DeepMind

Rapid training of deep neural networks without normalization, RELUs, or skip connections

Lead: James Martens

Collaborators: Andy Ballard, Guillaume Desjardins, Greg Swirszcz, Valentin Dalibard, Jascha Sohl-Dickstein, Sam Schoenholz

19/11/2021

The ubiquity of deep neural networks

- Deep neural networks are an essential component of most modern machine learning systems, such as:
  - Reinforcement Learning agents playing games
  - Machine translation systems and language models
  - Vision systems
  - Speech recognition
  - Search and recommendation systems
  - etc
- While practitioners have come up with many heuristic innovations to make them train faster at higher depths, theory has been relatively slow to catch up, and is rarely able to make an impact on practice

Current state of affairs

- Fast training of deep nets requires some combination of:
  - normalization layers (e.g. Batch Normalization [BN], Layer Normalization)
  - skip connections
  - specific choices for activation functions (e.g. ReLU, SELU)
- These come with various problems:
  - mechanism of action not well understood
  - not clear how to use them in new architectures
  - BN causes issues by sharing information over the batch
  - skip connections impose constraints on model specification
  - *more speculatively:* these techniques may be acting as a crutch, and our reliance on them could be holding us back from pushing DL theory and practice to the next level

Our contributions

- We develop **Deep Kernel Shaping (DKS)**, a general automated framework for transforming neural networks models to make them easier to train
- DKS enables rapid training of networks that are traditionally considered hard/impossible to train, including:
  - very deep vanilla convnets (without BN layers or skip connections)
  - networks with unpopular activation functions (e.g. tanh or softplus)
  - ***your new proposed model goes here***
- We also provide a comprehensive explanation for why things like ReLUs, BN layers and skip connections speed up training, and show how DKS makes them unnecessary (see the paper)

Network architecture assumptions

- DKS supports:
  - fully-connected, convolutional, pooling, element-wise nonlinear, weighted sum, and layer norm layers
  - Gaussian fan-in and orthogonal initializations (of the “Delta” type)
  - most types of weight sharing, such as in RNNs
  - arbitrary topologies with branches, multiple heads, etc

- ***Simplifying assumptions for this talk:***
  - only fully-connected, element-wise nonlinear, and sum layers
  - network input vectors x have norm $\sqrt{\dim(x)}$

Kernel function approximations for randomly initialized networks

- Let $f(x)$ be the vector output for our network, given input x. ***At initialization***, it turns out that we can closely approximate both

$$\frac{\|f(x)\|^2}{\dim(f(x))} \quad \text{and} \quad \frac{f(x)^\top f(x')}{\|f(x)\|\|f(x')\|}$$

Approximation becomes exact as layer widths grow

using only knowledge of the network's structure and the 3 scalar quantities:

$$\|x\|^2/\dim(x)\,, \qquad \|x'\|^2/\dim(x') \qquad \text{and} \qquad x^\top x'/(\|x\|\|x'\|)$$

- Approximated quantities are called **q values** and **c values** (resp.), and they are computed using **Q maps** and **C maps:**

$$Q_f(q) \approx \frac{\|f(x)\|^2}{\dim(f(x))} \qquad C_f(c) \approx \frac{f(x)^\top f(x')}{\|f(x)\|\|f(x')\|}$$

$$q \approx \|x\|^2/\dim(x) \qquad c \approx x^\top x'/(\|x\|\|x'\|)$$

Computing Q and C maps

- Q and C maps for linear layers are just the *identity function*. For a nonlinear layer f with activation ϕ they are given by:

$$Q_f(q) = \mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi(\sqrt{q}\,x)^2] \qquad C_f(c) = \frac{1}{Q_f(q)}\,\mathbb{E}_{\left[\begin{smallmatrix} v_1 \\ v_2 \end{smallmatrix}\right] \sim \mathcal{N}\left(0, \left[\begin{smallmatrix} 1 & c \\ c & 1 \end{smallmatrix}\right]\right)}[\phi(\sqrt{q}\,v_1)\,\phi(\sqrt{q}\,v_2)]$$

- To compute Q and C maps for whole networks, we compute them for component subnetworks and then compose:

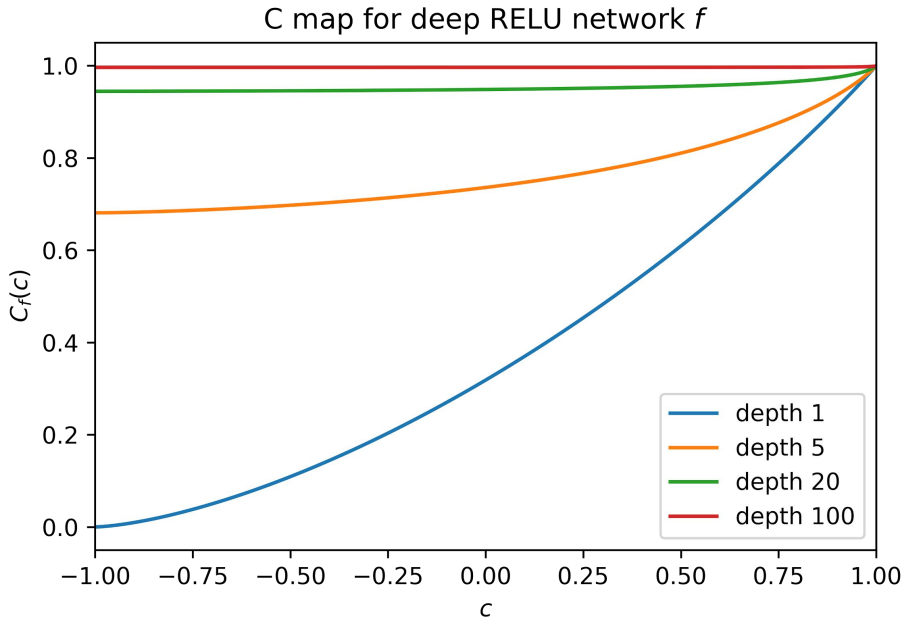
$$Q_{f \circ g}(c) = Q_f(Q_g(c)) \qquad C_{f \circ g}(c) = C_f(C_g(c))$$

