GRADIENTS EXPLODE
- DEEP NETWORKS ARE SHALLOW
- RESNET EXPLAINED

Anonymous authors
Paper under double-blind review

ABSTRACT

Whereas it is believed that techniques such as Adam, batch normalization and, more recently, SeLU nonlinearities “solve” the exploding gradient problem, we show that this is not the case and that in a range of popular MLP architectures, exploding gradients exist and that they limit the depth to which networks can be effectively trained, both in theory and in practice. We explain why exploding gradients occur and highlight the collapsing domain problem, which can arise in architectures that avoid exploding gradients.

ResNets have significantly lower gradients and thus can circumvent the exploding gradient problem, enabling the effective training of much deeper networks, which we show is a consequence of a surprising mathematical property. By noticing that any neural network is a residual network, we devise the residual trick, which reveals that introducing skip connections simplifies the network mathematically, and that this simplicity may be the major cause for their success.

1 INTRODUCTION

The exploding gradient problem has been a major challenge for training very deep feedforward neural networks at least since the advent of gradient-based parameter learning (Hochreiter, 1991). In a nutshell, it describes the phenomenon that as the gradient is backpropagated through the network, it may grow exponentially from layer to layer. This can make the application of vanilla SGD impossible for networks beyond a certain depth. Either the step size is too large for updates to lower layers to be useful or it is too small for updates to higher layers to be useful. While this intuitive notion is widely understood, there exist no rigorous tools for determining the presence of pathological exploding gradients in a given network.

It is a common notion that techniques such as introducing normalization layers (e.g. Ioffe & Szegedy (2015), Ba et al. (2016), Chunjie et al. (2017), Salimans & Kingma (2016)), careful initial scaling of weights (e.g. He et al. (2015), Glorot & Bengio (2015), Saxe et al. (2014)) or SeLU nonlinearities (Klambauer et al., 2017) largely eliminate exploding gradients by preserving the scale of activations in the forward pass. This notion was espoused in landmark papers. The paper that introduced batch normalization Ioffe & Szegedy (2015) states:

In traditional deep networks, too-high learning rate may result in the gradients that explode or vanish, as well as getting stuck in poor local minima. Batch Normalization helps address these issues.

The paper that introduced ResNet He et al. (2016a) states:

Is learning better networks as easy as stacking more layers? An obstacle to answering this question was the notorious problem of vanishing/exploding gradients, which hamper convergence from the beginning. This problem, however, has been largely addressed by normalized initialization and intermediate normalization layers, ...

More recently, the paper that introduced SeLU nonlinearities Klambauer et al. (2017) stated:

[Under SeLU nonlinearities] ... vanishing and exploding gradients are impossible.

Balduzzi et al. (2017) states:
A long-standing obstacle to progress in deep learning is the problem of vanishing and exploding gradients. Although, the problem has largely been overcome via carefully constructed initializations and batch normalization ...

These claims are mistaken. Not only do none of the mentioned techniques combat pathological exploding gradients in general, some may even exacerbate them. We intend to correct this crucial misconception.

At first glance, the exploding gradient problem is solved by introducing different step sizes for different layers or weights, as is done by many popular optimization algorithms such as RMSprop (Tieleman & Hinton, 2012) or Adam (Kingma & Ba, 2015). This raises an important unanswered question. Are exploding gradients merely a numerical quirk to be overcome by simply rescaling different parts of the gradient vector or are they reflective of an inherently difficult optimization problem that cannot be easily tackled by simple modifications to a stock algorithm?

ResNet (He et al., 2016a) and other neural network architectures utilizing skip connections (e.g. Huang et al. (2017), Szegedy et al. (2016)) have been highly successful recently. While the performance of networks without skip connections starts to degrade when depth is increased beyond a certain point, the performance of ResNet continues to improve until a much greater depth is reached. There is a general belief that skip connections somehow “improve” the gradient with respect to parameters in lower layers. While Balduzzi et al. (2017) demonstrated a favorable property of ResNet with batch normalization and ReLU nonlinearities, a rigorous explanation for the general performance boost brought about by skip connections has not been given.

Our contributions are as follows:

1. We introduce the gradient scale coefficient (GSC), a novel rigorous measurement for assessing the presence of pathological exploding gradients (section 2).
2. We demonstrate that exploding gradients are in fact present in a variety of popular MLP architectures, including architectures utilizing techniques that supposedly combat exploding gradients. We show that introducing normalization layers may even exacerbate the exploding gradient problem (section 3).
3. We show that exploding gradients are not a numerical quirk to be overcome by rescaling different parts of the gradient vector, but are indicative of an inherently complex optimization problem and that they limit the depth to which MLP architectures can be effectively trained, rendering very deep MLPs effectively much shallower (section 4).
4. We explain why exploding gradients are likely to occur in deep networks even when the forward activations do not explode (section 5).
5. We uncover a new problem for training very deep feedforward networks, which we term the collapsing domain problem. We show how this problem can arise precisely in architectures that avoid exploding gradients and that it can be at least as damaging to the training process (section 6).
6. We demonstrate that ResNet exhibits much lower gradients and thus largely circumvents the exploding gradient problem, rendering ResNets effectively deeper than corresponding networks without skip connections. We detail the surprising mathematical relationship that makes this possible (section 7).
7. We introduce the residual trick (section 4), which reveals that ResNets are a mathematically simpler version of networks without skip connections and thus approximately achieve what we term the orthogonal initial state. This provides, we argue, the major reason for their superior performance as well as an important criterion for neural network design in general (section 7).

In section 8 we conclude and derive practical recommendations for training deep networks.

In the appendix in section 9.2 we provide further high-level discussion. In section 9.2.1 we discuss the relationship of exploding gradients with other measures of network trainability, such as eigen-spectrum analysis (Saxe et al., 2014), shattering gradients (Balduzzi et al. 2017), trajectory lengths (Raghu et al., 2017), covariate shift (e.g. (Ioffe & Szegedy, 2015)) and Hessian conditioning (e.g. (Luo, 2017)). In section 9.2.2, we compare the exploding gradient problem as it occurs in feedforward networks to the exploding and vanishing gradient problems in RNNs (e.g. Pascanu et al. 2013).
(2013). In section 9.2.3 we further discuss the collapsing domain problem and challenges that arise when attempting to quantify it. In section 9.2.4 we discuss the implications of this work specifically for choosing network architectures. In section 9.2.5 we highlight potential future work.

2 EXPLODING GRADIENTS DEFINED - THE GRADIENT SCALE COEFFICIENT

2.1 NOTATION AND TERMINOLOGY

For the purpose of this paper, we define a neural network \( f \) as a succession of layers \( f_l, 0 \leq l \leq L \), where each layer is a vector-to-vector transformation. We assume a prediction framework, where the ‘prediction layer’ \( f_L \) is considered to output the prediction of the network and the goal is to minimize the value of the error layer \( f_0 \) over the network’s prediction and the true label \( y \), summed over some dataset \( D \).

\[
\arg\min_{\theta} E, \quad E = \frac{1}{|D|} \sum_{(x,y) \in D} f_0(y, f_1(\theta_1, f_2(\theta_2, f_3(\ldots f_L(\theta_L, x))))))
\]

Note that in contrast to standard notation, we denote by \( f_L \) the lowest layer and by \( f_0 \) the highest layer of the network. Let the dimension / width of layer \( l \) be \( d_l \) with \( d_0 = 1 \) and the dimension of the data input \( x \) be \( d \).

Each layer except \( f_0 \) is associated with a parameter sub-vector \( \theta_l \) that collectively make up the parameter vector \( \theta = (\theta_1, \ldots, \theta_L) \). This vector represents the trainable elements of the network. Depending on the type of the layer, the sub-vector might be empty. For example, a layer composed of tanh nonlinearities has no trainable elements, so its parameter sub-vector is empty. We call these layers ‘unparametrized’. In contrast, a fully-connected linear layer has trainable weights, which are encompassed in the parameter sub-vector. We call these layers ‘parametrized’.

We say a network that has layers \( f_0 \) through \( f_L \) has ‘nominal depth’ \( L \). In contrast, we say the ‘compositional depth’ is equal to the number of parametrized layers in the network, which is the quantity that is commonly referred to as “depth”. For example, a network composed of three linear layers, two tanh layers and a softmax layer has nominal depth 6, but compositional depth 3.

