DeepMind

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

James Martens

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

19/9/2022

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 change inductive bias of model (in ways which may be undesirable, depending on the application),
  - *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 so that they have better properties at *initialization time*, which makes them easier to train
- DKS enables rapid training of networks that are traditionally considered hard/impossible to train, including:
    - very deep vanilla convnets (without normalization layers or skip connections), when combined with power ***approximate natural gradient methods like K-FAC***
    - networks with unpopular activation functions (e.g. tanh or softplus)
    - *your new proposed model goes here*
- In the paper we also provide a comprehensive explanation for why things like ReLUs, normalization layers and skip connections speed up training, and show how DKS makes them mostly* unnecessary

Network architecture assumptions

- In general, DKS supports:
  - fully-connected, convolutional, pooling, weighted sum, layer norm, and element-wise nonlinear layers (which must be preceded by a linear layer)
  - Gaussian fan-in and orthogonal initializations (both of the "Delta" type), with zero-initialized biases
  - most types of weight sharing, such as in RNNs
  - arbitrary topologies with branches, multiple heads, etc

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

Kernel function approximations for randomly initialized networks

- Let $f(x)$ compute 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:

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

- We will call the approximations **"q values"** and **"c values"** (resp.). They are computed by functions called **"Q maps"** and **"C maps":**

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

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

Q and C map definitions

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

$$Q_g(q) = \mathbb{E}_{z\sim\mathcal{N}(0,1)}[\phi(\sqrt{q}\,z)^2] \qquad C_g(c) = \frac{1}{Q_g(q)}\mathbb{E}_{\left[\begin{smallmatrix} z_1 \\ z_2 \end{smallmatrix}\right]\sim\mathcal{N}\left(0,\left[\begin{smallmatrix} 1 & c \\ c & 1 \end{smallmatrix}\right]\right)}[\phi(\sqrt{q}\,z_1)\,\phi(\sqrt{q}\,z_2)]$$

  (Note we can treat the q value in the above C map equation as an input-independent constant thanks to our assumed input normalization.)

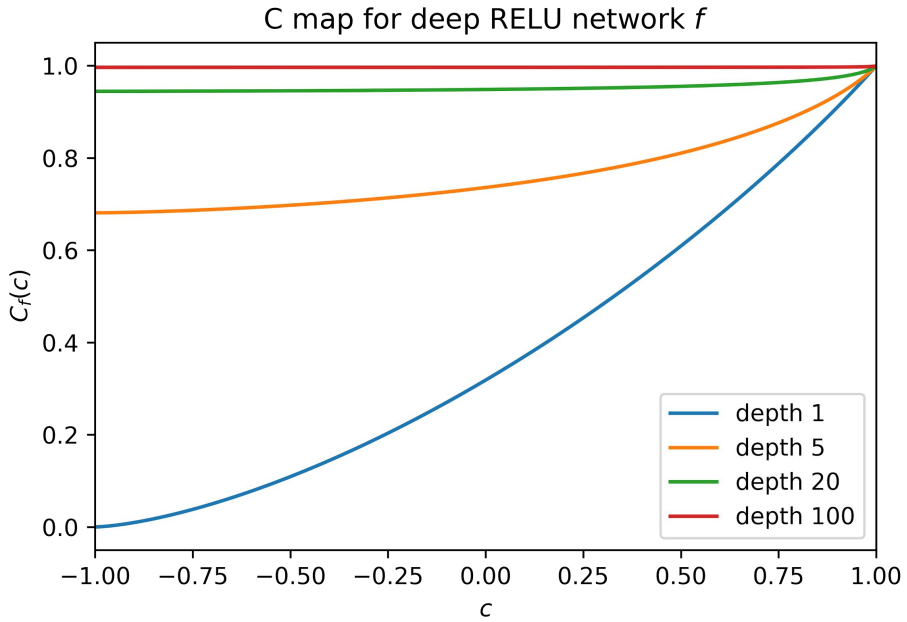
- To compute Q and C maps for whole networks, we compute them for component subnetworks and then compose:

$$Q_{f\circ h}(q) = Q_f(Q_h(q)) \qquad C_{f\circ h}(c) = C_f(C_h(c))$$

- Note that Q/C maps are only valid descriptions of the network at initialization. In general, the q/c values won't approximate any network-related quantities

C map degeneration in deep networks

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



Degenerate C maps imply difficult training

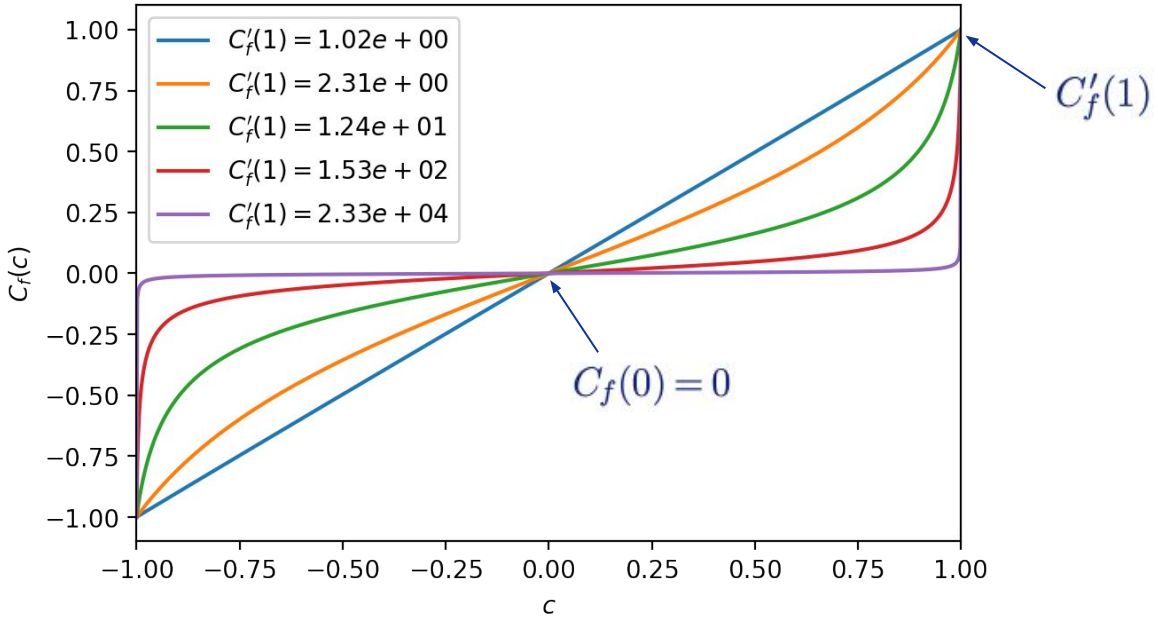
- The degenerate C maps seen in deep networks will squash entire ranges of input c values tightly around some output value c_0
- There are two basic cases for this, *both of which are bad for training*:
  - $c_0 \ll 1$: initial output vectors look “random”, and generalization is impossible. Early layers will have huge gradients compared to later layers
  - $c_0 \approx 1$: all initial 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 $C_g'(1) = 1$ for ***each nonlinear layer g*** (at initialization)
- This slows the asymptotic convergence with depth of c values to 1
- Unfortunately, given a deep enough network, they will still be pretty close to fully-converged, so the network's C map *will still be degenerate*
- For example, 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). They are also hard to train at high depths in practice without skip connections / BN layers

A new way to control C map properties

- C maps are *convex* on $[0, 1]$. Intuitively, this allows us to control the deviation of the ***network f's overall C map*** from the identity function by controlling the deviation of $C_f'(1)$ from 1, assuming $C_f(0) = 0$.