- Weighted sum operations are handled using:

$$q_{\text{out}} = w_1^2\,q_1 + \cdots + w_n^2\,q_n \qquad c_{\text{out}} = (w_1^2\,q_1\,c_1 + \cdots + w_n^2\,q_n\,c_n) \,/\, q_{\text{out}}$$

C map degeneration in deep networks

- C maps determine the angles between output vectors as a function of the angle between input vectors (subject to approximation error)
- In deep networks, C maps can easily become “degenerate”, so that information about input angles is obscured:



Degenerate C maps imply difficult training

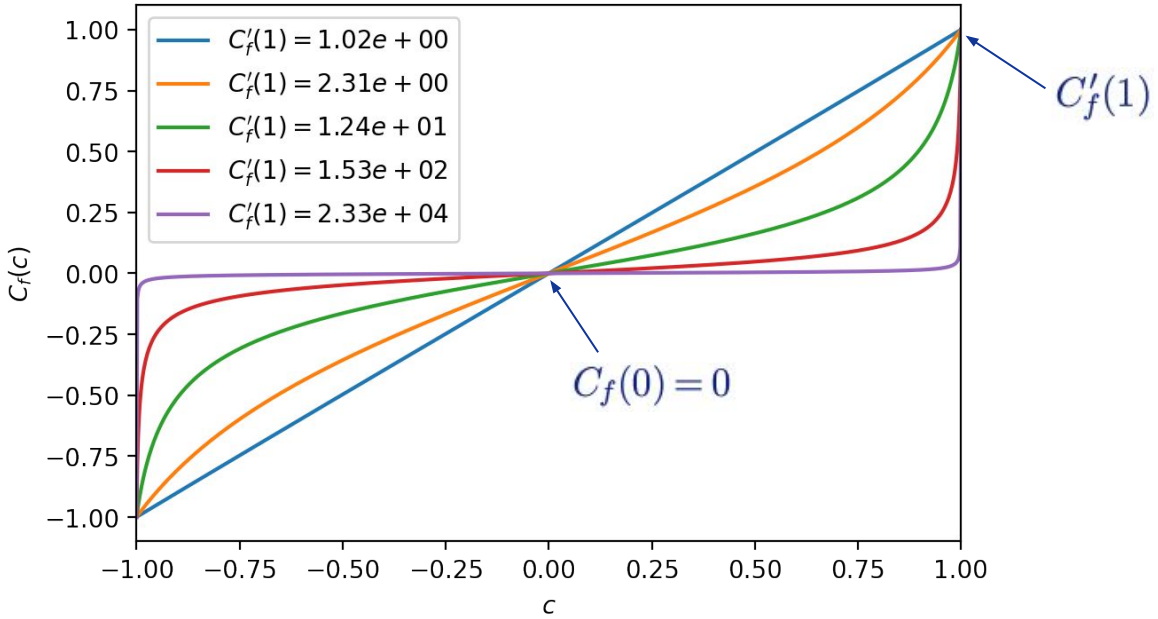
- The degenerate C maps seen in deep networks will squash entire ranges of input c values tightly around some output value c_0
- There are two basic cases for this, *both of which are bad for training*:
  - $c_0 \ll 1$: output vectors look “random”, and generalization is impossible. Early layers will have huge gradients compared to later layers.
  - $c_0 \approx 1$: all output vectors are essentially identical. Gradients of early layers vanish, and loss surface becomes very ill-conditioned.
- This is formalized in the paper using NTK theory

A previous solution: Initializing on the Edge of Chaos (EOC) (Schoenholz et al., 2016)

- One solution to this problem is to require $C_f'(1) = 1$ for each nonlinear layer f
- This is accomplished by setting the variance of the initial weights and biases
- This slows the asymptotic convergence of c values (to 1) with depth
- Unfortunately, given a deep enough network they will still be pretty close to fully-converged, meaning that the network's C map will still be degenerate
- For example, unmodified deep ReLU networks already satisfy the EOC condition (as long as the biases are zero), but they still have degenerate C maps (as seen in previous slide). Indeed, deep ReLU networks still require skip connections and BN layers to work well in practice

A new way to control C map properties

- C maps are convex on $[0, 1]$. Intuitively, this allows us to control the overall deviation of the C map from the identity by controlling the deviation of $C_f'(1)$ from 1, assuming $C_f(0) = 0$. (A rigorous argument is given in the paper.)



More rigorously...

