Lecture 7

Architecture 1: Signal propagation in depth

CS312 / Tatsu H

Logistics

A4 (Transformers vs RNNs) is out;

Next quiz Thu 10/29.

Tutorial Expectations

- First round of tutorials are now done
- Expectations will get higher further into the class
  - Know your quiz / assignment. Have thoughts on major concepts
  - Brush up on basics you might not be familiar with
  - Make sure to participate and talk.

Architectures

Reasoning about architectures is complicated!

GLU Variants Improve Transformer

Noam Shazeer
Google
noam@google.com

February 14, 2020

4 Conclusions

We have extended the GLU family of layers and proposed their use in Transformer. In a transfer-learning setup, the new variants seem to produce better perplexities for the de-noising objective used in pre-training, as well as better results on many downstream language-understanding tasks. These architectures are simple to implement, and have no apparent computational drawbacks. We offer no explanation as to why these architectures seem to work; we attribute their success, as all else, to divine benevolence.

Do Transformer Modifications Transfer Across Implementations and Applications?

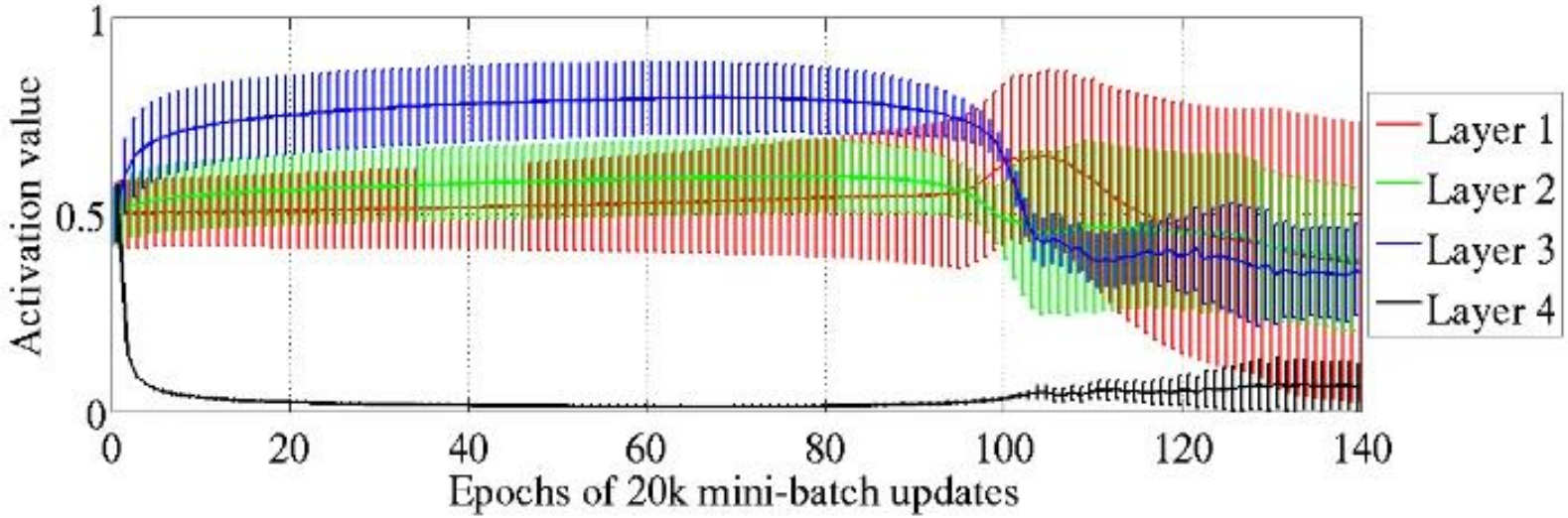
Sharan Narang* Hyung Won Chung Yi Tay William Fedus
Thibault Fevry† Michael Matena † Karishma Malkan† Noah Fiedel
Noam Shazeer Zhenzhong Lan† Yanqi Zhou Wei Li
Nan Ding Jake Marcus Adam Roberts Colin Raffel

Google Research

Model	Params	Ops	Step/s	Early loss	Final loss	SGLUE
Vanilla Transformer	$223M$	$11.1T$	3.50	2.182 ± 0.005	1.838	70.97
GeLU	$223M$	$11.1T$	3.58	2.179 ± 0.003	1.838	**73.67**
Swish	$223M$	$11.1T$	3.62	2.186 ± 0.003	1.847	**72.03**
ELU	$223M$	$11.1T$	3.56	2.270 ± 0.007	1.932	65.86
GLU	$223M$	$11.1T$	3.59	2.174 ± 0.003	**1.814**	**67.86**
GeGLU	$223M$	$11.1T$	3.55	2.130 ± 0.006	**1.792**	**74.86**
ReGLU	$223M$	$11.1T$	3.57	2.145 ± 0.004	**1.803**	**73.40**
SeLU	$223M$	$11.1T$	3.55	2.315 ± 0.004	1.948	66.13
SwiGLU	$223M$	$11.1T$	3.53	2.127 ± 0.003	**1.789**	**74.21**
LiGLU	$223M$	$11.1T$	3.59	2.149 ± 0.005	**1.798**	**73.97**
Sigmoid	$223M$	$11.1T$	3.63	2.291 ± 0.019	1.867	**71.27**
Softplus	$223M$	$11.1T$	3.47	2.207 ± 0.011	1.850	70.86
RMS Norm	$223M$	$11.1T$	3.68	2.167 ± 0.008	**1.821**	**73.73**
Rezero	$223M$	$11.1T$	3.51	2.262 ± 0.003	1.939	57.21
Rezero + LayerNorm	$223M$	$11.1T$	3.26	2.223 ± 0.006	1.858	68.16
Rezero + RMS Norm	$223M$	$11.1T$	3.34	2.221 ± 0.009	1.875	65.14
Fixup	$223M$	$11.1T$	2.95	2.382 ± 0.012	2.067	55.71
24 layers, $d_{\text{ff}} = 1536$, $H = 6$	$224M$	$11.1T$	3.33	2.200 ± 0.007	1.843	**72.55**
18 layers, $d_{\text{ff}} = 2048$, $H = 8$	$223M$	$11.1T$	3.38	2.185 ± 0.005	**1.831**	**74.74**
8 layers, $d_{\text{ff}} = 4608$, $H = 18$	$223M$	$11.1T$	3.69	2.190 ± 0.005	1.847	**72.17**
6 layers, $d_{\text{ff}} = 6144$, $H = 24$	$223M$	$11.1T$	3.70	2.201 ± 0.010	1.857	70.28

Deep networks were hard to train for decades

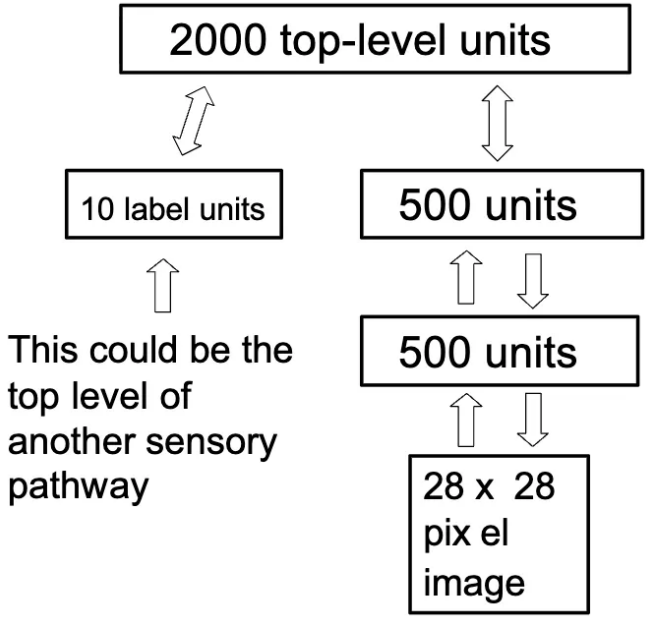
Early deep learning history: training 5 layers was hard!

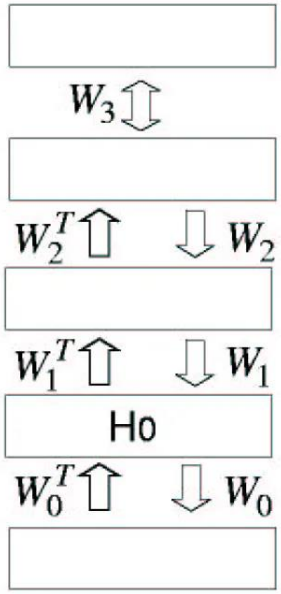


Glorot & Bengio 2010

Initial workarounds: build depth one layer at a time

All sorts of non end-to-end workarounds. Example: Hinton, Osindero and Teh stacked Restricted Boltzmann Machines, trained them greedily layer by layer, then fine-tuned.

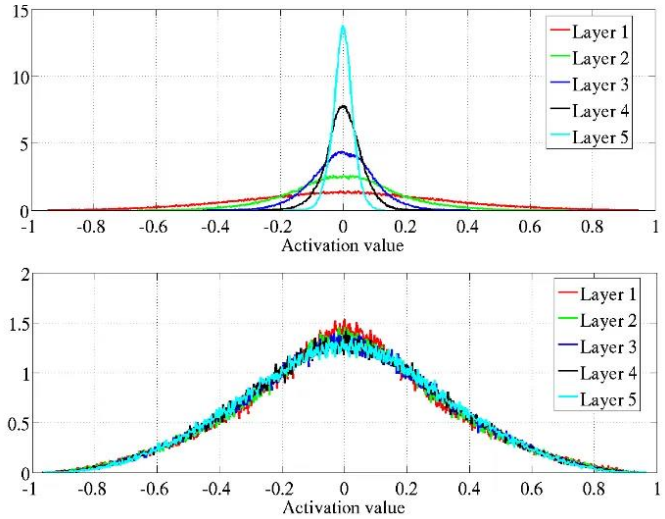


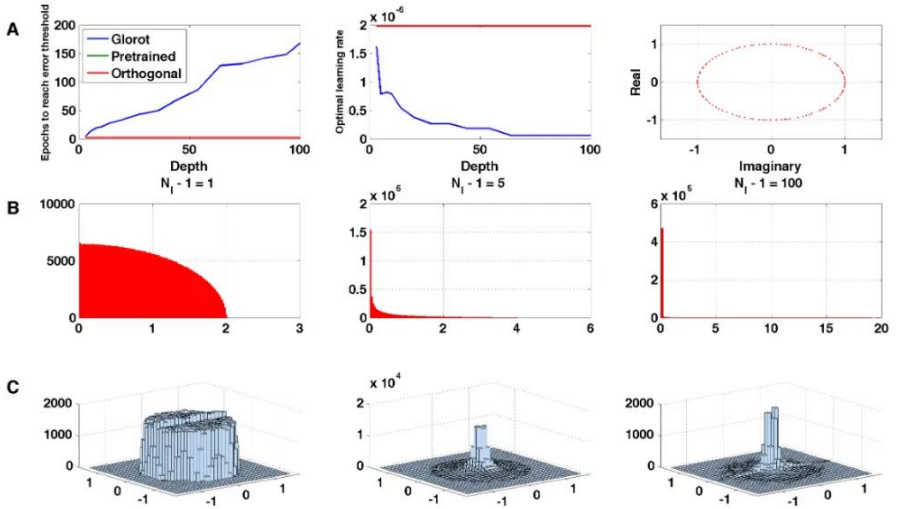


General consensus for deep learning – reliable, end-to-end training of deep networks was hard but critical

Advancements in initialization and normalization

Careful scaling (glorot, left), batch norm and ReLU activations pushed the trainable depth further. Still, Glorot style initialization had a hard time pushing to hundreds of layers..

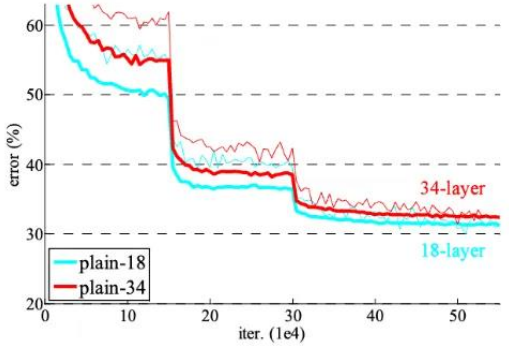


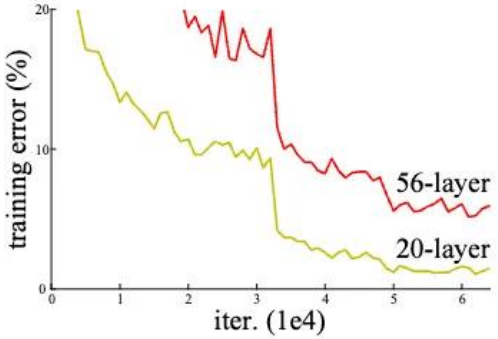


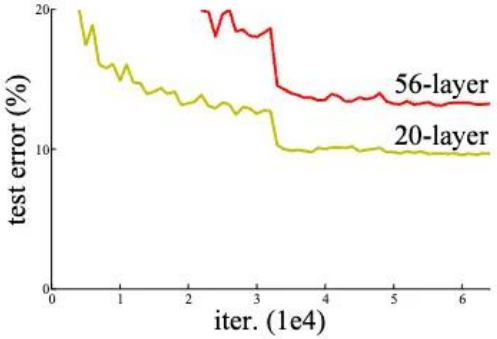
But, limits to this set of tricks

Even then, the demands of modern, very deep training are high!

With ReLU, BatchNorm and He init, a 34-layer plain net has higher training error than an 18-layer one, on ImageNet and on CIFAR-10.