More rigorously...

Theorem. *If* $C_f(0) = 0$, *then we have that*

$$\max_{c \in [-1,1]} |C_f(c) - c| \leqslant 2\,(C_f'(1) - 1) \qquad \textit{and} \qquad \max_{c \in [-1,1]} |C_f'(c) - 1| \leqslant 3\,(C_f'(1) - 1)\,.$$

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 away from linear behavior

  e.g. for each ReLU activation, add 10^{10} to its input and subtract 10^{10} from its output, making it effectively the *identity function*

  → Model class is *technically* the same as a standard ReLU network, but optimizer will have to move the params by ~10^{10}

- ***Solution***: require that $C_g'(1)$ is maximized for nonlinear layers g, subject to our conditions on the network's overall C map

And another failure mode: high kernel approximation error

- The validity of Q and C maps relies on the kernel approximation error being sufficiently low
- Unfortunately, error can be very high in deep networks unless width is extremely large
- Issue is most obvious for q values. Up to first order, error in $Q_f(q)$ is proportional to $Q_f'(q)$ times the error in q. And $Q_f'(q)$ can get ***very*** big in deep networks.
- q value perturbations will also affect validity of C map computations (which require constant q values in order to be valid)
- **Solution:** require that $Q_f'(q) \leqslant 1$ for values of q we expect to see

Target Q/C map conditions (at initialization)

- DKS enforces the following conditions on Q/C maps for all subnetworks f:
  - a) $Q_f(1) = 1$ → standardizes map computations across layers, prevents issues with huge/tiny inputs to final loss
  - b) $Q'_f(1) = 1$ → controls kernel approximation error
  - c) $C_f(0) = 0$ → prevents C map degeneration
  - d) $C'_f(1) \leqslant \zeta$ for a moderate ζ → prevents C map degeneration
  - e) $\min_{g \text{ nonlinear layer}} C'_g(1)$ is maximized → prevents "nearly linear" networks
- For (a), (b) & (c), it's sufficient to have them hold for the Q/C maps of all nonlinear layers (and to "normalize" any weighted sums in the network)
- (d) & (e) are equivalent to the condition $C'_g(1) = \zeta^{1/D}$ for each nonlinear layer g, where D is the depth

Transforming activation functions

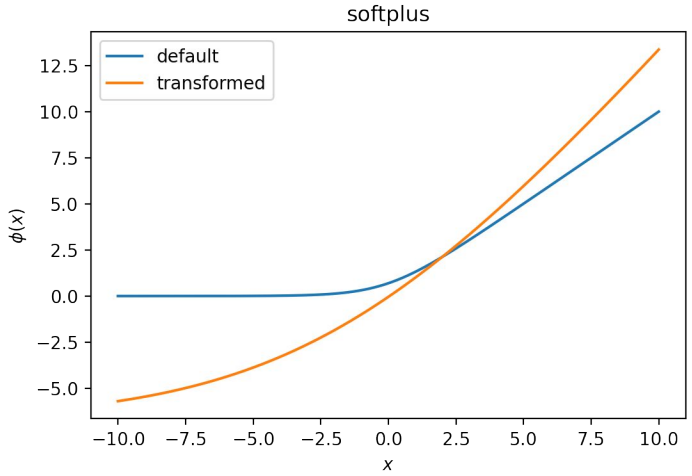
- To achieve these conditions, we introduce non-trainable scale and shift constants (for both input and output) to each activation function:

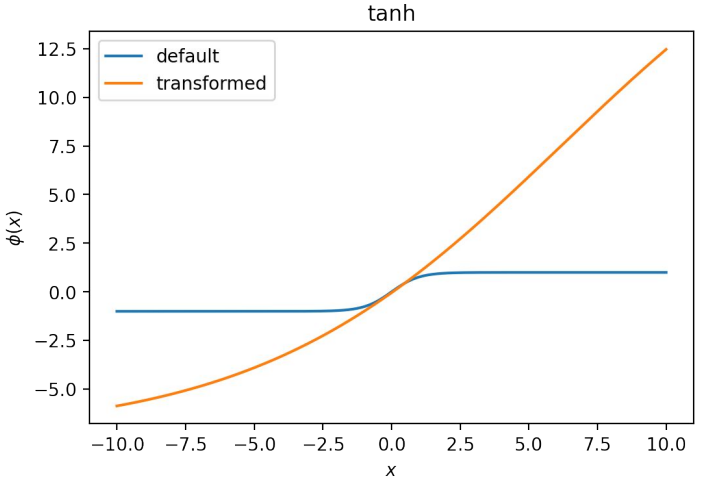
$$\phi(x) \quad \longrightarrow \quad \gamma\left(\phi(\alpha x + \beta) + \delta\right)$$

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

Examples:

(for a vanilla 100 layer MLP)