Theorem 13 *For any subnetwork f we have*

$$\frac{1}{4}(1 - C'_f(0)) \leqslant \max_{c\in[-1,1]} |C_f(c) - c| \leqslant 2(1 - C'_f(0))$$

and

$$\max_{c\in[-1,1]} |C'_f(c) - 1| \leqslant 2(1 - C'_f(0)) + (C'_f(1) - 1).$$

If $C_f(0) = 0$, then we additionally have that

$$\max_{c\in[-1,1]} |C_f(c) - c| \leqslant 2(C'_f(1) - 1)$$

and

$$\max_{c\in[-1,1]} |C'_f(c) - 1| \leqslant 3(C'_f(1) - 1).$$

A different failure mode: networks that are “nearly linear”

- We require non-degenerate C maps for training to go well, but this isn’t always enough
- Linear networks have nice C maps but their model class is very limited
- “Nearly linear” networks also have nice C maps, but may be hard to optimize

  e.g. for each ReLU activation, add 10^{10} to its input and subtract 10^{10} from its output

  → Model class is *technically* the same as a standard ReLU network, but optimizer will struggle to find nonlinear behavior

Target Q/C map properties

- In DKS we will enforce the following properties:
  a) $Q_f(1) = 1$ for all subnetworks f → standardizes map computations across layers, prevents issues with huge/tiny inputs to final loss
  b) $Q'_f(1) = 1$ for all subnetworks f → controls kernel approximation error
  c) $C_f(0) = 0$ for all subnetworks f → prevents C map degeneration
  d) $C'_f(1) \leqslant \zeta$ for all subnetworks f for a moderate ζ → prevents C map degeneration
  e) $\min_g C'_g(1)$ is maximized → prevents "nearly linear" networks

- For (a), (b) & (c), it's good enough to have them hold for all nonlinear layers (and to "normalize" any weighted sums in the network)
- For (d) & (e), it's sufficient that $C'_f(1) = \chi_1$ for each nonlinear layer f, where χ_1 is special constant we can compute from the model architecture

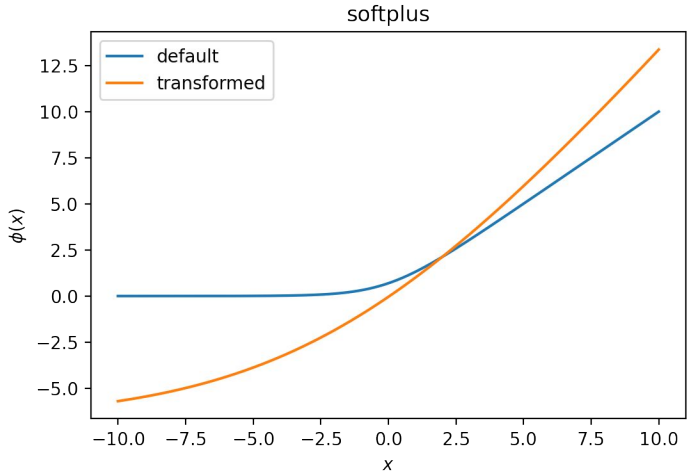
Transforming activation functions

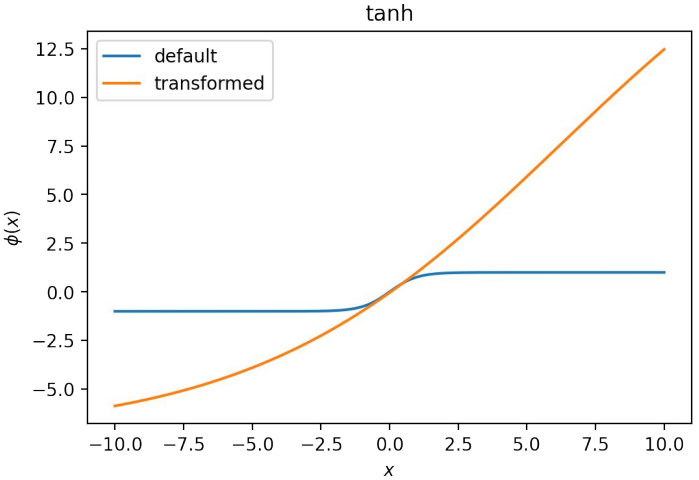
- To achieve these conditions, we introduce non-trainable scale and shift constants (for both input and output) to each activation function:

$$\phi(x) \quad \longrightarrow \quad \gamma\left(\phi(\alpha x + \beta) + \delta\right)$$

- Because careful choices for weights and biases can simulate the same thing, *this won't change the model class*

Examples:





Consequences of using DKS on the NTK

- The Neural Tangent Kernel tells us a lot about how a network will behave under gradient descent training
- Is an exact theory in the wide layer limit
- The following theorem describes the effect on using DKS on the NTK function Θ_i:

