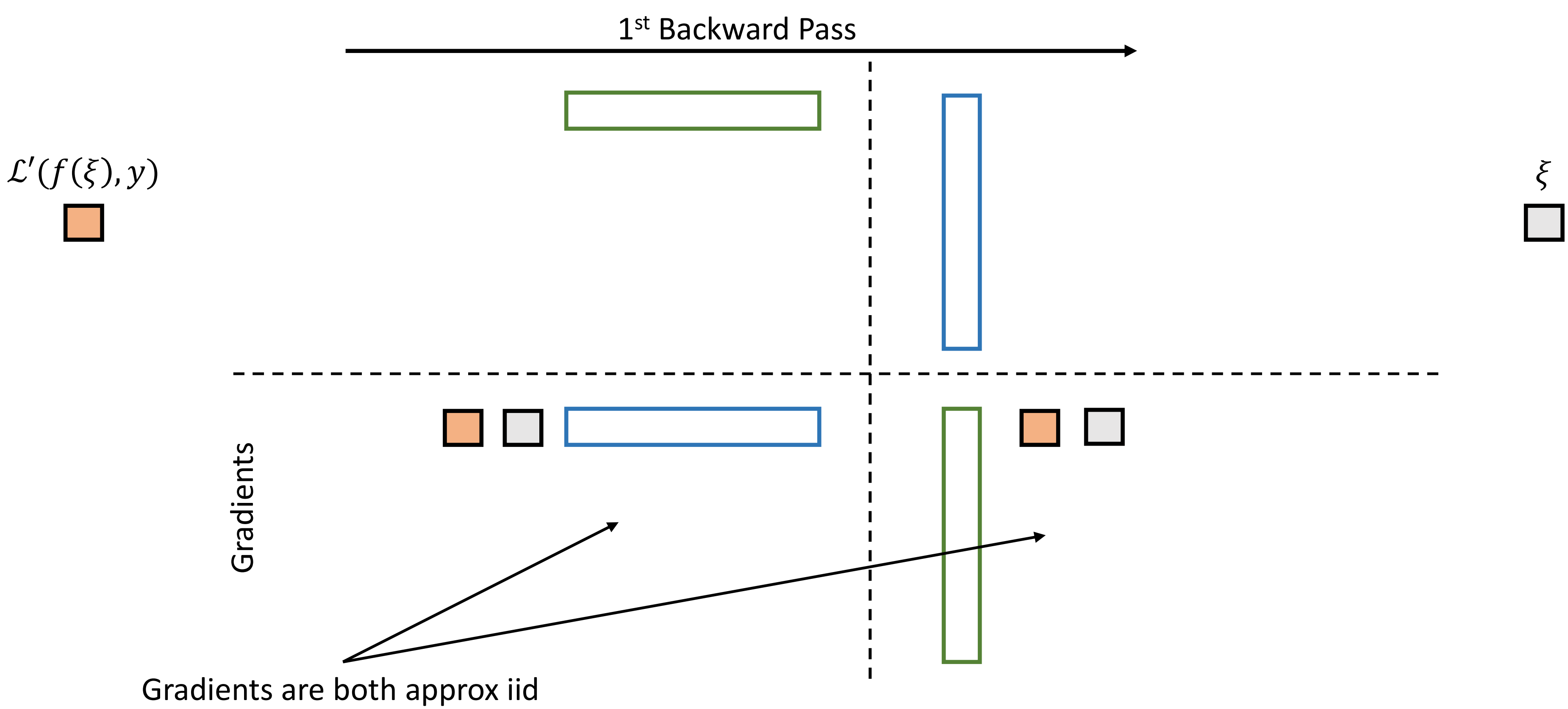


Motivating Example: Linear 1-Hidden-Layer

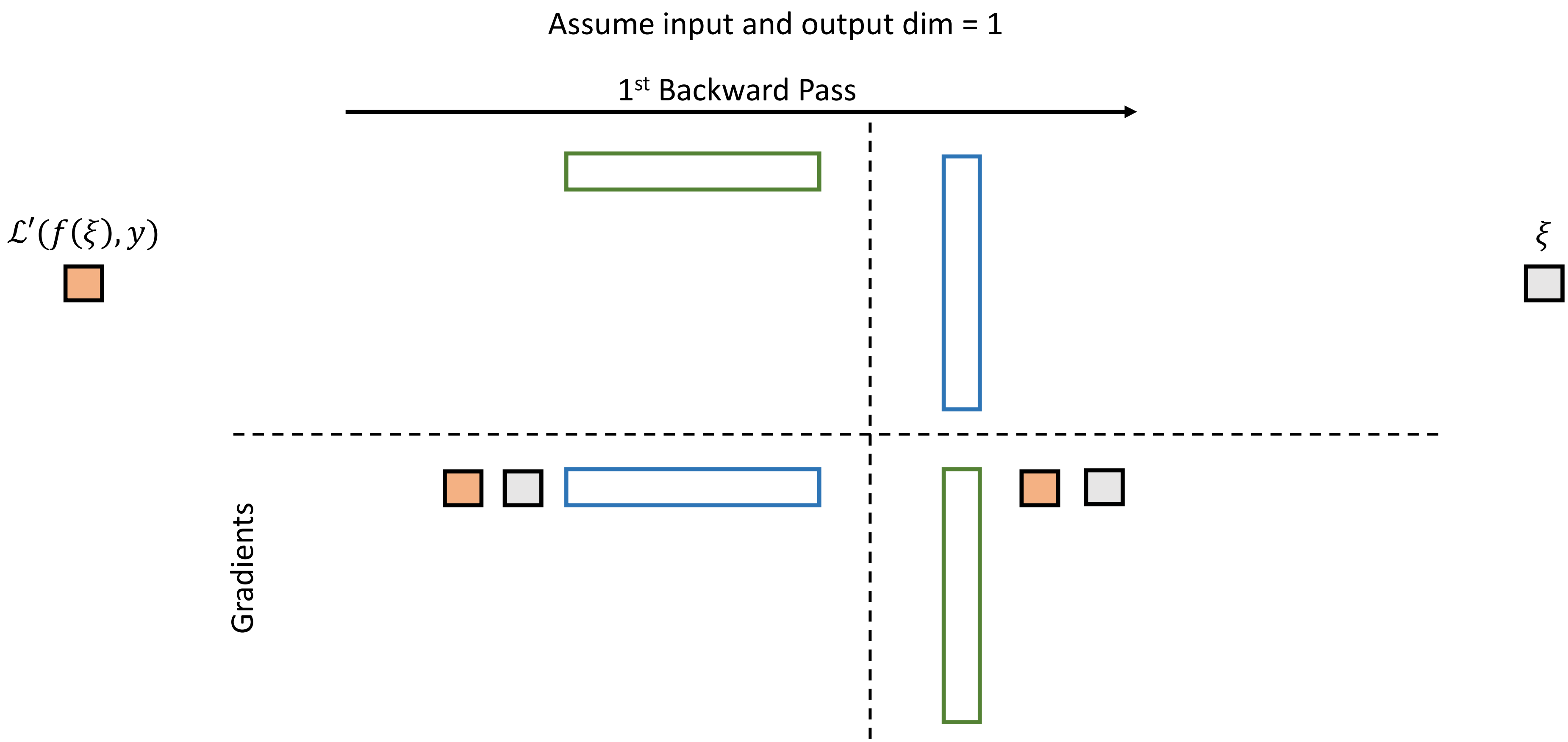


Motivating Example: Linear 1-Hidden-Layer

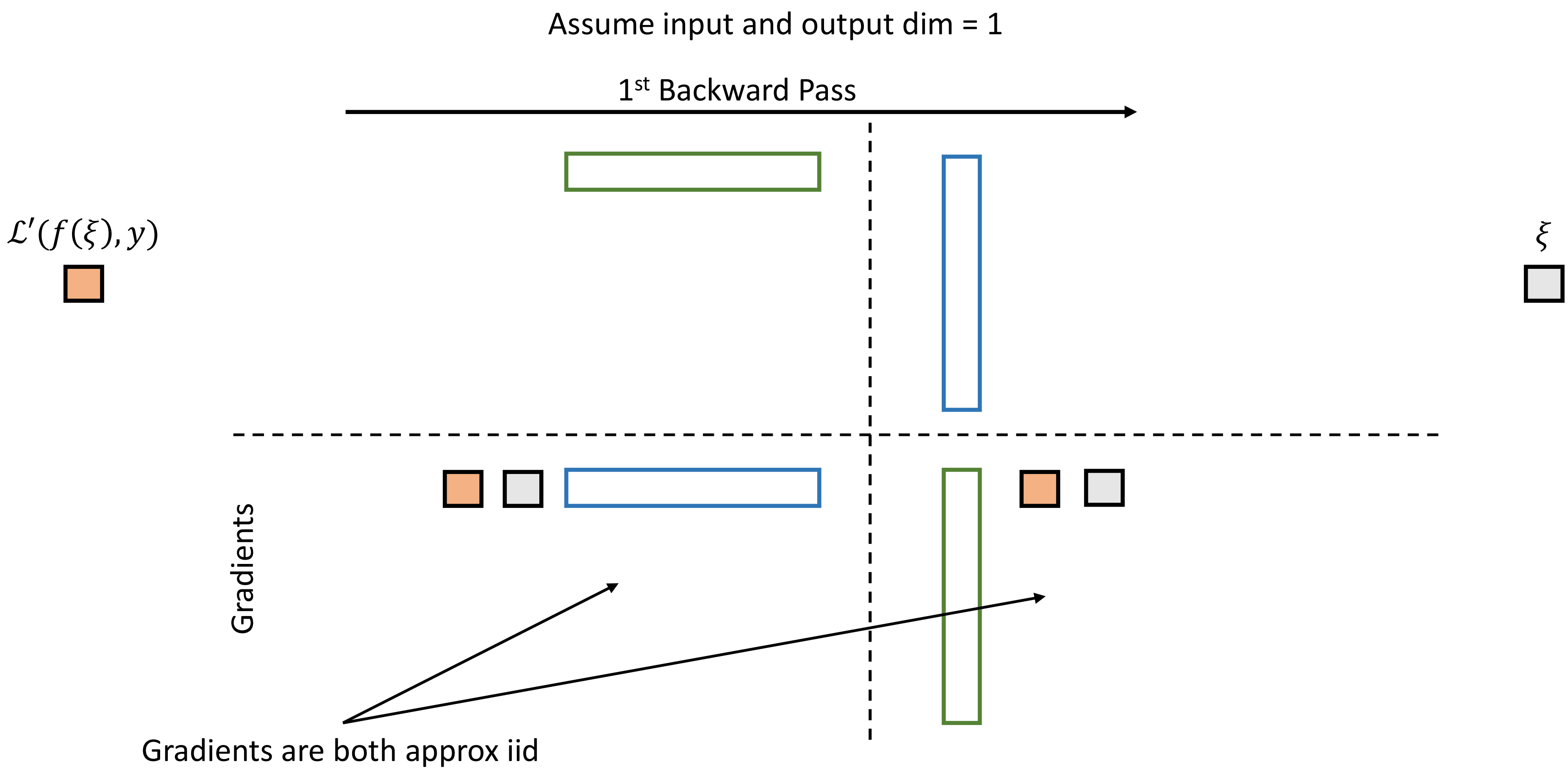
Assume input and output dim = 1



Motivating Example: Linear 1-Hidden-Layer

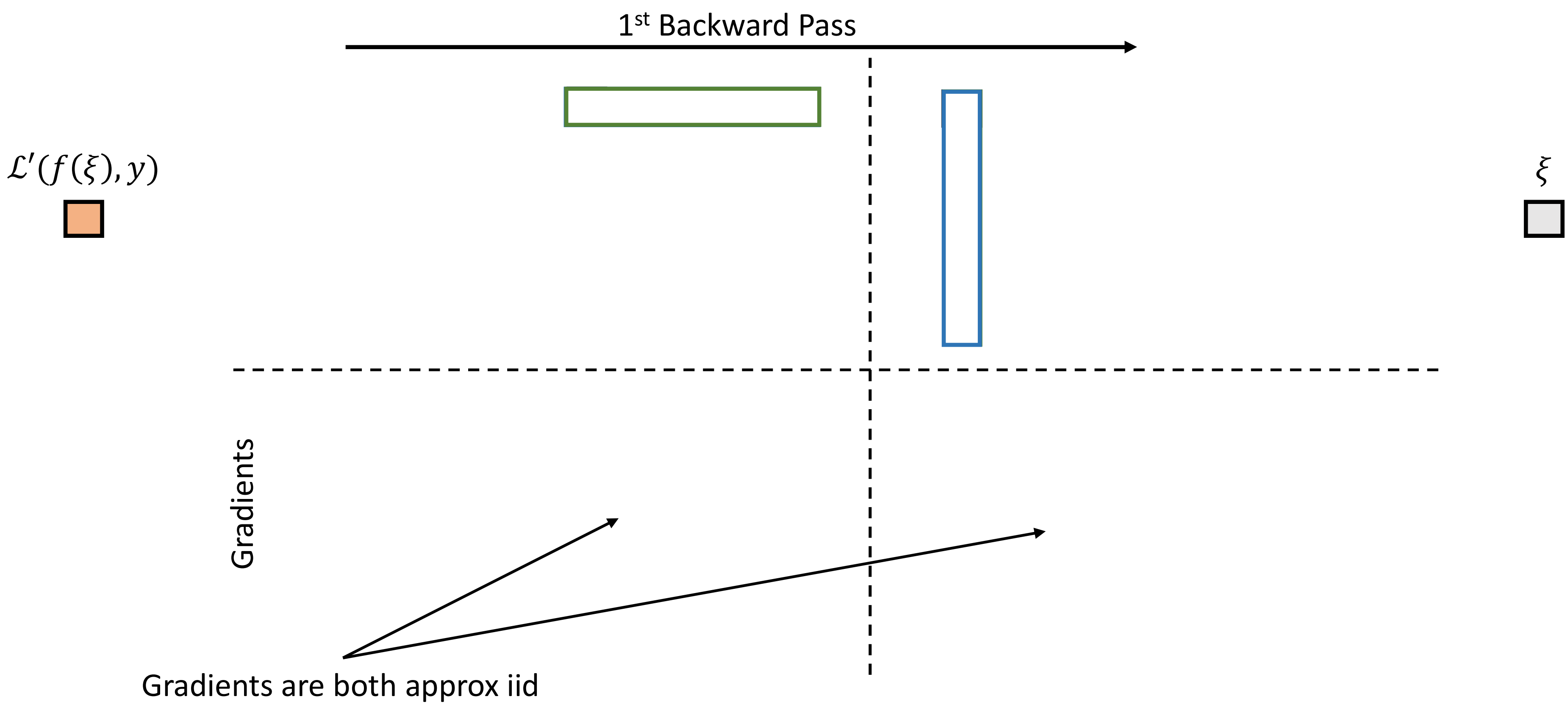


