How to Start Training: The Effect of Initialization and Architecture

Boris Hanin[1], David Rolnick[2]

[1]Texas A&M University, [2]Massachusetts Institute of Technology

Motivation

Neural networks may underperform due to local optima, saddle points, overfitting, etc. More fatally, **learning may not start at all.**

Goals: (1) to ensure that networks at least get above random chance.
(2) to develop simple principles for architecture and initialization.

We identify two simple reasons learning fails and **prove how to avoid them**.

Summary for Mathematicians

We study ReLU networks at initialization, with randomly initialized weights. Consider the **mean** and **variance** of the length of each layer's activation vector.

FC & ConvNets: Mean is **exponential** in **variance** of i.i.d. **weights**.
Variance is **exponential** in **sum of reciprocals** of **layer widths.**

ResNets: Mean and variance are both **exponential** in **residual block weights**.

Summary for Engineers

Poor **initialization** and poor **architecture** both stop networks from learning.

Initialization: Use i.i.d. weights with variance **2/fan-in** (e.g. He normal).
Watch out for truncated normals!

Architecture: **Width** (or **#features** in ConvNets) should grow with **depth**.
Even a single narrow layer makes training hard.

ResNets avoid the architecture issues, but **residual blocks** should be **weighted**.

Effect of i.i.d. Weight Variance on Activation Length and Early Training

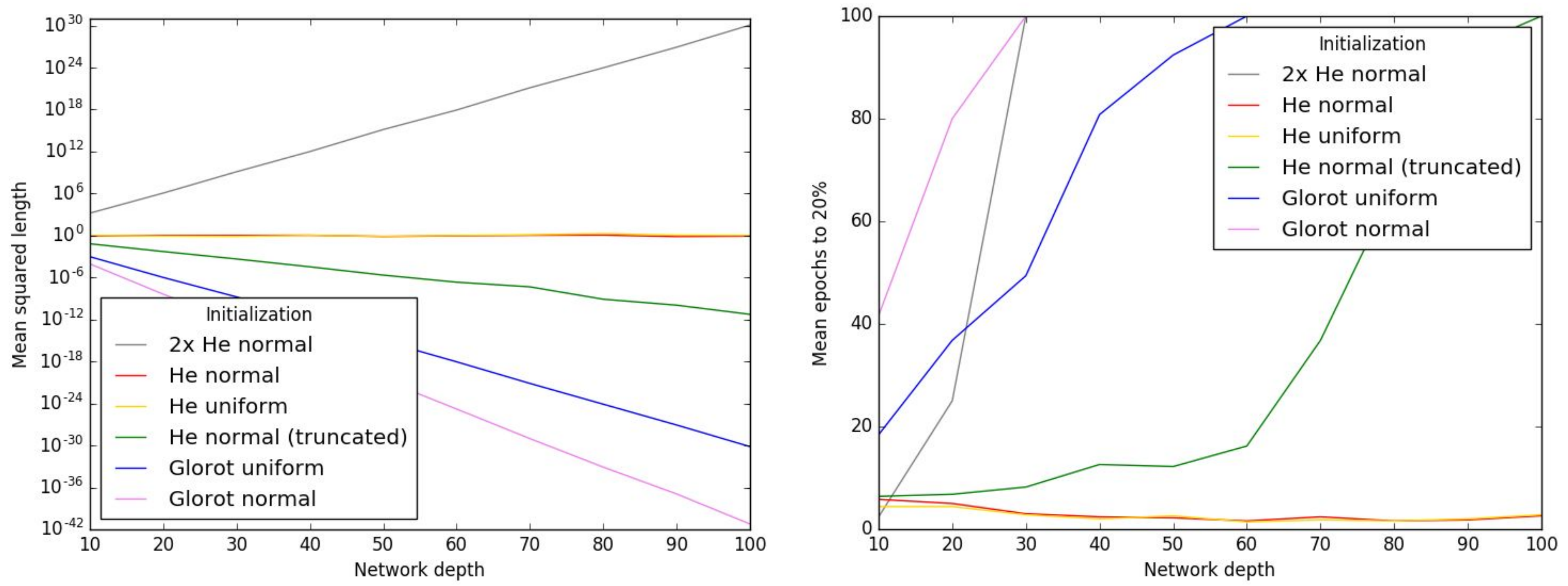

Fig. 1: *Left panel shows exponential growth of the mean squared length of the output vector for many popular initializations. Right panel shows the negative impact of very large and small output lengths on early training over MNIST.*