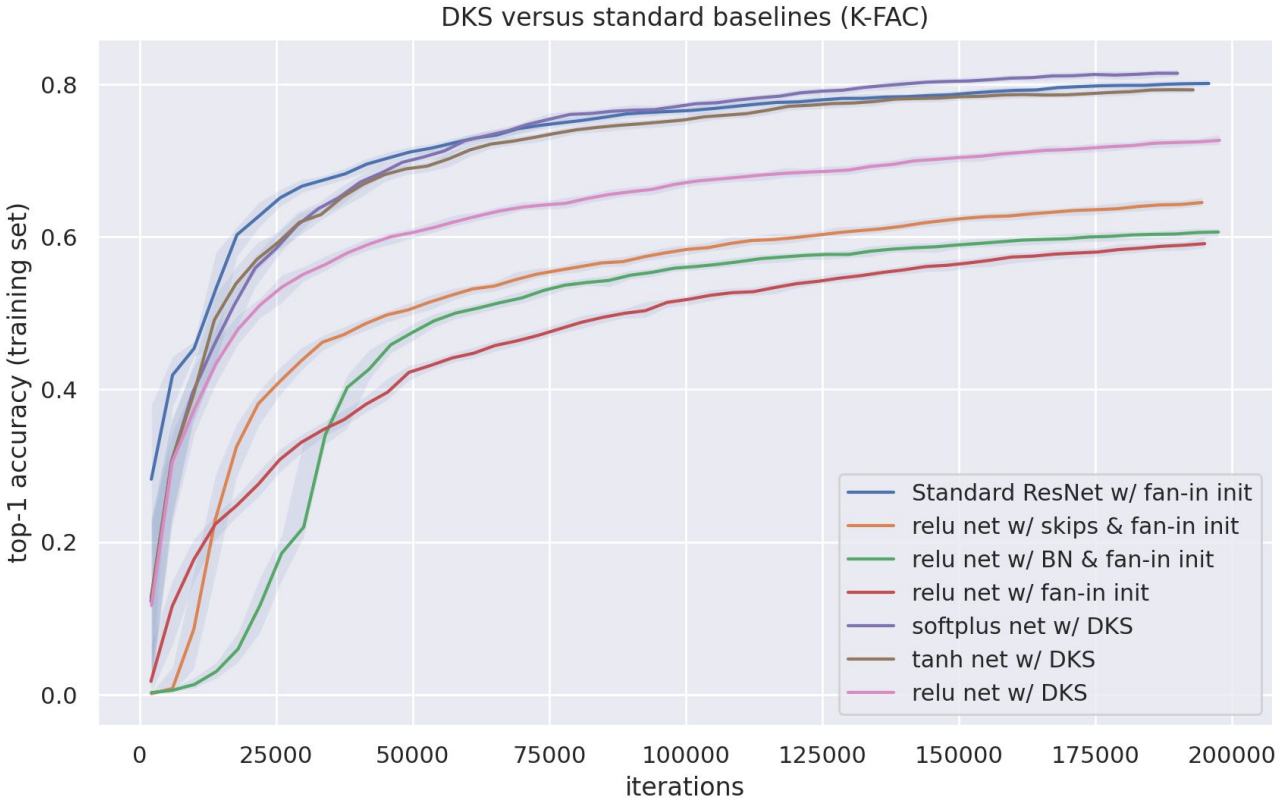
Theorem 27 *Suppose that Θ_i is the per-layer NTK (for layer i) of a network conforming to the assumptions of Section 24.1 which has been transformed using DKS with global slope bound ζ. Then we have*

$$\left|\Theta_i(x, x') - \frac{1}{d_0} x^\top x'\right| \leqslant 11(\zeta - 1).$$

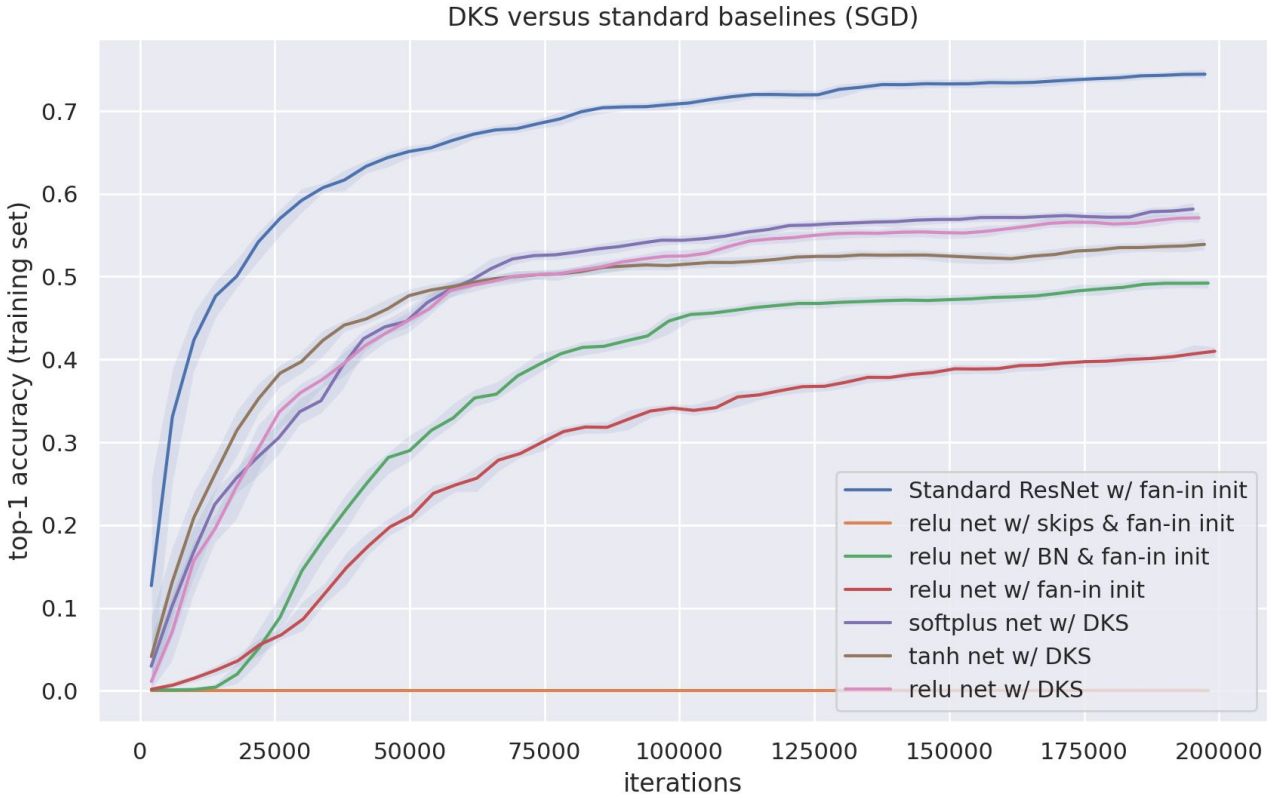
Experimental setup

- We trained a ResNet-101-V2 style architecture on Imagenet, with and without Batch Normalization layers and skip connections
- Batch size was 512
- Learning rate schedules were optimized dynamically using FIRE PBT (Dalibard et al., 2021) to maximize optimization speed
- Other optimization hyperparameters were lightly tuned
- **Lots** of additional experiments and ablations in the paper

