ReZero is All You Need:
Fast Convergence at Large Depth

Thomas Bachlechner[1*], Bodhisattwa Prasad Majumder[2*], Huanru Henry Mao[3*],
Garrison W. Cottrell[2], Julian McAuley[2]

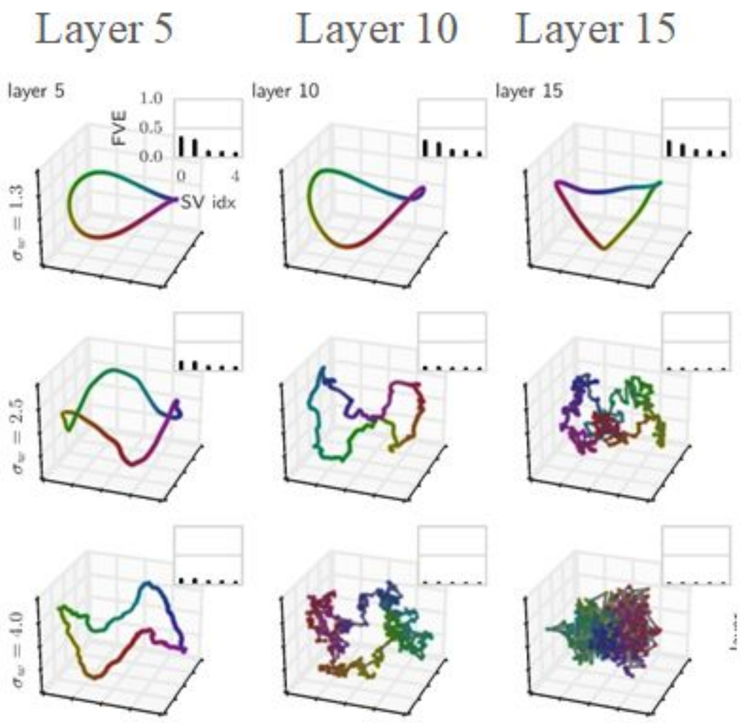
[1]MeetElise, USA
[2]UC San Diego, USA
[3]Altum Inc., USA

Deeper Networks

Deep networks have expressive power that **scales exponentially** with depth:

This figure shows networks with random weights of different distributions propagating a circle through the network.

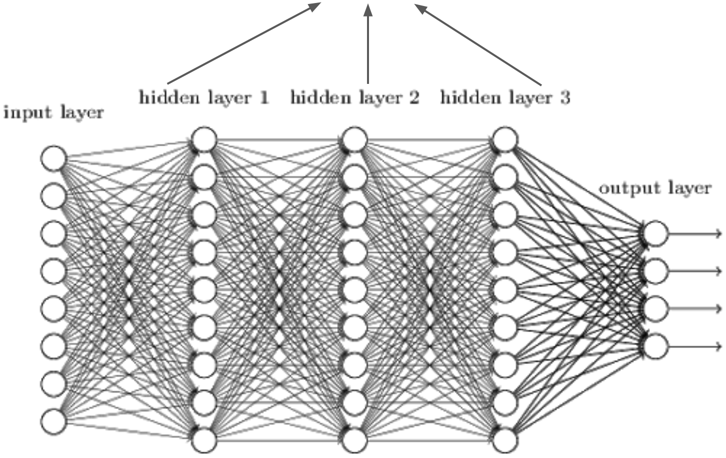
This flexibility makes them **harder to train**.



Why are deep networks harder to train?

Consider a deep network as a series of width-preserving function:

$$x_{i+1} = F[\mathcal{W}_i](x_i)$$



Assuming depth L, if the magnitude of a perturbation is changed by a factor r in each layer, both signals and gradients **vanish** or **explode** at a rate of r^L.

An Analogy

An Analogy

Signal propagation in randomly initialized networks

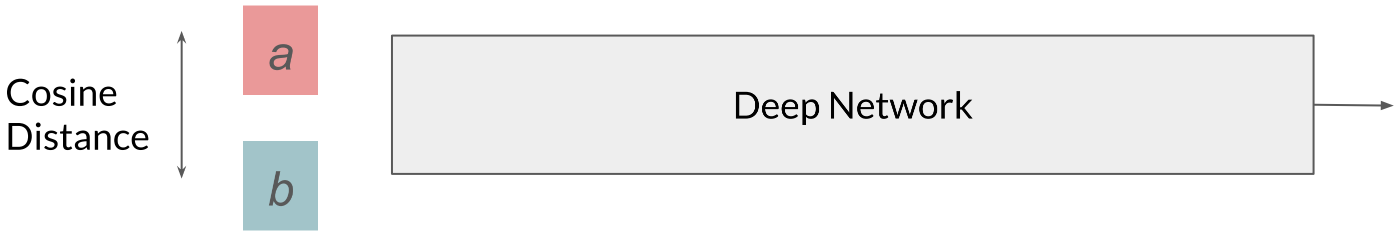
Recently, analysis (using **Mean Field Theory**) of how signals propagate in deep networks has found interesting properties:

Consider two input examples *a* and *b*. The cosine distance of *a* and *b* approaches a **fixed point** as it moves through the deep network.