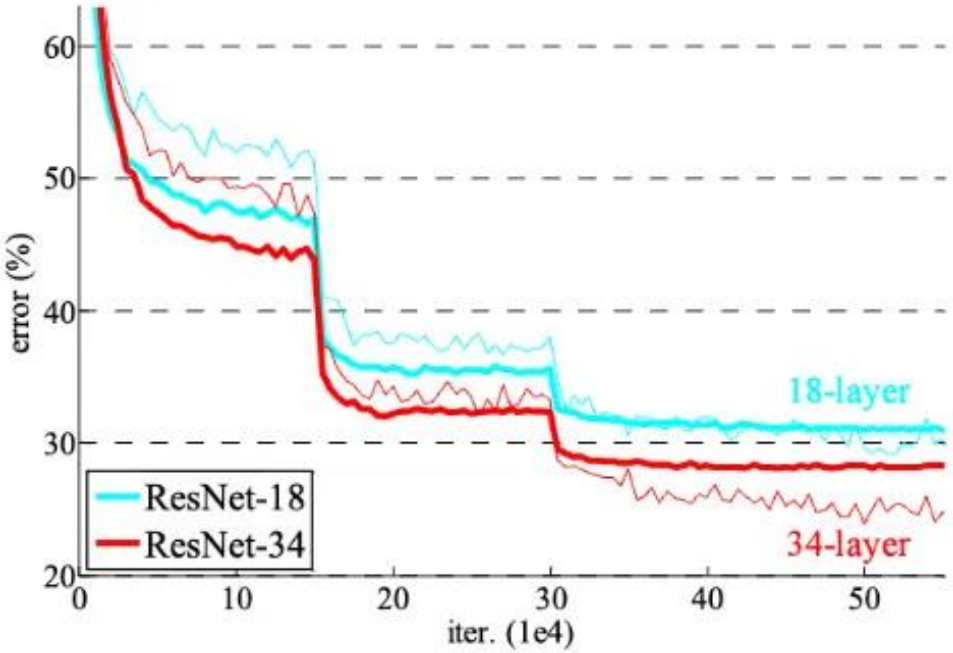


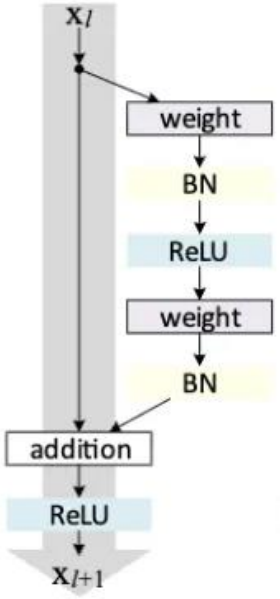
Importantly, note that this is *training error* so this is purely a optimization issue!

One of the core modern advances: residual connections

One of the most important advances in architecture design – residual connections

ResNet-34 beats ResNet-18 (25.0 vs 27.9 top-1) with the same layers

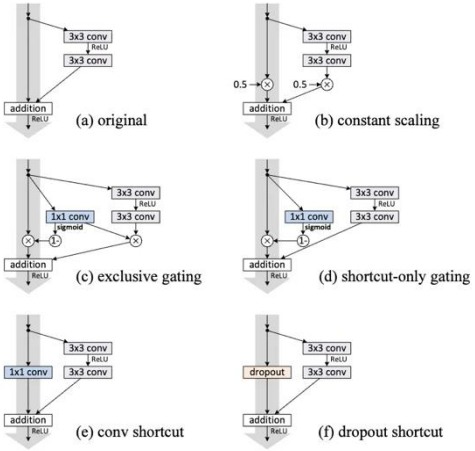




Lecture today

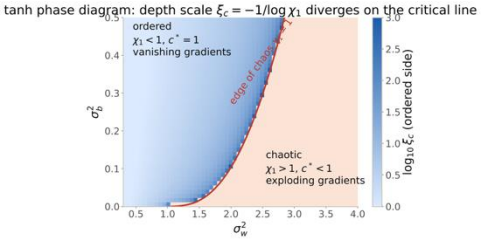
Part 1

Empirical analysis of ResNets



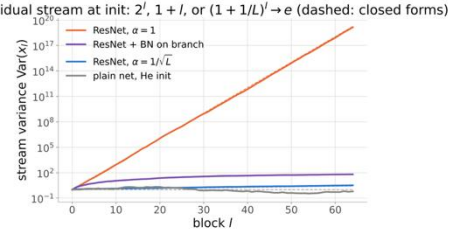
Part 2

Signal propagation theory



Part 3

Testing the theory



Jul 2016

Identity Mappings in Deep Residual Networks

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun

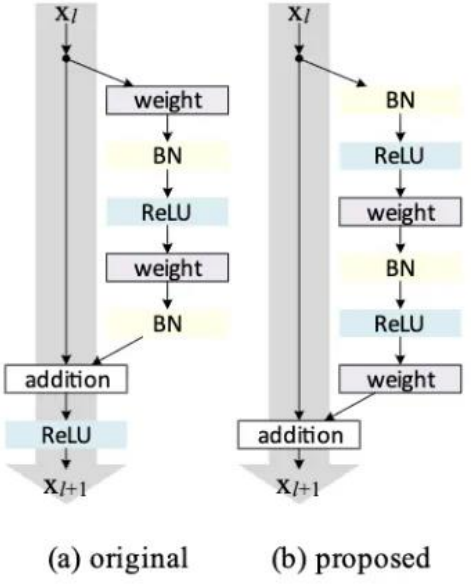
Microsoft Research

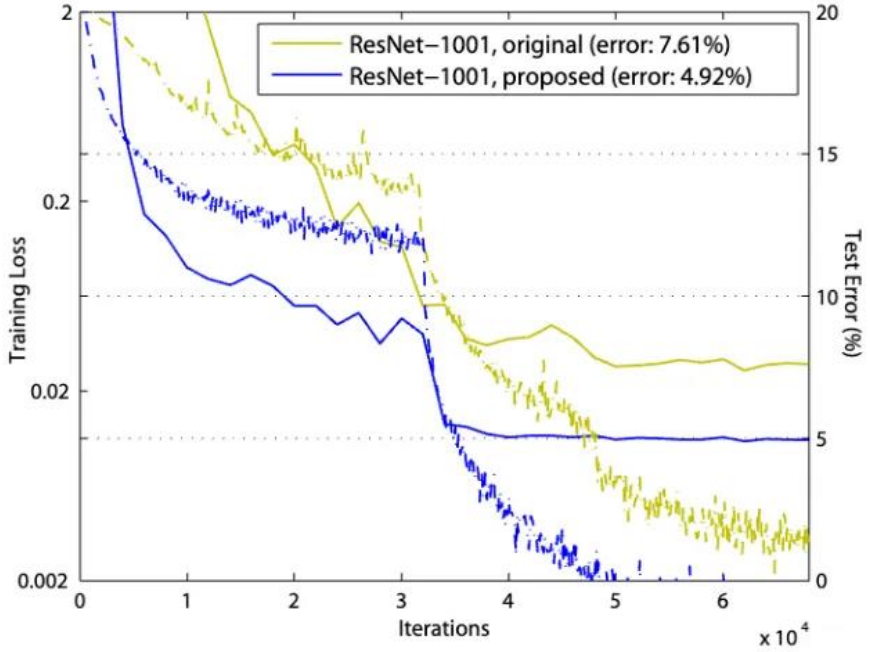
Core to ResNets (v1 and v2) – residual stream

ResNet philosophy: modular 'residual blocks' (cf. transformers)

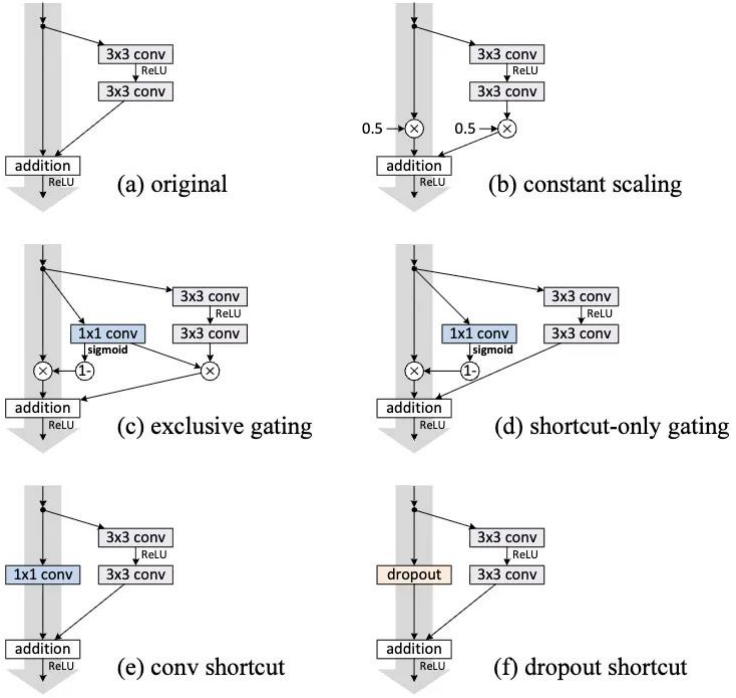
V1: ReLU after each merge.

V2: Addition only - **x_L = x_l + Σ F(x_i)** – gradients and signal both add straight through





What happens if we mess with the shortcuts?



Interventions:

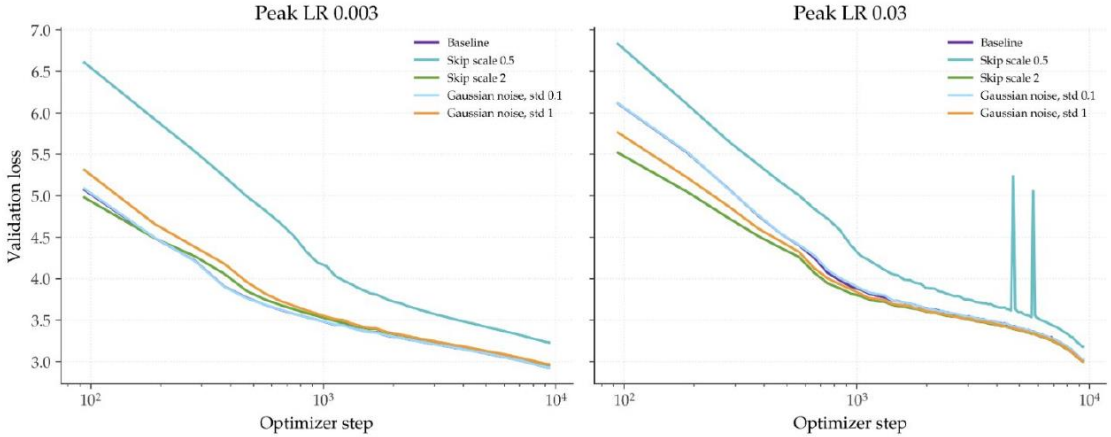
- **Scaling**: λ gives λ^(L−l): gain 0.5 fails outright.
- **Conv**: 1 × 1 conv shortcut doubles the error.
- **Gating**: Fails unless init is trivial
- **Dropout**: Fails

case	Fig.	on shortcut	on $\mathcal{F}$	error (%)	remark
original [1]	Fig. 2(a)	1	1	**6.61**	
constant scaling	Fig. 2(b)	0	1	fail	This is a plain net
		0.5	1	fail	
		0.5	0.5	12.35	frozen gating
exclusive gating	Fig. 2(c)	$1-g(\mathbf{x})$	$g(\mathbf{x})$	fail	init b_g=0 to −5
		$1-g(\mathbf{x})$	$g(\mathbf{x})$	8.70	init b_g=-6
		$1-g(\mathbf{x})$	$g(\mathbf{x})$	9.81	init b_g=-7
shortcut-only gating	Fig. 2(d)	$1-g(\mathbf{x})$	1	12.86	init b_g=0
		$1-g(\mathbf{x})$	1	6.91	init b_g=-6
1×1 conv shortcut	Fig. 2(e)	1×1 conv	1	12.22	
dropout shortcut	Fig. 2(f)	dropout 0.5	1	fail	

Our own experiment (lecture 1)

Similar issues – scaling down fails.

Gaussian noise is surprisingly unimportant (maybe due to the SNR?)

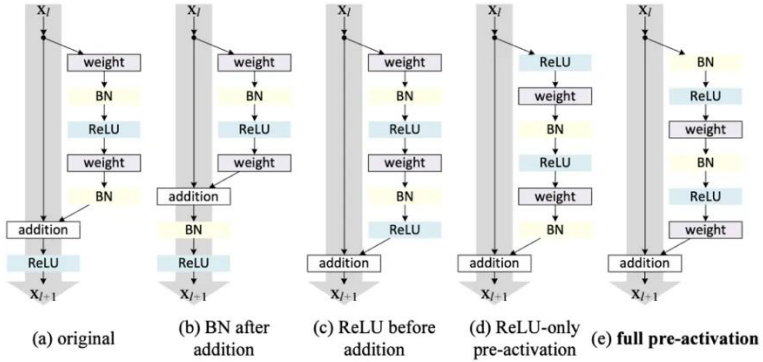


Residual pathway	LR 0.003	LR 0.03
Baseline	2.9267	3.0214
Skip scale 0.5	3.2269	3.1752
Skip scale 2	2.9500	2.9993
Gaussian noise, std 0.1	2.9280	3.0206
Gaussian noise, std 1	2.9662	3.0016

What about the batch norm?

Is 'clean residual' the only thing that matters?