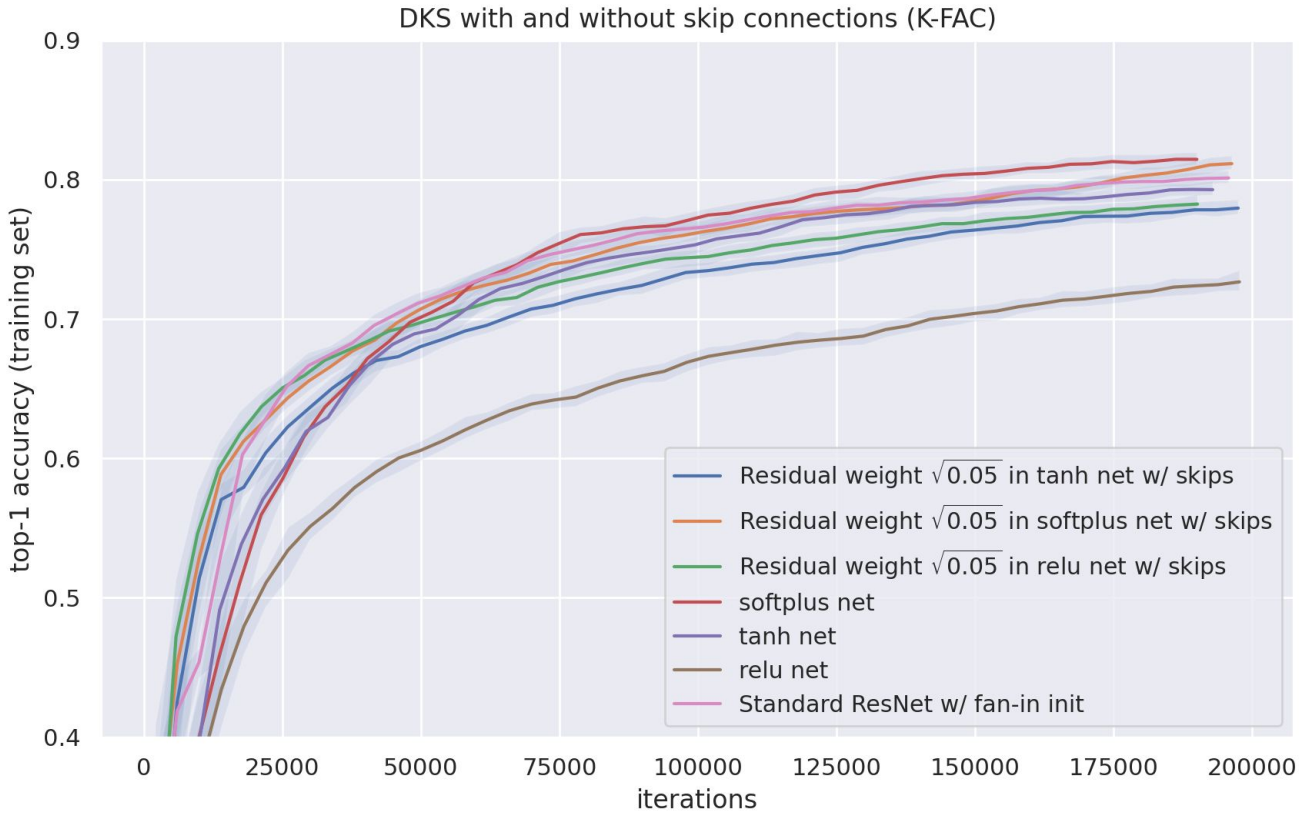
Vanilla networks w/ DKS compared to ResNets (K-FAC)



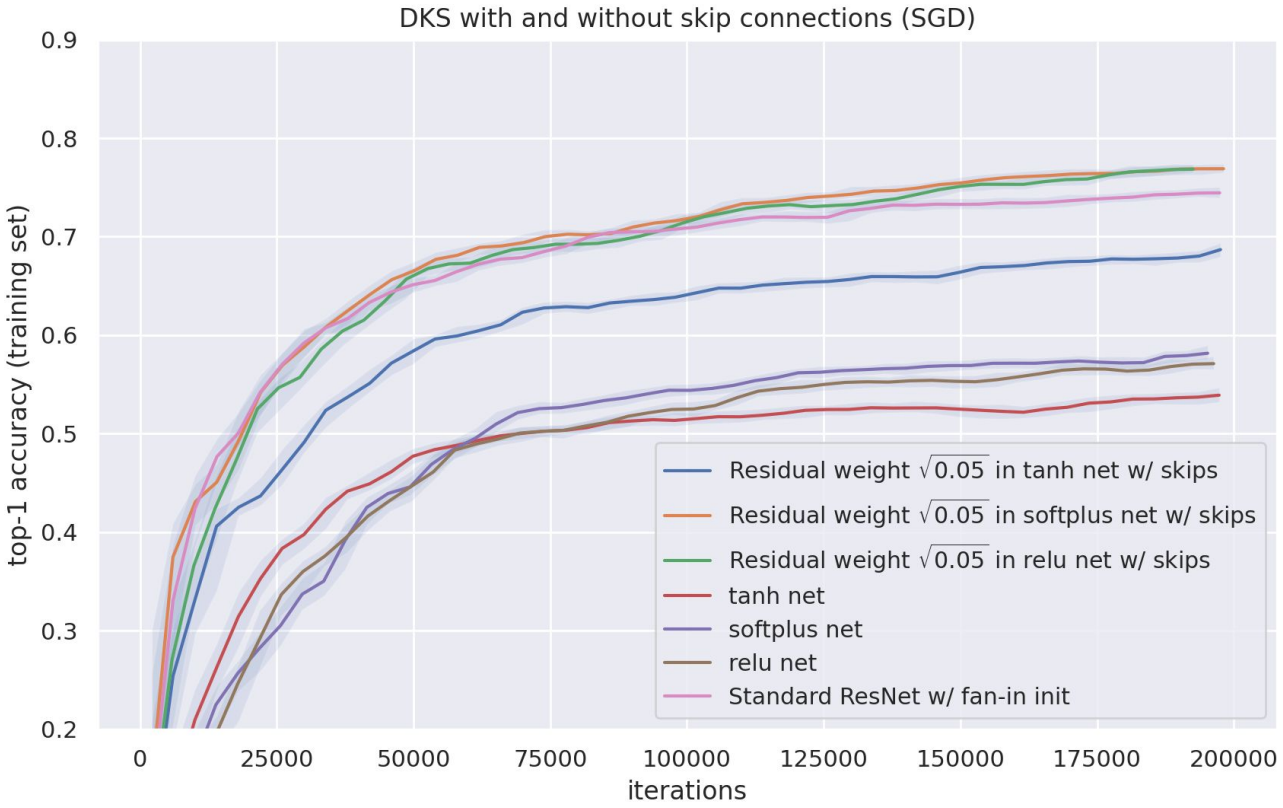
Vanilla networks w/ DKS compared to ResNets (SGD)