Motivating Example: Linear 1-Hidden-Layer



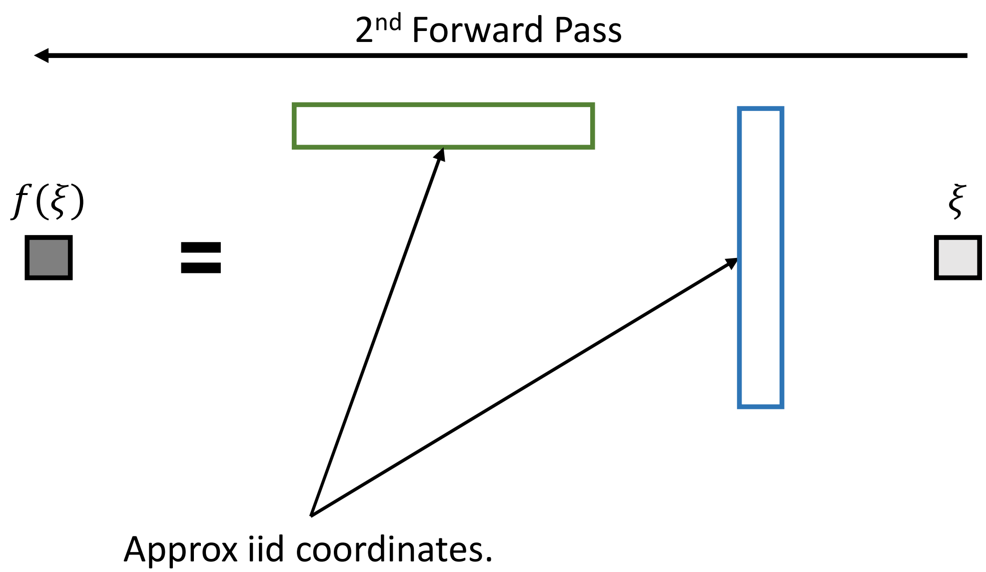
Motivating Example: Linear 1-Hidden-Layer

Assume input and output dim = 1



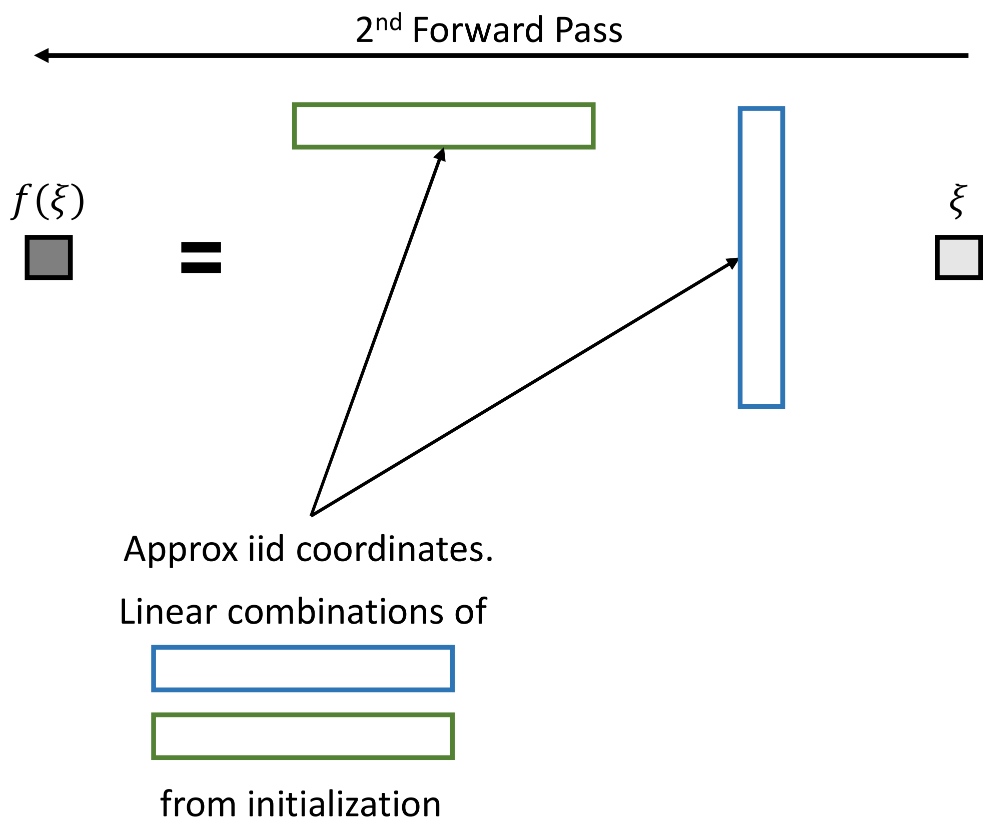
Motivating Example: Linear 1-Hidden-Layer

Assume input and output dim = 1



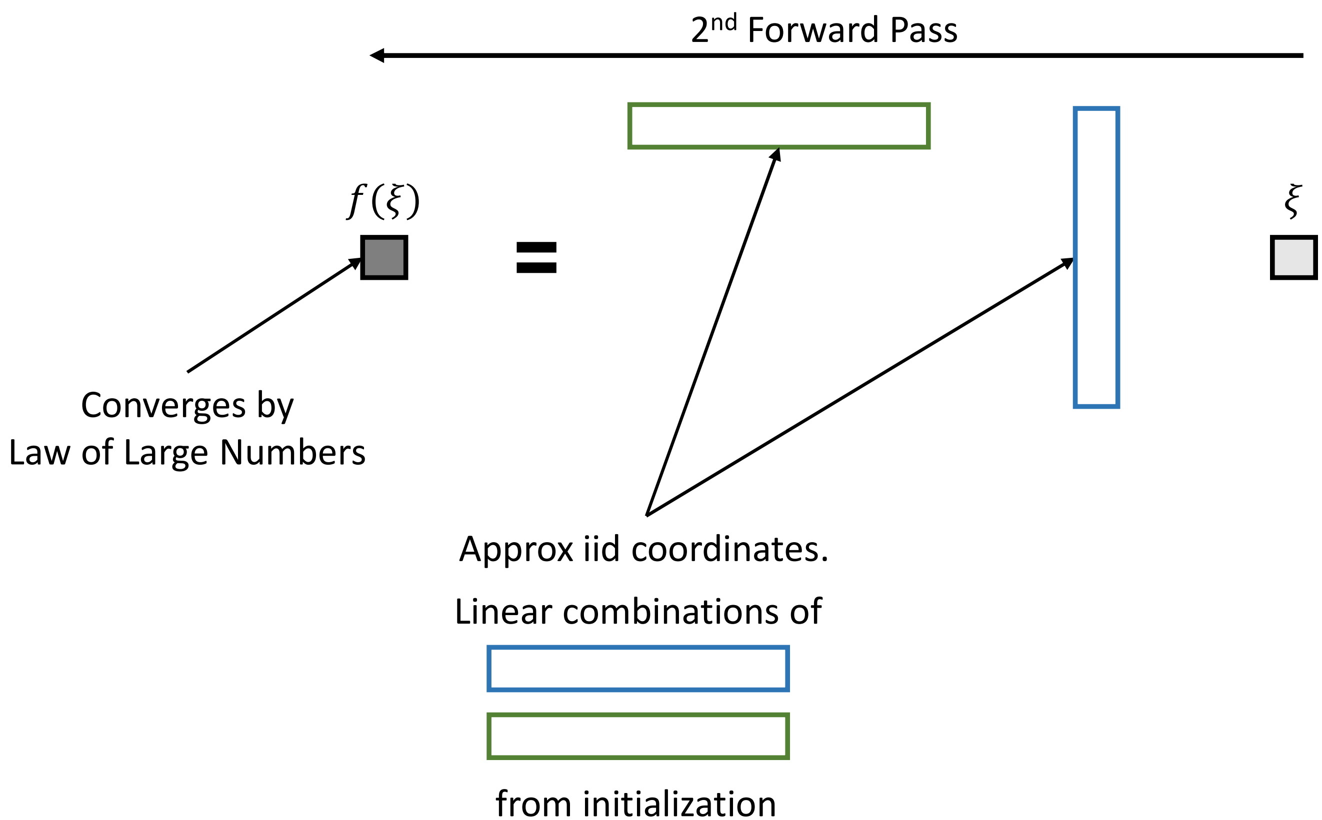
Motivating Example: Linear 1-Hidden-Layer