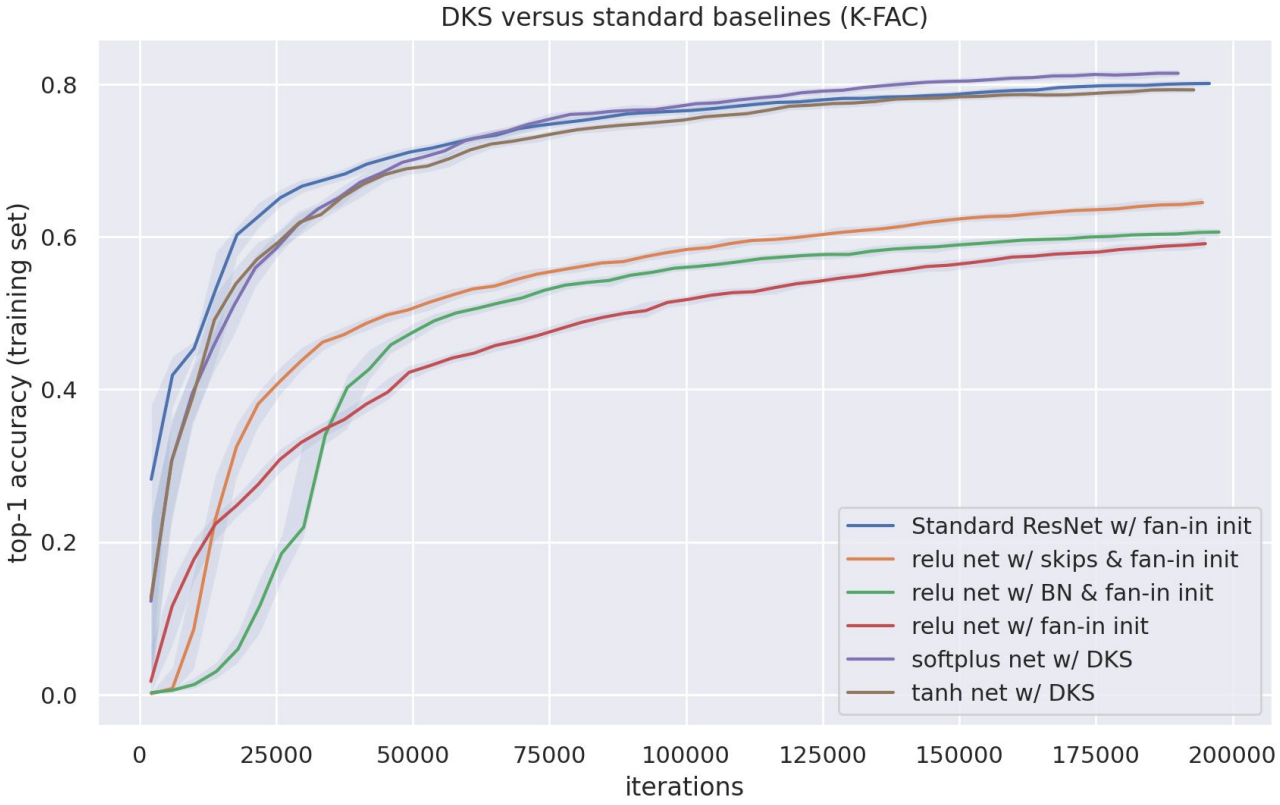




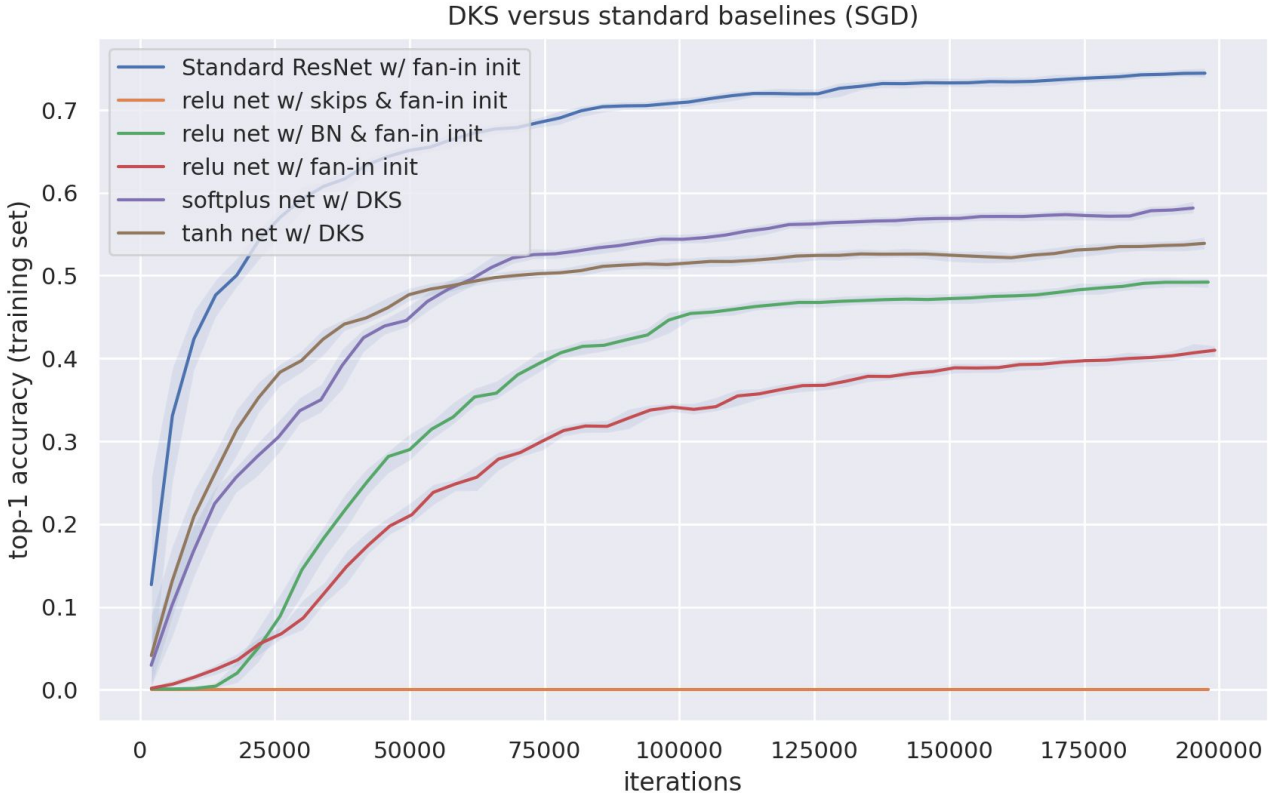
Experimental setup

- 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)