Depending on the Jacobian of the network, the cosine of two input vectors (*a* and *b*) will converge to 0 **(orthogonal)** or 1 **(aligned)**.

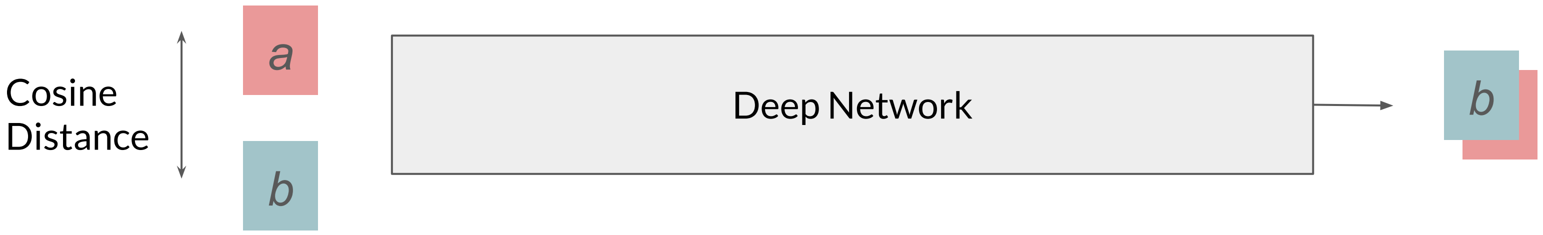
Signal propagation in randomly initialized networks

Depending on the Jacobian of the network, the cosine of two input vectors (*a* and *b*) will converge to **0 (orthogonal)** or **1 (aligned)**.



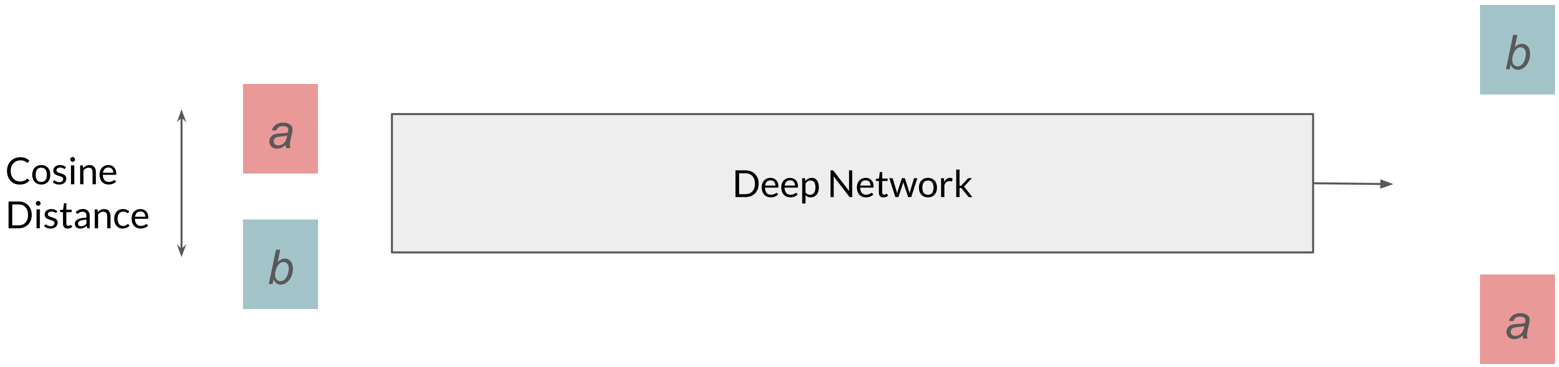
Signal propagation in randomly initialized networks

If this fixed point is **1 (aligned)**: Network is **stable** and every input maps to the same output, so the **gradient vanishes** - even very different inputs will be aligned.



Signal propagation in randomly initialized networks

If this fixed point is 0 **(orthogonal)**: Network is **chaotic** and similar inputs map to very different outputs, leading to **exploding gradients**.



Ideally, we want to initialize our network to be at the ***edge of chaos***.

Dynamical Isometry

To find out if a network is stable or chaotic, we can compute:

$$\boldsymbol{J}_{\mathrm{io}} \equiv \frac{\partial \boldsymbol{x}_L}{\partial \boldsymbol{x}_0}$$

Here, x_L is the output of the network, x_0 is the input.

The mean squared singular values (χ) of J determines the growth or decay of the average signal as it moves through the deep network. When χ is approximately 1, the average signal strength is neither enhanced or attenuated.

Dynamical Isometry (strong condition): All singular values of *J* must be close to 1.