Effect of Network Architecture on Early Training

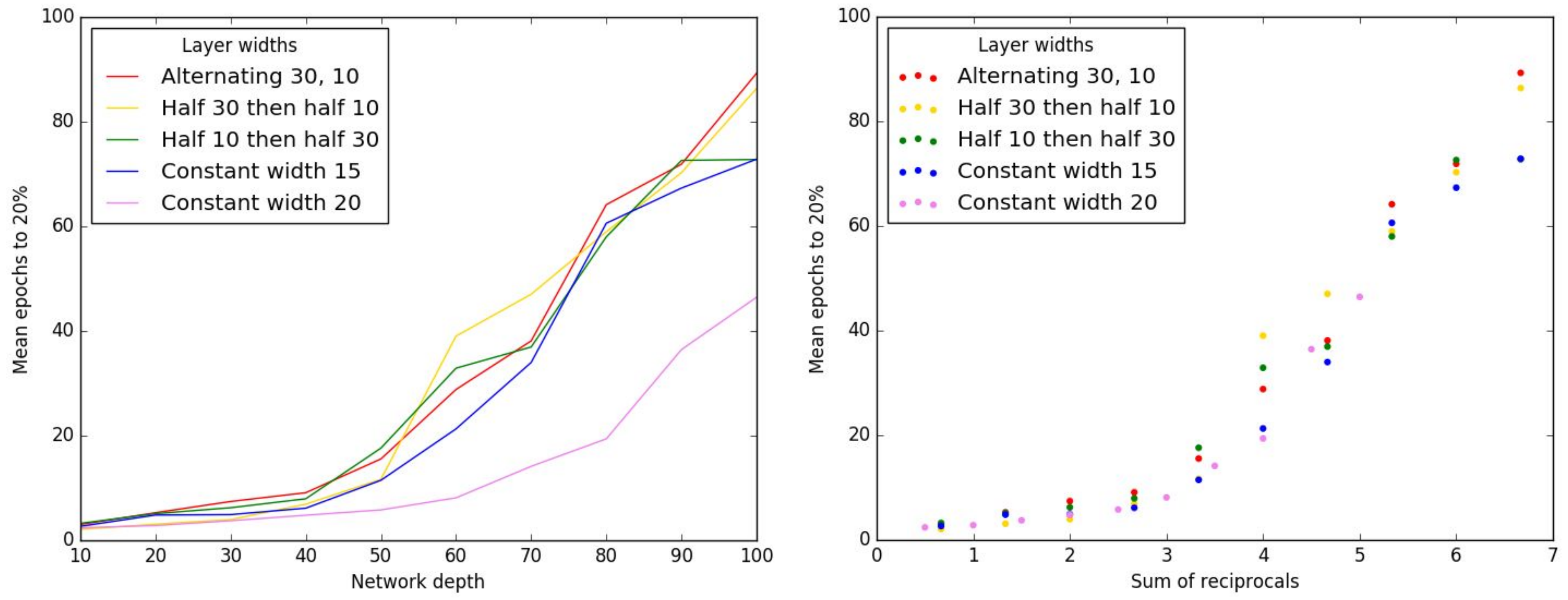

Fig. 2: *Both panels show early training dynamics for a variety of architectures, when training on MNIST. In the left panel, the pink curve has a smaller sum of reciprocals at each depth, while all other curves have the same (larger) sum.*