Assume input and output dim = 1

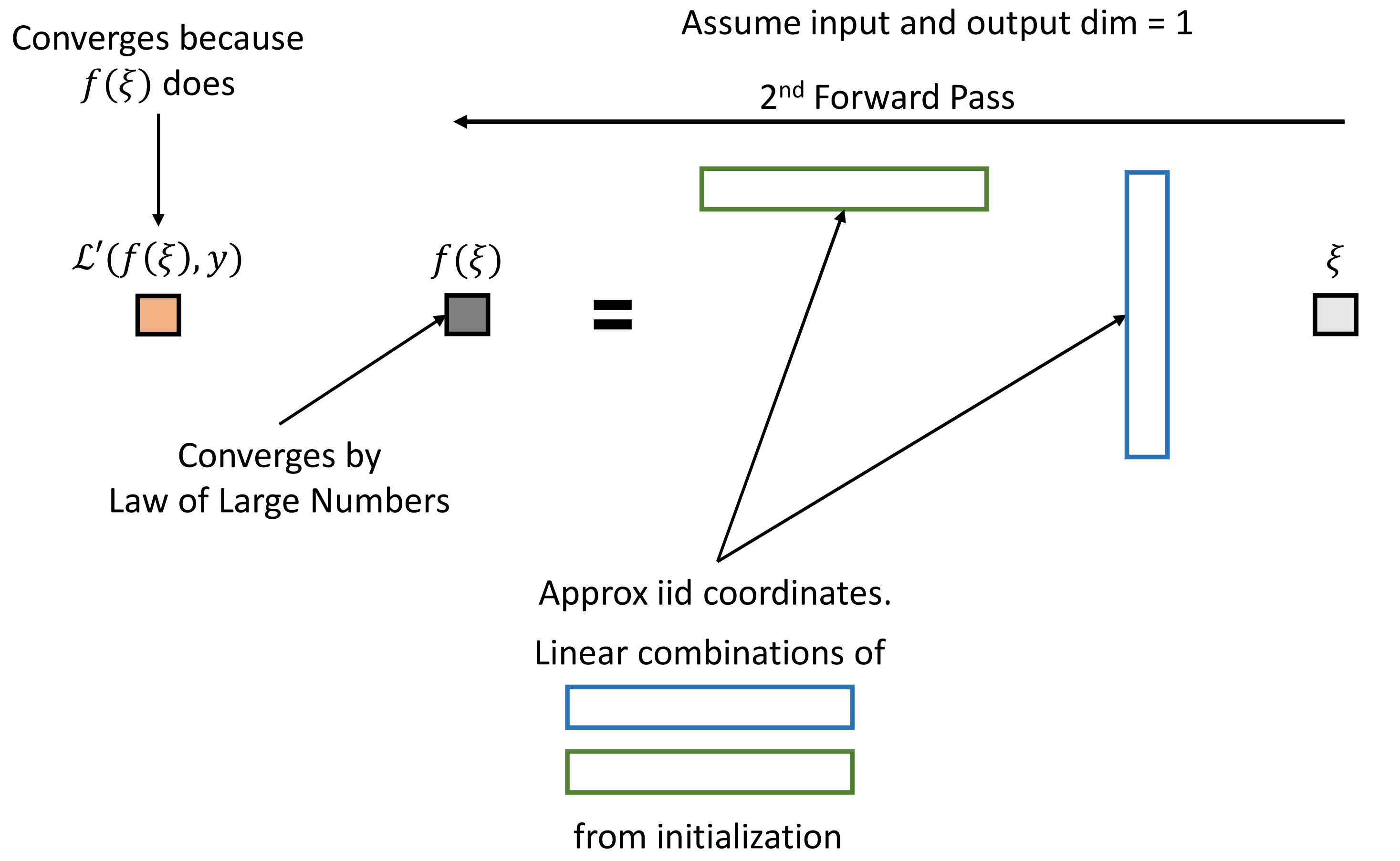


Motivating Example: Linear 1-Hidden-Layer

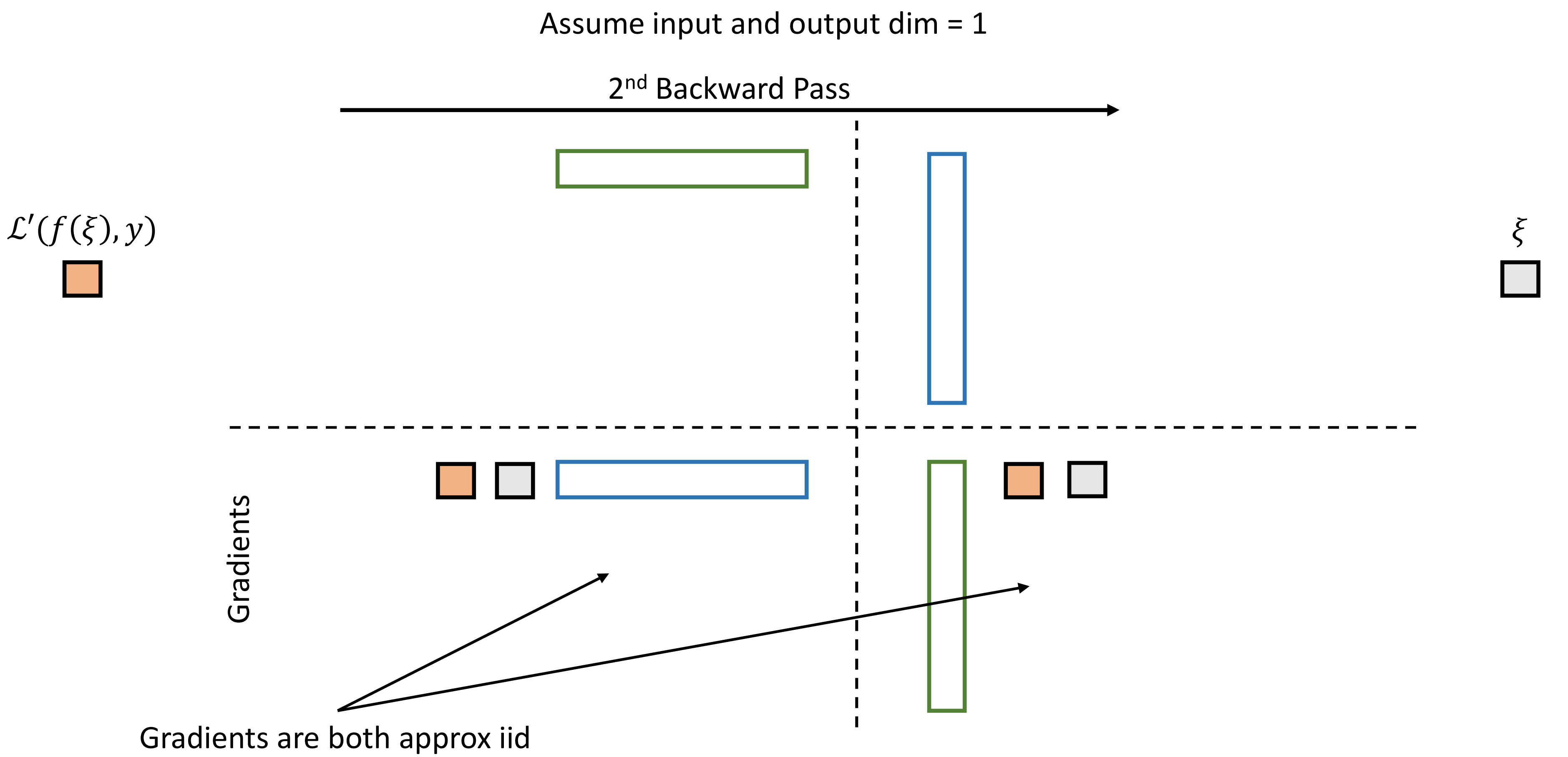
Assume input and output dim = 1



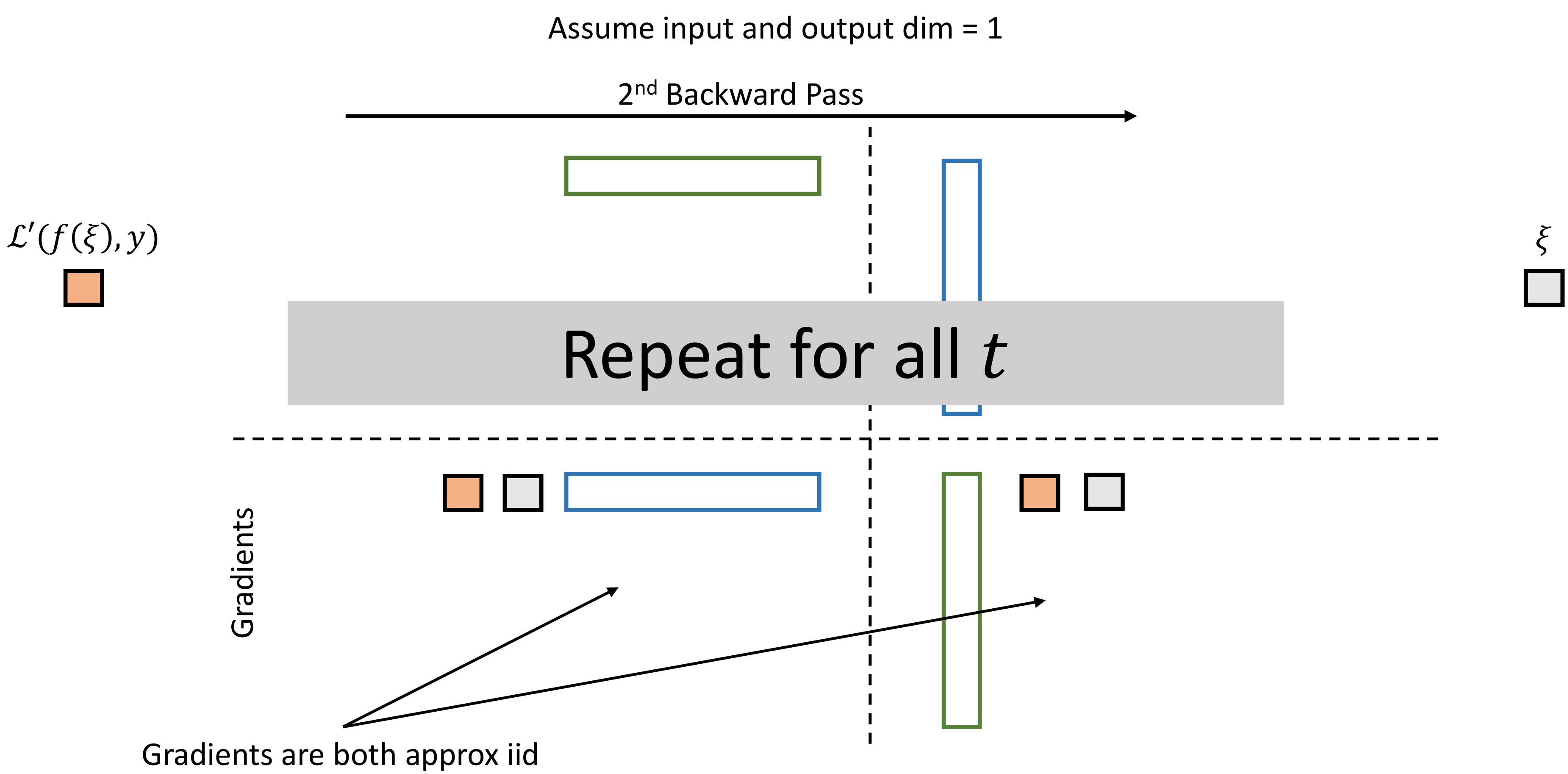
Motivating Example: Linear 1-Hidden-Layer