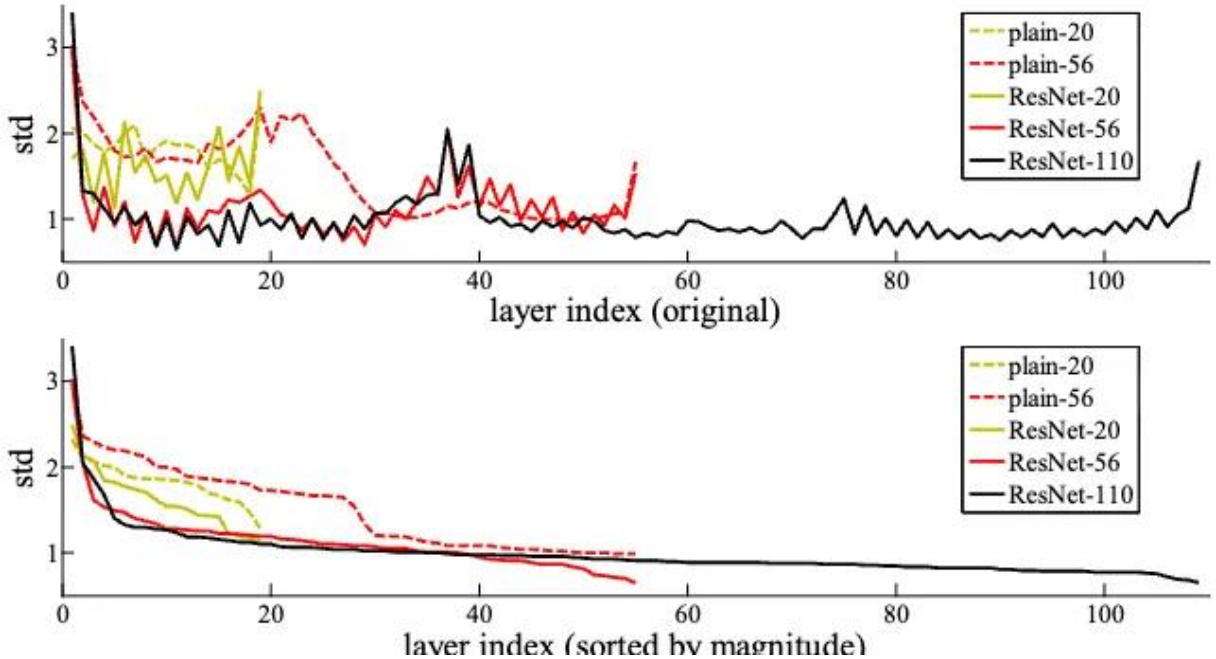
No, batch norm placements can be quite important!



case	Fig.	ResNet-110	ResNet-164
original Residual Unit [1]	Fig. 4(a)	6.61	5.93
BN after addition	Fig. 4(b)	8.17	6.50
ReLU before addition	Fig. 4(c)	7.84	6.14
ReLU-only pre-activation	Fig. 4(d)	6.71	5.91
full pre-activation	Fig. 4(e)	**6.37**	**5.46**

Residual blocks learn small corrections

Observation: not all residual layers contribute equally to the residual stream



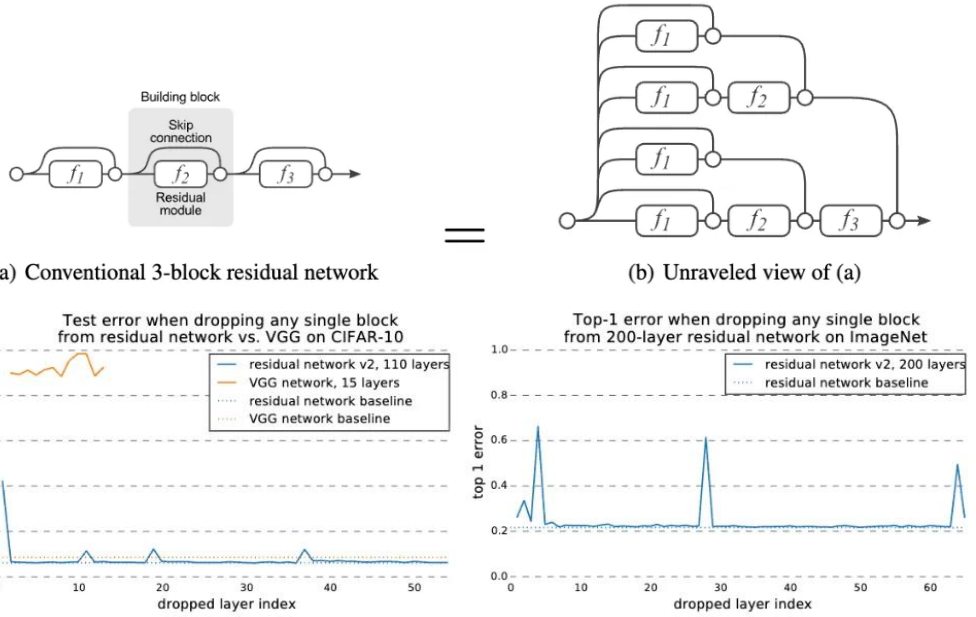
Note the long tail of (small) contributions, and comparison to plain networks

A ResNet is an ensemble of short paths

Residual Networks Behave Like Ensembles of Relatively Shallow Networks

Andreas Veit Michael Wilber Serge Belongie
Department of Computer Science & Cornell Tech
Cornell University
{av443, mjw285, sjb344}@cornell.edu

Residual networks can be viewed as an ensemble / collection of blocks

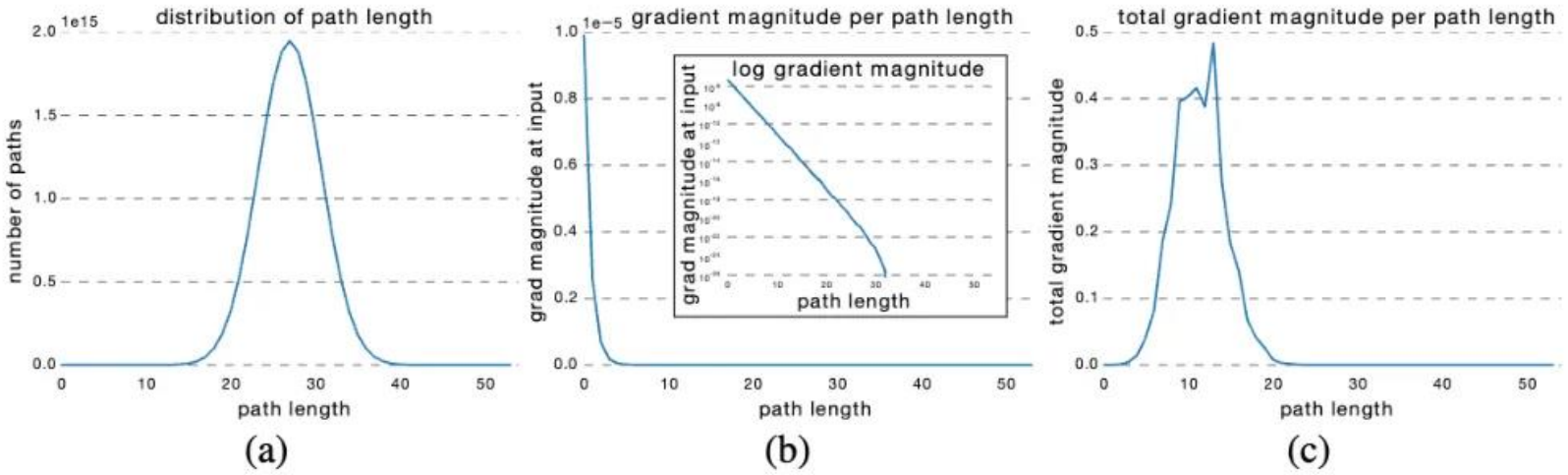


This suggests that each block will be ‘small’, and much of the signal takes short paths

The gradient comes from the short paths

Path lengths of this ‘ensemble’ follows a binomial

Surprisingly, most of the gradients are on ‘short paths’ (5-15 blocks)

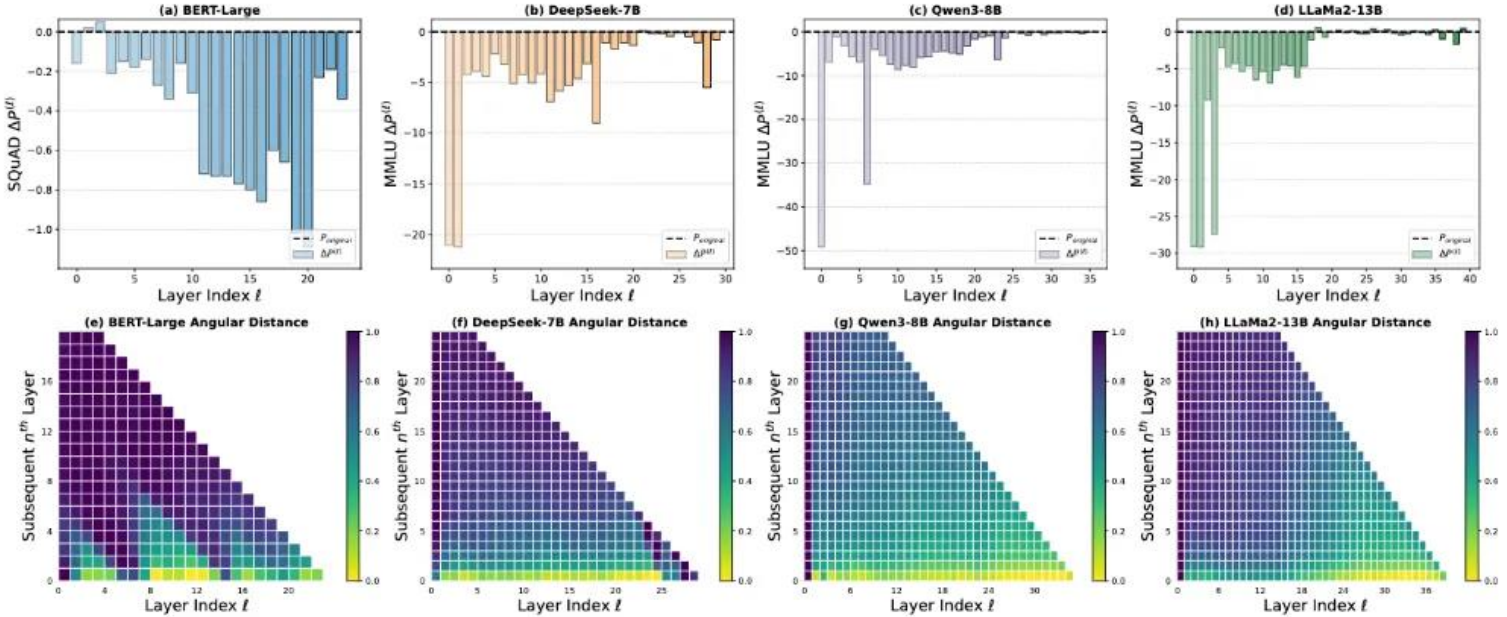


The same experiment on a 2025 LLM

The Curse of Depth in Large Language Models

Wenfang Sun*[1], Xinyuan Song*[2], Pengxiang Li*[3], Lu Yin[4], Yefeng Zheng[1], Shiwei Liu†[5]
[1]Westlake University, China
[2]Emory University, USA
[3]Dalian University of Technology, China
[4]University of Surrey, UK
[5]University of Oxford, UK

LLaMA-2, DeepSeek-7B and Qwen3: beyond layer ~18, removing any single layer barely moves MMLU



Sun et al. 2025, The Curse of Depth, Fig. 2

Summary

Residual connections – one of the most important innovations in architecture!

What makes residual connections work –‘clean’ residuals that allow signal propagation

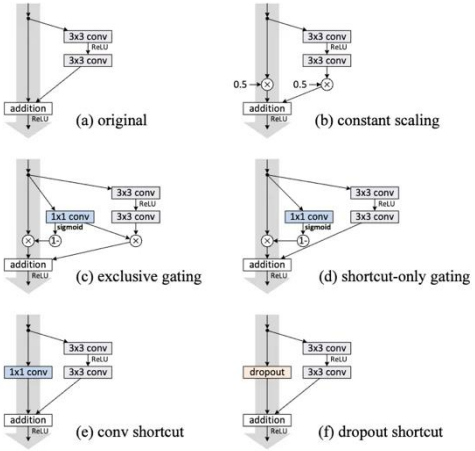
Subtleties in residual connections – But! Stable training doesn’t mean all layers used

Much of the prediction signal in a ResNet comes through shallow paths

Lecture today

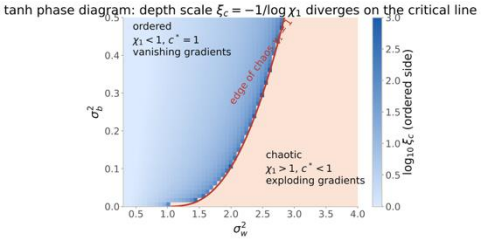
Part 1

Empirical analysis of ResNets



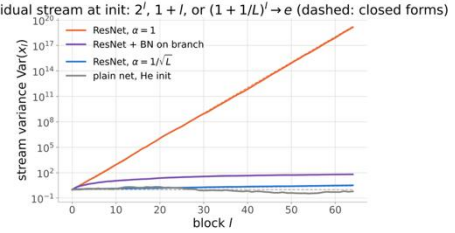
Part 2

Signal propagation theory



Part 3

Testing the theory



The return of the muP

What makes resnets good?

- Can we go beyond ‘have a clean residual’?
- Can we explain detailed choices like norm placements?
- Are the requirements of a good network the same as the training ones for muP?
- Can we replicate the goodness of resnets without a residual connection?

Published as a conference paper at ICLR 2017

DEEP INFORMATION PROPAGATION

Samuel S. Schoenholz* Google Brain
Justin Gilmer* Google Brain
Surya Ganguli Stanford University
Jascha Sohl-Dickstein Google Brain

Mean Field Residual Networks: On the Edge of Chaos

Greg Yang*
Microsoft Research AI
gregyang@microsoft.com

Samuel S. Schoenholz
Google Brain
schsam@google.com

Setup and notation

We're going back to muP style analysis of 'scaling'.

Setup: Simple MLP style model parametrized by (σ_w, σ_b)

$$h^l = W^l \phi\left(h^{l-1}\right) + b^l, \text{ with } W_{ij}^l \sim N(0, \sigma_w^2/n) \text{ and } b_i^l \sim N(0, \sigma_b^2)$$

What's same as before: tracking activation / gradient variances (magnitudes) $(q^l, \tilde{q}^l)$.