Let the ‘quadratic expectation’ \( Q \) of a random variable \( X \) be defined as \( Q[X] = E[X^2]^{1/2} \), i.e. the generalization of the quadratic mean to random variables. Similarly, let the ‘inverse quadratic expectation’ \( Q^{-1} \) of a random variable \( X \) be defined as \( Q[X] = E[X^{-2}]^{-1/2} \). Further terminology used only in the appendix is given in section 9.3.

2.2 COLLOQUIAL NOTION OF EXPLODING GRADIENTS

Colloquially, the exploding gradient problem is understood approximately as follows:

When the error is backpropagated through a neural network, it may increase exponentially from layer to layer. In those cases, the gradient with respect to the parameters in lower layers may be exponentially greater than the gradient with respect to parameters in higher layers. This makes the network hard to train if it is sufficiently deep.

Let \( J_k^l(\theta, x, y) \) be the Jacobian of the \( l \)’th layer \( f_l \) with respect to the \( k \)’th layer \( f_k \) evaluated with parameter \( \theta \) at \( (x, y) \), where \( 0 \leq l \leq k \leq L \). Similarly, let \( T_k^l(\theta, x, y) \) be the Jacobian of the \( l \)’th layer \( f_l \) with respect to the parameter sub-vector of the \( k \)’th layer \( \theta_k \). Then we might take this colloquial notion to mean that if \( ||J_k^l|| \) and \( ||T_k^l|| \) grow exponentially in \( k - l \), according to some to-be-determined norm \( ||\cdot|| \), the network is hard to train if it is sufficiently deep.

However, this notion is insufficient because we can construct networks that are easy to train yet have Jacobians that grow exponentially at arbitrary rates. In a nutshell, all we have to do to construct such a network is to take an arbitrary network of desired depth that is easy to train and scale each layer function \( f_l \) and each parameter sub-vector \( \theta_l \) by \( R^{-l} \) for some constant \( R > 1 \). During training, all we have to do to correct for this change is to scale the gradient sub-vector corresponding to each layer by \( R^{-2l} \).
Proposition 1. Consider any \( r > 1 \) and any neural network \( f \) which can be trained to some error level in a certain number of steps by some gradient-based algorithm. There exists a network \( f' \) that can also be trained to the same error level as \( f \) and to make the same predictions as \( f \) in the same number of steps by the same algorithm, and has exponentially growing Jacobians with rate \( r \). (See section 9.5.7 for details.)

Therefore, we need a definition of ‘exploding gradients’ different from ‘exponentially growing Jacobians’ if we hope to derive from it that training is hard.

Note that all propositions and theorems are stated informally in the main body of the paper, for the purpose of readability and brevity. In the appendix in sections 9.5 and 9.6 respectively, they are re-stated in rigorous terms, proofs are provided and the practicality of conditions is discussed.

2.3 The Gradient Scale Coefficient

In this section, we outline our definition of ‘exploding gradients’ which can be used to show hardness of training because it does not suffer from the confounding effect outlined in the previous section.

Definition 1. Let the quadratic mean norm or \( qm \) norm of an \( m \times n \) matrix \( A \) be the quadratic mean of its singular values where the sum of squares is divided by its right dimension \( n \). If \( s_1, s_2, \ldots, s_{\min(m,n)} \) are the singular values of \( A \), we write:

\[
\|A\|_{qm} = \sqrt{\frac{s_1^2 + s_2^2 + \ldots + s_{\min(m,n)}^2}{n}}
\]

An equivalent definition would be \( \|A\|_{qm} = Q_u \|Au\|_2 \), where \( u \) is a uniformly random unit length vector. In plain language, it measures the expected impact the matrix has on the length of a vector with uniformly random orientation.

Definition 2. Let the gradient scale coefficient (GSC) for \( 0 \leq l \leq k \leq L \) be as follows:

\[
\text{GSC}(k, l, f, \theta, x, y) = \frac{\|\mathcal{J}^l\|_{qm}^2 \|f_k\|_2^2}{\|f_l\|_2^2}
\]

Definition 3. We say that the network \( f(\theta) \) has exploding gradients with rate \( r \) and intercept \( c \) at some point \((x, y)\) if for all \( k \) and \( l \) we have \( \text{GSC}(k, l, f, \theta, x, y) \geq cr^{k-l} \), and in particular \( \text{GSC}(l, 0, f, \theta, x, y) \geq cr^l \).

Of course, under this definition, any network of finite depth has exploding gradients for sufficiently small \( c \) and \( r \). There is no objective threshold for \( c \) and \( r \) beyond which exploding gradients become pathological. Informally, we will say that a network has ‘exploding gradients’ if the GSC can be well-approximated by an exponential function.

The GSC combines the norm of the Jacobian with the ratio of the lengths of the forward activation vectors. In plain language, it measures the size of the gradient flowing backward relative to the size of the activations flowing forward. Equivalently, it measures the relative sensitivity of layer \( l \) with respect to small random changes in layer \( k \).

Proposition 2. \( \text{GSC}(k, l) \) measures the quadratic expectation of the relative size of the change in the value of \( f_l \) in response to a small random change in \( f_k \). (See section 9.5.2 for details.)

What about the sensitivity of layers with respect to the parameter? For fully-connected linear layers, we obtain a similar relationship.

Proposition 3. When \( f_k \) is a fully-connected linear layer without trainable bias parameters and \( \theta_k \) contains the entries of the weight matrix, \( \text{GSC}(k, l) \sqrt{\frac{\|\theta_k\|_2\|f_{k+1}\|_2}{\|f_k\|_2\sqrt{\|d_{k+1}\|}}} \) measures the quadratic expectation of the relative size of the change in the value of \( f_l \) in response to a small random change in \( \theta_k \). Further, if the weight matrix is randomly initialized, \( \mathbb{E}_{\theta_k} \frac{\|\theta_k\|_2\|f_{k+1}\|_2}{\|f_k\|_2\sqrt{\|d_{k+1}\|}} = 1 \). (See section 9.5.3 for details.)
In this paper, for reasons of space and mathematical simplicity, we focus our analysis for now on multi-layer perceptrons (MLPs) which are comprised only of fully-connected linear layers with no trainable bias parameters and unparametrized layers. Therefore we also do not use trainable bias and variance parameters in the normalization layers.

In section 2.2, we showed that we can construct easily trainable networks with exponentially growing Jacobians by simple multiplicative rescaling of layers, parameters and gradients. Crucially, the GSC is invariant to this rescaling as it affects both the forward activations and the Jacobian equally, so the effects cancel out.

**Proposition 4.** \( GSC(k, l) \) is invariant under multiplicative rescalings of the network that do not change the function computed. (See section 9.5.4 for details.)

3 Gradients Explode - Despite Bounded Activations

In this section, we show that exploding gradients exist in a range of popular MLP architectures. Consider the decomposability of the GSC.

**Proposition 5.** Assuming the approximate decomposability of the norm of the product of Jacobians, i.e. \( \|J_k \cdot J_{k+1} \cdot \cdots J_{k-m}\|_{qm} \approx \|J_k\|_{qm}\|J_{k+1}\|_{qm} \cdots \|J_{k-m}\|_{qm} \), we have \( GSC(k, l) = GSC(k-k) \cdot GSC(k-k-1) \cdots GSC(l+1, l) \). (See section 9.5.5 for the proof.)

This suggests that as long as the GSC of individual layers is approximately \( r > 1 \), we may have an exponential growth of \( GSC(k, l) \) in \( k - l \). In figure 1, we show \( GSC(l, 0) \) for seven popular MLP architectures (left graph). A linear layer is followed by (i) a ReLU nonlinearity (Dahl et al., 2013) (‘ReLU’), (ii) layer normalization (Ba et al., 2016) followed by a ReLU nonlinearity (‘layer-ReLU’), (iii) batch normalization plus ReLU (‘batch-ReLU’), (iv) tanh, (v) layer norm plus tanh (‘layer-tanh’), (vi) batch norm plus tanh (‘batch-tanh’), (vii) SeLU. All networks have compositional depth 50 (i.e. 50 linear layers) and each layer has 100 neurons. Both data input and labels are Gaussian noise and the error layer computes the dot product between the label and the prediction. The entries of weight matrices are drawn from independent Gaussian distributions with mean zero. Weight matrix entries for ReLU architectures are initialized with variance \( \frac{2}{100} \) as suggested by He et al. (2015), weight matrix entries for tanh architectures with variance \( \frac{1}{100} \) as suggested by Sax et al. (2014) and Glorot & Bengio (2015), and weight matrix entries for SeLU architectures with variance \( \frac{1}{100} \) as suggested by Klambauer et al. (2017). For further experimental details, see section 9.9.