Problem with Initialization

- **Not all architectures** can satisfy this
  - RELUs
  - Self-attention

- In practice, you could use **costly** normalization instead to “fix” the issue
  - BatchNorm has shown to not work well with sequential data
  - LayerNorm can work, but there are problems
  - They incur computational cost

- *What if* there is a **simpler** way?

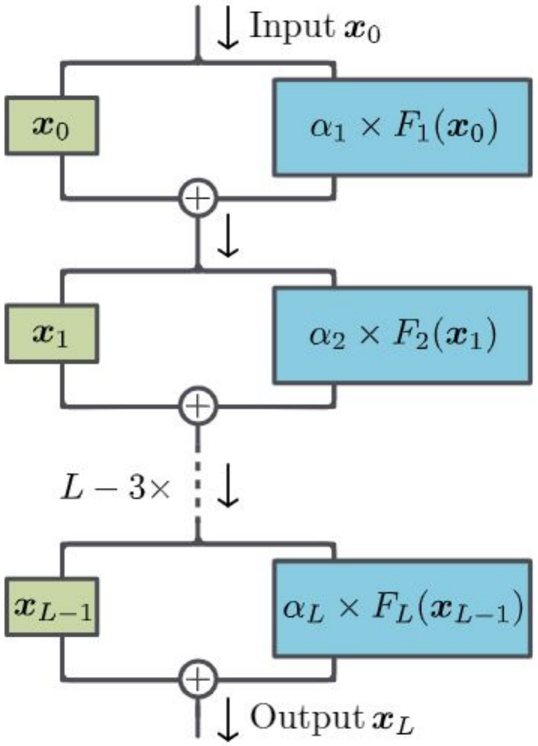
ReZero

ReZero: **re**sidual with **zero** initialization

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \alpha_i F[\mathcal{W}_i](\boldsymbol{x}_i)$$

Initialize this learned scalar to **zero**

- Initializes to **identity map**, trivially satisfies dynamical isometry
- Train as **deep** as you want
- Train much **faster**



Why would ReZero train faster?

Consider a **toy residual network** that has a **single neuron**, single weight and L layers deep:

$$x_L = (1 + \alpha w)^L x_0$$
$$J_{\text{io}} = (1 + \alpha w)^L$$

Why would ReZero train faster?

Consider a **toy residual network** that has a **single neuron**, single weight and L layers deep:

$$x_L = (1 + \alpha w)^L x_0$$
$$J_{\text{io}} = (1 + \alpha w)^L$$

When alpha = 1: The network would be very sensitive the small perturbations of the input. You need a learning rate that is exponentially small in depth.

Why would ReZero train faster?

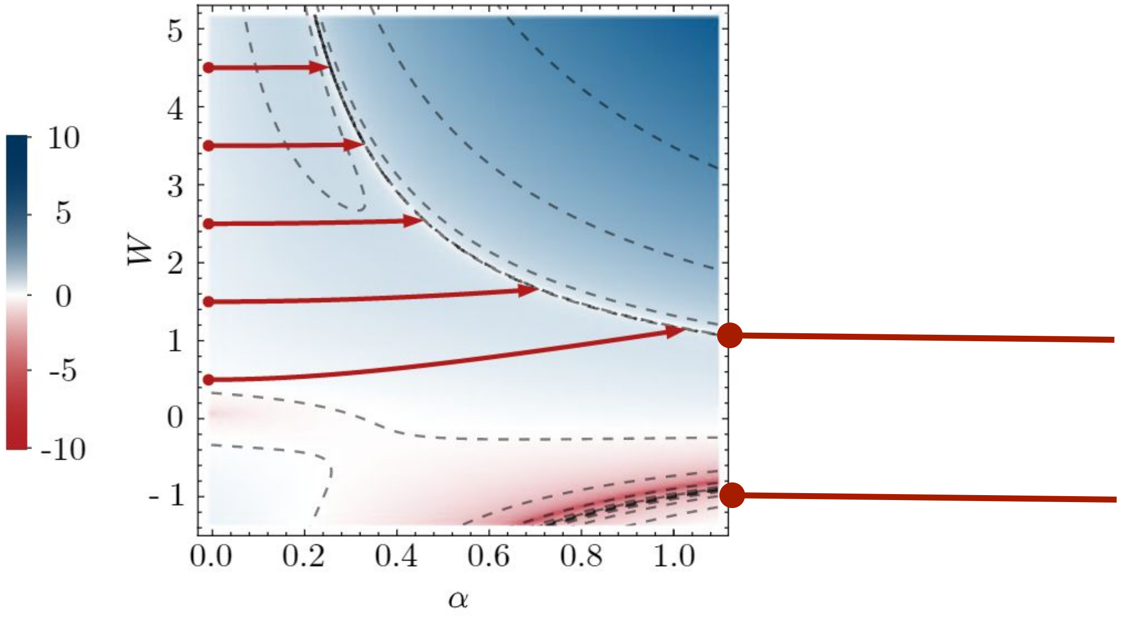
Consider a **toy residual network** that has a **single neuron**, single weight and L layers deep:

$$x_L = (1 + \alpha w)^L x_0$$
$$J_{\text{io}} = (1 + \alpha w)^L$$

When alpha = 1: The network would be very sensitive the small perturbations of the input. You need a learning rate that is exponentially small in depth.