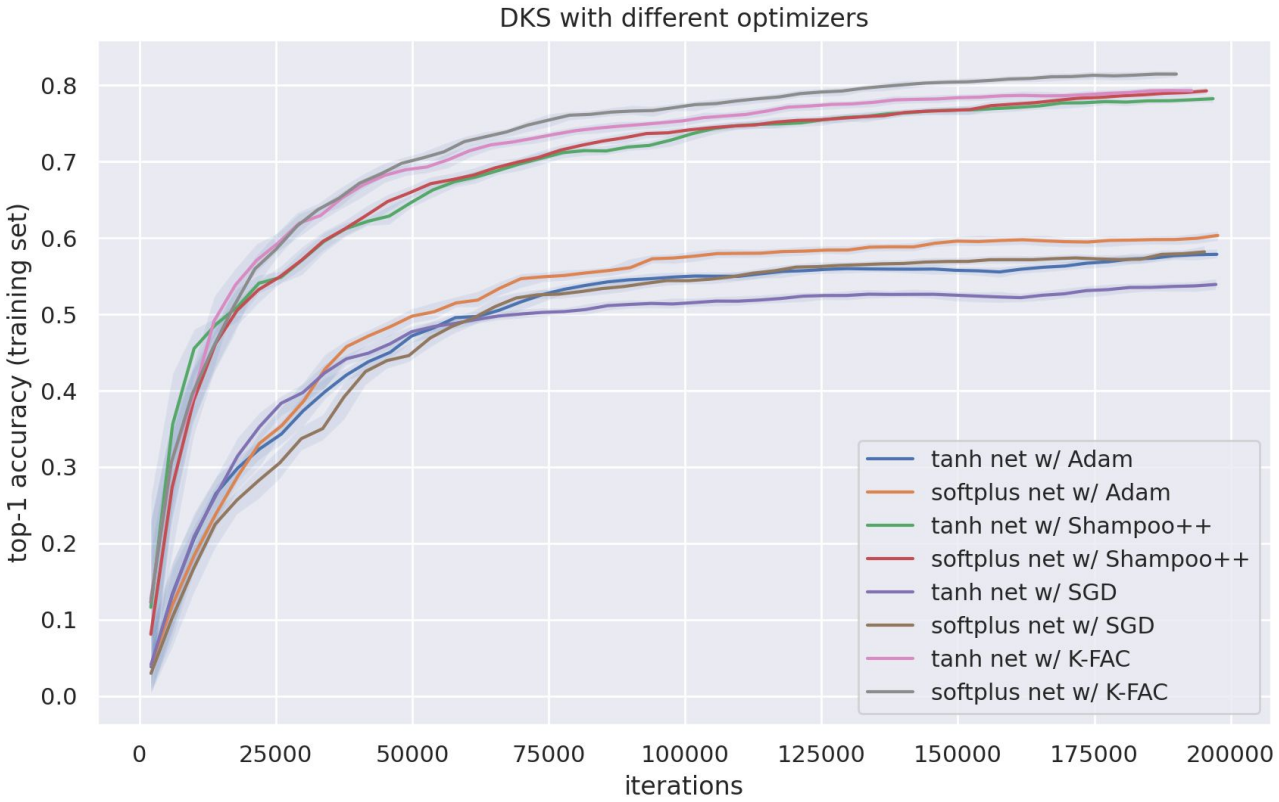
Networks with skip connections and DKS (K-FAC)