We find that in four architectures (batch-ReLU, layer-tanh, batch-tanh and SeLU), \( GSC(l, 0) \) grows almost perfectly linearly in log-space. This corresponds to gradient explosion. We call those architectures ‘exploding architectures’. Among these architectures, a range of techniques that supposedly reduce or eliminate exploding gradients are used: careful initial scaling of weights, normalization...
layers, SeLU nonlinearities. Adding normalization layers may even bring about or exacerbate exploding gradients.

In light of proposition 4, it is not surprising that these techniques are not effective. Multiplicative scaling does not combat exploding gradients as defined by the GSC. Normalization layers are used to scale the activations. Carefully choosing the initial scale of weights corresponds to a multiplicative scaling of weights. SeLU nonlinearities, again, act to scale down large activations and scale up small activations. While these techniques may of course impact the GSC by changing the fundamental mathematical properties of the network (as can be seen, for example, when comparing ReLU and batch-ReLU), they do not combat gradient explosion simply by virtue of controlling the size of forward activations. We note that all four exploding architectures, in addition to exhibiting exploding gradients as defined by the GSC, also exhibit exponentially growing Jacobians.

In contrast, the other three architectures (ReLU, layer-ReLU and tanh) do not exhibit exploding gradients. However, this apparent advantage comes at a cost, as we further explain in section 5.

All curves in figure 1 (left graph) exhibit small jitters. This is because we plotted the value of the GSC at every linear layer, every normalization layer and every nonlinearity layer in the graph and then connected the points corresponding to these values. Layers were placed equispaced on the x axis in the order they occurred in the network. Not every type of layer affects the GSC equally. In fact, we find that as gradients pass through linear layers, they tend to shrink relative to forward activations. In the exploding architectures, this is more than counterbalanced by the relative increase the gradient experience as it passes through e.g. normalization layers. Despite these layer-dependent differences, it is worth noting that each individual layer used in the architectures studied has only a small impact on the GSC. This would not be true for either the forward activations or gradients taken by themselves. For example, passing through a ReLU layer reduces the length of both activation and gradient vector by \( \approx \sqrt{2} \). This invariance suggests that the GSC measures not just a superficial quantity, but a deep property of the network. This hypothesis is confirmed in the following sections.

Finally, we note that the GSC is also robust to changes in width and depth. Changing the depth has no impact on the rate of explosion of the four exploding architectures as the layer-wise GSC, i.e. \( \text{GSC}(l+1, l) \), is itself independent of depth. In figure 1 we also show the results for the SeLU architecture where each layer contains 200 neurons instead of 100 (‘SeLU wide’). We found that the rate of gradient explosion decreases slightly when width increases. We also studied networks with exploding architectures where the width oscillated from layer to layer. \( \text{GSC}(k, 0) \) still increased approximately exponentially and at a similar rate to corresponding networks with constant width.

A summary of results can be found in table 1 in the appendix.

4 Exploding Gradients Limit Depth - The Residual Trick

4.1 Background: Effective Depth

In this section, we introduce the concept of effective depth as defined for the ResNet architecture by Veit et al. (2016). We denote a residual network by writing each layer \( f_l \) (except \( f_0 \)) as the sum of a fixed initial function \( i_l \) and a residual function \( r_l \). We define the optimization problem for a residual net analogously to equation 1.

\[
\arg \min_{\theta} E, \quad \text{where} \quad E = \frac{1}{|D|} \sum_{(x, y) \in D} f_0(y, (i_1 + r_1(\theta_1)) \circ (i_2 + r_2(\theta_2)) \circ \ldots \circ (i_L + r_L(\theta_L)) \circ x) \quad (2)
\]

Let’s assume for the sake of simplicity that the dimension of each layer is identical. In that case, the initial function for ResNet is generally chosen to be the identity function. Then, writing the identity matrix as \( I \), we have

\[
\frac{df_0}{dx} = \frac{df_0}{df_1}(I + \frac{dr_1}{df_2})(I + \frac{dr_2}{df_3})\ldots(I + \frac{dr_{L-1}}{df_L})(I + \frac{dr_L}{dx})
\]
Multiplying out, this becomes the sum of $2^L$ terms. Almost all of those terms are the product of approximately $\frac{1}{2}$ identity matrices and $\frac{1}{2}$ residual Jacobians. However, if the operator norm of the residual Jacobians is less than $p$ for some $p < 1$, the norm of terms decreases exponentially in the number of residual Jacobians they contain. Let the terms in $\frac{df}{d\theta}$ containing $\lambda$ or more residual Jacobians be called ‘$\lambda$-residual’ and let $res^\lambda$ be the sum of all $\lambda$-residual terms. Then:

$$||res^\lambda||_2 \leq ||\frac{df_0}{df_1}||_2 \sum_{l=\lambda}^L p^L$$

Again, if $p < 1$, the right hand side decreases exponentially in $\lambda$ for sufficiently large $\lambda$, for example when $\lambda > \frac{1}{p}$. As long as $\frac{df}{d\theta}$ in bounded, the combined size of $\lambda$-residual terms is exponentially small. Therefore, Veit et al. (2016) argue, the full set of network layers does not jointly co-adapt during training because the information necessary for such co-adaptation is contained in terms that contain many or all residual Jacobians. Only sets of layers of size at most $\lambda$ where $res^\lambda$ is not negligibly small co-adapt. The largest such $\lambda$ is called the ‘effective depth’ of the network. Veit et al. (2016) argue that if the effective depth is less than the compositional depth of a residual network, the network is not really as deep as it appears, but rather behaves as an ensemble of relatively shallow networks. This argument is bolstered by the success of the stochastic depth (Huang et al., 2016) training technique, where random sets of residual functions are deleted for each mini-batch update.

Veit et al. (2016) introduced the concept of effective depth somewhat informally. We give our formal definition in section 9.4. There, we also provide a more detailed discussion of the concept and point out limitations.

4.2 The Residual Trick

Now we make a crucial observation. Any neural network can be expressed as a residual network as defined in 2. We can simply choose arbitrary initial functions $i_l$ and define $r_l(\theta_l) := f_l(\theta_l) - i_l$. Specifically, if we train a network $f$ from some fixed initial parameter $\theta^{(0)}$, we can set $i_l := f_l(\theta_l^{(0)})$ and thus $r_l(\theta_l) := f_l(\theta_l) - f_l(\theta_l^{(0)})$. Then training begins with all residual functions being zero functions. Therefore, all analysis devised for ResNet that relies on the small size of the residual Jacobians can then be brought to bear on arbitrary networks. We term this the residual trick. Indeed, the analysis by Veit et al. (2016) does not rely on the network having skip connections in the computational sense, but only on the mathematical framework of equation 2. Therefore, as long as the operator norms of $\frac{df_l}{df_{l+1}} - \frac{df_l(\theta_l^{(0)})}{df_{l+1}}$ are small, $f$ is effectively shallow.

Terminology From now on, we will make a distinction between the terms ‘ResNet’ and ‘residual network’. The former will be used to refer to networks that have an architecture as in He et al. (2016a) that uses skip connections. The latter will be used to refer to arbitrary networks expressed in the framework of equation 2. Networks without skip connections will be referred to as ‘vanilla networks’. Note that, again, for reasons of space and mathematical simplicity, we focus our analysis for now on ResNets with fully-connected layers.

4.3 Theoretical Analysis

We will now derive our first main result. Namely, we will show that an exploding gradient causes the effective training time of deep MLPs to be exponential in depth and thus limits the effective depth that can be achieved.

The proof is based on the insight that the relative size of a gradient-based update $\Delta \theta_l$ on $\theta_l$ is bounded by the inverse of the GSC if that update is to be useful. The basic assumption underlying gradient-based optimization is that the function optimized is locally approximated by a linear function as indicated by the gradient. Any update made based on a local gradient computation must be small enough so that the updated value lies in the region around the original value where the linear approximation is sufficiently accurate. Let’s assume we apply a random update to $\theta_l$ with relative magnitude $\frac{1}{GSC(0,l)}$, i.e. $\frac{||\Delta \theta_l||_2}{||\theta_l||_2} = \frac{1}{GSC(0,l)}$. Then under the local linear approximation,
according to proposition this would change the output \( f_0 \) approximately by a value with quadratic expectation \( f_0 \). Hence, with significant probability, the error would become negative. This is not reflective of the true behavior of the function \( f_0 \) in response to changes in \( \theta_t \) of this magnitude. Since gradient-based updates impact the function value even more than random updates, useful gradient-based updates are even more likely to be bounded in relative magnitude by \( \mathcal{GSC}([0,1]) \).