Different: tracking 'geometry' – how much the network keeps two signals apart

- Forward pass: $c^l = \frac{q_{ab}^l}{\sqrt{q_{aa}^l \cdot q_{bb}^l}}$ is the correlation of two inputs
- Backward pass: the Jacobian J shows how gradients propagate

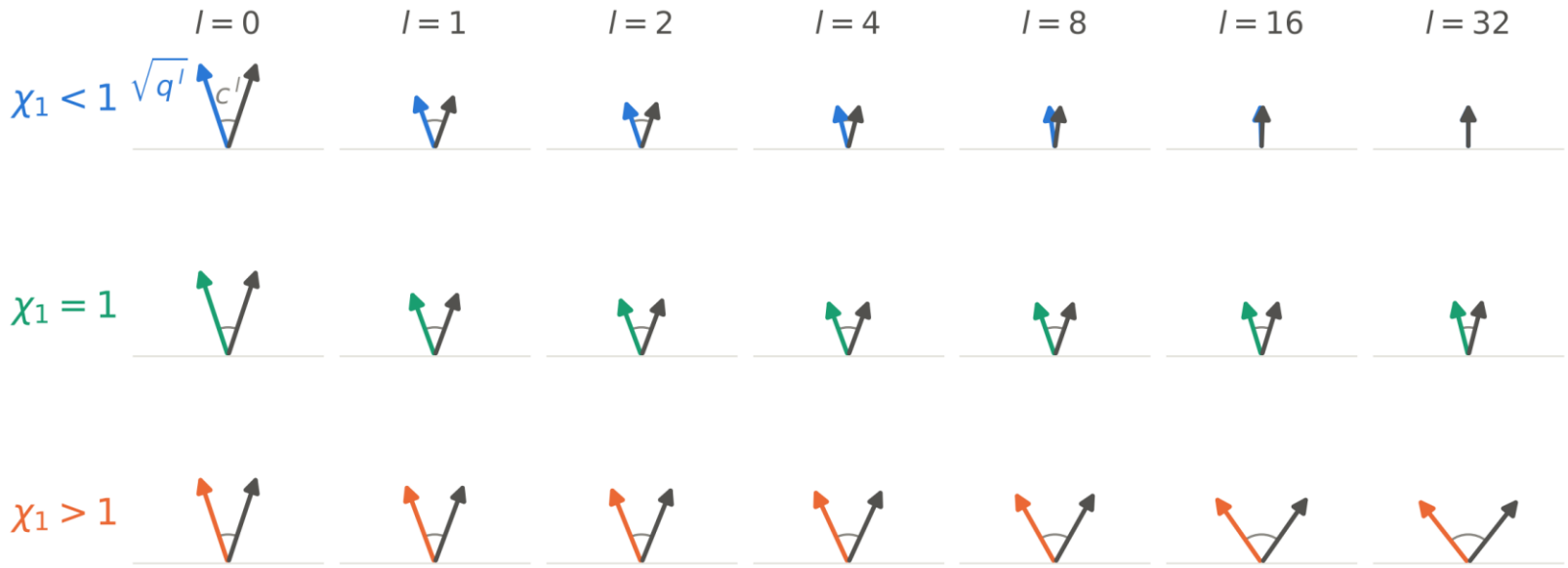
	Forward	Backward
Magnitude	q^l (activation variance)	$\tilde{q}^l$ (gradient variance)
Geometry	c^l (correlation of two inputs)	spectrum of J

What is our goal this time?

The core analysis in the forward case:

Want to ensure that magnitudes and angles across layers do not become degenerate!

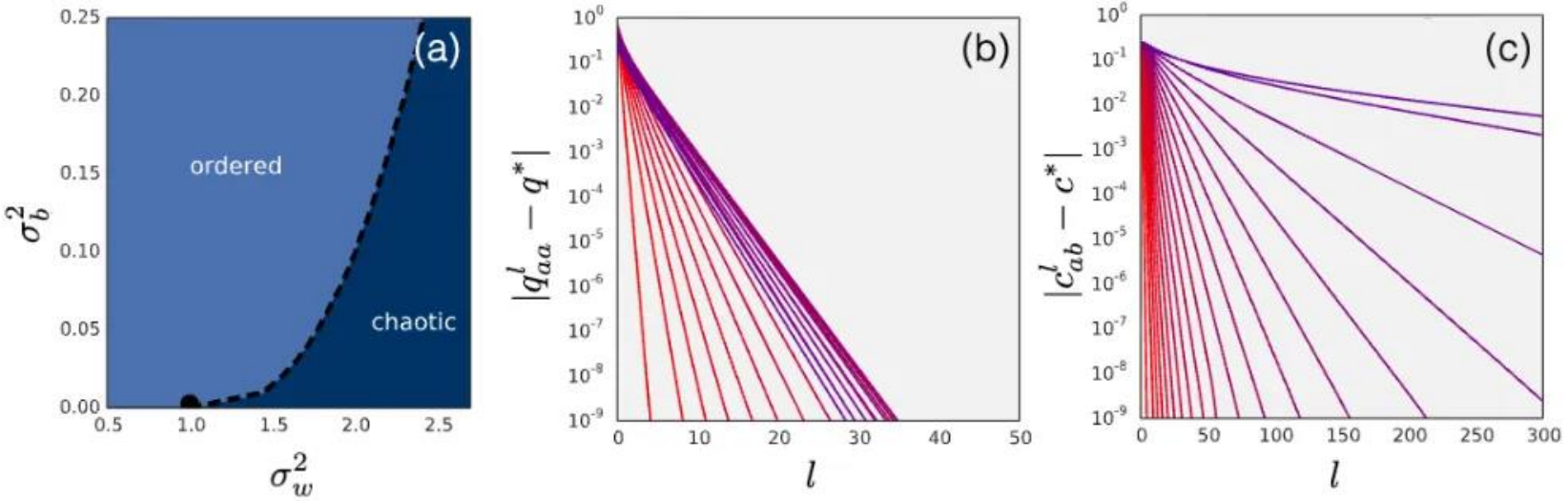
Variables χ_1 defines the multiplicative decay/growth of the angle.



The phase diagram

Ordered ($\chi_1 < 1$): everything looks alike. Chaotic ($\chi_1 > 1$): nothing looks alike.

At the boundary where $\chi_1(\sigma_w, \sigma_b) = 1$, information propagates across depth.

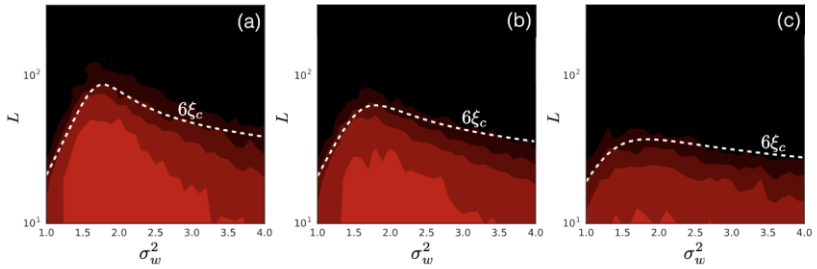


Prediction: trainable depth

The χ_1 multiplier maps to a 'trainable depth' $\xi_c = \frac{1}{-\log \chi_1}$ (kind of like a half life for layers)

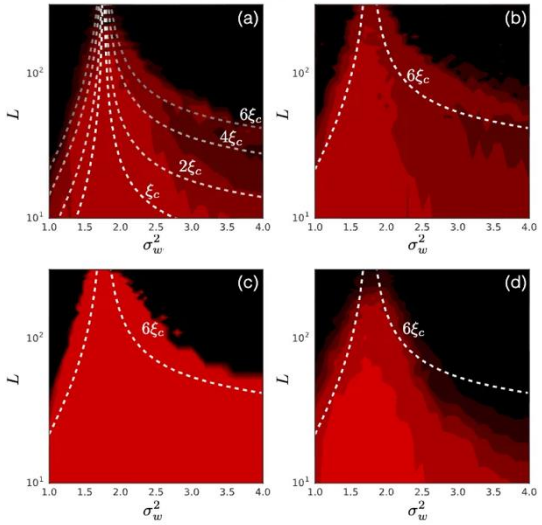
This maps surprisingly nicely to accuracies of trained, deep models on MNIST/CIFAR-10 for SGD/RMSprop

MNIST SGD, now with varying dropout / keep rate



MNIST SGD, 200

CIFAR SGD, 2k



MNIST SGD 14k

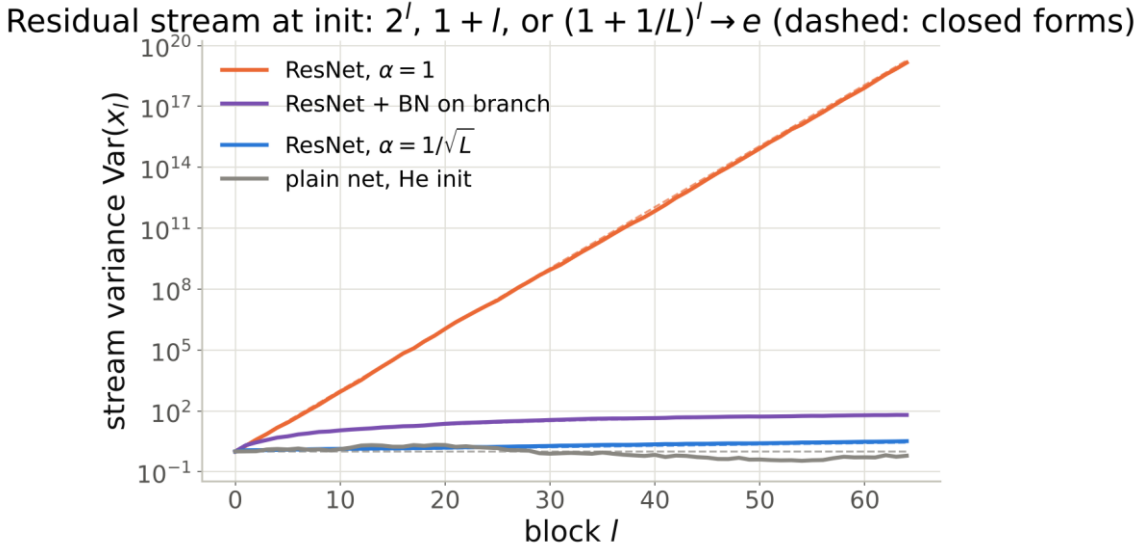
MNIST RMSprop, 300

Dashed lines (theory predictions), red (accuracy on benchmark).

Understanding the batch norm via variance calculations

Variances add as Var(x_l) = (1+α²)^l,

- **Unnormalized**: 2^l
- **Batch norm:** $1 + l$
- **Normalize by** $\frac{\mathbf{1}}{\sqrt{\boldsymbol{L}}}$: (1+1/L)^l → e



Norms help because it shrinks the branch

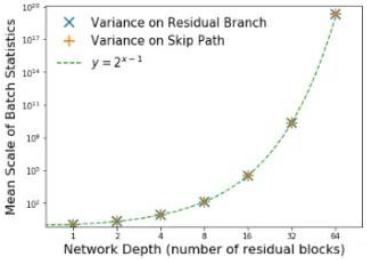
Batch Normalization Biases Residual Blocks Towards the Identity Function in Deep Networks

Soham De
DeepMind, London
sohamde@google.com

Samuel L. Smith
DeepMind, London
slsmith@google.com

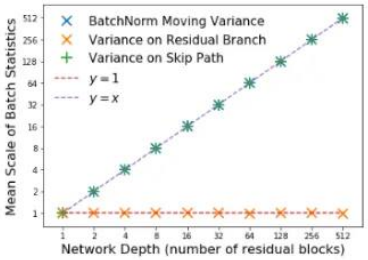
Why does batch norm (and layernorm) help us train much deeper models?

- Batchnorm descreases the relative contribution of residual blocks
- The stream's variance grows linearly (norm as sqrt)
- Batch norm keeps contributions O(1), so the relative contributions decay



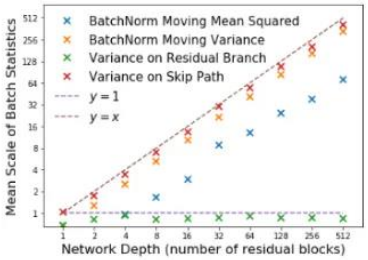
(a)

No Norm resnet
Gaussian data



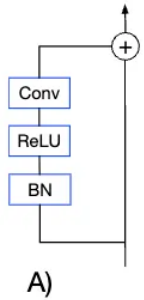
(b)

ResNet on
Gaussians

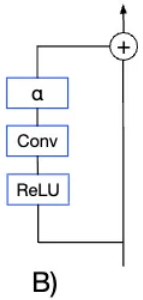


(c)

CIFAR residual
convnet



A)



B)

The pass/fail table

Early experiments into residual connection scaling (alpha)

Batch Normalization

Depth	Test accuracy	Learning rate
16	93.5 ± 0.1	2^{-1} (2^{-1} to 2^{-1})
100	94.7 ± 0.1	2^{-1} (2^{-2} to 2^{-0})
1000	94.6 ± 0.1	2^{-2} (2^{-3} to 2^{-0})

SkipInit ($\alpha = 1$)

Depth	Test accuracy	Learning rate
16	93.0 ± 0.1	2^{-2} (2^{-2} to 2^{-1})
100	–	–
1000	–	–

SkipInit ($\alpha = 1/\sqrt{d}$)

Depth	Test accuracy	Learning rate
16	93.0 ± 0.1	2^{-2} (2^{-2} to 2^{-1})
100	94.2 ± 0.1	2^{-1} (2^{-2} to 2^{-1})
1000	94.2 ± 0.0	2^{-1} (2^{-2} to 2^{-1})

Divide residual block by $\sqrt{2}$

Depth	Test accuracy	Learning rate
16	92.4 ± 0.1	2^{-2} (2^{-2} to 2^{-1})
100	88.9 ± 0.4	2^{-5} (2^{-5} to 2^{-5})
1000	–	–

SkipInit ($\alpha = 0$)

Depth	Test accuracy	Learning rate
16	93.3 ± 0.1	2^{-2} (2^{-2} to 2^{-2})
100	94.2 ± 0.1	2^{-2} (2^{-2} to 2^{-2})
1000	94.3 ± 0.2	2^{-2} (2^{-3} to 2^{-1})

SkipInit without L2 ($\alpha = 0$)

Depth	Test accuracy	Learning rate
16	89.8 ± 0.2	2^{-3} (2^{-3} to 2^{-3})
100	91.7 ± 0.2	2^{-2} (2^{-2} to 2^{-2})
1000	92.1 ± 0.1	2^{-2} (2^{-2} to 2^{-2})

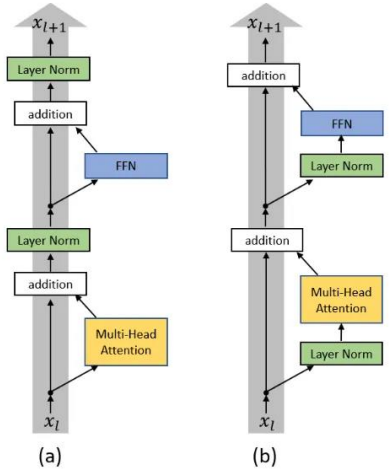
Initializing the models with small, learnable scalers work (even without batchnorm!)