Effect of Residual Module Weights

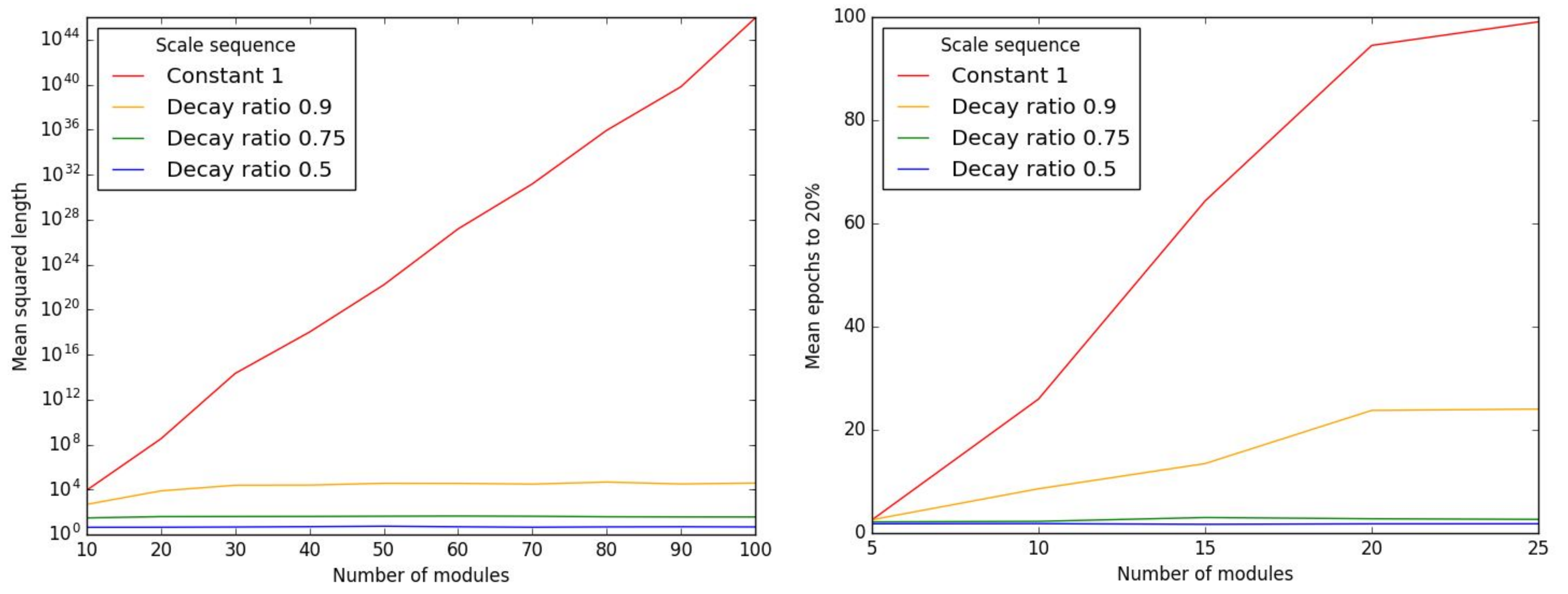

Fig. 3: *Left panel shows the effect of residual module weights, given by different geometric sequences, on the mean squared length of the output. Right panel shows the corresponding impact on early training dynamics over MNIST.*

Definitions

We consider ReLU networks $\mathcal{N}$ at initialization.

FC & ConvNet: Depth d, layer widths $n_0, \ldots, n_d$ (or #features for ConvNets).
Independent weights/biases with marginals symmetric around 0.
Let $\mathrm{act}^{(j)}$ be the vector of activations at layer j, and set:

$$M_j = ||\mathrm{act}^{(j)}||_2^2 \,/ n_j$$

ResNet: We consider a network with residual modules $\mathcal{N}_j$ weighted by s_j:

$$\mathcal{N}(x) = x + s_1\mathcal{N}_1(x) + s_2\mathcal{N}_2(x + s_1\mathcal{N}_1(x)) + \ldots$$

Formal statements

Suppose: $\mathrm{Var}[\text{weights}] \;=\; \kappa \cdot 2/\text{fan-in}$

FC & ConvNet:
- $\mathbb{E}[M_d] \;=\; \kappa^d$
- $\kappa \;=\; 1 \quad\Longrightarrow\quad \widehat{\mathrm{Var}}[M],\ \mathrm{Var}[M_d] \;\sim\; \exp\left[C\sum_{j=1}^{d} n_j^{-1}\right]$

ResNet:
- $\kappa \;=\; 1 \quad\Longrightarrow\quad \mathbb{E}[M_d],\ \mathrm{Var}[M_d] \;\sim\; \exp\left[C\sum_{j=1}^{d} s_j\right]$

Proof approach

(1) Layer activations in feedforward neural nets form a **Markov chain**.

(2) Squared size of layer activation vectors is an integrable **submartingale**.
Therefore may apply **Doob's Pointwise Martingale Convergence Theorem**.

(3) Variance of squared size of layer activations is exponential in sum of reciprocals of hidden layer widths / residual module weights. Thus, need uniform bounds on sum of reciprocals or layer widths / residual module weights to apply **Doob's L_p Martingale Convergence Theorem.**