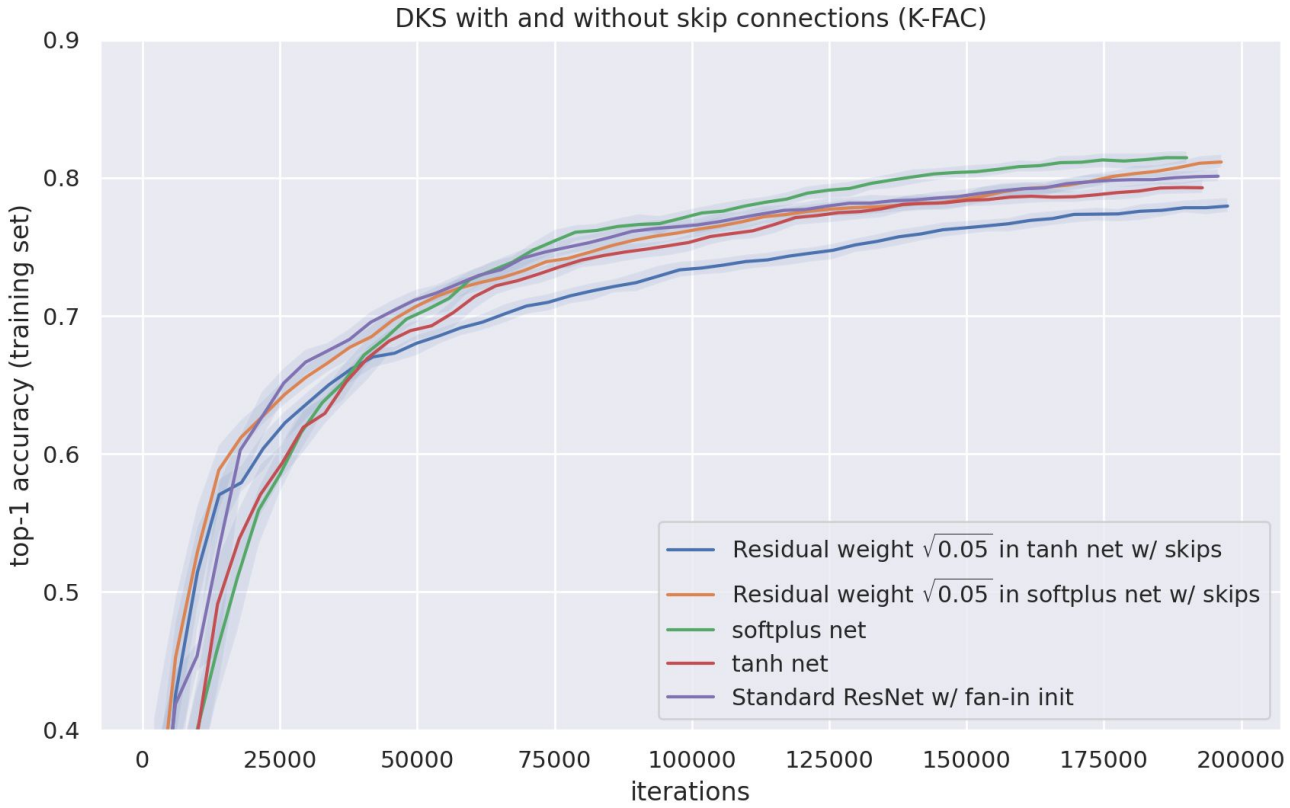
Vanilla networks w/ DKS compared to ResNets (K-FAC)



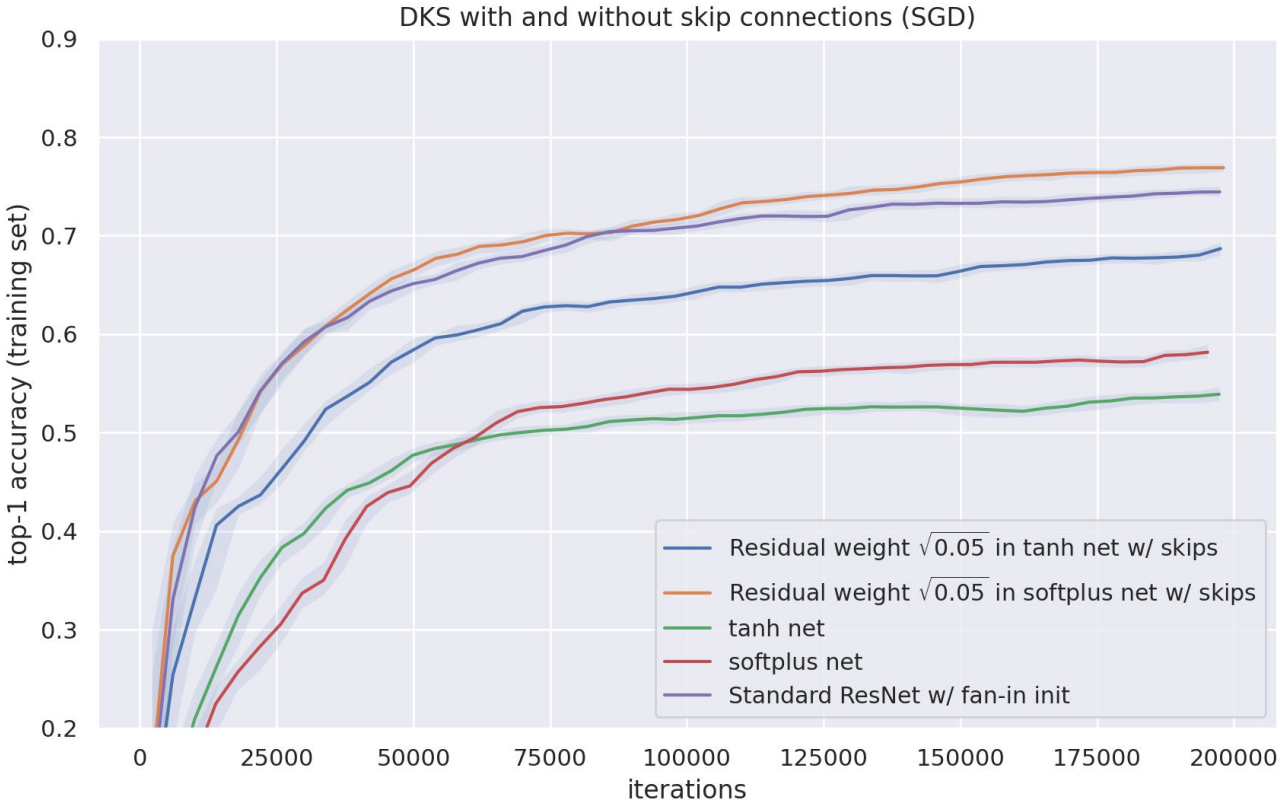
Vanilla networks w/ DKS compared to ResNets (SGD)