**Theorem 1.** Under certain conditions, if an MLP has exploding gradients with explosion rate \( r \) and intercept \( c \) on some dataset, then there exists a constant \( c' \) such that training this neural network with a gradient-based algorithm to have effective depth \( \lambda \) takes at least \( c' cr^{-\frac{1}{\lambda}} \) updates. (See section 9.6.7 for details.)

**Proof Summary.** Since the relative size of updates on the \( l \)th layer is bounded to within \( \frac{1}{\mathcal{GSC}([0,1])} = O(r^{-1}) \), after \( T \) updates, residual functions have relative size at most \( O(Tr^{-1}) \). Thus the relative combined size of \( \lambda \)-residual terms is at most \( O((T\lambda)^{\lambda} r^{-\frac{\lambda(\lambda-1)}{2}} (\ln r)^{-\lambda}) \). Thus after \( T \) updates, the cumulative relative size of updates applied that stem from \( \lambda \)-residual terms is at most \( O(T^{\lambda+1}\lambda^r r^{-\frac{\lambda(\lambda-1)}{2}} (\ln r)^{-\lambda}) \). Thus, to reach a given target level, we require at least \( T = O(r^{-\frac{1}{\lambda}}) \).

Importantly, the lower bound on the number of updates required to reach a certain effective depth stated by theorem is independent of the depth of the network. While the constant \( c' \) depends on some constants that arise in the conditions of the theorem, as long as those constants do not change when depth is increased, neither does the lower bound.

**Corollary 1.** In the scenario of theorem if the number of updates to convergence is bounded, so is effective depth.

Here we simply state that if we reach convergence after a certain number of updates, but theorem indicates that more would be required to attain a greater effective depth, then that greater effective depth is unreachable with that algorithm.

### 4.4 Experiments

To practically validate our theory of limited effective depth, we train our four exploding architectures (batch-ReLU, layer-tanh, batch-tanh and ScLU) on CIFAR10. All networks studied have a compositional depth of 51 (i.e. 51 linear layers) and 100 neurons in each layer except for the input, prediction and error layers. Full experimental details can be found in section 9.9.

First, we determined the approximate best step size for SGD for each individual linear layer. We started by pre-training the highest layers of each network with a small uniform step size until the training classification error was below 85%, but at most for 10 epochs. Then, for each linear layer, we trained only that layer for 1 epoch with various step sizes while freezing the other layers. The step size that achieved the lowest training classification error after that epoch was selected. Note that we only considered step sizes that induce relative update sizes of 0.1 or less, because larger updates can cause weight instability. The full algorithm for step size selection and a justification is given in section 9.9.4.

In figure 2 (top left), we show the relative update size induced on each linear layer by what was selected to be the best step size as well as \( \frac{1}{\mathcal{GSC}([0,1])} \) as a dashed line. In section 4.3, we argued that \( \frac{1}{\mathcal{GSC}([0,1])} \) is an upper bound for the relative size of a useful update. We find that this bound holds and is conservative except for a small number of outliers. Even though our algorithm for determining the best step size for each layer gives noisy results, there is a clear trend that lower layers require relatively smaller updates, and that this effect is more pronounced if the gradient explodes with a larger rate.

We then smoothed these best step size estimates and trained each network for 500 epochs with those smoothed estimates. Periodically, we scaled all step sizes jointly by \( \frac{1}{\lambda} \). In figure 2 (top middle), we show the training classification error of each architecture. There is a trend that architectures with less gradient explosion attain a lower final error. Note that, of course, all these error values are still much higher than the state of the art on CIFAR10. The goal of this paper is to understand pathological architectures rather than find optimal ones.
In figure 2 (top right), we show the GSC across the entire network, i.e. $GSC(L, 0)$, as training progresses. During the initial pre-training phase, this value drops significantly but later regains or even exceeds its original value. In the top left graph, the dashed line indicates the inverse of $GSC(l, 0)$ at each linear layer after pre-training. We find that the GSC actually falls below 1 as the gradient passes through the pre-trained layers, but then resumes explosion once it reached the layers that were not pre-trained. We find this behavior surprising and unexpected. We conclude that nonstandard training procedures can have a significant impact on the GSC but that there is no evidence that when all layers are trained jointly, which is the norm, the GSC either significantly increases or decreases during training.

We then went on to measure the effective depth of each network. We devised a conservative, computationally tractable estimate of the cumulative size of updates that stem from $\lambda$-residual terms. See section 9.4.2 for details. The effective depth depicted in figure 2 (bottom left) is the largest value of $\lambda$ such that this estimate has a length exceeding $10^{-6}$. As expected, none of the architectures reach an effective depth equal to their compositional depth, and there is a trend that architectures that use relatively smaller updates achieve a lower effective depth. It is worth noting that the effective depth increases most sharply at the beginning of training. Once all step sizes have been multiplied by $\frac{1}{3}$ several times, effective depth no longer changes significantly while the error, on the other hand, is still going down. This suggests that, somewhat surprisingly, high-order co-adaptation of layers takes place towards the beginning of training and that as the step size is reduced, layers are fine-tuned relatively independently of each other.

SeLU and especially tanh-batch reach an effective depth close to their compositional depth. In figure 2 (bottom middle), we show the operator norm of the residual weight matrices after training. All architectures except SeLU, which has a $GSC(L, 0)$ close to 1 after pre-training, show a clear downward trend in the direction away from the error layer. If this trend were to continue for networks that have a much greater compositional depth, then those networks would not achieve an effective depth significantly greater than our 51-linear layer networks.

Veit et al. (2016) argue that a limited effective depth indicates a lack of high-order co-adaptation. We wanted to verify that our networks, especially layer-tanh and batch-ReLU, indeed lack these high-order co-adaptations by using a strategy independent of the concept of effective depth to measure this effect. We used Taylor expansions to do this. Specifically, we replaced the bottom $k$ layers...
of the fully-trained networks by their first-order Taylor expansion around the initial functions. See section 9.7 for how this is done. This reduces the compositional depth of the network by \( k - 2 \). In figure 2 (bottom right), we show the training classification error in response to compositional depth reduction. We find that the compositional depth of layer-tanh and batch-ReLU can be reduced enormously without suffering a significant increase in error. In fact, the resulting layer-tanh network of compositional depth 15 greatly outperforms the original batch-tanh and batch-ReLU networks. This confirms that these networks lack high-order co-adaptations. Note that cutting the depth by using the Taylor expansion not only eliminates high-order co-adaptations among layers, but also co-adaptations of groups of 3 or more layers among the bottom \( k \) layers. Hence, we expect the increase in error induced by removing only high-order co-adaptations to be even lower than what is shown in figure 2. Unfortunately, this cannot be tractably computed.

Finally, we trained each of the exploding architectures by using only a single step size for each layer that was determined by grid search, instead of custom layer-wise step sizes. As expected, the final error was higher. The results are found in table 2.

4.5 A NOTE ON BATCH NORMALIZATION

We used minibatches of size 1000 to train all architectures except batch-ReLU, for which we conducted full-batch training. When minibatches were used on batch-ReLU, the training classification error stayed above 89% throughout training. (Random guessing achieves a 90% error.) In essence, no learning took place. This is because of the pathological interplay between exploding gradients and the noise inherent in batch normalization. Under batch normalization, the activations at a neuron are normalized by their mean and standard deviation. These values are estimated using the current batch. Hence, if a minibatch has size \( b \), we expect the noise induced by this process to have relative size \( \approx \sqrt{\frac{1}{b}} \). But we know that according to proposition 2 under the local linear approximation, this noise leads to a change in the error layer of relative size \( \approx \frac{GSC}{\sqrt{r}} \). Hence, if the GSC between the error layer and the first batch normalization layer is larger than \( \sqrt{\frac{1}{b}} \), learning should be seriously impaired. For the batch-ReLU architecture, this condition was satisfied and consequently, the architecture was untrainable using minibatches. Ironically, the gradient explosion that renders the noise pathological was introduced in the first place by adding batch normalization layers.