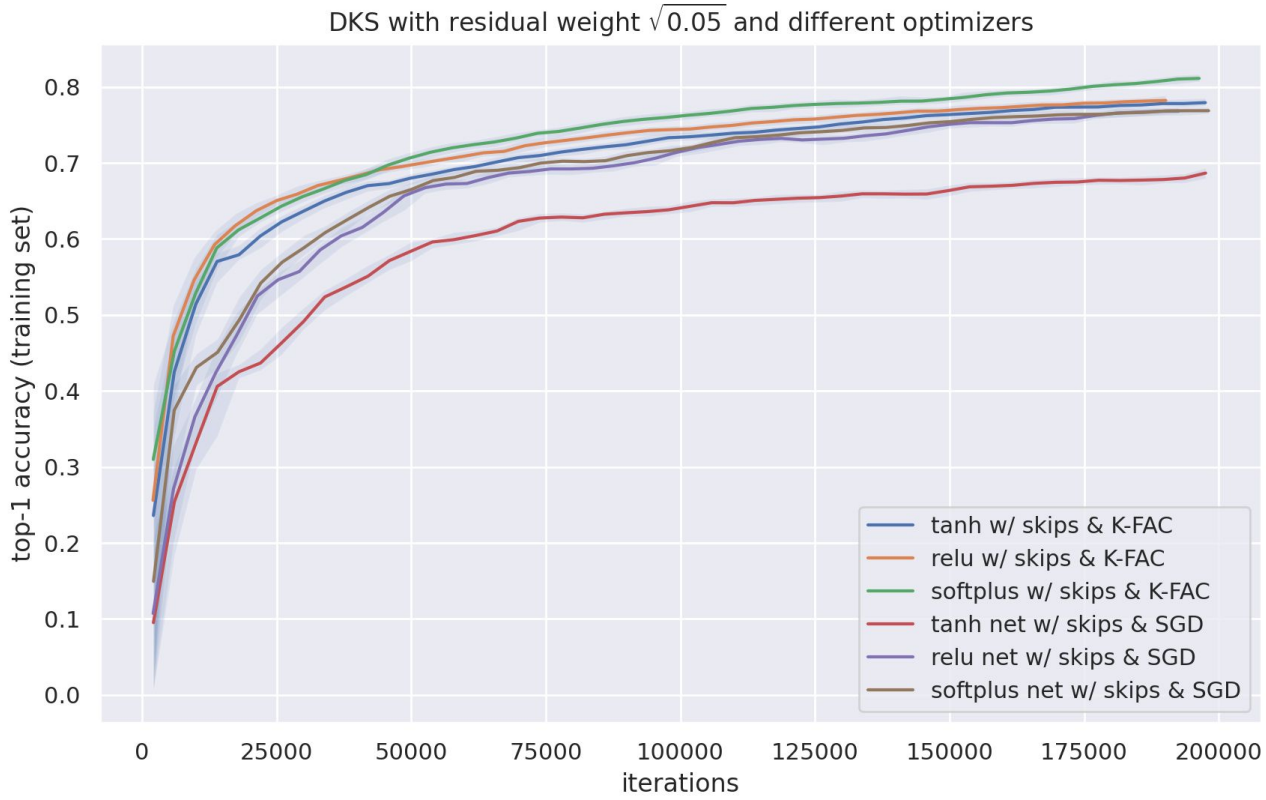
Networks with skip connections and DKS (SGD)



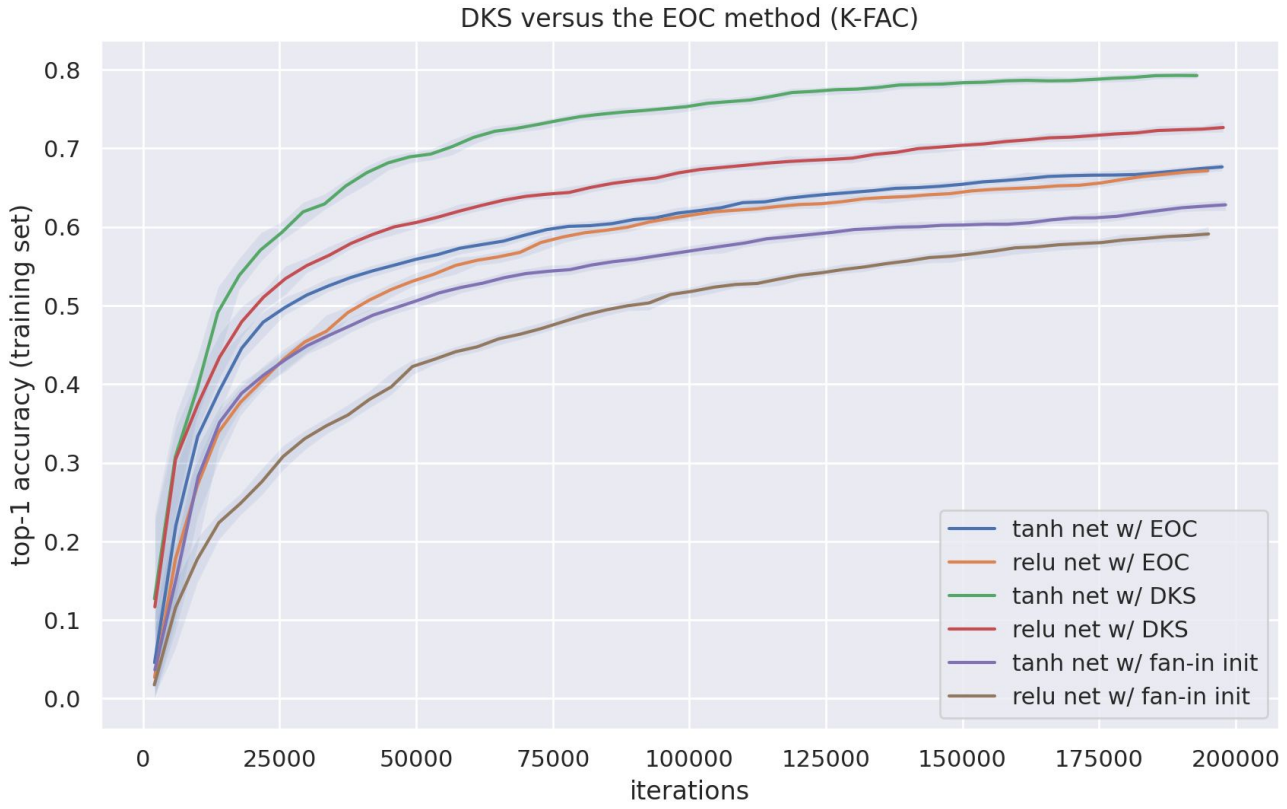
The effect of different optimizers on vanilla networks w/ DKS