Similar logic in LMs for LayerNorm: Pre-LN vs Post-LN

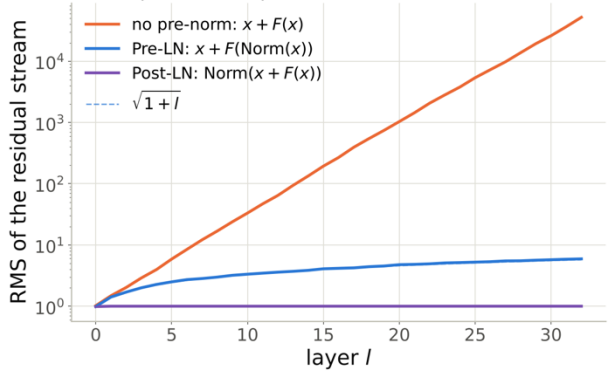
Pre-LN has a clean residual and the variance grows as l.

Post-LN normalises the stream, so each block multiplies the identity path by ‖x_l‖/‖x_l + F‖ ≈ 1/√(1 + Var F)

Notice that early signals are repeatedly attenuated by the norm layer, as we divide the residuals repeatedly



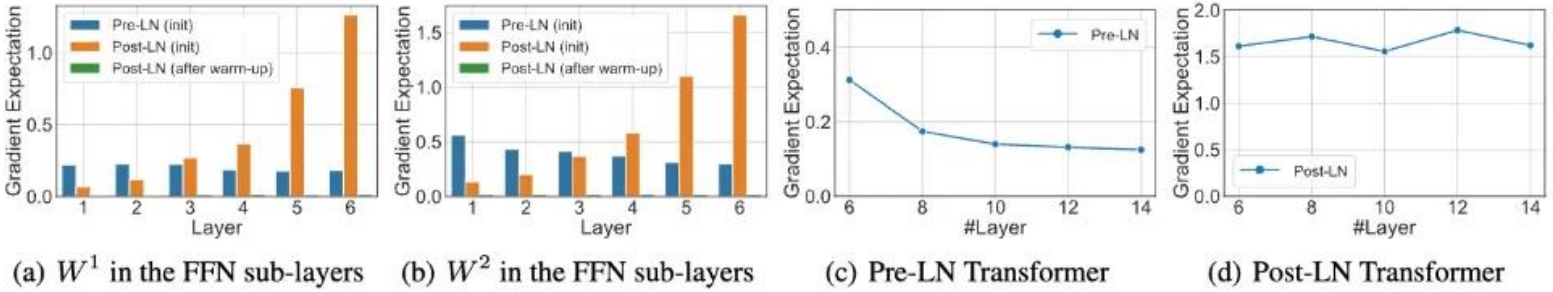
Random MLP-residual stack (width 512): where the normalization sits sets the stream scale



Gradients at init depend on norm placement

This ‘repeated division’ effect leads to (at certain inits), gradients concentrated at the end

Xiong et al note this gradient concentration, and link it to the need for warmup in their transformers



(a) W^1 in the FFN sub-layers (b) W^2 in the FFN sub-layers (c) Pre-LN Transformer (d) Post-LN Transformer

The Curse of Depth in Large Language Models

Wenfang Sun*[1], Xinyuan Song*[2], Pengxiang Li*[3], Lu Yin[4], Yefeng Zheng[1], Shiwei Liu[†5]
[1]Westlake University, China
[2]Emory University, USA
[3]Dalian University of Technology, China
[4]University of Surrey, UK
[5]University of Oxford, UK

Impact of layernorm on depth utilization

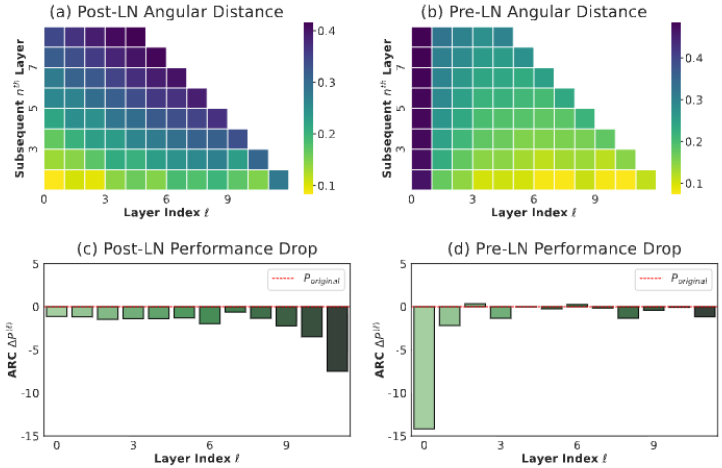


Figure 3: Results of in-house small-scale LLaMa-130M. **Angular Distance (a, b)**: Each column represents the angular distance from the initial layer ℓ (x-axis) and its subsequent n^{th} layer (y-axis). The distance is scaled to the range [0, 1], where yellow indicates smaller distances and purple indicates larger distances. **Performance Drop (c, d)**: ARC-e performance drop of removing each single layer from LLaMa-130M.

Post norm – Early layers are ‘redundant’

Pre norm – Late layers are more redundant

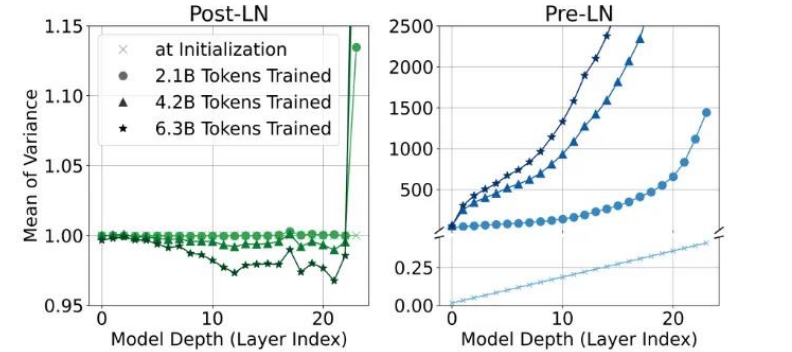
Prenorm enables training deeper models, but not without its limitations..

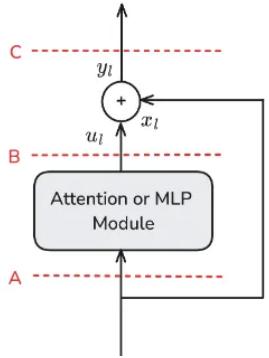
Other tricks – layernorms both before and after.

Peri-LN: Revisiting Normalization Layer in the Transformer Architecture

Jeonghoon Kim [1 2] Byeongchan Lee [2] Cheonbok Park [1 2] Yeontaek Oh [1] Beomjun Kim [2] Taehwan Yoo [1]
Seongjin Shin [1] Dongyoon Han [3] Jinwoo Shin [† 2] Kang Min Yoo [† 1]

Norms both before and after (common in models like olmo, gemma) controls output growth





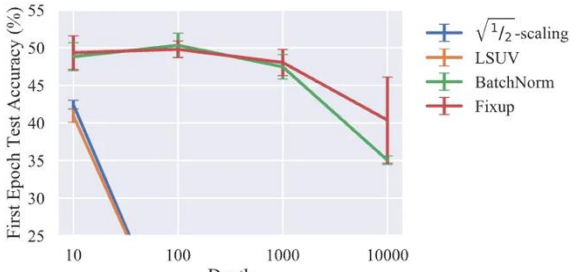
	A	B	C
Post-LN	×	×	✓
Pre-LN	✓	×	×
Peri-LN	✓	✓	×

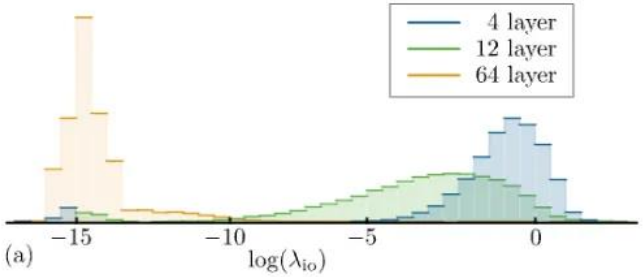
(One intuition – prenorm controls activation pass behavior, peri-norm also controls backwards)

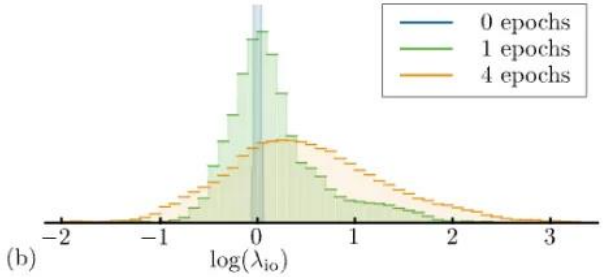
Beyond batchnorm.

Several 'batch norm free' architectures that have been successful.

- SkipInit (already covered): Learnable scalar ($\alpha = 0$), forcing identity at init.
- Fixup: Zero init each branch + learnable scalar + shrink the branch to force $O\left(\frac{\eta}{L}\right)$ update size.
- ReZero: Learnable scalar ($\alpha = 0$),







Summary

MuP-style analyses justify ‘keeping the residual an identity’ – this is stable!

Generally, in toy systems the predictions line up with reality on trainability

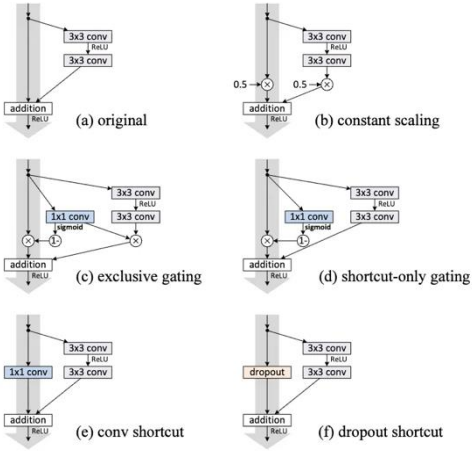
Some differences from the optimizer section is the focus on ‘c’ (representation of pairs of signal), and less of a focus on optimizer dynamics and behavior.

Variance tracking also lets us understand the role of the norms and their placement

Lecture today

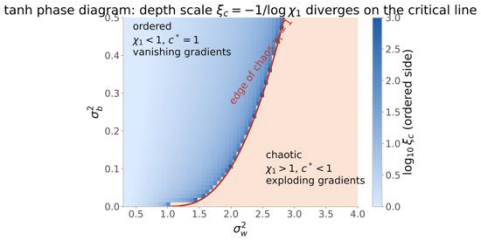
Part 1

Empirical analysis of ResNets



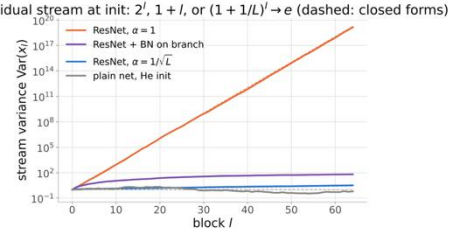
Part 2

Signal propagation theory



Part 3

Testing the theory



How can we test our ‘theory of signal propagation’

Current hypothesis:

Signal propagation involves controlling activation and gradient magnitudes in depth

If we have stable signal propagation, we can train a very very deep network.

(So far) Residual networks are an easy way of getting stable signal propagation

What about other, more nontrivial networks that have good signal propagation?

Residual less deep networks

Plain convnet

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]

Transformer language models

Published as a conference paper at ICLR 2023

DEEP TRANSFORMERS WITHOUT SHORTCUTS: MODIFYING SELF-ATTENTION FOR FAITHFUL SIGNAL PROPAGATION

Bobby He[1] **James Martens**[2] **Guodong Zhang**[2] **Aleksandar Botev**[2]
Andrew Brock[2] **Samuel L. Smith**[2] **Yee Whye Teh**[1,2]
[1]University of Oxford, [2]DeepMind
Correspondence to: bobby.he@stats.ox.ac.uk, jamesmartens@google.com.

One view: dynamical isometry

$$x_i^{l+1} = \sum_j g\, W_{ij}^{(l+1,l)}\, \phi(x_j^l),$$

Consider the “end to end” jacobian

$$J_{ij}^{N_l,1}(x^{N_l}) \equiv \left.\frac{\partial x_i^{N_l}}{\partial x_j^1}\right|_{x^{N_l}}$$

This represents how gradients at the last layer propagate to the earlier (or first) layer.

The spectrum of the Jacobian tell us how the matrix shrinks or explodes gradients across layers

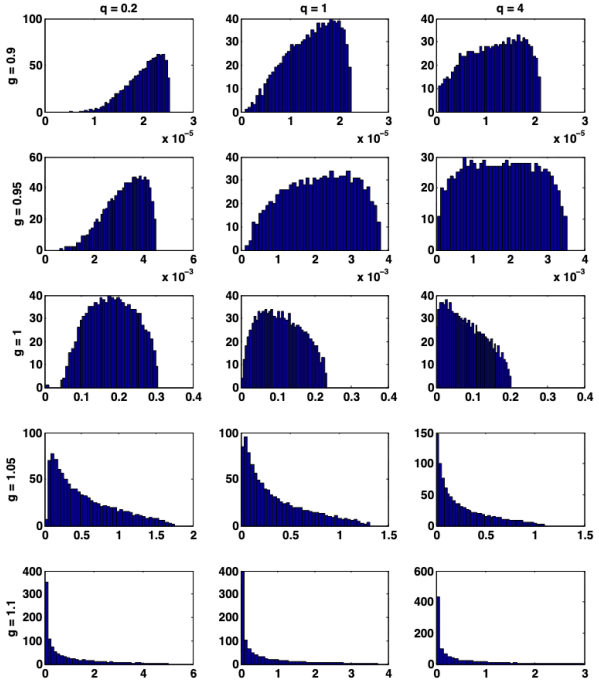
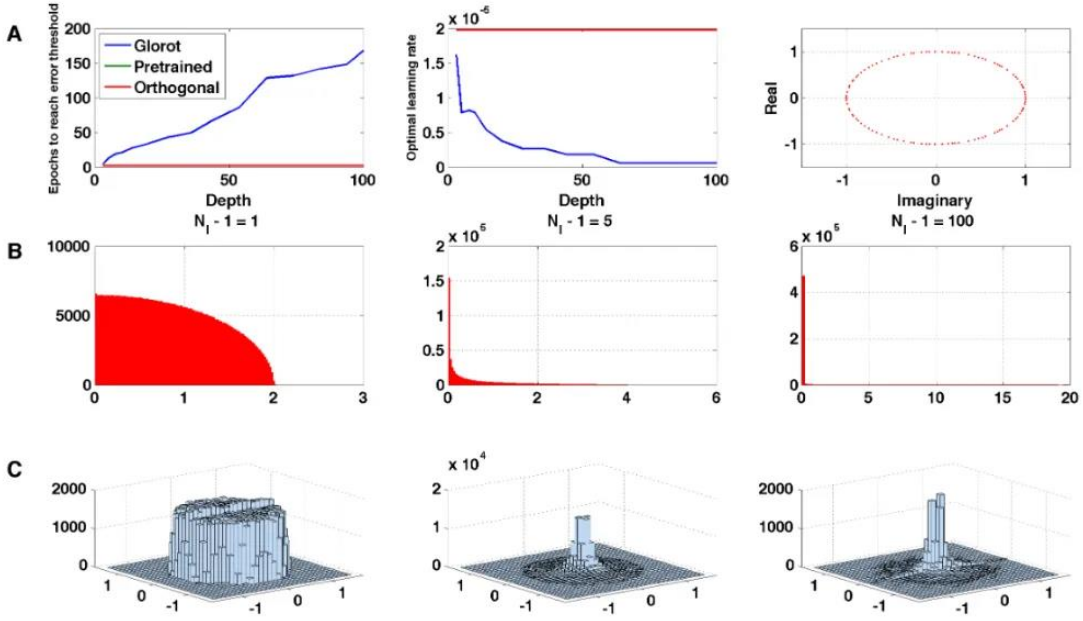


Figure 7: Singular value distribution of the end to end Jacobian, defined in (22), for various values of the gain g in (20) and the input layer population variance $q = q^1$ in (21). The network architecture consists of $N_l = 100$ layers with $N = 1000$ neurons per layer, as in the linear case in Fig. 6B.