Motivating Example: Linear 1-Hidden-Layer

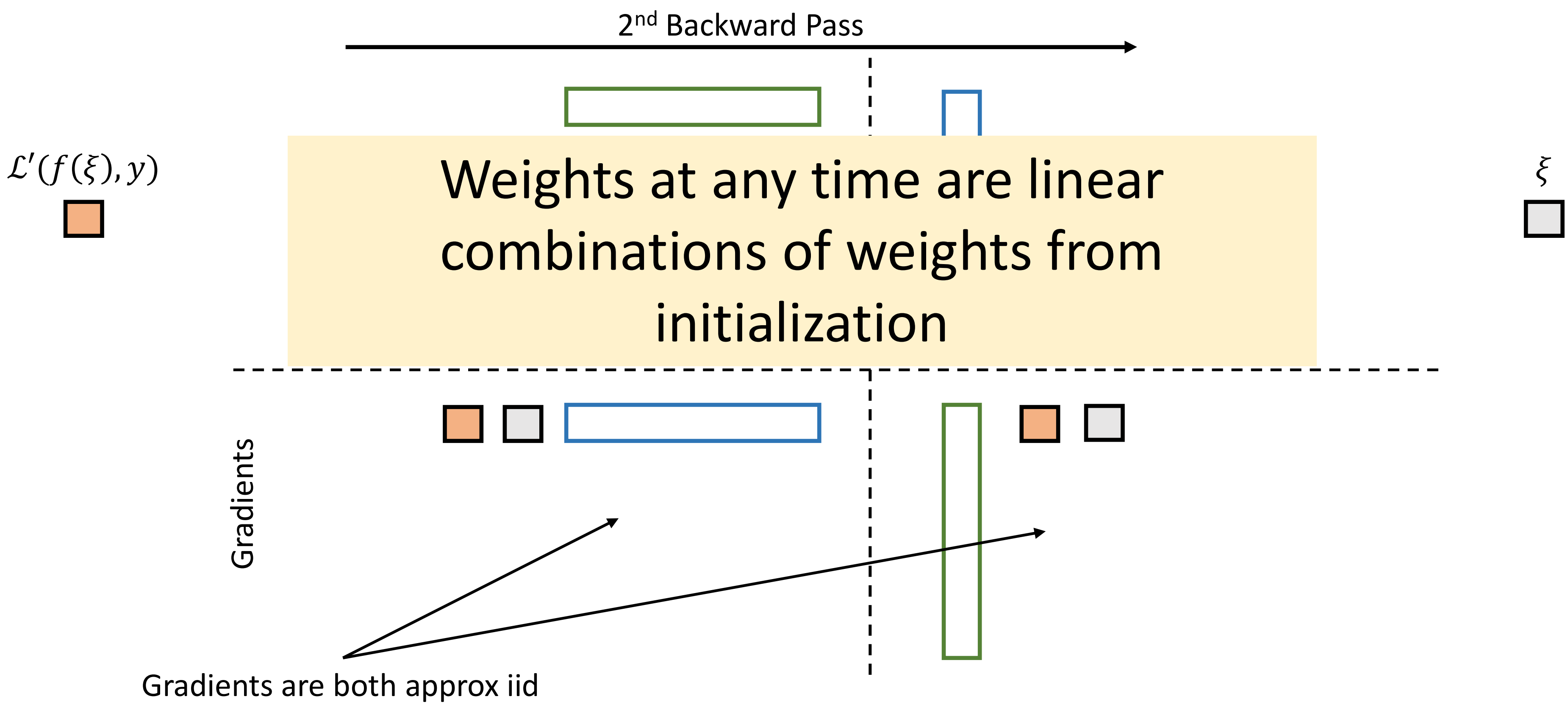


Motivating Example: Linear 1-Hidden-Layer



Motivating Example: Linear 1-Hidden-Layer

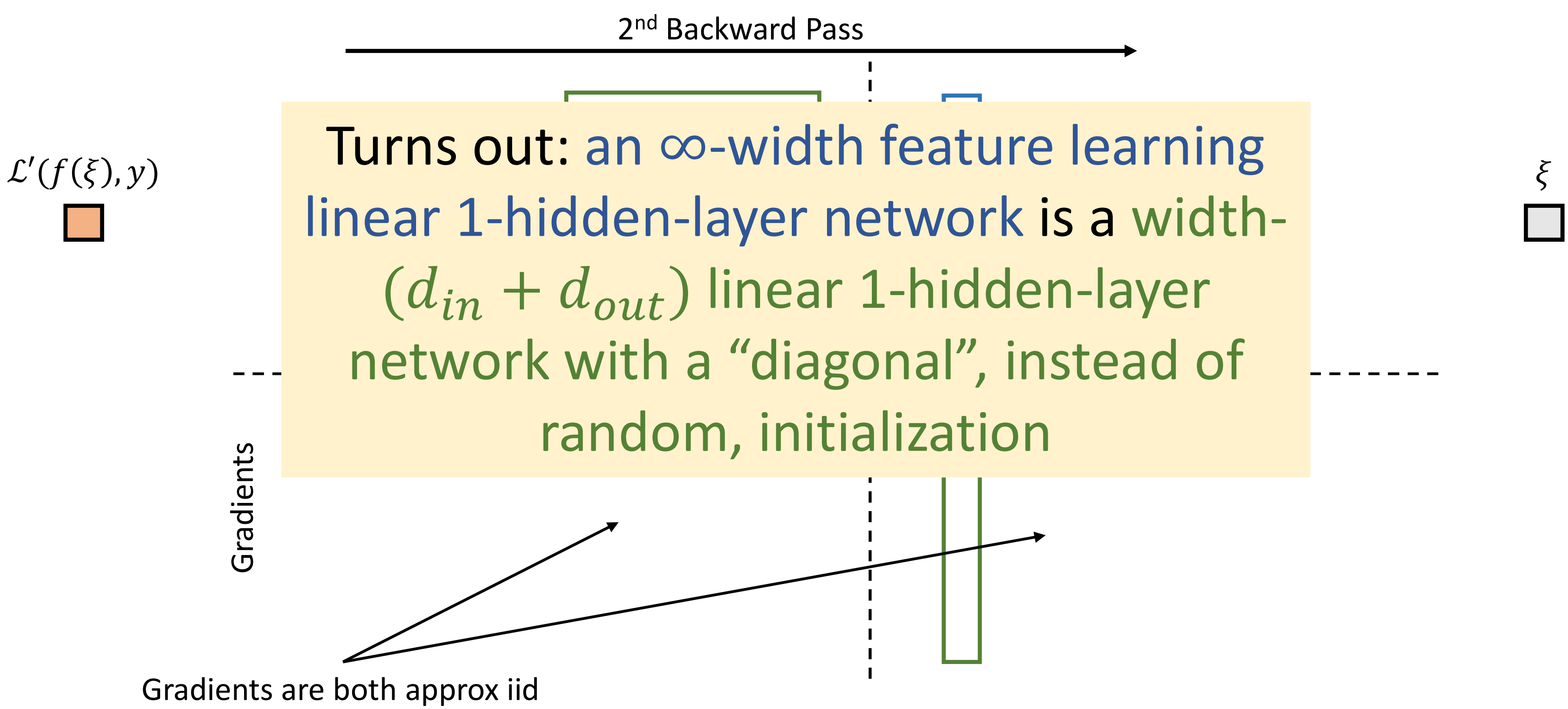
Assume input and output dim = 1



Weights at any time are linear combinations of weights from initialization

Motivating Example: Linear 1-Hidden-Layer

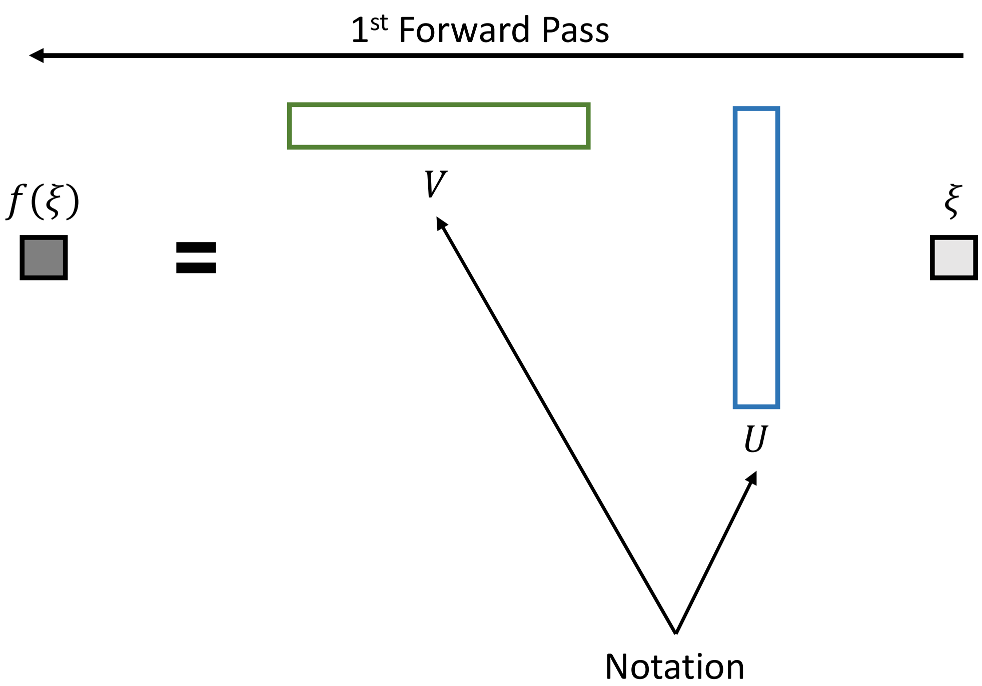
Assume input and output dim = 1



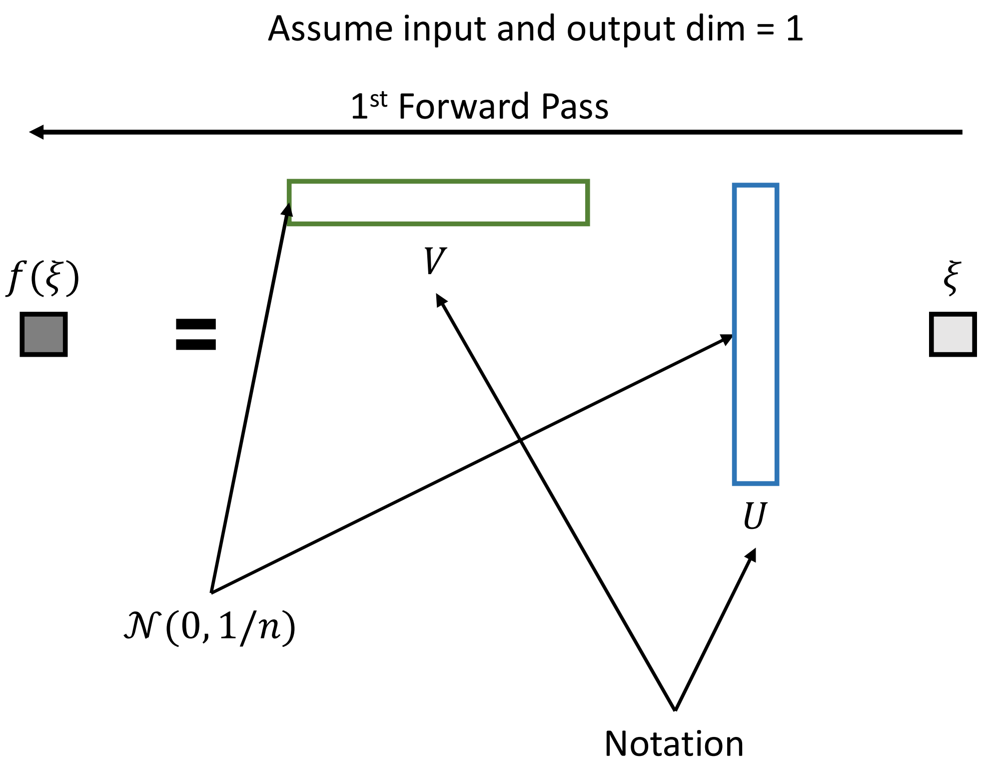
Turns out: an ∞-width feature learning linear 1-hidden-layer network is a width-$(d_{in} + d_{out})$ linear 1-hidden-layer network with a "diagonal", instead of random, initialization