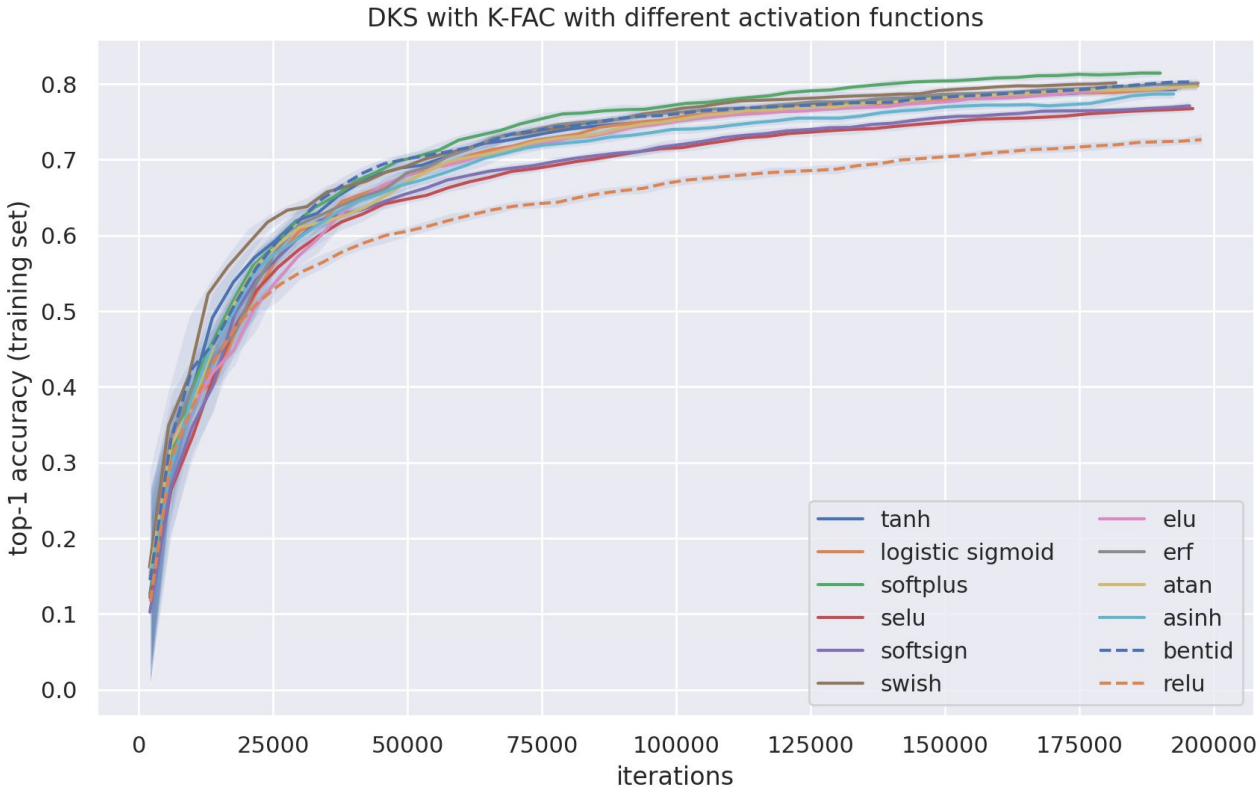
Interaction of skip connections and DKS (K-FAC)



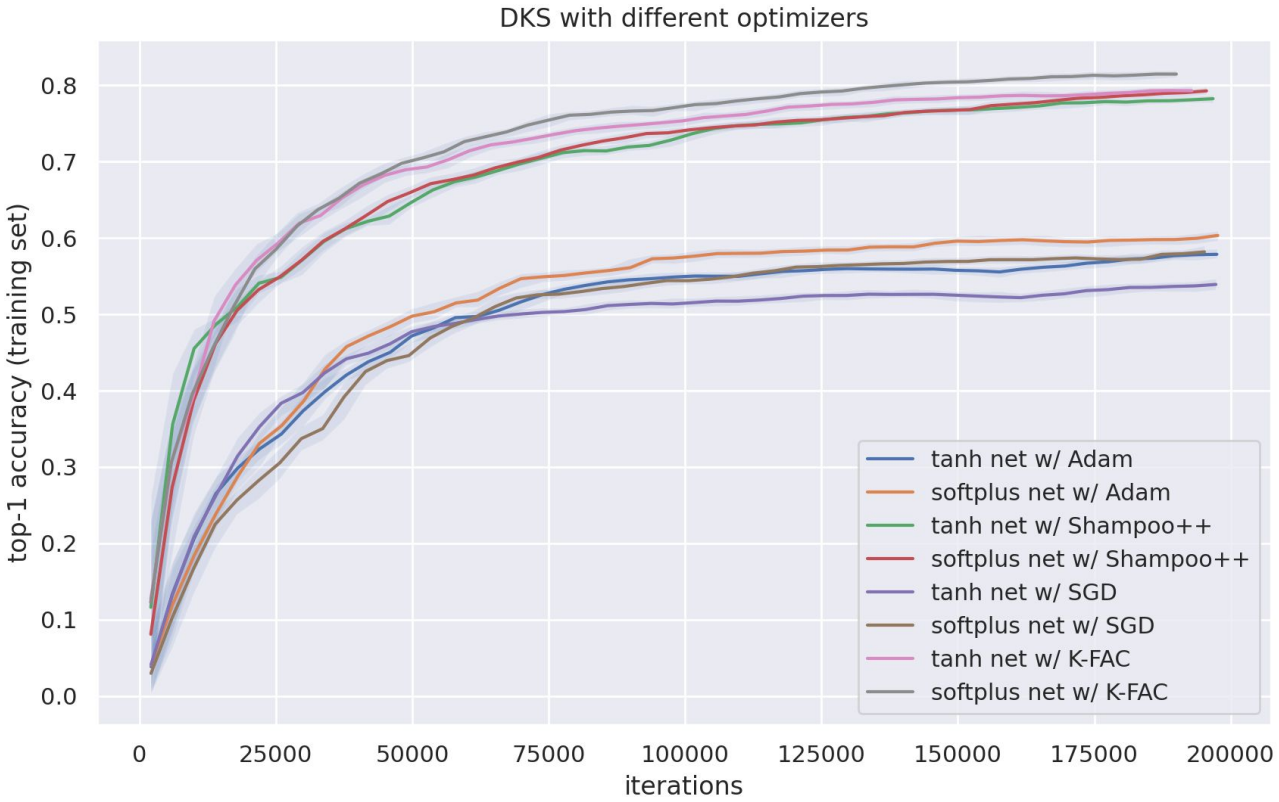
Interaction of skip connections and DKS (SGD)