5 WHY GRADIENTS EXPLODE - DETERMINANTS VS QM NORM

Why do exploding gradients occur? As mentioned in section 3, gradients explode with rate \( r > 1 \) as long as we have (A) \( GSC(k, l) \approx GSC(l + 1, 1)GSC(l + 2, l + 1)\ldots GSC(k, k - 1) \) and (B) \( GSC(l + 1, 1) \approx r \) for all \( k \) and \( l \). It turns out that we can show both of these hold in expectation under fairly realistic conditions if we view the network parameter as a random variable.

**Theorem 2.** Under certain conditions, for any neural network \( f \) with random parameter \( \theta \) composed of layer functions \( f_\ell \) that are surjective endomorphisms on the hypersphere, where the absolute singular values of the Jacobian of each layer differ by at least \( \epsilon \) with probability \( \delta \), gradients explode in expectation with rate \( r(\delta, \epsilon) \). (See section 9.6.3 for details.)

**Proof Summary.** Consider a surjective, differentiable endomorphism \( f_\ell \) on the hypersphere and a random input \( x \) distributed uniformly on that hypersphere. Surjectivity implies that the absolute determinant of the Jacobian, in expectation over the input, is at least 1. The absolute determinant is the product of the absolute singular values. If those absolute singular values are IID and their expected product is at least 1, the expectation of each absolute singular value is also at least 1. So if these singular values are sufficiently different from each other with sufficient probability, the expected quadratic mean of the singular values is at least \( r \). Since both input and output of \( f_\ell \) have length 1, the expected GSC will also be at least \( r \).

**Theorem 2** reveals an important insight. Exploding gradients arise because in order to preserve the domain of the forward activations from layer to layer, layer-Jacobians need to have unit absolute
determinants in expectation, and this tends to cause their qm norm values to be greater than 1, and then these values tend to compound exponentially.

To determine which architectures might suffer from exploding gradients, we can determine whether a given architecture or an architecture similar to it is composed of bijective endomorphisms on the hypersphere. Let ‘length-only layer normalization’ or ‘LOlayer’ be a function that divides its input vector by its quadratic mean.

**Proposition 6.** Any endomorphism on the hypersphere composed of (i) a strictly monotonic, continuous nonlinearity $\sigma$ that has $\sigma(0) = 0$, (ii) multiplication with a full-rank matrix and (iii) length-only layer normalization is bijective. (See section 9.5.6 for the proof.)

Hence, LOlayer-tanh and LOlayer-SeLU architectures are composed of bijective endomorphisms on the hypersphere. Hence, we expect that these architectures have exploding gradients. We found that indeed, these architectures exhibit exploding gradients at rates very similar to layer-tanh and SeLU respectively. See table 1 for details.

Further, the theorem presents two clear avenues for avoiding exploding gradients: (i) use non-surjective layer functions, (ii) ensure that Jacobians get progressively closer to multiples of orthogonal matrices as we go deeper. It turns out that these are exactly the strategies employed by ReLU and ResNet respectively to avoid exploding gradients, and we will discuss these in the next two sections.

6 THE COLLAPSING DOMAIN PROBLEM - HOW NOT TO REDUCE GRADIENTS

In the previous section, we showed how surjective endomorphisms can exhibit exploding gradients. This suggests that we can avoid exploding gradients if we reduce the domain of the forward activations from layer to layer. Informally, this can be understood as follows. Consider some layer function $f_l$. If we shrink its co-domain by a factor $c$, we reduce the eigenvalues of the Jacobian and hence its qm norm by $c$. If we also ensure that the length of the output stays the same, the GSC is also reduced by $c$. Similarly, inflating the co-domain would cause the qm norm to increase.

This suggests that in neural network design, we can actively trade off exploding gradients and shrinkage of the domain and that eliminating one effect may exacerbate the other. This is precisely what we find in practice. Returning to figure [1] we now turn our attention to the middle graph. Here, we plot the standard deviation of the activation values at each neuron across datapoints in the layers before each nonlinearity layer (‘pre-activations’), averaged over all neurons in the same layer. The four exploding architectures exhibit a near constant standard deviation, whereas the other three architectures (ReLU, layer-ReLU and tanh) exhibit a rapidly collapsing standard deviation, which shows that the activations corresponding to different datapoints become more and more similar with depth. We term this effect the collapsing domain problem. But why is this effect a problem? Two reasons.

Collapsing depth causes pseudo-linearity If the pre-activations that are fed into a nonlinearity are highly similar, the nonlinearity can be well-approximated by a linear function. In the tanh architecture we studied, for example, activation values become smaller and smaller as they are propagated forward. If the pre-activations of a tanh nonlinearity have small magnitude, the tanh nonlinearity can be approximated by the identity function. But if a tanh layer is approximately equal to an identity layer, the entire network becomes equivalent to a linear network. We say the network becomes ‘pseudo-linear’. Of course, linear networks of any depth have the representational capacity of a linear network of depth 1 and are unable to model nonlinear functions. Hence, a tanh network that is pseudo-linear beyond compositional depth $k$ approximately has the representational capacity of a compositional depth $k + 1$ tanh network. Based on the decrease in pre-activation standard deviation exhibited by the tanh architecture in figure [1], a reasonable estimate is that the network of compositional depth 50 has the representational capacity of a network of compositional depth 10.

Similarly, for a ReLU nonlinearity, if either all or most pre-activations are positive or all or most pre-activations are negative, the nonlinearity can be approximated by a linear function. If all or most pre-activations are positive, ReLU can be approximated by the identity function. If all or most pre-activations are negative, ReLU can be approximated by the identity function.
activations are negative, ReLU can be approximated by the zero function. In the right graph of figure 1, we plot the proportion of pre-activations for each neuron in a nonlinearity layer that are positive or negative, whichever is smaller, averaged over each layer. We call this metric ‘sign diversity’. For both ReLU and layer-ReLU, sign diversity decreases rapidly to cause pseudo-linearity from at least, say, the 20th linear layer onwards. None of the four exploding architectures suffers a significant loss in sign diversity.

Collapsing depth is exploding gradient in disguise
In theorem 1, we used the fact that the output of the error layer of the network was positive to bound the size of a useful gradient-based update. In other words, we used the fact that the domain of the error layer was bounded. However, the collapsing domain problem causes not just a reduction of the size of the domain of the error layer, but of all intermediate layers. Hence, we expect the largest useful update to shrink in proportion with the reduction of the size of the domain. Therefore, we suspect that a collapsing domain will ultimately have the same effect on the largest useful update size of each layer as exploding gradients, that is to reduce them and thus cause a low effective depth.

In table 2 we show the final error values achieved by training ReLU, layer-ReLU and tanh on CIFAR10. The errors are substantially higher than those achieved by the exploding architectures, except for batch-ReLU. Also, training with layer-wise step sizes did not help compared to training with a single step size. In figure 4 (top left), we show the estimated best relative update sizes for each layer. This time, there is no downward trend towards lower layers, which is likely why training with a single step size is “sufficient”. Interestingly, we note that the difference between the \( \frac{1}{GSC} \) bound and the empirically optimal relative update sizes is now much larger than it is for the exploding architectures (see figure 2). This suggests that indeed, collapsing domains may similarly reduce the optimal relative update size, just as exploding gradients. In figure 4 (center left), we find that again, the effective depth reached is significantly lower than the compositional depth of the network and is comparable to that of architectures with exploding gradients.

In the bottom row of figure 4, we plot the pre-activation standard deviation and sign diversity at the highest nonlinearity layer throughout training. Interestingly, pseudo-linearity declines significantly early in training. The networks become less linear through learning.

6.1 Architecture-specific results
In this section, we analyze why linear MLPs, ReLU MLPs and related architectures avoid exploding gradients but suffer from collapsing domains.

**Proposition 7.** For a linear MLP with randomly initialized weight matrices, \( Q^{-1}_{\theta}GSC(k, l) = 1 \) for all \( k \) and \( l \). (See section 9.5.7 for details.)

The key insight here is that linear layers multiply the forward activations and the gradient with the same weight matrix and so affect them in a similar way.

Let a ‘self-similar’ nonlinearity \( \sigma_{self-sim} \) with parameters \( c_{neg} \) and \( c_{pos} \) be defined as \( \sigma_{self-sim}(a) = c_{pos}a \) when \( a \geq 0 \) and \( \sigma_{self-sim} = c_{neg}a \) when \( a < 0 \).