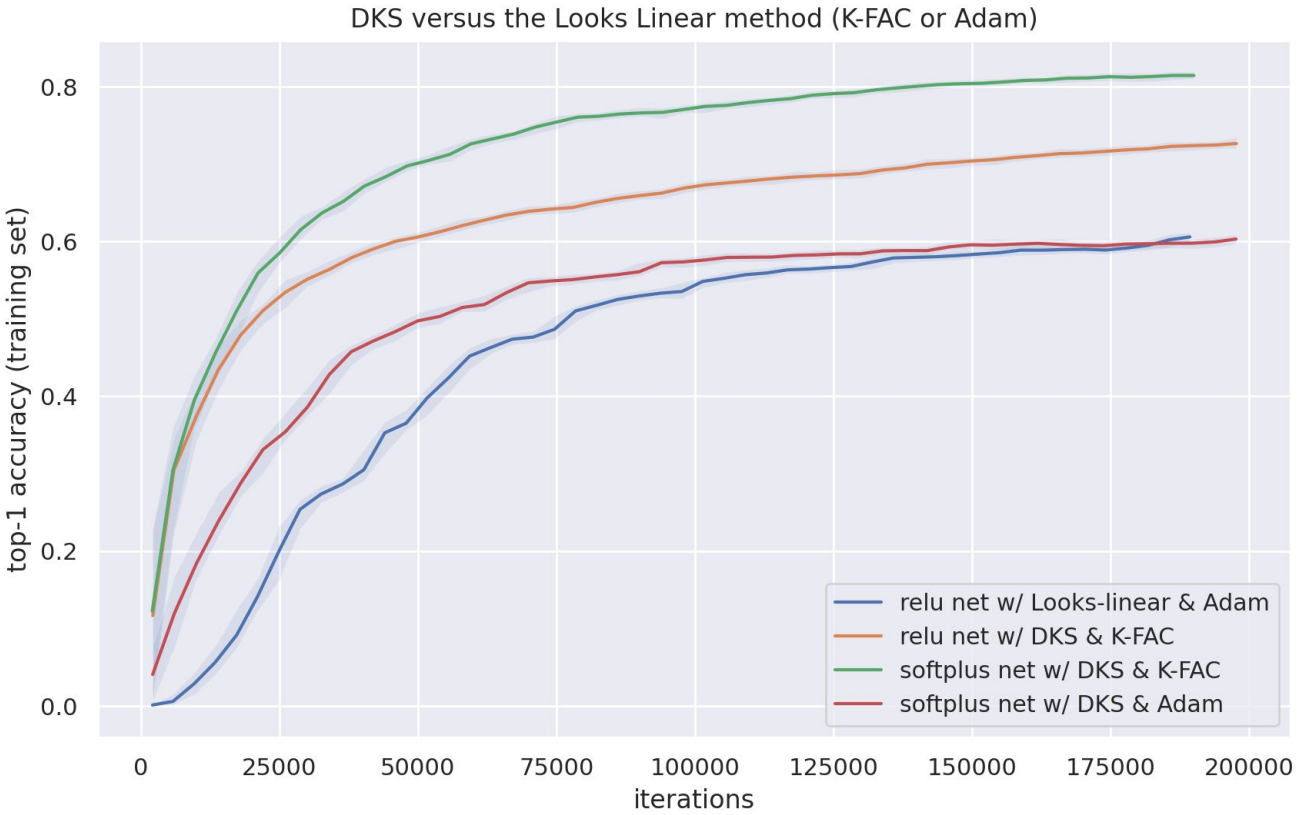
The effect of different optimizers on networks w/ DKS + skip connections



Comparisons to EOC



Comparisons to Looks Linear



Limitations

- Multiplicative units (as in Transformers) are not supported, but an extension in future work seems possible
- Vanilla networks using DKS seem to generalize worse than standard ResNets, although this has been largely addressed in follow-up work submitted to ICLR 2022
- To match the training speed of ResNets using purely vanilla networks we currently require K-FAC. With SGD we require at least 2x more iterations

Outlook

- DKS could be a useful tool for:
    - unlocking new model classes,
    - enable existing models that have optimization issues to train better,
    - elimination of deep learning “tricks” like batch normalization and skip connections where these cause problems, or can’t be used.

Selected related work

- B. Poole, S. Lahiri, M. Raghu, J. Sohl-Dickstein, and S. Ganguli. **Exponential expressivity in deep neural networks through transient chaos.** Advances in neural information processing systems, 29:3360–3368, 2016.
- A. Daniely, R. Frostig, and Y. Singer. **Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity.** Advances In Neural Information Processing Systems, 29:2253–2261, 2016
- S. S. Schoenholz, J. Gilmer, S. Ganguli, and J. Sohl-Dickstein. **Deep information propagation.** In International Conference on Learning Representations, 2017.
- D. Balduzzi, M. Frean, L. Leary, J. Lewis, K. W.-D. Ma, and B. McWilliams. **The shattered gradients problem: If resnets are the answer, then what is the question?** In International Conference on Machine Learning, pages 342–350. PMLR, 2017.
- A. Jacot, C. Hongler, and F. Gabriel. **Neural tangent kernel: Convergence and generalization in neural networks.** In Advances in neural information processing systems, 2018.
- L. Xiao, Y. Bahri, J. Sohl-Dickstein, S. Schoenholz, and J. Pennington. **Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks.** In International Conference on Machine Learning, pages 5393–5402, 2018.