Engineering the spectrum

There are choices, such as orthogonal weights which do maintain units Jacobian throughout



But this is a very very toy setting (random linear, or MNIST) experiments. Can this be made real?

Nonlinearities can be a key bottleneck

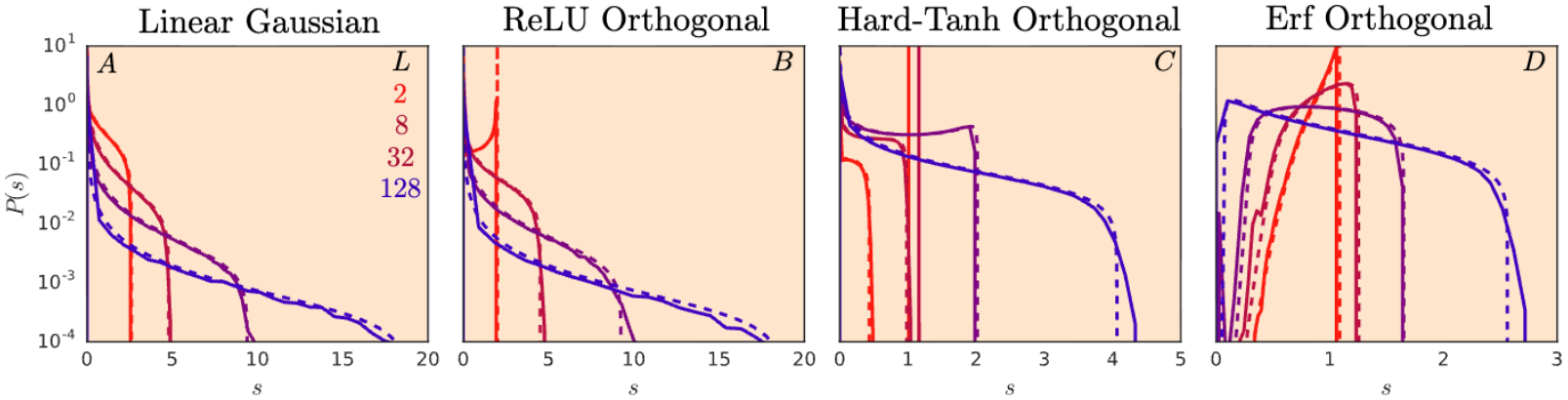
The Emergence of Spectral Universality in Deep Networks

Jeffrey Pennington
Google Brain

Samuel S. Schoenholz
Google Brain

Surya Ganguli
Google Brain
Applied Physics, Stanford University

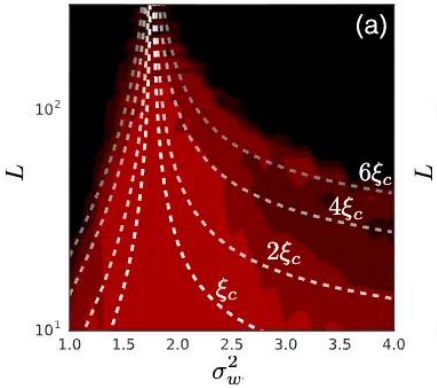
Even orthogonal matrices are maybe not enough, looking at the spectrum.
ReLU (unlike Tanh) can cause singular values to spread out – some explode and others vanish



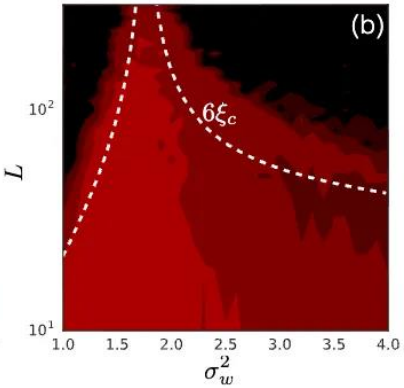
Key ideas for training a 10000 layer non-residual network

1. **Use a Tanh** – this has a linear region that can propagate gradients
2. **Keep preactivation small at init** – Keeps init near the linear region
3. **Pick weights and variances carefully** to be at ‘critical’ signal propagation
4. Carefully pick an orthogonal (or similar) initialization

MNIST SGD, 200



CIFAR SGD, 2k



Training a 10k layer network

It trains! And not too slowly

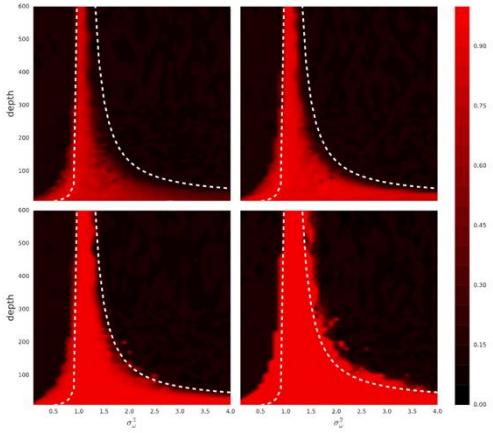
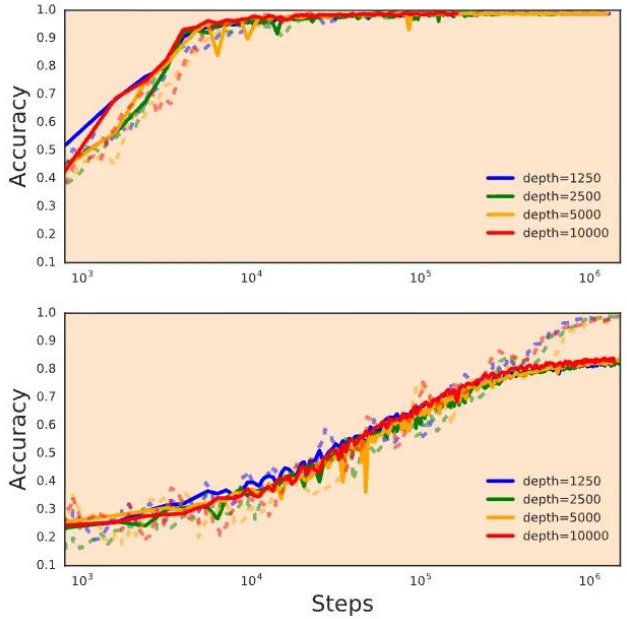


Figure 2. Mean field theory predicts the maximum trainable depth for CNNs. For fixed bias variance $\sigma_b^2 = 2 \times 10^{-5}$, the heat map shows the training accuracy on MNIST obtained for a given depth L network and weight variance σ_w, after (a) 500, (b) 2,500, (c) 10,000, and (d) 100,000 training steps. Also plotted (white dashed line) is a multiple ($6\xi_c$) of the characteristic depth scale governing convergence to the fixed point.



Caveats: no gain beyond 1.2k layers

Basically starts out by being a big fat linear model

Beyond ‘basically linear’

Further improvements possible..

- Start w/ resnet, remove the skips
- Remove batchnorm
- Use special, leakly relu w/ scale
- Orthogonal init + KFAC

Fundamentally similar tracking of variances / correlations

DEEP LEARNING WITHOUT SHORTCUTS: SHAPING THE KERNEL WITH TAILORED RECTIFIERS

Guodong Zhang[1,2], **Aleksandar Botev**[3], **James Martens**[3]
[1]University of Toronto, [2]Vector Institute, [3]DeepMind
gdzhang@cs.toronto.edu, {botev,jamesmartens}@google.com

ABSTRACT

Training very deep neural networks is still an extremely challenging task. The common solution is to use shortcut connections and normalization layers, which are both crucial ingredients in the popular ResNet architecture. However, there is strong evidence to suggest that ResNets behave more like ensembles of shallower networks than truly deep ones. Recently, it was shown that deep vanilla networks (i.e. networks without normalization layers or shortcut connections) can be trained as fast as ResNets by applying certain transformations to their activation functions. However, this method (called Deep Kernel Shaping) isn’t fully compatible with ReLUs, and produces networks that overfit significantly more than ResNets on ImageNet. In this work, we rectify this situation by developing a new type of transformation that is fully compatible with a variant of ReLUs – Leaky ReLUs. We show in experiments that our method, which introduces negligible extra computational cost, achieves validation accuracies with deep vanilla networks that are competitive with ResNets (of the same width/depth), and significantly higher than those obtained with the Edge of Chaos (EOC) method. And unlike with EOC, the validation accuracies we obtain do not get worse with depth.

Attention is not *all* you need:
pure attention loses rank doubly exponentially with depth

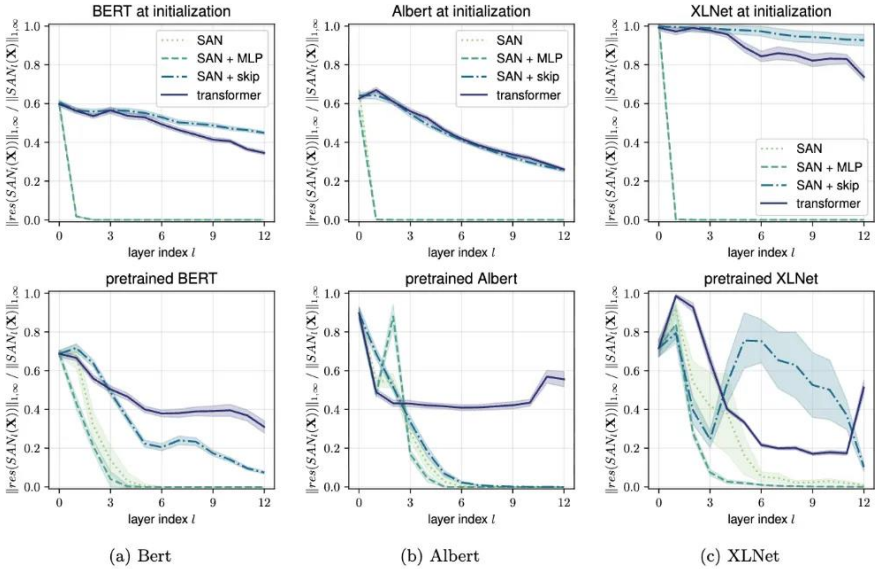
Yihe Dong
Google
yihed@google.com

Jean-Baptiste Cordonnier
EPFL
jean-baptiste.cordonnier@epfl.ch

Andreas Loukas
EPFL
andreas.loukas@epfl.ch

Back to language models + transformers..

Skipless transformers are surprisingly hard – attention alone is 'contractive'
Rapidly collapses the token representations together (the correlation c goes to 1)



(a) Bert (b) Albert (c) XLNet

SAN - Network without MLP/skip

Walking through a simplified attention

Very simplified toy model of attention (at init)

- For a layer l, we have that $X_l = A_l X_{l-1}$, with the attention matrix $A = \sigma\left(\frac{QK^\top}{\sqrt{d_k}} + B\right)$
- The token-token linear gram matrix is $\Sigma_l = \frac{1}{d} X_l X_l^\top$
- We can unroll this as $\Sigma_L = (A_L \dots A_1)\Sigma_0(A_L \dots A_1)^\top$

For this simple model – can we control the behavior of Σ_l? Yes, via the Cholesky Decomp.