**Proposition 8.** For any MLP composed of layers which are themselves composed of (i) a self-similar nonlinearity layer where the activation pattern does not depend on the parameter \( \theta \) and (ii) multiplication with a randomly initialized matrix, \( Q^{-1}_{\theta}GSC(k, l) = 1 \) for all \( k \) and \( l \). (See section 9.5.8 for details.)

Examples of self-similar nonlinearities are ReLU with \( c_{neg} = 0 \) and \( c_{pos} = 1 \) as well as leaky ReLU (Maas et al., 2013) where \( c_{neg} \) is arbitrary and \( c_{pos} = 1 \). The key insight here is that self-similar nonlinearity layers can be locally approximated as multiplication with a diagonal matrix, which can then be interpreted as just another weight matrix in a linear MLP.

To understand why linear MLPs comprised of random non-orthogonal weight matrices have a collapsing domain, we only need to remind ourselves that as these matrices are multiplied, the singular values of the product diverge. Therefore all inputs are ultimately collapsed onto the dominant eigenvector of the product of the weight matrices. Saxe et al. (2014) investigated this effect in detail.

**Proposition 9.** In any MLP composed of layers which are endomorphisms on the \( d \)-dimensional hypersphere composed of (i) a ReLU nonlinearity, (ii) multiplication with a full-rank, randomly
initialized matrix and (iii) length-only layer normalization, the volume of the domain collapses by factor $2^d$ per layer in expectation. (See section 9.5.9 for details.)

This helps explain why LOlayer-ReLU, and ultimately ReLU and layer-ReLU, exhibit a collapsing domain. As the forward activations pass through a ReLU layer, all but the positive $d$-dimensional quadrant is eliminated from the domain.

In this paper, we do not give a rigorous definition of the collapsing domain problem, because it is hard to assess and measure. For now, we are content to discuss the issues that arise when attempting to quantify it and to give a further overview of the problem itself. See section 9.2.3 for this discussion.

7 ResNet reduces the gradient - and ResNet explained

ResNet and related architectures that utilize skip connections have been very successful recently because they can be successfully trained to much greater depths than vanilla networks. In this section, we show how skip connections are able to greatly reduce the GSC and thus largely circumvent the exploding gradient problem.

There are a variety of styles of ResNet architectures. In this section, we focus our analysis on ResNets that are made up of $B$ residual blocks, each of which is bypassed by a skip connection. Each block is composed of 2 sub-blocks of 3 layers each: first a normalization layer, then a nonlinearity layer, and then a linear layer. Skip connections between layers of the same width are identity connections and skip connections between layers of different widths are slices of a fixed orthogonal matrix. Finally, we insert a single linear layer between the input and the first block and a normalization layer after the last block. This style of architecture is representative of a range of popular styles.

Let $s_b, 1 \leq b \leq B$ be the function computed by the $b$th skip connection. Let $\rho_b$ be the function computed by the $b$th residual block. $b = B$ corresponds to the lowest block and $b = 1$ corresponds to the highest block. Let $f_b = s_b + \rho_b$.

**Definition 4.** We say a function $f_b$ is $k$-diluted with respect to a random vector $v$ if there exists a matrix $S_b$ and a function $\rho_b$ such that $f_b(v) = S_b v + \rho_b(v)$ and $\frac{\|S_b v\|_2}{\|\rho_b(v)\|_2} = k$.

$k$-dilution expresses the idea that the kinds of functions that $f_b$ represents are of a certain form if $s_b$ is restricted to matrix multiplication. (The identity function can be viewed as matrix multiplication with the identity matrix.) The larger the value of $k$, the more $\rho_b$ is “diluted” by a linear function, bringing $f_b$ itself closer and closer to a linear function.

**Theorem 3.** Under certain conditions, if a function $\rho_b$ would cause the GSC to grow with expected rate $r_b$ then ignoring higher-order terms, $k$-diluting $\rho_b$ reduces this rate to $1 + \frac{r_{b+1}}{k^{b+1}}$. (See section 9.6.1 for details.)

This reveals the reason why ResNet circumvents the exploding gradient problem. Diluting the block function by a factor $k$ does not just reduce the growth of the GSC by factor $k$, but by $k^2+1$. Therefore what appears to be a relatively mild reduction in representational capacity achieves a relatively large amount of gradient reduction, and therefore ResNet can be trained successfully to “unreasonably” great depths.

To validate our theory, we repeated the experiments in figure 1 with 5 ResNet architectures: layer-ReLU, batch-ReLU, layer-tanh, batch-tanh and layer-SeLU. As we see in figure 3 (left graph), the gradient growth is indeed much reduced, with much of it taking place in the lower layers. In the middle graph, we find that the rate of domain collapse for layer-ReLU, as measured by sign diversity, is also significantly slowed.

We then went on to check whether the gradient reduction experienced is in line with theorem 3. We measured the dilution level $k_b$ and growth rate $r_b$ at each residual block $b$ and then replaced the growth rate with $1 + (k_{b+1}^2 + 1)(r_b - 1)$. The result of this post-processing is found in figure 3 (right graph). Indeed, the GSC now again grows almost linearly in log space. Therefore, the estimate of the magnitude of gradient reduction from theorem 3 is accurate. It is interesting to note that this
surrogate explosion rate in the ResNet architectures is not quite the same as the explosion rate of corresponding vanilla networks. See figure 1 to compare. Hence, introducing skip connections alters the mathematical properties of the network beyond the effect captured by theorem 3.

We then repeated the CIFAR10 experiments depicted in figure 2 with our 5 ResNet architectures. The results are shown in figure 5. As expected, in general, ResNet enables higher relative update sizes, achieves lower error, a higher effective depth and is less “robust” to taylor approximation than vanilla networks. The only exception to this trend is the layer-SeLU ResNet when compared to the SeLU vanilla network, which already has a relatively slowly exploding gradient to begin with. Also see table 2 to compare final error values. Note that in order to make the effective depth results comparable to those in figure 2, we applied the residual trick to ResNet. We let the initial function \( \mathcal{I} \) encompass not just the skip function \( \mathcal{S} \), but also the initial block function \( \mathcal{P} \). Hence, we set \( \mathcal{I}_b = \mathcal{S}_b + \mathcal{P}_b(\theta(0)) \) and \( \mathcal{R}_b = \mathcal{P}_b(\theta) - \mathcal{P}_b(\theta(0)) \). Note that our effective depth values for ResNet are much higher than those of Veit et al. (2016). This is because we use a much more conservative estimate of this intractable quantity.

Veit et al. (2016) argues that ResNets behave like an ensemble of shallow networks. We argue that vanilla networks behave like ensembles of even shallower networks.

Note that while the level of dilution increases as we go deeper in the style of ResNet architecture we studied in this section, this need not be so. The skip function \( \mathcal{S} \) and block function \( \mathcal{P} \) can be scaled with constants to achieve arbitrary, desired levels of dilution (Szegedy et al., 2016). Alternatively, instead of putting all normalization layers in the residual blocks, we can insert them between blocks / skip connections. This would keep the dilution level constant and hence cause gradients to explode again, though at a lower rate compared to vanilla networks. Finally, note that gradient reduction is achieved not just by identity skip connections but, as theorem 3 suggests, also by skip connections that multiply the incoming value with a fixed linear randomly initialized matrix. Results for those skip connections can be found in table 1.

### 7.1 The Limits of Dilution

\( k \)-dilution has its limits. Any \( k \)-dilated function with large \( k \) is close to a linear function. Hence, we can view \( k \)-dilution as another form of pseudo-linearity that can damage representational capacity. It also turns out that under similar conditions to those used in theorem 3, dilution only disappears slowly as diluted functions are composed.

**Proposition 10.** Under certain conditions, the composition of \( B \) random functions that are \( k_b \)-diluted in expectation respectively is \( \left( \prod_l (1 + \frac{1}{r_b}) \right)^{-\frac{1}{k_b}} \)-diluted in expectation. (See section 9.5.10 for details.)
More simply, assume all the $k_b$ are equal to some $k$. Ignoring higher-order terms, the composition is $\frac{1}{\sqrt{B}}k$-diluted. The flipside of an $O(k^2)$ reduction in gradient via dilution is thus the requirement of $O(k^2)$ layers to eliminate that dilution. This indicates that the overall amount of gradient reduction achievable through dilution may be limited.