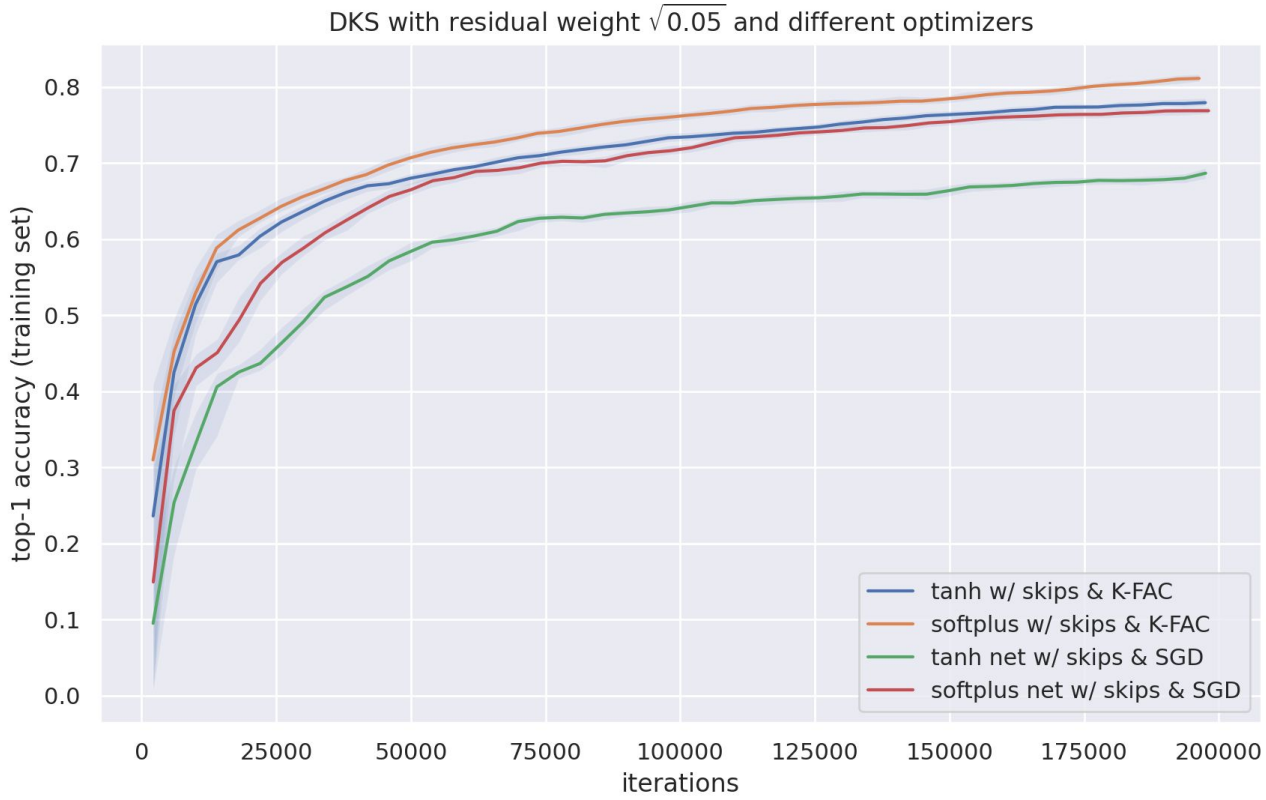
DKS applied to different activation functions



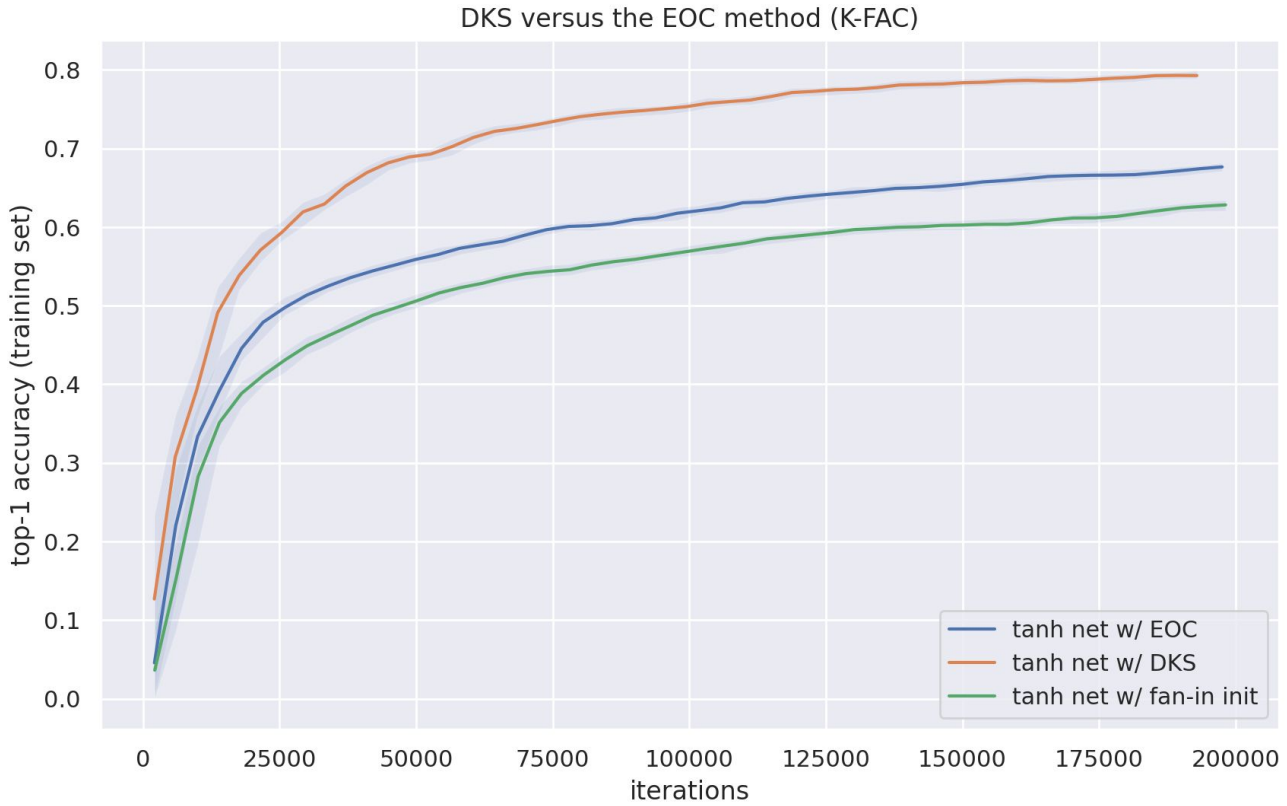
The effect of different optimizers on vanilla networks w/ DKS