When alpha = 0: The input signal is preserved.

Why would ReZero train faster?



Contour log plots of gradient norm

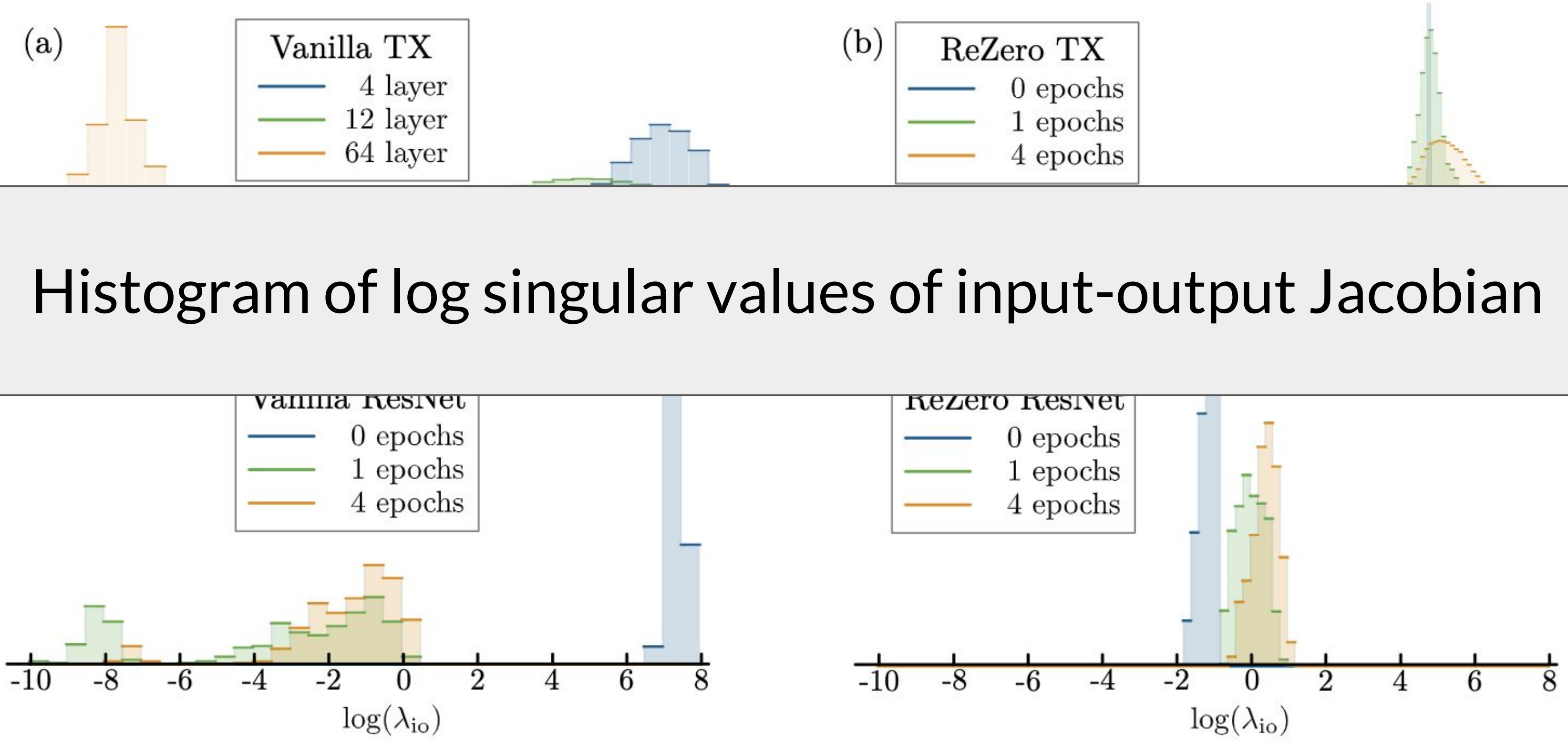
Solution at αw = 1.2

ReZero avoids poorly conditioned regions around α ≈ 1

ReZero for Deep Networks

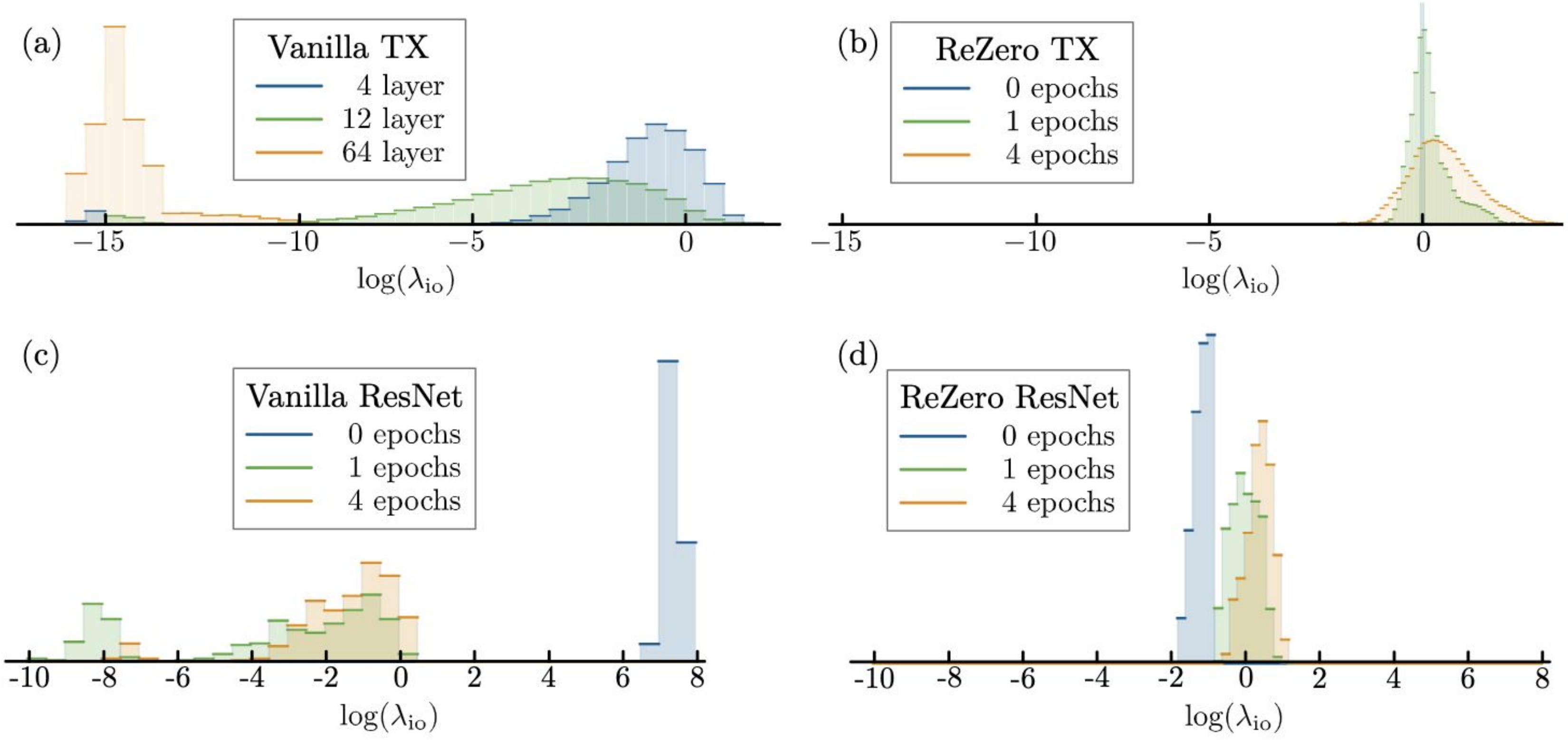
(1) Deep Network (Net.)	$\boldsymbol{x}_{i+1} = F(\boldsymbol{x}_i)$
(2) Residual Network	$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + F(\boldsymbol{x}_i)$
(3) Deep Net. + Norm	$\boldsymbol{x}_{i+1} = \mathrm{Norm}\big(F(\boldsymbol{x}_i)\big)$
(4) Residual Net. + Pre-Norm	$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + F(\mathrm{Norm}\big(\boldsymbol{x}_i)\big)$
(5) Residual Net. + Post-Norm	$\boldsymbol{x}_{i+1} = \mathrm{Norm}\big(\boldsymbol{x}_i + F(\boldsymbol{x}_i)\big)$
(6) **ReZero**	$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \alpha_i F(\boldsymbol{x}_i)$