7.2 THE ORTHOGONAL INITIAL STATE

Applying the residual trick to ResNet reveals several insights. The difference between ResNet and vanilla networks in terms of skip connections is superficial, because both ResNet and vanilla networks are residual networks in the framework of equation 2. Also, both ResNet and vanilla networks have nonlinear initial functions, because $\rho_b(\theta_b(0))$ must be included in $i_b$ for ResNet. However, there is one key difference. The initial functions of ResNet are closer to a linear transformation and indeed closer to an orthogonal transformation because they are composed of a nonlinear function $\rho_b(\theta_b(0))$ that is significantly diluted by an orthogonal transformation $s_b$. Therefore, ResNet, while being conceptually more complex, is mathematically simpler.

We have shown how ResNets achieve a reduced gradient via $k$-dilution. And just as with effective depth, the residual trick allows us to generalize this notion to arbitrary networks.

Definition 5. We say a residual network $f(\theta)$ has an orthogonal initial state if each initial function $i_l$ is multiplication with an orthogonal matrix or a slice / multiple thereof and $r_l(\theta_l)$ is the zero function.

Any network that is trained from an (approximate) orthogonal initial state benefits from reduced gradients via dilution. ResNet is a style of architecture that achieves this, but it is far from being the only one. Balduzzi et al. (2017) introduced the ‘looks-linear initialization’ for ReLU networks, which achieves not only a perfectly orthogonal initial state, but outperformed ResNet in their experiments. We detail this initialization scheme in section 9.8. In Table 2 we show that a simple ReLU network with looks-linear initialization can achieve an ever lower training error than ResNet on CIFAR10. In Figure 6 (top right), we find that indeed looks-linear initialization reduces the gradient growth of batch-ReLU drastically not just in the initialized state, but throughout training even as the residual functions grow significantly beyond the size they achieve under Gaussian initialization. A strategy related to looks-linear initialization that is even simpler is to initialize weight matrices as orthogonal matrices instead of Gaussian matrices. This reduces the gradient growth in the initialized state somewhat (Table 1).

Using the ensemble view of very deep networks reveals another significant disadvantage of non-orthogonal initial functions. The output computed by an ensemble member must pass through the initial functions of the layers not contained in that ensemble member to reach the prediction layer. Therefore, having non-orthogonal initial functions is akin to taking a shallow network and adding additional, untrainable non-orthogonal layers to it. This has obvious downsides such as a collapsing domain and / or exploding gradient, and an increasingly unfavorable eigenspectrum of the Jacobian (Saxe et al., 2014). One would ordinarily not make the choice to insert such untrainable layers. While there has been some success with convolutional networks where lower layers are not trained (e.g. Saxe et al. (2011); He et al. (2016b)), it is not clear whether such networks are capable of outperforming other networks where such layers are trained.

The big question is now: What is the purpose of not training a network from an orthogonal initial state? We are not aware of such a purpose. Since networks with orthogonal initial functions are mathematically simpler than other networks, we argue they should be the default choice. Using non-orthogonality in the initial function, we believe, is what requires explicit justification.

Balduzzi et al. (2017) asks in the title: If ResNet is the answer, then what is the question? We argue that a better question would be: Is there a question to which vanilla networks are the answer?

8 CONCLUSION

Summary In this paper, we revisited the classical problem of exploding gradients, drawing attention to a pathology that was mistakenly thought of as overcome, as well as clarifying the need
to view exploding gradients in relation to the scale of forward activations. We provided a rigorous link between gradient explosion and hardness of training, explained why gradients tend to explode even when forward activations do not and showcased the tradeoff between exploding gradients and a collapsing domain. Finally, we explained the favorable gradient flow of ResNet and proposed a foundational criterion for both the success of ResNet and deep network design in general: the orthogonal initial state.

Practical Recommendations

- **Train from an orthogonal initial state**, i.e. initialize the network such that it is a series of orthogonal linear transformations. This can reduce the growth of the GSC and domain collapse not just in the initial state, but also as training progresses. It can prevent the forward activations from having to pass through unnecessary non-orthogonal transformations. Even if a perfectly orthogonal initial state is not achievable, an architecture that approximates this such as ResNet can still confer significant benefit.

- **When not training from an orthogonal initial state, avoid low effective depth**. A low effective depth signifies that the network is composed of an ensemble of networks significantly shallower than the full network. If the initial functions are not orthogonal, the values computed by these ensemble members have to pass through what may be unnecessary nonlinear transformations. Low effective depth may be caused by, for example, exploding gradients or a collapsing domain.

- **Avoid pseudo-linearity**. For the representational capacity of a network to grow with depth, linear layers must be separated by nonlinearities. If those nonlinearities can be approximated by linear functions, they are ineffective. Pseudo-linearity can be caused by, for example, a collapsing domain.

- **As the GSC grows, adjust the step size**. If it turns out that some amount of growth of the GSC is unavoidable or desirable, weights in lower layers could benefit from experiencing a lower relative change during each update. Optimization algorithms such as RMSprop or Adam may partially address this.

We provide continued discussion in section 9.2.

REFERENCES


David Balduzzi, Marcus Frean, Lennox Leary, JP Lewis, Kurt Wan-Duo Ma, and Brian McWilliams. The shattered gradients problem: If resnets are the answer, then what is the question? In *ICML*, 2017.