Linear 1-Hidden-Layer: Taking the Limit

Assume input and output dim = 1

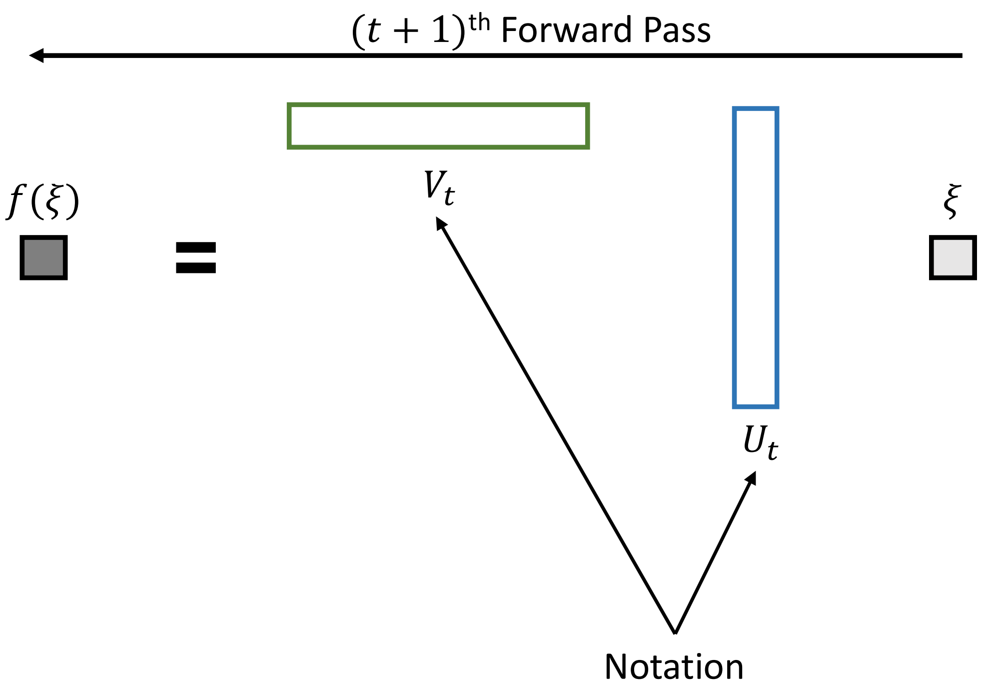


Linear 1-Hidden-Layer: Taking the Limit



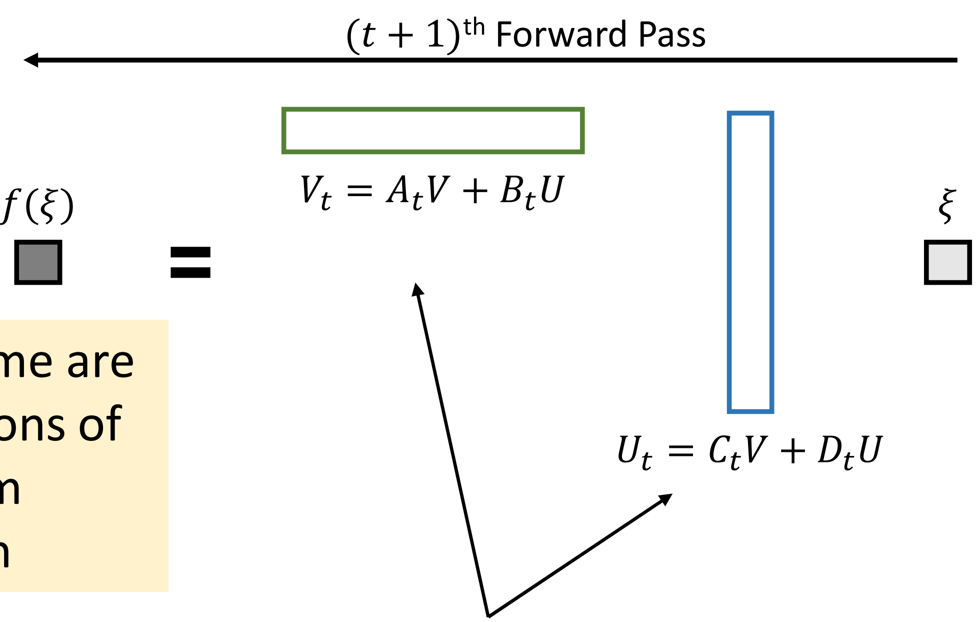
Linear 1-Hidden-Layer: Taking the Limit

Assume input and output dim = 1



Linear 1-Hidden-Layer: Taking the Limit

Assume input and output dim = 1

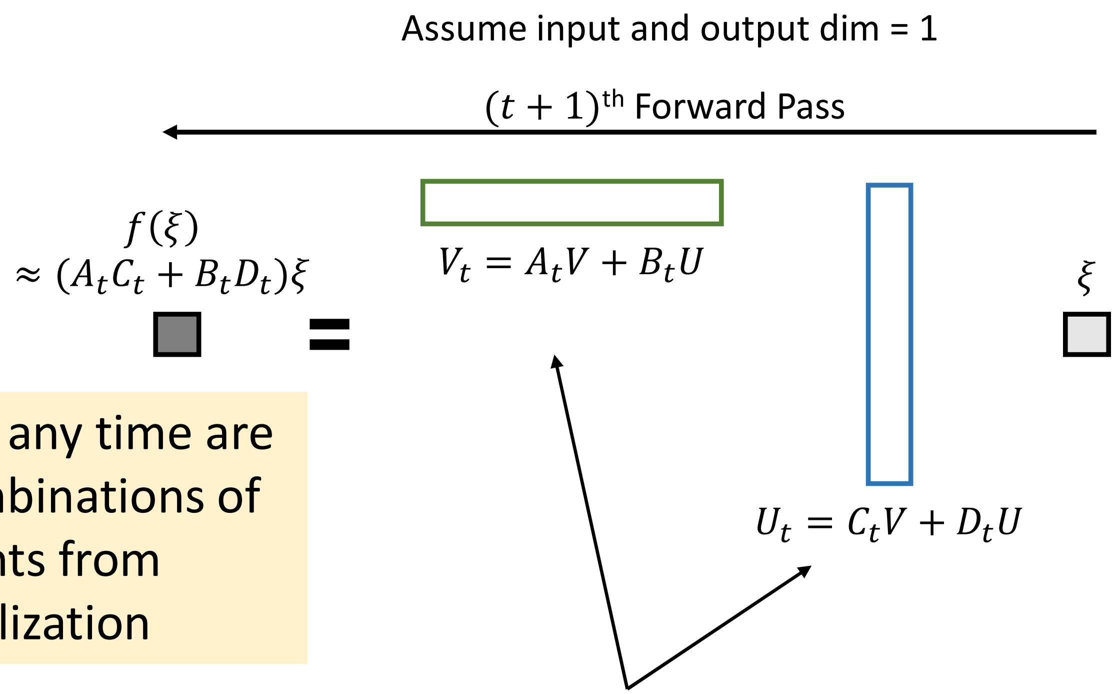


Weights at any time are linear combinations of weights from initialization

For any t, there are coefficients $A_t, B_t, C_t, D_t \in \mathbb{R}$, which converge deterministically, such that

Initial condition: for $t = 0$, $A_t = D_t = 1, B_t = C_t = 0$

Linear 1-Hidden-Layer: Taking the Limit

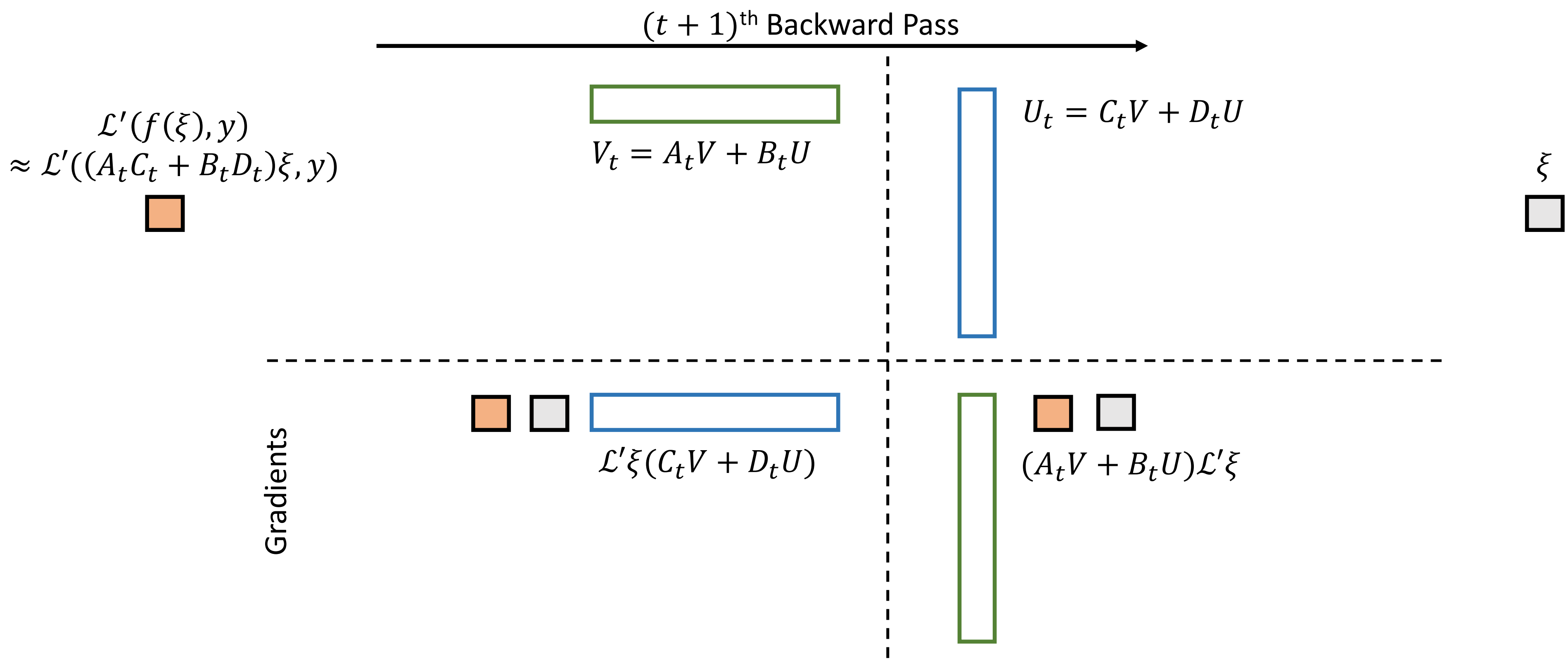


Weights at any time are linear combinations of weights from initialization

For any t, there are coefficients $A_t, B_t, C_t, D_t \in \mathbb{R}$, which converge deterministically, such that