Signal Propagation in ReZero

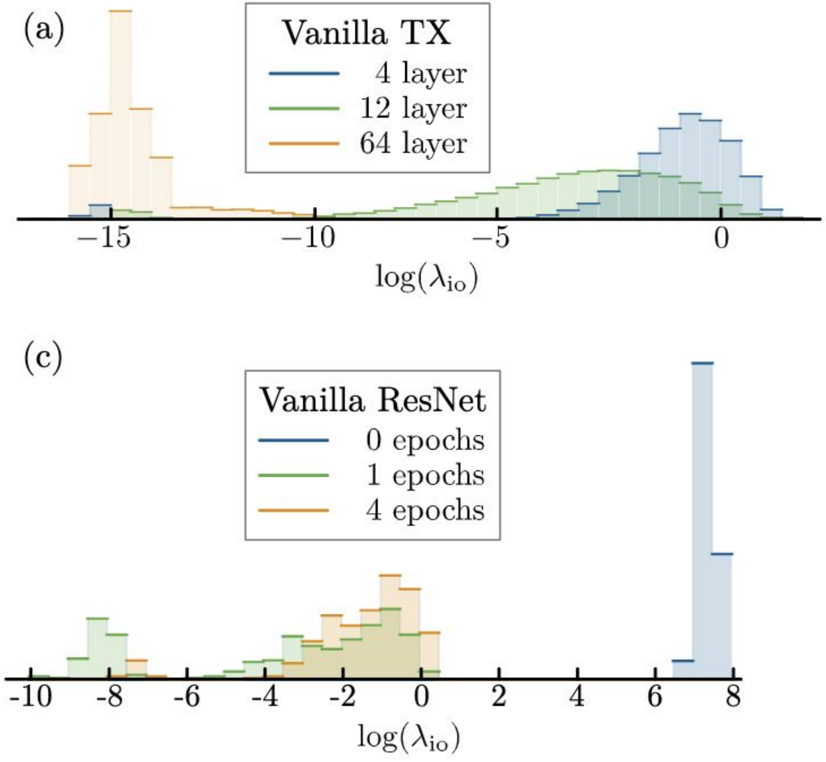


Histogram of log singular values of input-output Jacobian

Signal Propagation in ReZero



Signal Propagation in ReZero



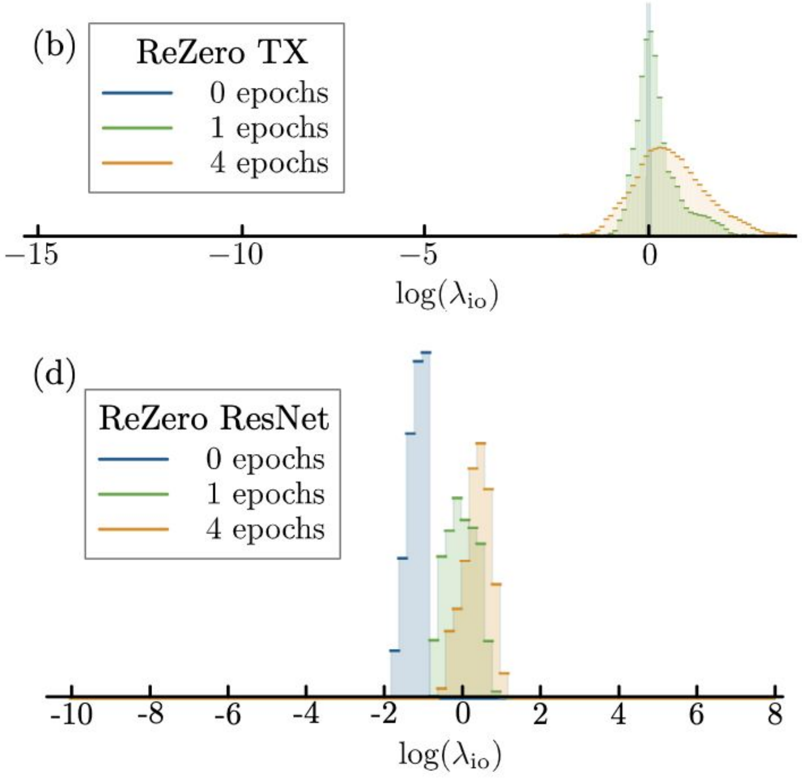
For deeper networks, or at initialization, log of singular values **are far from 0**.

This leads to **vanishing** or exploding **gradients**.

Signal Propagation in ReZero

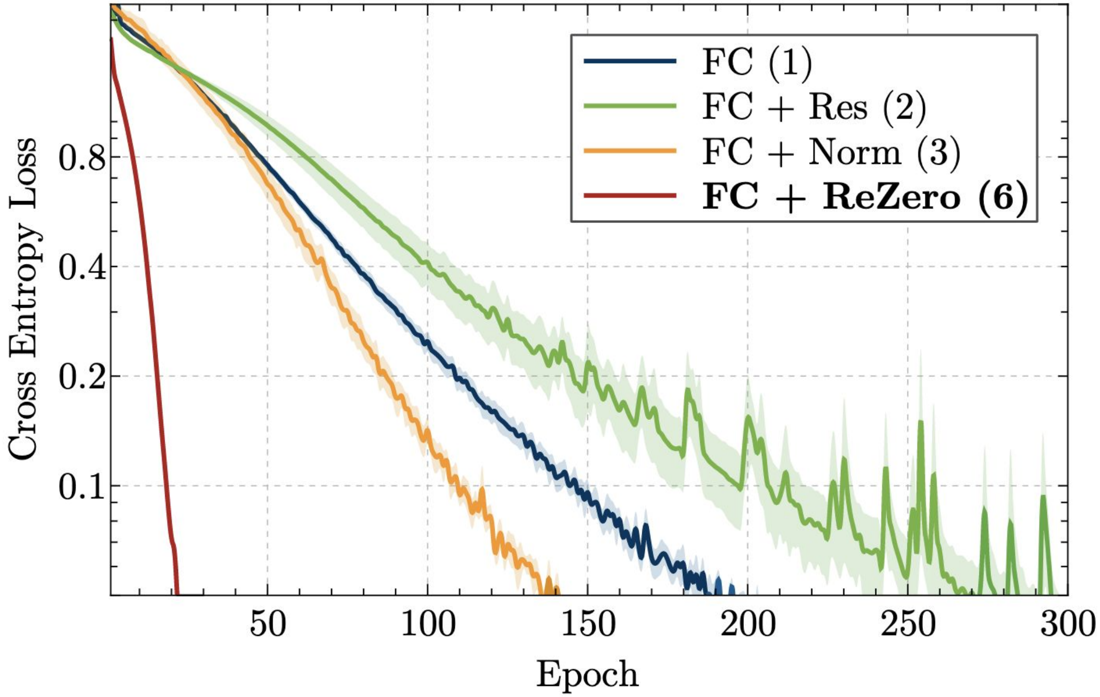
For **ReZero**, for both deeper networks and at initialization, log of singular values **are close to 0**.