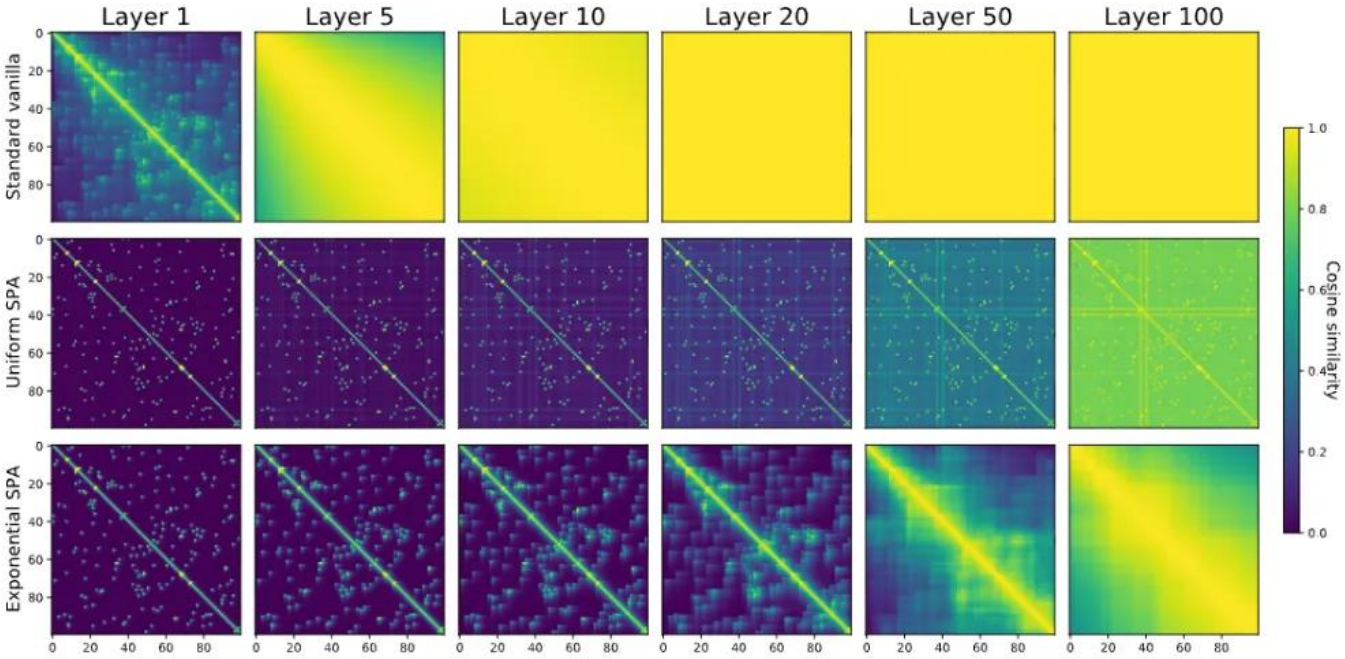
- Let $\Sigma_l = L_\ell L_\ell^\top$ be the Cholesky factors of the matrix Σ
- Initialize $A_\ell = L_\ell L_{\ell-1}^{-1}$
- Then the telescoping product above gives exactly the desired Σ

(The real method is more complicated.. Since $\mathrm{L}_\ell \mathrm{L}_{\ell-1}^{-1}$ is not a valid self-attention)

Initializing to force ‘reasonable’ attention maps in depth

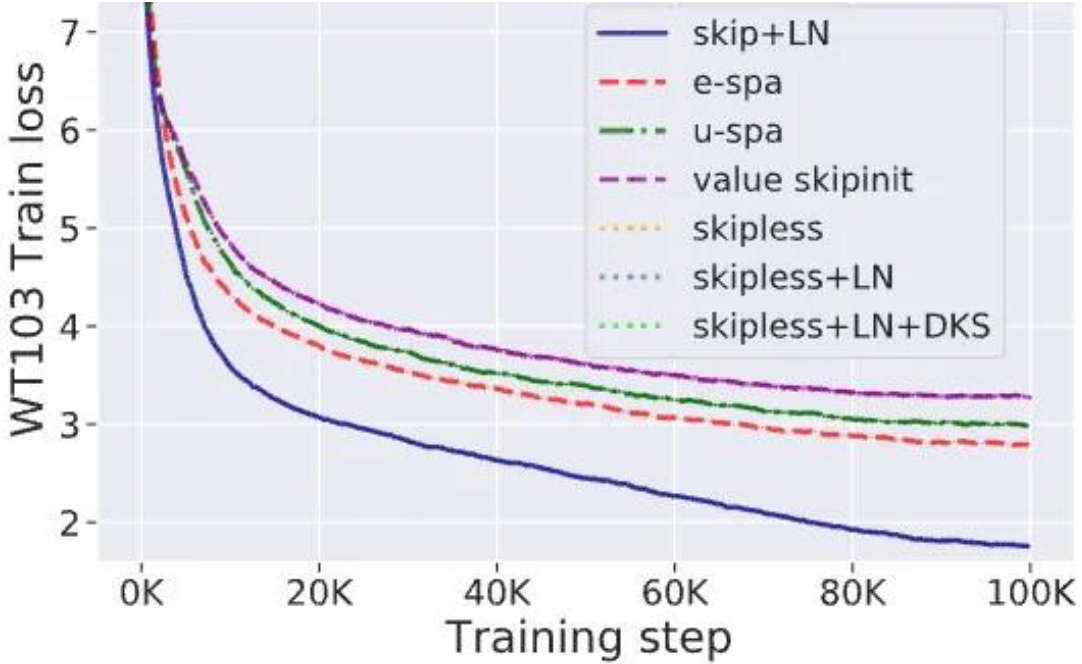
Solve for attention matrices A_l = L_l L_{l−1}^{-1} that realize a target kernel.

In this case, pick a pattern that is exponentially decaying with token



Skipless transformers train…

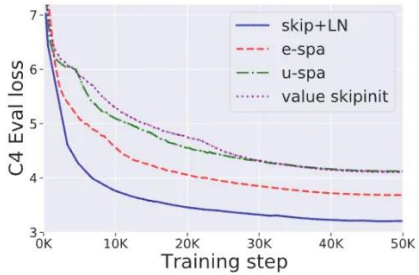
A 36-block transformer without skips goes from untrainable (loss ≈ 7.5) to 2.79 on WikiText-103 with these tricks



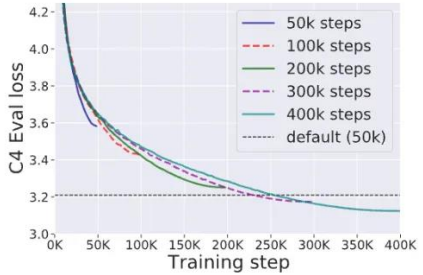
…about five times slower

Skipless E-SPA needs ~5 × the iterations to match Pre-LN's training loss

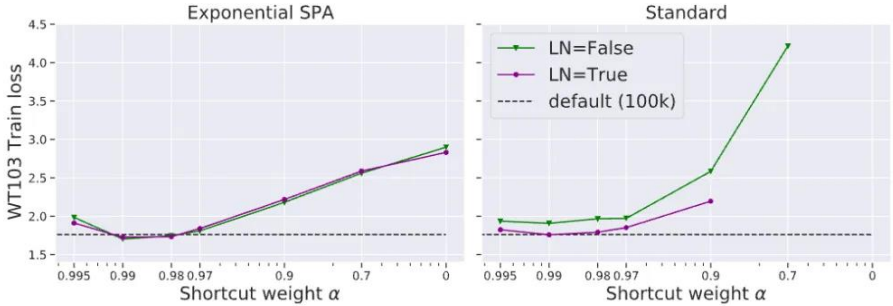
Very far from useful, but surprising that this is possible at all!



(a) Validation loss of vanilla transformers.



(b) Longer training.



Summary

Signal propagation theory (kind of) works – training *deep skipless* models

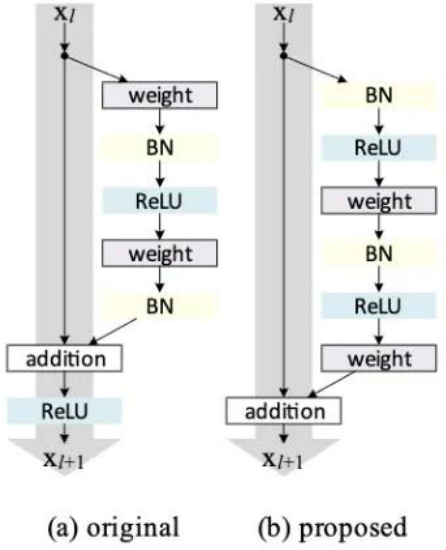
The prescription is basically ‘track the Jacobian’ – Keep things close to the identity

This is both hard and tractable

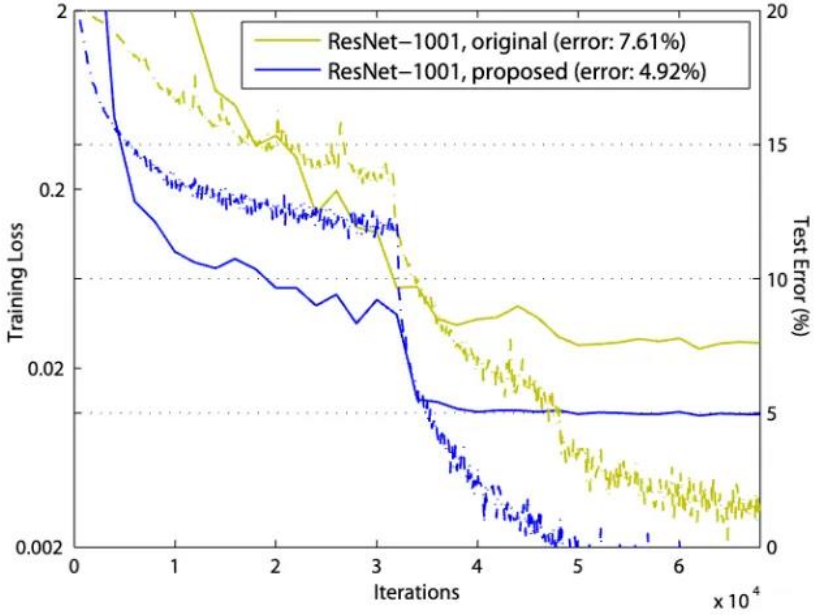
- **Hard** – this is the ‘end to end’ jacobian. We have to reason about multiple layers
- **Tractable** – only care about the init + local behaviors

Can we improve upon the residual?

Current logic: the closer to 'identity' you are, the better



(a) original (b) proposed



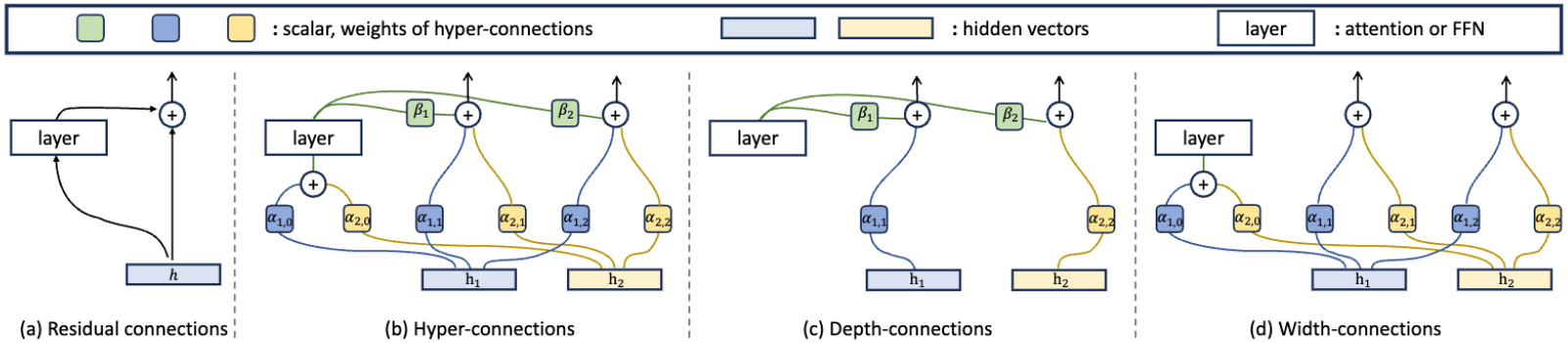
But is this *strictly* true? What about some emerging / modern residual ideas in LLMs?

Published as a conference paper at ICLR 2025

Core idea: have ‘more’ residual streams

HYPER-CONNECTIONS

Defa Zhu, Hongzhi Huang, Zihao Huang, Yutao Zeng, Yunyao Mao, Banggu Wu, Qiyang Min, Xun Zhou
Seed-Foundation-Model Team, ByteDance
{zhudefa,huanghongzhi.51,huangzihao.notabot,yutao.zeng,
maoyunyao.myy,wubanggu,minqiyang,zhouxun}@bytedance.com



Hyper-connections:

- Have multiple ‘residual streams’
- Streams have multiplicative weights on how information mixes in and out
- Weights can be learned, data independent scalars or data dependent gates
- From Kaiming’s paper – gating is risky! Have to be careful with gate weights.

Training tricks..

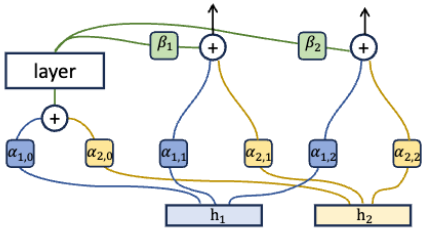
Initialization

Pick the residual mixing A = I at initialization (unit jacobian)

Pick one residual stream for input mixing.

Pick output mixing B=1 at init (equal write into all residuals)

This reduces to a initialization at the pre-norm residual network.



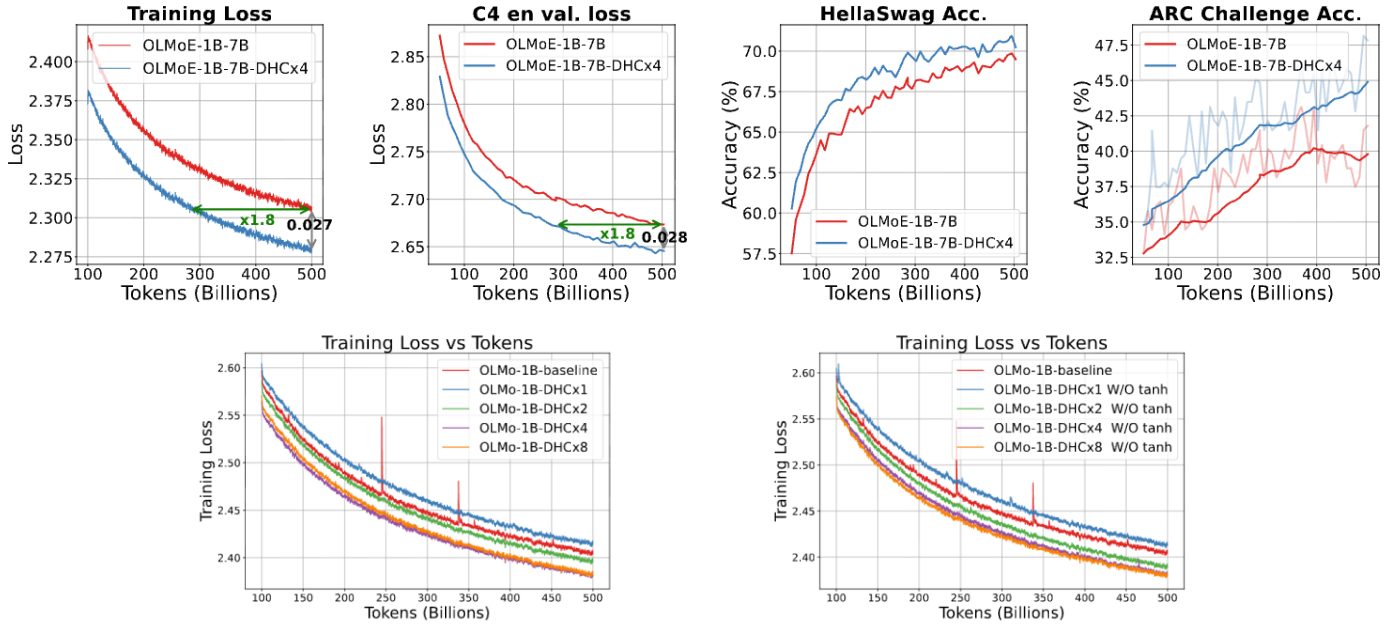
(b) Hyper-connections

Learning

Constrain the ‘learned matrix’ to be close to the init

$$\overline{\mathbf{H}} = \texttt{norm}(\mathbf{H})$$
$$\mathcal{B}(\mathbf{H}) = s_\beta \circ \texttt{tanh}(\overline{\mathbf{H}}\mathbf{W}_\beta)^\intercal + \mathbf{B} \in \mathbb{R}^{1\times n}$$
$$\mathcal{A}_m(\mathbf{H}) = s_\alpha \circ \texttt{tanh}(\overline{\mathbf{H}}\mathbf{W}_m) + \mathbf{A}_m \in \mathbb{R}^{n\times 1}$$
$$\mathcal{A}_r(\mathbf{H}) = s_\alpha \circ \texttt{tanh}(\overline{\mathbf{H}}\mathbf{W}_r) + \mathbf{A}_r \in \mathbb{R}^{n\times n}$$

Hyper connections in action



Faster training of a mixture of experts model (olmoe).