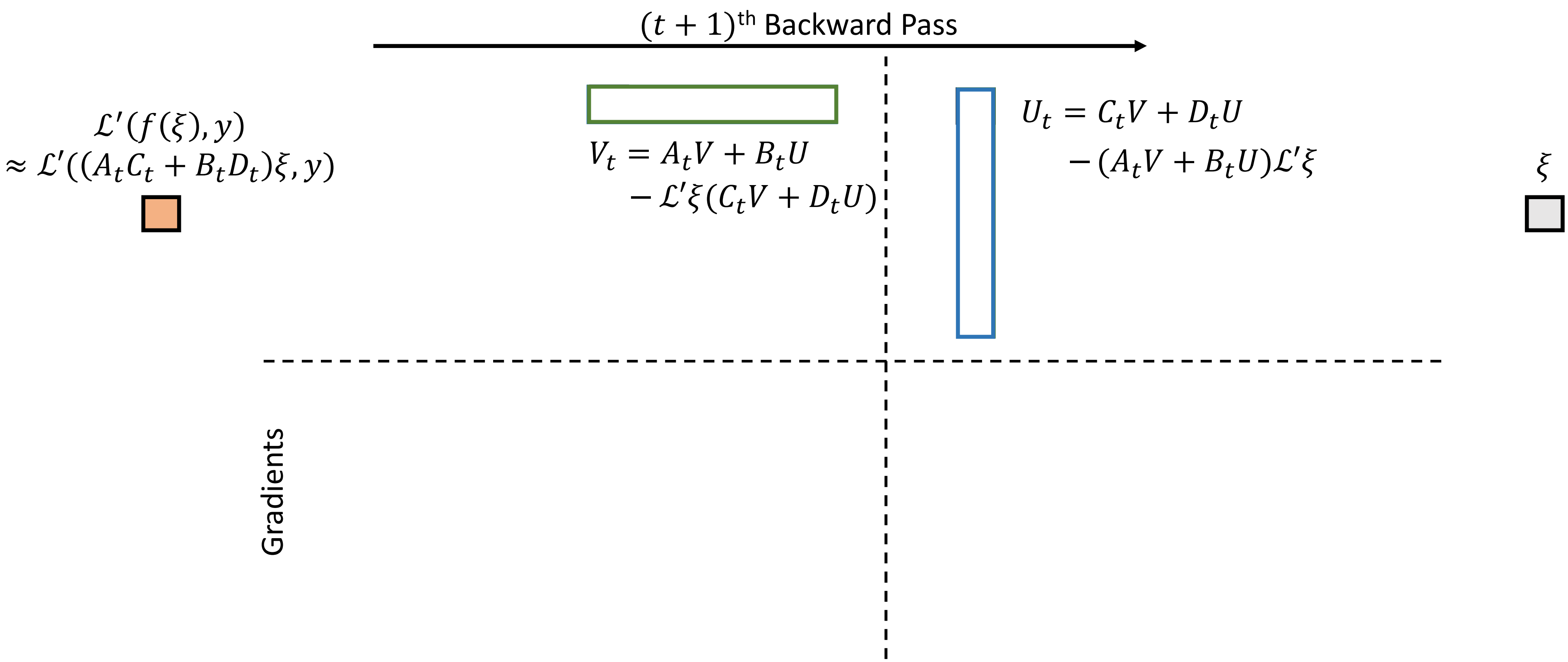
Linear 1-Hidden-Layer: Taking the Limit

Assume input and output dim = 1



Linear 1-Hidden-Layer: Taking the Limit

Assume input and output dim = 1



Linear 1-Hidden-Layer: Taking the Limit

Assume input and output dim = 1

$(t+2)^{\text{th}}$ Forward Pass

$f(\xi)$ =

$$V_{t+1} = A_{t+1}V + B_{t+1}U$$
$$A_{t+1} = A_t - \mathcal{L}'\xi C_t$$
$$B_{t+1} = B_t - \mathcal{L}'\xi D_t$$

$$U_{t+1} = C_{t+1}V + D_{t+1}U$$
$$C_{t+1} = C_t - \mathcal{L}'\xi A_t$$
$$D_{t+1} = D_t - \mathcal{L}'\xi B_t$$

ξ

Maximal Update Limit of Linear 1-Hidden-Layer NN

Suppose $d_{in} = d_{out} = 1$, *LR = 1. Then in* $n \to \infty$ *limit,*

$$f_t(\xi) = (A_t C_t + B_t D_t)\xi$$

$$(A_{t+1}, B_{t+1}) = (A_t, B_t) - \mathcal{L}'\xi(C_t, D_t)$$

$$(C_{t+1}, D_{t+1}) = (C_t, D_t) - \mathcal{L}'\xi(A_t, B_t)$$

where $\mathcal{L}' = \mathcal{L}'(f_t(\xi), y)$

with initial condition $A_0 = D_0 = 1, B_0 = C_0 = 0.$

Comparison with Mean Field Limit

- MF Limit expresses everything in terms of their PDF and is continuous-time
- This turns update equations into convolution equations and obscures their simplicity

Maximal Update Limit of Linear 1-Hidden-Layer NN

Suppose $d_{in} = d_{out} = 1$, *LR = 1. Then in* $n \to \infty$ *limit,*

$$f_t(\xi) = (A_t C_t + B_t D_t)\xi$$

$$(A_{t+1}, B_{t+1}) = (A_t, B_t) - \mathcal{L}'\xi(C_t, D_t)$$

$$(C_{t+1}, D_{t+1}) = (C_t, D_t) - \mathcal{L}'\xi(A_t, B_t)$$

where $\mathcal{L}' = \mathcal{L}'(f_t(\xi), y)$

with initial condition $A_0 = D_0 = 1, B_0 = C_0 = 0.$

Comparison with Mean Field Limit

- MF Limit expresses everything in terms of their PDF and is continuous-time
- This turns update equations into convolution equations and obscures their simplicity

Linear 1-hidden-layer width-2 NN

Maximal Update Limit of Linear 1-Hidden-Layer NN

Suppose $d_{in} = d_{out} = 1$, *LR = 1. Then in* $n \to \infty$ *limit,*

$$f_t(\xi) = (A_t C_t + B_t D_t)\xi$$

$$(A_{t+1}, B_{t+1}) = (A_t, B_t) - \mathcal{L}'\xi(C_t, D_t)$$

$$(C_{t+1}, D_{t+1}) = (C_t, D_t) - \mathcal{L}'\xi(A_t, B_t)$$

where $\mathcal{L}' = \mathcal{L}'(f_t(\xi), y)$

with initial condition $A_0 = D_0 = 1, B_0 = C_0 = 0.$

Comparison with Mean Field Limit

- MF Limit expresses everything in terms of their PDF and is continuous-time
- This turns update equations into convolution equations and obscures their simplicity

Maximal Update Limit of Linear 1-Hidden-Layer NN

Linear 1-hidden-layer width-2 NN

2nd layer weights

Suppose $d_{in} = d_{out} = 1$, *LR = 1. Then in* $n \to \infty$ *limit,*

$$f_t(\xi) = (A_t C_t + B_t D_t)\xi$$

$$(A_{t+1}, B_{t+1}) = (A_t, B_t) - \mathcal{L}'\xi(C_t, D_t)$$

$$(C_{t+1}, D_{t+1}) = (C_t, D_t) - \mathcal{L}'\xi(A_t, B_t)$$

where $\mathcal{L}' = \mathcal{L}'(f_t(\xi), y)$

with initial condition $A_0 = D_0 = 1, B_0 = C_0 = 0.$

Comparison with Mean Field Limit

- MF Limit expresses everything in terms of their PDF and is continuous-time
- This turns update equations into convolution equations and obscures their simplicity

Maximal Update Limit of Linear 1-Hidden-Layer NN

Linear 1-hidden-layer width-2 NN

2nd layer weights

1st layer weights

Suppose $d_{in} = d_{out} = 1$, *LR = 1. Then in* $n \to \infty$ *limit,*

$$f_t(\xi) = (A_t C_t + B_t D_t)\xi$$

$$(A_{t+1}, B_{t+1}) = (A_t, B_t) - \mathcal{L}'\xi(C_t, D_t)$$

$$(C_{t+1}, D_{t+1}) = (C_t, D_t) - \mathcal{L}'\xi(A_t, B_t)$$

where $\mathcal{L}' = \mathcal{L}'(f_t(\xi), y)$

with initial condition $A_0 = D_0 = 1, B_0 = C_0 = 0.$

Comparison with Mean Field Limit
- MF Limit expresses everything in terms of their PDF and is continuous-time
- This turns update equations into convolution equations and obscures their simplicity

Maximal Update Limit of Linear 1-Hidden-Layer NN

Linear 1-hidden-layer width-2 NN

2nd layer weights

1st layer weights

Suppose $d_{in} = d_{out} = 1$, *LR = 1. Then in* $n \to \infty$ *limit,*

$$f_t(\xi) = (A_t C_t + B_t D_t)\xi$$

$$(A_{t+1}, B_{t+1}) = (A_t, B_t) - \mathcal{L}'\xi(C_t, D_t)$$

$$(C_{t+1}, D_{t+1}) = (C_t, D_t) - \mathcal{L}'\xi(A_t, B_t)$$

where $\mathcal{L}' = \mathcal{L}'(f_t(\xi), y)$

with initial condition $A_0 = D_0 = 1, B_0 = C_0 = 0.$

"Diagonal" initialization: $\begin{pmatrix} A_0 & B_0 \\ C_0 & D_0 \end{pmatrix} = I$

Comparison with Mean Field Limit

- MF Limit expresses everything in terms of their PDF and is continuous-time
- This turns update equations into convolution equations and obscures their simplicity

Maximal Update Limit of Linear 1-Hidden-Layer NN with random init **=** *Width-$(d_{in} + d_{out})$ Linear 1-Hidden-Layer NN with "diagonal" init*

Maximal Update Limit of Linear 1-Hidden-Layer NN

Suppose $d_{in} = d_{out} = 1$, LR = 1. Then in $n \to \infty$ limit,

$$f_t(\xi) = (A_t C_t + B_t D_t)\xi$$

$$(A_{t+1}, B_{t+1}) = (A_t, B_t) - \mathcal{L}'\xi(C_t, D_t)$$

$$(C_{t+1}, D_{t+1}) = (C_t, D_t) - \mathcal{L}'\xi(A_t, B_t)$$

where $\mathcal{L}' = \mathcal{L}'(f_t(\xi), y)$

with initial condition $A_0 = D_0 = 1, B_0 = C_0 = 0$.