Allows **faster** signal propagation at **large depth**.



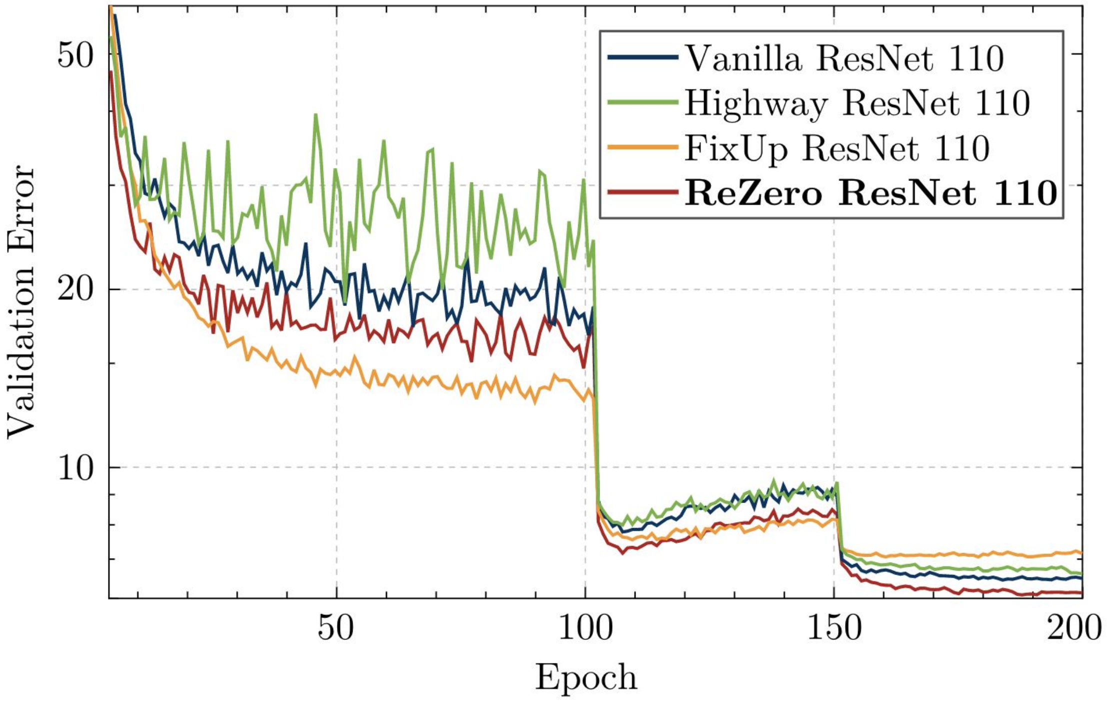
Faster Convergence for Fully-connected Networks

Four variants of
32 layer fully-connected
networks with width
256 and ReLU
activations on CIFAR-10