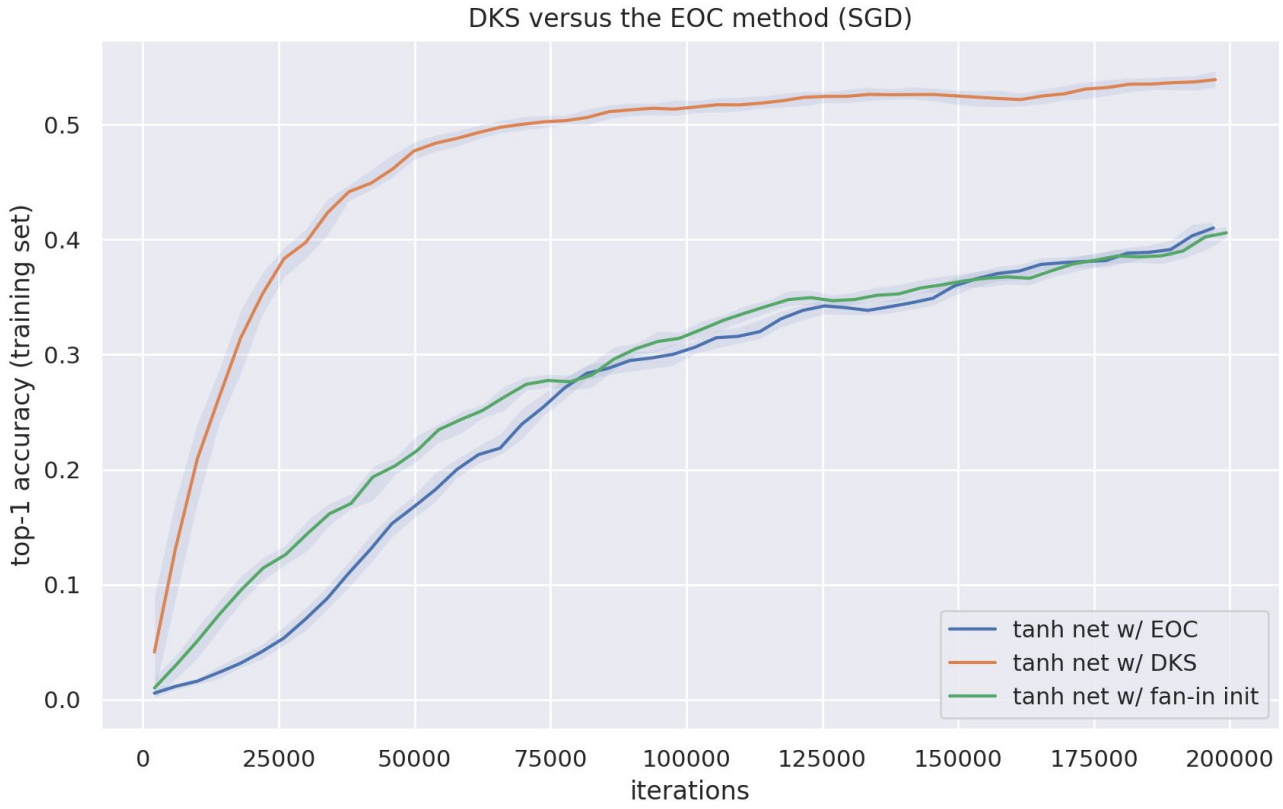
The effect of different optimizers on networks w/ skip connections & DKS



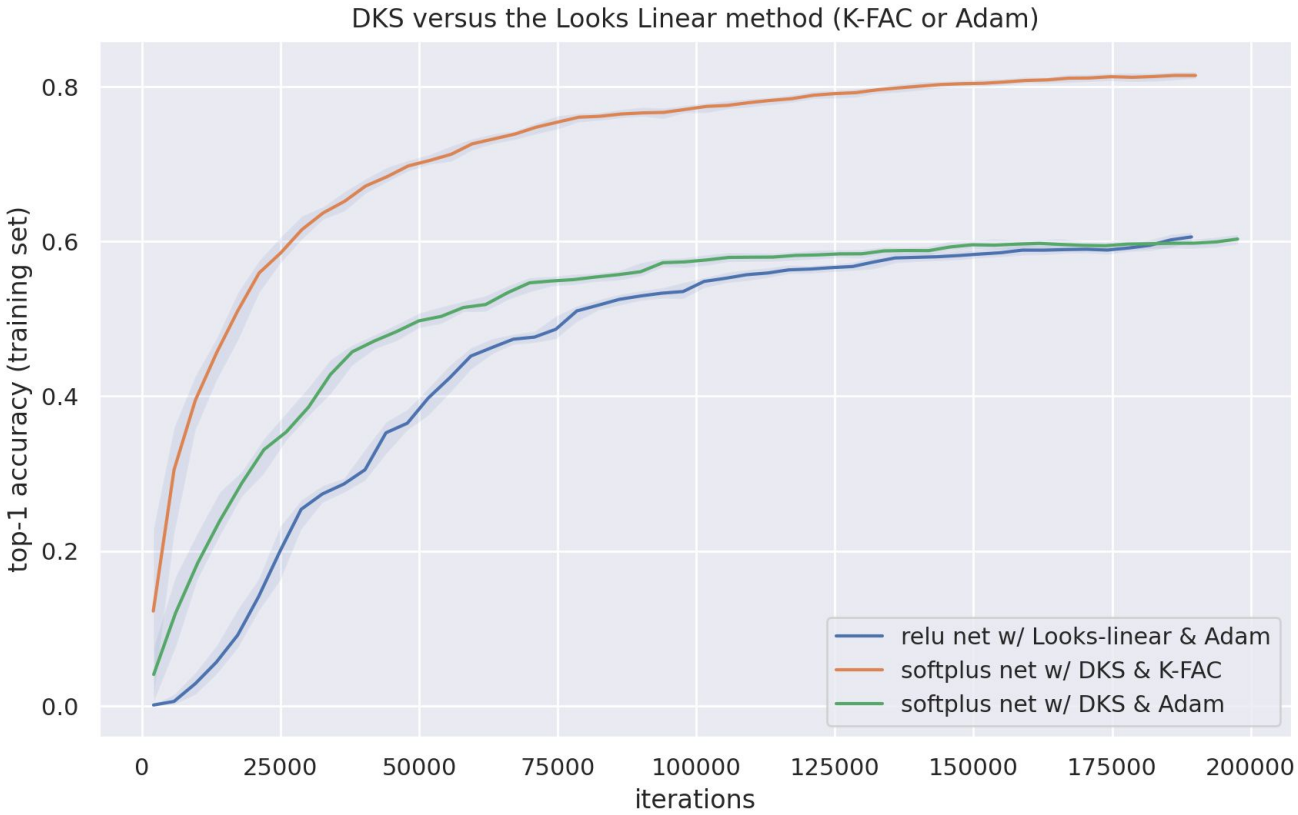
Comparisons to EOC (K-FAC)