Linear 1-hidden-layer width-2 NN

2nd layer weights

1st layer weights

"Diagonal" initialization: $\begin{pmatrix} A_0 & B_0 \\ C_0 & D_0 \end{pmatrix} = I$

Maximal Update Limit of Linear 1-Hidden-Layer NN with random init

=

Width-$(d_{in} + d_{out})$ Linear 1-Hidden-Layer NN with "diagonal" init

This is what we compute for large-scale experiments like Word2Vec

$d_{in} = d_{out} = \text{vocab size}$

$d_{in} + d_{out} = 140k$ on `text8`

$d_{in} + d_{out} = 280k$ on `fil9`

1-Hidden-Layer: Summary

- The weight matrices have iid coordinates at initialization
- The function output converges due to Law of Large Numbers
- Gradients have approx. iid coordinates
- So after gradient update, weight coordinates are still approx. iid
- Repeat
- In the linear case, we express weights at any time as linear combinations of weights from initialization
  - This allows us to have efficient calculation of limit

L-Hidden-Layer: An Appetizer

- $n \times n$ Gaussian random matrix W in the middle of network
  - Central limit behavior
    - Wx is a Gaussian vector if x independent of W
  - Correlation with $W^\top$
    - Appears after 1 step of SGD
      - no effect in 1st step due to Gradient Independence Phenomenon
- See paper for details of the L-hidden-layer limit

The Tensor Program Framework
Automates All of These Derivations

Outline of This Talk

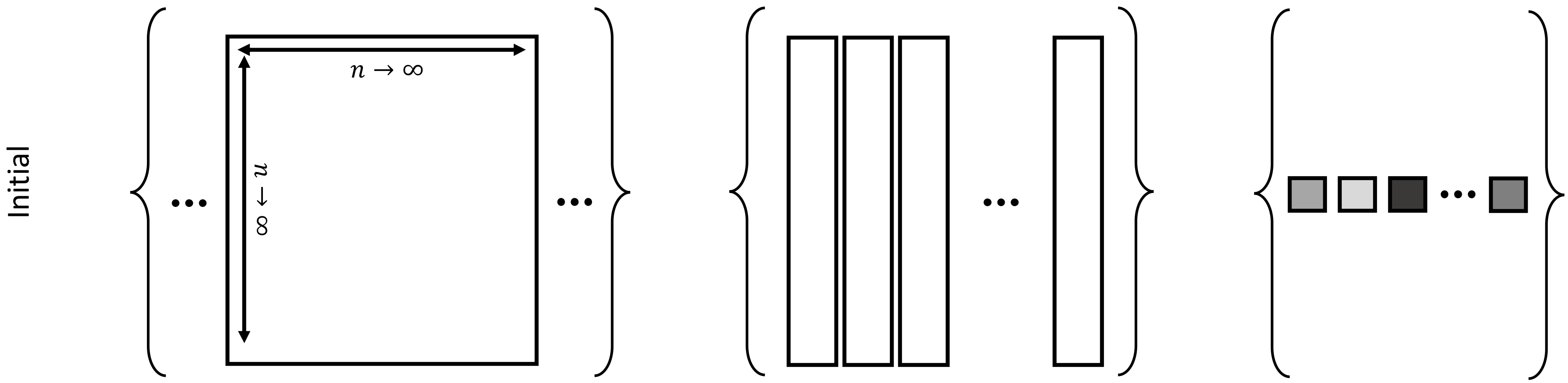
- Parametrizations of Neural Networks
- Dichotomy of Parametrizations
- The “Maximal” Feature Learning Limit
- The *Tensor Programs* technique for deriving the limit
  - Key Idea
  - Motivating Example: Linear 1-Hidden-Layer NN
  - What is a Tensor Program?
  - Master Theorem
  - Infinite-Width Limit for All

Outline of This Talk

- Parametrizations of Neural Networks
- Dichotomy of Parametrizations
- The "Maximal" Feature Learning Limit
- The *Tensor Programs* technique for deriving the limit
  - Key Idea
  - Motivating Example: Linear 1-Hidden-Layer NN
  - What is a Tensor Program?
  - Master Theorem
  - Infinite-Width Limit for All

What is a Tensor Program?

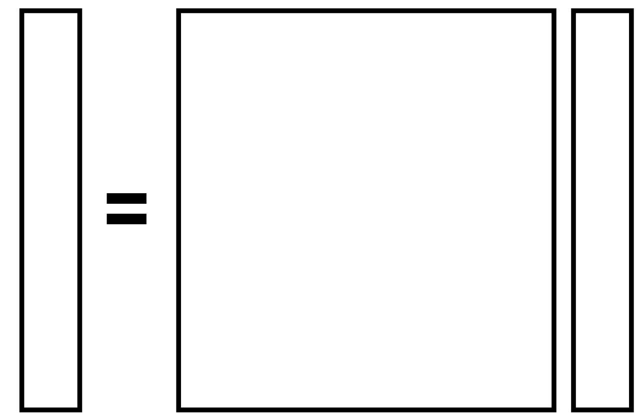
- A set of inductively generated vectors and scalars, given an initial set of matrices, vectors, and scalars



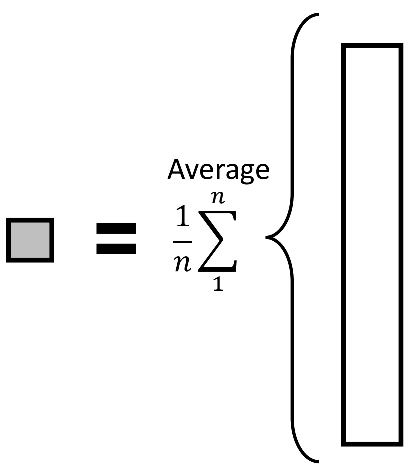
What is a Tensor Program?

- A set of inductively generated vectors and scalars, given an initial set of matrices, vectors, and scalars
- Generation rules

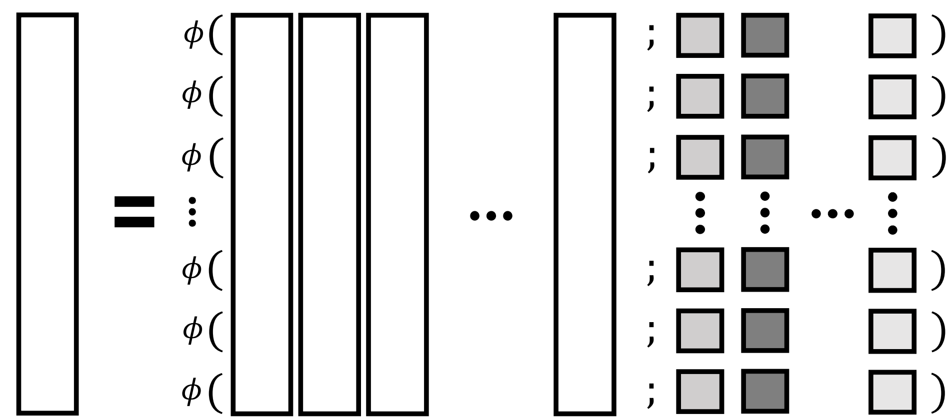
MatMul



Moment

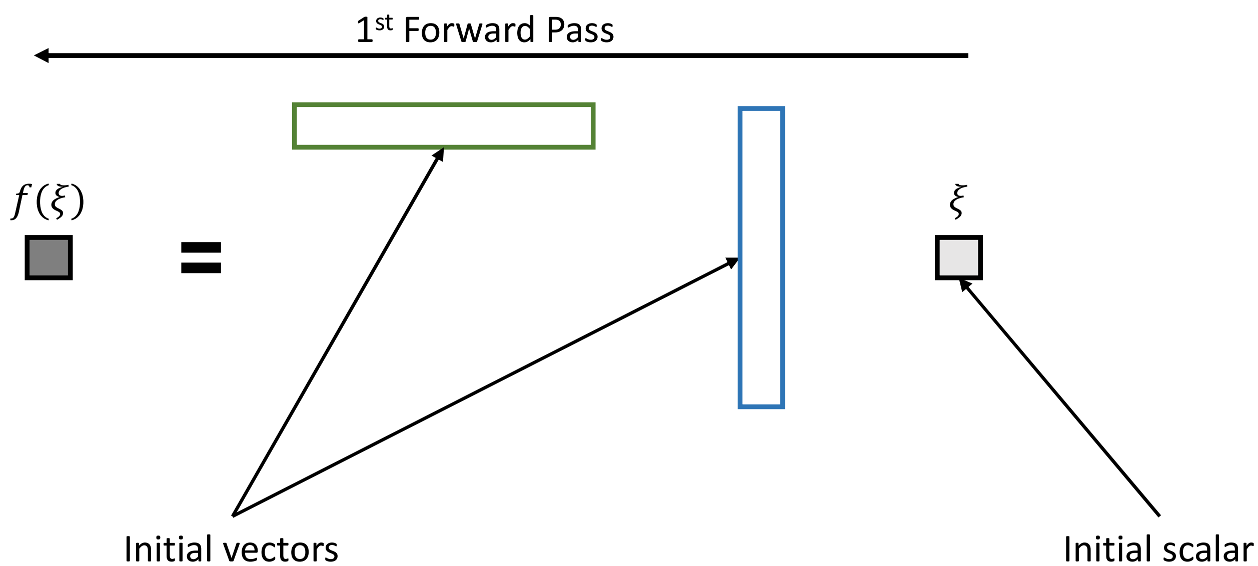


Nonlin



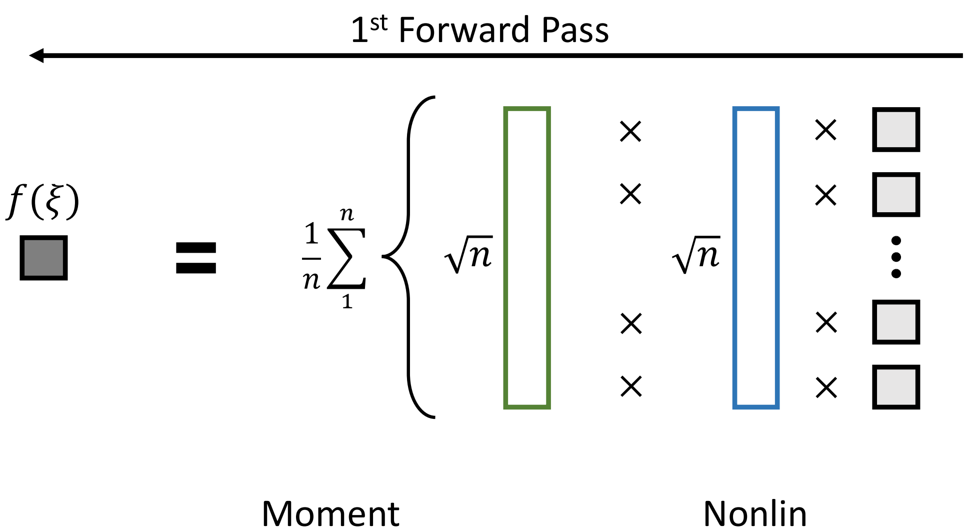
Linear 1-Hidden-Layer as a Tensor Program