Gains with more residual streams

But, hyperconnections can be unstable

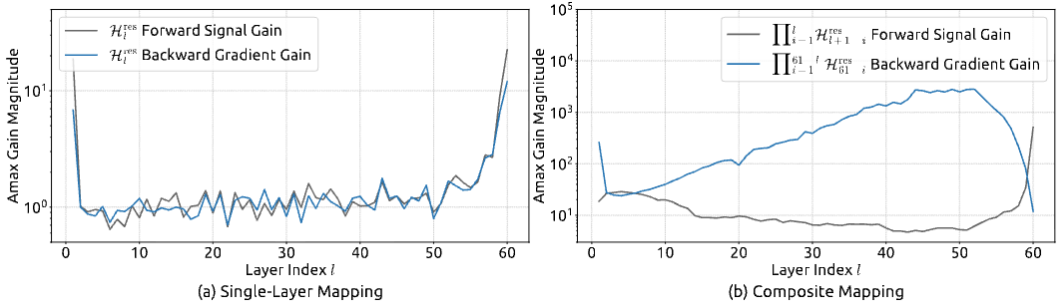
deepseek

*m*HC: Manifold-Constrained Hyper-Connections

Zhenda Xie*†, Yixuan Wei*, Huanqi Cao*,
Chenggang Zhao, Chengqi Deng, Jiashi Li, Damai Dai, Huazuo Gao, Jiang Chang,
Kuai Yu, Liang Zhao, Shangyan Zhou, Zhean Xu, Zhengyan Zhang, Wangding Zeng,
Shengding Hu, Yuqing Wang, Jingyang Yuan, Lean Wang, Wenfeng Liang
DeepSeek-AI

However, as the training scale increases, HC introduces potential risks of instability. The primary concern is that the unconstrained nature of HC compromises the identity mapping property when the architecture extends across multiple layers. In architectures comprising multiple parallel streams, an ideal identity mapping serves as a conservation mechanism. It ensures that the average signal intensity across streams remains invariant during both forward and backward propagation. Recursively extending HC to multiple layers via Eq. (3) yields:

$$\mathbf{x}_L = \left(\prod_{i=1}^{L-l} \mathcal{H}_{L-i}^{\text{res}}\right) \mathbf{x}_l + \sum_{i=l}^{L-1} \left(\prod_{j=1}^{L-1-i} \mathcal{H}_{L-j}^{\text{res}}\right) \mathcal{H}_i^{\text{post}\,\top} \mathcal{F}(\mathcal{H}_i^{\text{pre}} \mathbf{x}_i, \mathcal{W}_i), \tag{4}$$



(a) Single-Layer Mapping (b) Composite Mapping

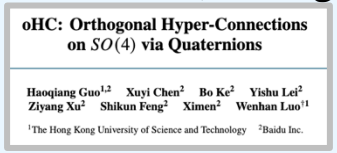
Even with all this, training hyperconnections can result in ‘uneven’ gradient magnitudes

Hyperconnections with signal propagation

How can we maintain signal propagation?

- Want to preserve the norm and relationships of vectors (Orthogonal)?

  No, though there is a paper about this



oHC: Orthogonal Hyper-Connections on $SO(4)$ via Quaternions

Haoqiang Guo[1,2] Xuyi Chen[2] Bo Ke[2] Yishu Lei[2]
Ziyang Xu[2] Shikun Feng[2] Ximen[2] Wenhan Luo[†1]

[1]The Hong Kong University of Science and Technology [2]Baidu Inc.

- Use a doubly stochastic matrix: at least one eigenvalue is equal to 1, rest <= 1.

$$X = \frac{1}{12}\begin{pmatrix} 7 & 0 & 5 \\ 2 & 6 & 4 \\ 3 & 6 & 3 \end{pmatrix}$$

Prevents blow-up, though some directions may vanish over layers.

Manifold hyperconnections propagate signal

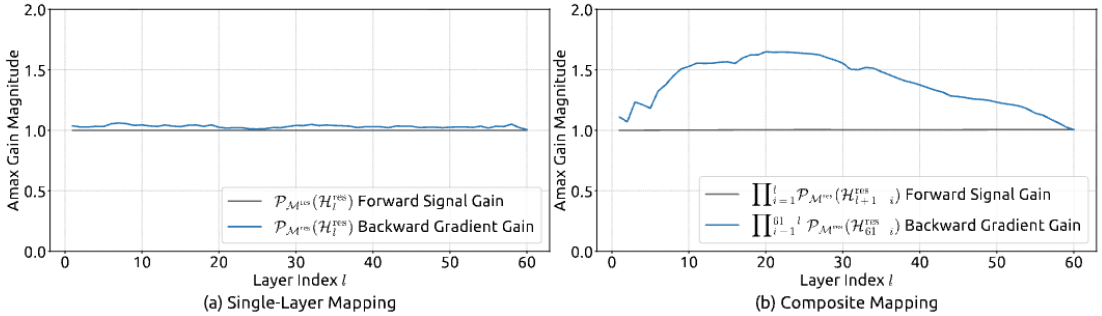
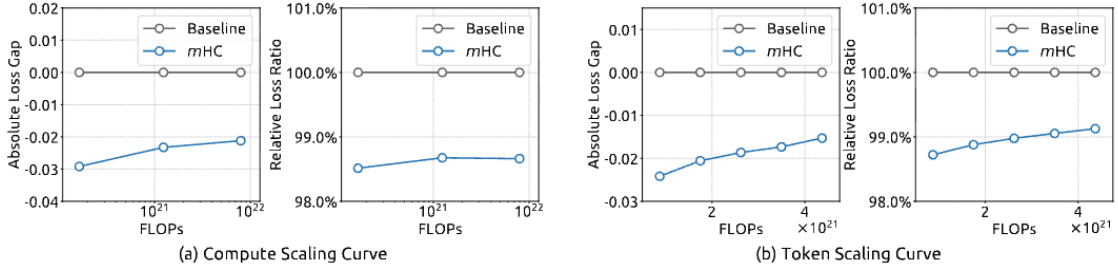


Figure 7 | **Propagation Stability of Manifold-Constrained Hyper-Connections (*m*HC).** This figure illustrates the propagation dynamics of (a) the single-layer mapping $\mathcal{P}_{\mathcal{M}^{\text{res}}}(\mathcal{H}_l^{\text{res}})$ and (b) the composite mapping $\prod_{i=1}^{L-l} \mathcal{P}_{\mathcal{M}^{\text{res}}}(\mathcal{H}_{L-i}^{\text{res}})$ within the 27B model. The results demonstrate that *m*HC significantly enhances propagation stability compared to HC.

The same propagation diagrams now look quite well behaved with the doubly stochastic matrix constraint (the ‘birkhoff polytope’)

Overall impact of this change



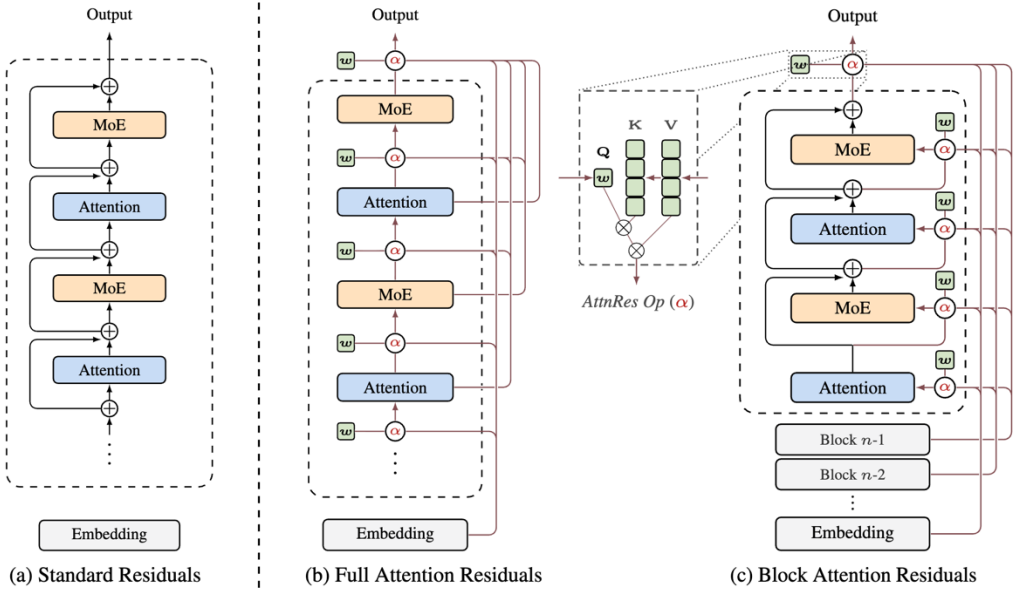
2.2. Manifold-Constrained Hyper-Connections

As shown in Figure 2, DeepSeek-V4 series incorporate Manifold-Constrained Hyper-Connections (*m*HC) (Xie et al., 2026) to strengthen the conventional residual connections between adjacent Transformer blocks. Compared with naive Hyper-Connections (HC) (Zhu et al., 2025), the core idea of *m*HC is to constrain the residual mapping onto a specific manifold, and thus enhance the stability of signal propagation across layers while preserving model expressivity. This subsection briefly introduces the standard HC and describes how we design *m*HC for stable training.

Great scaling in the mHC paper, and used in Deepseek v4!

Last quick trick: attention residuals

Allow for weighted averages from the distant past



ATTENTION RESIDUALS

TECHNICAL REPORT OF ATTENTION RESIDUALS

Kimi Team

https://github.com/MoonshotAI/Attention-Residuals

$$\boldsymbol{h}_l = \alpha_{0\to l}\cdot \boldsymbol{h}_1 + \sum_{i=1}^{l-1} \alpha_{i\to l}\cdot f_i(\boldsymbol{h}_i)$$

$$\alpha_{i\to l} = \frac{\phi(\boldsymbol{q}_l, \boldsymbol{k}_i)}{\sum_{j=0}^{l-1}\phi(\boldsymbol{q}_l, \boldsymbol{k}_j)}$$

$$\boldsymbol{q}_l = \boldsymbol{w}_l, \qquad \boldsymbol{k}_i = \boldsymbol{v}_i = \begin{cases} \boldsymbol{h}_1 & i = 0 \\ f_i(\boldsymbol{h}_i) & 1 \le i \le l-1 \end{cases}$$

Query is a learned vector per layer. Key and values are the ‘residual output’

Details with attention residuals

No longer identity init

Suggested initialization is *uniform* over previous layers. Very much not identity

Could enable ‘dead layers’

Softmax could concentrate on a few layers, or skip layers

In practice, seems to works well

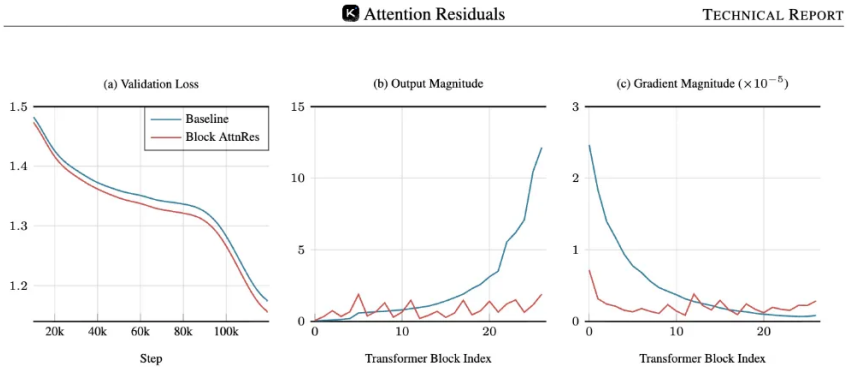


Table 4: Ablation on key components of AttnRes (16-layer model).

Variant		Loss
Baseline (PreNorm)		1.766
DenseFormer [36]		1.767
mHC [59]		1.747
AttnRes	Full	1.737
	w/ input-dependent query	1.731
	w/ input-independent mixing	1.749
	w/ sigmoid	1.741
	w/o RMSNorm	1.743
	SWA ($W = 1 + 8$)	1.764
	Block ($S = 4$)	1.746
	w/ multihead ($H = 16$)	1.752
	w/o RMSNorm	1.750

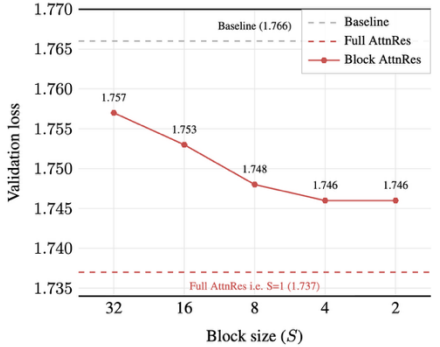


Figure 6: Effect of block size on validation loss (16-layer model).

Recap

Deep learning is about training deep networks

But as we've seen, this can be surprisingly subtle.

Residual networks are important for you to understand

Empirically, one of the core innovations enabling 'easy' training

Theory wise, one of the few things we understand well

Some tools, like variance tracking, end-to-end Jacobians

Simplicity of depth-wise layers lead to simple tools for understanding

But not fully predictive! (AttnRes, mHC, etc)