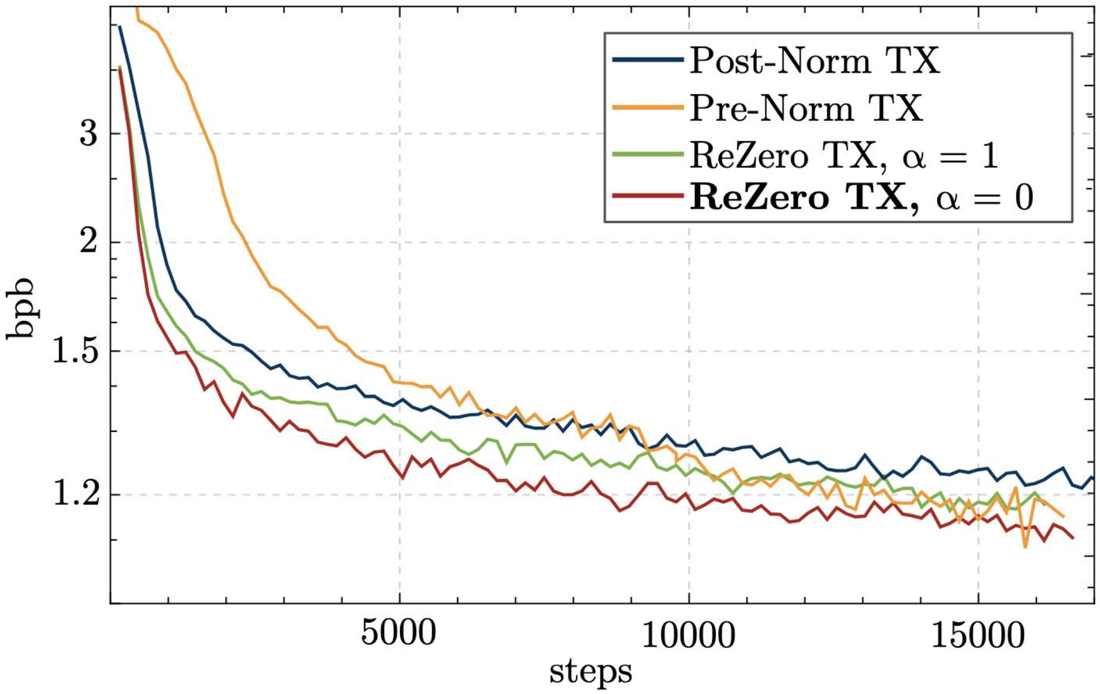
Faster Convergence for ResNets

Validation error for four variants of ResNet-110 on CIFAR-10

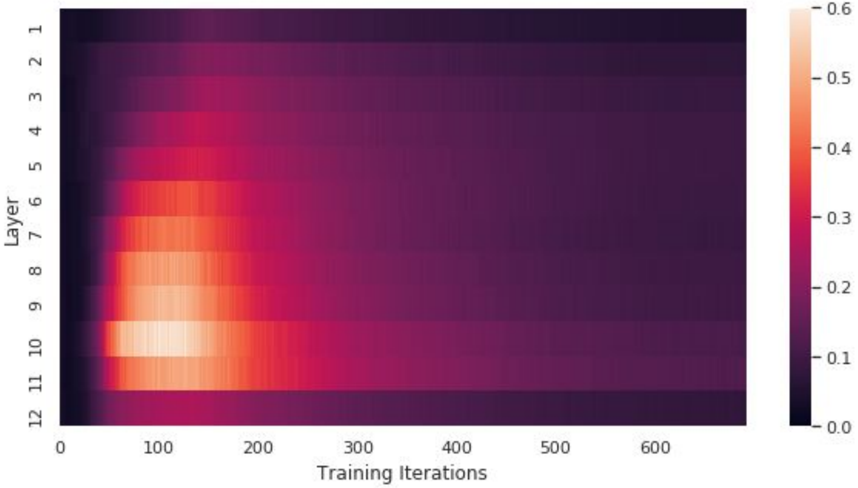


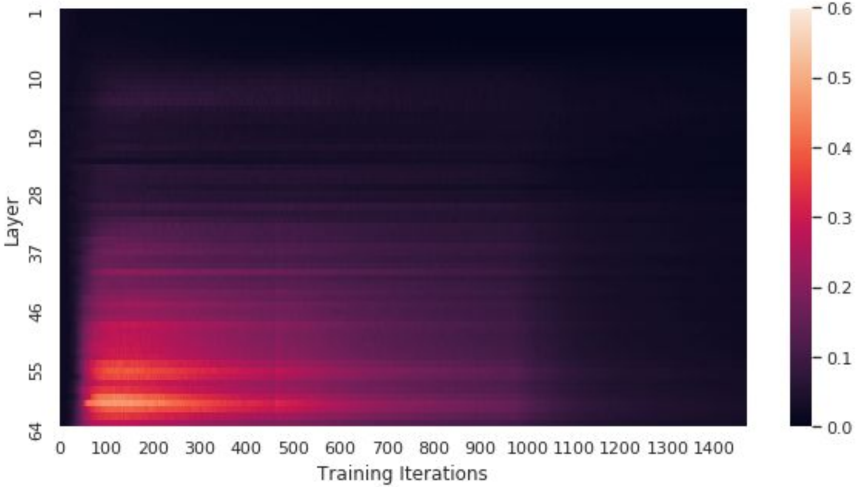
Faster Convergence for Transformers

Three variants of 12 layer Transformers normalization variants against ReZero on enwiki8



Analysis on Alphas





Alphas become big near the deepest layer early in training, then drop back to small values

Conclusion

- A really **simple way** to get much faster convergence in deep networks.
- You can train **arbitrary deep networks** as you desire.
- **Flexible to many architectures**, no need for complex initialization schemes.

Future work:

  - Explore the meaning behind residual weights.
  - Progressively grow your ReZero network.

`pip install rezero`