Assume input and output dim = 1



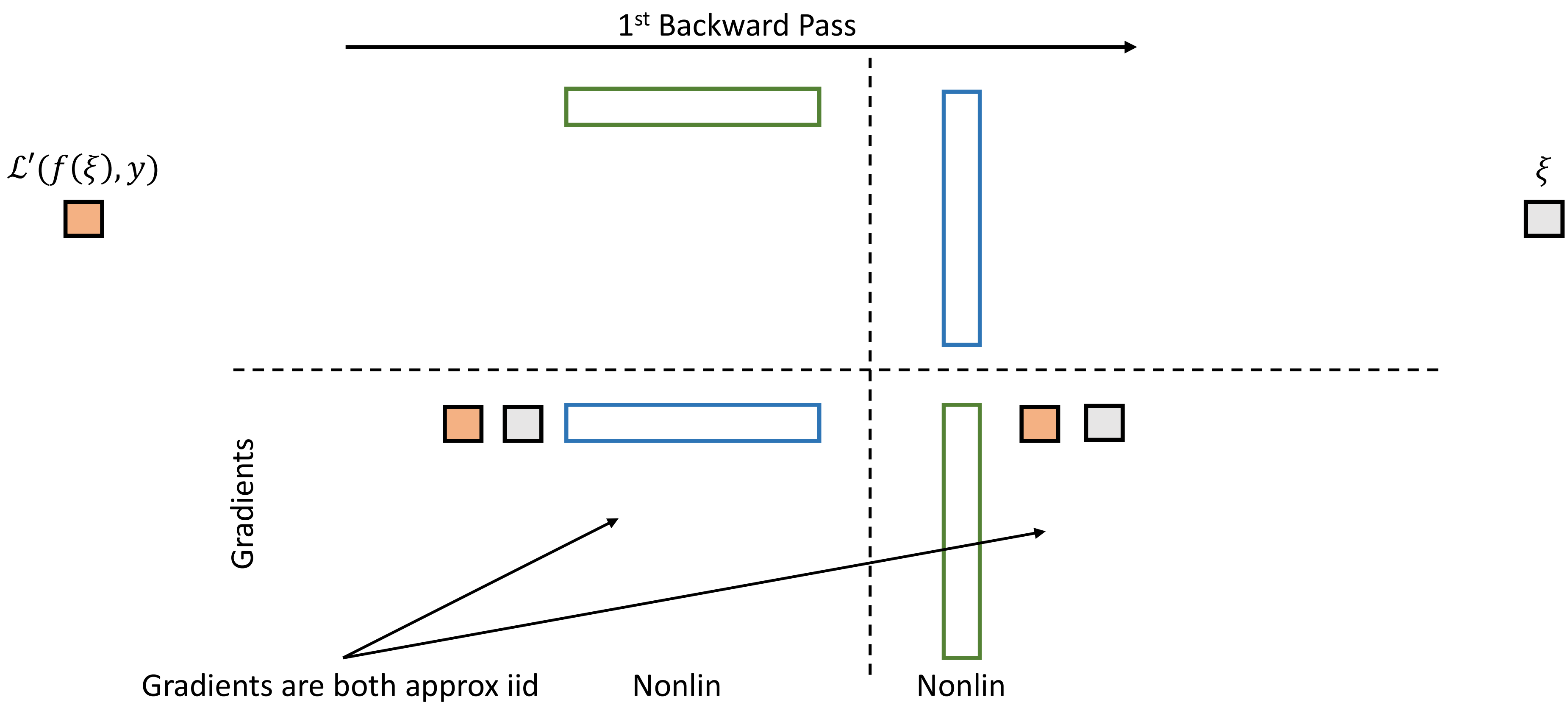
Linear 1-Hidden-Layer as a Tensor Program

Assume input and output dim = 1

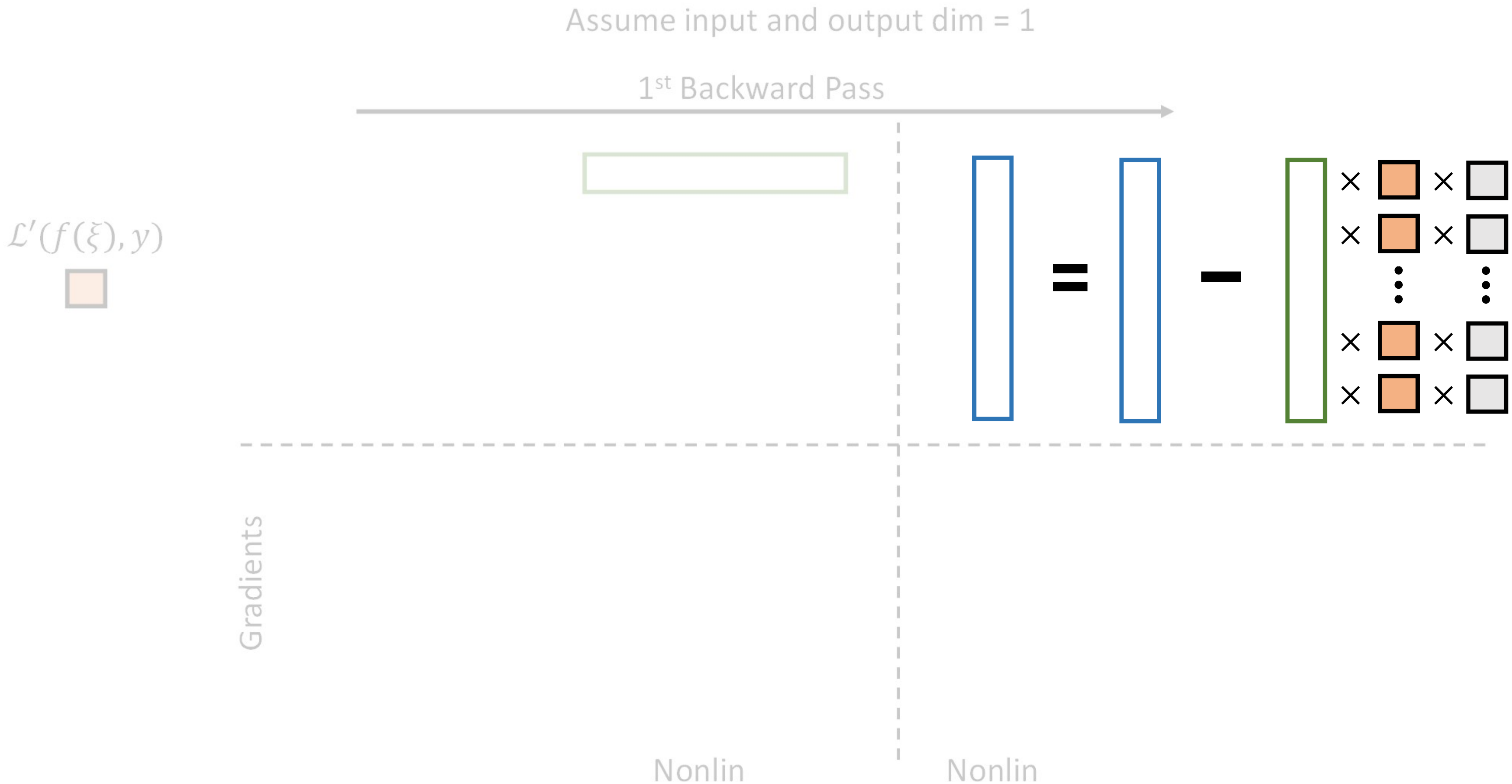


Linear 1-Hidden-Layer as a Tensor Program

Assume input and output dim = 1



Linear 1-Hidden-Layer as a Tensor Program



SGD as a Tensor Program

- 1-hidden-layer
  - Only need to use Nonlin and Moment
- L-hidden-layer
  - Need to use MatMul because of $n \times n$ Gaussian matrix in the middle of network

Outline of This Talk

- Parametrizations of Neural Networks
- Dichotomy of Parametrizations
- The “Maximal” Feature Learning Limit
- The *Tensor Programs* technique for deriving the limit
  - Key Idea
  - Motivating Example: Linear 1-Hidden-Layer NN
  - What is a Tensor Program?
  - Master Theorem
  - Infinite-Width Limit for All

Outline of This Talk

- Parametrizations of Neural Networks
- Dichotomy of Parametrizations
- The “Maximal” Feature Learning Limit
- The *Tensor Programs* technique for deriving the limit
  - Key Idea
  - Motivating Example: Linear 1-Hidden-Layer NN
  - What is a Tensor Program?
  - Master Theorem
  - Infinite-Width Limit for All

Master Theorem

Tells you how a Tensor Program behaves in the $n \to \infty$ *limit.*

Master Theorem

If 1) initial scalars converge deterministically and 2) initial matrices & vectors are sampled as iid Gaussians, then

- all vectors generated in the program have iid coordinates in the $n \to \infty$ limit, and there are rules to calculate such limit distributions.
- all scalars generated in the program converge to deterministic values, and there are rules to calculate such limit scalars.

Embedding $x^l(\xi)$ of input ξ of trained network

Output (i.e. logits) of trained network

Outline of This Talk

- Parametrizations of Neural Networks
- Dichotomy of Parametrizations
- The "Maximal" Feature Learning Limit
- The *Tensor Programs* technique for deriving the limit
    - Key Idea
    - Motivating Example: Linear 1-Hidden-Layer NN
    - What is a Tensor Program?
    - Master Theorem
    - Infinite-Width Limit for All

Outline of This Talk

- Parametrizations of Neural Networks
- Dichotomy of Parametrizations
- The "Maximal" Feature Learning Limit
- The *Tensor Programs* technique for deriving the limit
  - Key Idea
  - Motivating Example: Linear 1-Hidden-Layer NN
  - What is a Tensor Program?
  - Master Theorem
  - Infinite-Width Limit for All

Church-Turing Thesis for Deep Learning

Any “useful” deep learning computation can be expressed as a Tensor Program.

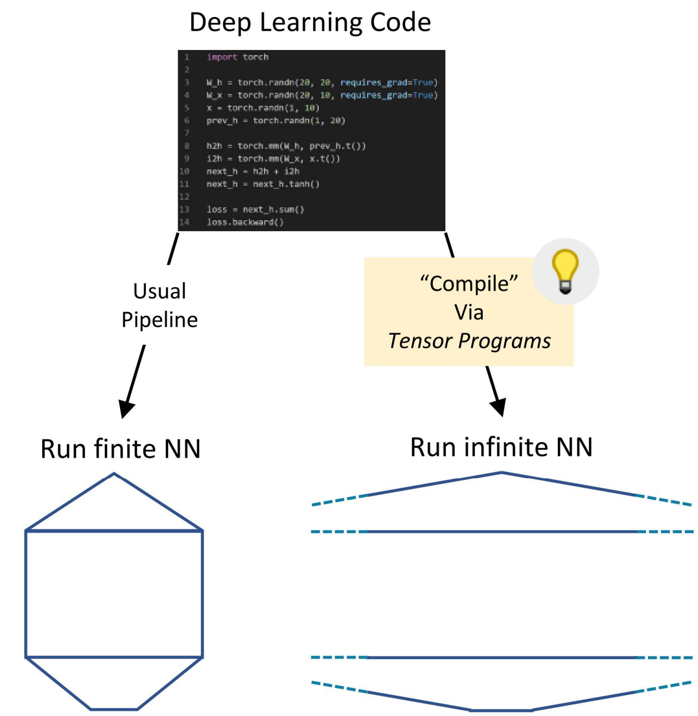
Consequence: Infinite-Width Limits for All

- SOTA architectures: ResNet, Transformers, etc
- SGD with momentum, weight decay
- Adam, Adadelta, Adafactor, etc
- Natural Gradient Descent
- Pretraining and Finetuning
- Metalearning (e.g. MAML)
- Deep Reinforcement Learning (DQN, DDPG, etc)
- Image Generation (GANs, VAEs, etc)

Tensor Programs "Compile" Finite-Width Computation to Infinite-Width Computation

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.



Outline of This Talk

- Parametrizations of Neural Networks
- Dichotomy of Parametrizations
- The “Maximal” Feature Learning Limit
- The *Tensor Programs* technique for deriving the limit
  - Key Idea
  - Motivating Example: Linear 1-Hidden-Layer NN
  - What is a Tensor Program?
  - Master Theorem
  - Infinite-Width Limit for All

Summary

Our Contributions

- Classify all abc-parametrizations and their ∞-width limits
- Identify Maximal Update Parametrization for maximizing feature learning
- Propose the *Tensor Programs* technique for deriving its equations, and more generally, the limit of any neural computation

Significance

- Framework for studying feature learning in overparametrized NN
- Formulas for training feature-learning ∞-width NN in variety of settings (e.g. pretraining, metalearning, reinforcement learning, GANs, etc)
- Mostly solves the problem of taking ∞-width limits

Looking Ahead

- What kinds of representations are learned?
- How does it inform us about finite neural networks?
- How does this feature learning affect training and generalization?
- How does this jibe with the scaling law of language models?
- Can we train an infinite-width GPT…so GPT∞?
- ... and so many more questions are now ripe for the picking!

Thank you! Question?

Scan to link to paper