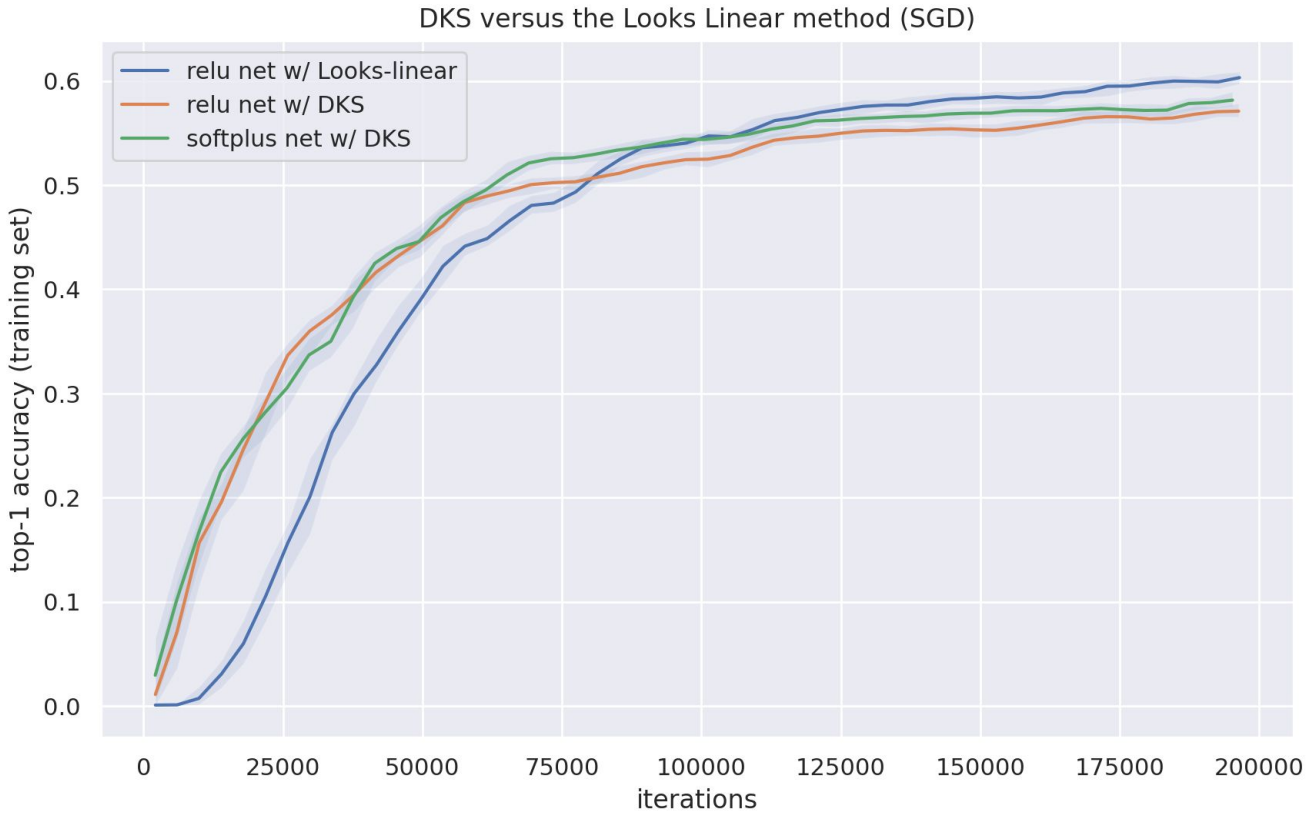
Comparisons to EOC (SGD)



Comparisons to Looks Linear (K-FAC)



Comparisons to Looks Linear (SGD)



Current Limitations

- Self-attention layers (as in Transformers) are not supported, but we are currently working on this
- Vanilla networks using DKS seem to generalize worse than standard ResNets, although this has been largely addressed in follow-up work (published in ICLR 2022)
- To match the training speed of ResNets using purely vanilla networks we currently require K-FAC. With SGD we require at least 3x 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.

Resources

DEEP KERNEL SHAPING

Rapid training of deep neural networks without skip connections or normalization layers using Deep Kernel Shaping

James Martens JAMESMARTENS@DEEPMIND.COM
Andy Ballard AYBJ@DEEPMIND.COM
Guillaume Desjardins GDESJARDINS@DEEPMIND.COM
Grzegorz Swirszcz SWIRSZCZ@DEEPMIND.COM
Valentin Dalibard VDALIBARD@DEEPMIND.COM
DeepMind, London, UK

Jascha Sohl-Dickstein JASCHAS@GOOGLE.COM
Samuel S. Schoenholz SCHSAM@GOOGLE.COM
Google, Mountainview, USA

Abstract

Using an extended and formalized version of the Q/C map analysis of Poole et al. (2016), along with Neural Tangent Kernel theory, we identify the main pathologies present in deep networks that prevent them from training fast and generalizing to unseen data, and show how these can be avoided by carefully controlling the "shape" of the network's initialization-time kernel function. We then develop a method called Deep Kernel Shaping (DKS), which accomplishes this using a combination of precise parameter initialization, activation function transformations, and small architectural tweaks, all of which preserve the model class. In our experiments we show that DKS enables SGD training of residual networks without normalization layers on Imagenet and CIFAR-10 classification tasks at speeds comparable to standard ResNetV2 and Wide-ResNet models, with only a small decrease in generalization performance. And when using K-FAC as the optimizer, we achieve similar results for networks *without* skip connections. Our results apply for a large variety of activation functions, including those which traditionally perform very badly, such as the logistic sigmoid. In addition to DKS, we contribute a detailed analysis of skip connections, normalization layers, special activation functions like RELU and SELU, and various initialization schemes, explaining their effectiveness as alternative (and ultimately incomplete) ways of "shaping" the network's initialization-time kernel.

[arXiv paper]

```python
def get_transformed_activations(
    activation_names, method="DKS", dks_params=None, tat_params=None,
    max_slope_func=None, max_curv_func=None, max_cmap_func=None,
    activation_getter=_get_numpy_activation_function):
  """Gets transformed activation functions using the DKS or TAT method.

  See the DKS paper (https://arxiv.org/abs/2110.01765) and the TAT paper
  (https://openreview.net/forum?id=U0k7XNTiFEq) for details about what these
  are, how they are computed, and what their parameters mean.

  Note that if you are using JAX, PyTorch, or TensorFlow, you probably want to
  be using the version of get_transformed_activations() in the corresponding
  subdirectories of this package.

  Args:
    activation_names: An iterable of string names for the activation functions.
      Supported activation function names are: "tanh", "sigmoid", "erf", "relu",
      "softplus", "selu", "elu", "swish", "bentid", "atan", "asinh", "square",
      "softsign", and "leaky_relu".
    method: A string representing the method used to transform the activation
      functions. Can be "DKS", "TAT", or "untransformed". The latter choice
      will return activation functions without any transformations.
      (Default: "DKS")
    dks_params: A dictionary containing the parameters to use for DKS. Keys
      should be a subset of {"c_slope_1_target", "local_q_slope_target"}.
      "c_slope_1_target" gives the target maximal slope value for the network
      (corresponding to "zeta" from the paper), and defaults to 1.5.
      "local_q_slope_target" gives the target value for the local Q map slope
```

[official implementation]

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.

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 (assuming NTK parameterization)
- 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*

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