## Appendix

### 9.1 Additional Experimental Results

<table>
<thead>
<tr>
<th>Nonlinearity</th>
<th>Normalization</th>
<th>Matrix type</th>
<th>Skip type</th>
<th>Width</th>
<th>$GSC(L, 0)$</th>
<th>St. Dev.</th>
<th>Sign Div.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReLU</td>
<td>none</td>
<td>Gaussian</td>
<td>none</td>
<td>100</td>
<td>1.52</td>
<td>0.22</td>
<td>0.030</td>
</tr>
<tr>
<td>ReLU</td>
<td>layer</td>
<td>Gaussian</td>
<td>none</td>
<td>100</td>
<td>1.16</td>
<td>0.096</td>
<td>0.029</td>
</tr>
<tr>
<td>ReLU</td>
<td>LOlayer</td>
<td>Gaussian</td>
<td>none</td>
<td>100</td>
<td>1.30</td>
<td>0.10</td>
<td>0.030</td>
</tr>
<tr>
<td>ReLU</td>
<td>batch</td>
<td>Gaussian</td>
<td>none</td>
<td>100</td>
<td>5728</td>
<td>1.00</td>
<td>0.41</td>
</tr>
<tr>
<td>tanh</td>
<td>none</td>
<td>Gaussian</td>
<td>none</td>
<td>100</td>
<td>1.26</td>
<td>0.096</td>
<td>0.50</td>
</tr>
<tr>
<td>tanh</td>
<td>layer</td>
<td>Gaussian</td>
<td>none</td>
<td>100</td>
<td>72.2</td>
<td>1.00</td>
<td>0.50</td>
</tr>
<tr>
<td>tanh</td>
<td>LOlayer</td>
<td>Gaussian</td>
<td>none</td>
<td>100</td>
<td>71.9</td>
<td>1.00</td>
<td>0.50</td>
</tr>
<tr>
<td>tanh</td>
<td>batch</td>
<td>Gaussian</td>
<td>none</td>
<td>100</td>
<td>93.6</td>
<td>1.00</td>
<td>0.50</td>
</tr>
<tr>
<td>SeLU</td>
<td>none</td>
<td>Gaussian</td>
<td>none</td>
<td>100</td>
<td>6.36</td>
<td>0.97</td>
<td>0.42</td>
</tr>
<tr>
<td>SeLU</td>
<td>LOlayer</td>
<td>Gaussian</td>
<td>none</td>
<td>100</td>
<td>7.00</td>
<td>0.98</td>
<td>0.42</td>
</tr>
<tr>
<td>ReLU</td>
<td>batch</td>
<td>Gaussian</td>
<td>none</td>
<td>100/200</td>
<td>5556</td>
<td>1.00</td>
<td>0.42</td>
</tr>
<tr>
<td>ReLU</td>
<td>batch</td>
<td>Gaussian</td>
<td>none</td>
<td>100/200</td>
<td>5527</td>
<td>1.00</td>
<td>0.41</td>
</tr>
<tr>
<td>SeLU</td>
<td>none</td>
<td>Gaussian</td>
<td>none</td>
<td>100/200</td>
<td>5.86</td>
<td>0.99</td>
<td>0.45</td>
</tr>
<tr>
<td>SeLU</td>
<td>none</td>
<td>Gaussian</td>
<td>none</td>
<td>100/200</td>
<td>6.09</td>
<td>0.98</td>
<td>0.43</td>
</tr>
<tr>
<td>ReLU</td>
<td>none</td>
<td>orthogonal</td>
<td>none</td>
<td>100</td>
<td>1.29</td>
<td>0.20</td>
<td>0.03</td>
</tr>
<tr>
<td>ReLU</td>
<td>layer</td>
<td>orthogonal</td>
<td>none</td>
<td>100</td>
<td>1.00</td>
<td>0.10</td>
<td>0.03</td>
</tr>
<tr>
<td>ReLU</td>
<td>batch</td>
<td>orthogonal</td>
<td>none</td>
<td>100</td>
<td>5014</td>
<td>1.00</td>
<td>0.42</td>
</tr>
<tr>
<td>tanh</td>
<td>none</td>
<td>orthogonal</td>
<td>none</td>
<td>100</td>
<td>1.18</td>
<td>0.10</td>
<td>0.50</td>
</tr>
<tr>
<td>tanh</td>
<td>layer</td>
<td>orthogonal</td>
<td>none</td>
<td>100</td>
<td>56.3</td>
<td>1.00</td>
<td>0.50</td>
</tr>
<tr>
<td>tanh</td>
<td>batch</td>
<td>orthogonal</td>
<td>none</td>
<td>100</td>
<td>54.6</td>
<td>1.00</td>
<td>0.50</td>
</tr>
<tr>
<td>SeLU</td>
<td>none</td>
<td>orthogonal</td>
<td>none</td>
<td>100</td>
<td>5.47</td>
<td>1.00</td>
<td>0.49</td>
</tr>
<tr>
<td>ReLU</td>
<td>none</td>
<td>looks-linear</td>
<td>none</td>
<td>100</td>
<td>1.00</td>
<td>1.00</td>
<td>0.50</td>
</tr>
<tr>
<td>ReLU</td>
<td>layer</td>
<td>looks-linear</td>
<td>none</td>
<td>100</td>
<td>1.00</td>
<td>1.00</td>
<td>0.50</td>
</tr>
<tr>
<td>ReLU</td>
<td>batch</td>
<td>looks-linear</td>
<td>none</td>
<td>100</td>
<td>1.00</td>
<td>1.00</td>
<td>0.50</td>
</tr>
<tr>
<td>ReLU</td>
<td>layer</td>
<td>Gaussian</td>
<td>identity</td>
<td>100</td>
<td>1.08</td>
<td>0.56</td>
<td>0.19</td>
</tr>
<tr>
<td>ReLU</td>
<td>batch</td>
<td>Gaussian</td>
<td>identity</td>
<td>100</td>
<td>4.00</td>
<td>1.00</td>
<td>0.48</td>
</tr>
<tr>
<td>tanh</td>
<td>layer</td>
<td>Gaussian</td>
<td>identity</td>
<td>100</td>
<td>1.63</td>
<td>1.00</td>
<td>0.50</td>
</tr>
<tr>
<td>tanh</td>
<td>batch</td>
<td>Gaussian</td>
<td>identity</td>
<td>100</td>
<td>1.57</td>
<td>1.00</td>
<td>0.50</td>
</tr>
<tr>
<td>SeLU</td>
<td>layer</td>
<td>Gaussian</td>
<td>identity</td>
<td>100</td>
<td>1.31</td>
<td>0.99</td>
<td>0.48</td>
</tr>
<tr>
<td>ReLU</td>
<td>layer</td>
<td>Gaussian</td>
<td>Gaussian</td>
<td>100</td>
<td>1.17</td>
<td>0.56</td>
<td>0.18</td>
</tr>
<tr>
<td>ReLU</td>
<td>batch</td>
<td>Gaussian</td>
<td>Gaussian</td>
<td>100</td>
<td>4.50</td>
<td>1.00</td>
<td>0.48</td>
</tr>
<tr>
<td>tanh</td>
<td>layer</td>
<td>Gaussian</td>
<td>Gaussian</td>
<td>100</td>
<td>1.97</td>
<td>1.00</td>
<td>0.50</td>
</tr>
<tr>
<td>tanh</td>
<td>batch</td>
<td>Gaussian</td>
<td>Gaussian</td>
<td>100</td>
<td>1.71</td>
<td>1.00</td>
<td>0.50</td>
</tr>
<tr>
<td>SeLU</td>
<td>layer</td>
<td>Gaussian</td>
<td>Gaussian</td>
<td>100</td>
<td>1.53</td>
<td>9.97</td>
<td>0.48</td>
</tr>
</tbody>
</table>

Table 1: Key metrics for popular architectures in their randomly initialized state evaluated on Gaussian noise. In the ‘Normalization’ column, ‘layer’ refers to layer normalization, ‘batch’ refers to batch normalization, ‘LOlayer’ refers to length-only layer normalization and ‘none’ refers to an absence of a normalization layer. In the ‘Matrix type’ column, ‘Gaussian’ refers to matrices where each entry is drawn from an independent Gaussian distribution with mean zero and a standard deviation that is constant across all entries. ‘orthogonal’ refers to a uniformly random orthogonal matrix and ‘looks-linear’ refers to the initialization scheme proposed by Balduzzi et al. (2017) and expounded in section 9.8. In the ‘Skip type’ column, ‘identity’ refers to identity skip connections, ‘Gaussian’ refers to skip connections that multiply the incoming value with a matrix where each entry is drawn from an independent unit Gaussian and ‘none’ refers to an absence of skip connections. In the ‘Width’ column, ‘100/200’ refers to linear layers having widths alternating between 100 and 200. ‘St. Dev.’ refers to pre-activation standard deviation as introduced in section 3. ‘Sign Div.’ refers to pre-activation sign diversity as introduced in section 3. For further details, see section 9.9. Red values indicate gradient explosion or pseudo-linearity.
Under review as a conference paper at ICLR 2018

<table>
<thead>
<tr>
<th>Nonlinearity</th>
<th>Normalization</th>
<th>Matrix type</th>
<th>Skip type</th>
<th>Error (custom step size)</th>
<th>Error (single step size)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReLU</td>
<td>none</td>
<td>Gaussian</td>
<td>none</td>
<td>31.48%</td>
<td>19.24%</td>
</tr>
<tr>
<td>ReLU</td>
<td>layer</td>
<td>Gaussian</td>
<td>none</td>
<td>42.48%</td>
<td>21.23%</td>
</tr>
<tr>
<td>ReLU</td>
<td>batch</td>
<td>Gaussian</td>
<td>none</td>
<td>34.83%</td>
<td>76.65%</td>
</tr>
<tr>
<td>tanh</td>
<td>none</td>
<td>Gaussian</td>
<td>none</td>
<td>23.42%</td>
<td>16.22%</td>
</tr>
<tr>
<td>tanh</td>
<td>layer</td>
<td>Gaussian</td>
<td>none</td>
<td>1.92%</td>
<td>17.5%</td>
</tr>
<tr>
<td>tanh</td>
<td>batch</td>
<td>Gaussian</td>
<td>none</td>
<td>12.31%</td>
<td>23.8%</td>
</tr>
<tr>
<td>SeLU</td>
<td>none</td>
<td>Gaussian</td>
<td>none</td>
<td>0.24%</td>
<td>1.78%</td>
</tr>
<tr>
<td>ReLU</td>
<td>none</td>
<td>looks-linear</td>
<td>none</td>
<td><strong>0.002%</strong></td>
<td><strong>0.008%</strong></td>
</tr>
<tr>
<td>ReLU</td>
<td>layer</td>
<td>looks-linear</td>
<td>none</td>
<td><strong>0.77%</strong></td>
<td>1.2%</td>
</tr>
<tr>
<td>ReLU</td>
<td>batch</td>
<td>looks-linear</td>
<td>none</td>
<td>0.38%</td>
<td><strong>0.19%</strong></td>
</tr>
<tr>
<td>tanh</td>
<td>layer</td>
<td>Gaussian</td>
<td>id</td>
<td>0.35%</td>
<td><strong>0.27%</strong></td>
</tr>
<tr>
<td>tanh</td>
<td>batch</td>
<td>Gaussian</td>
<td>id</td>
<td><strong>0.13%</strong></td>
<td>0.24%</td>
</tr>
<tr>
<td>ReLU</td>
<td>layer</td>
<td>Gaussian</td>
<td>id</td>
<td>2.09%</td>
<td><strong>1.49%</strong></td>
</tr>
<tr>
<td>ReLU</td>
<td>batch</td>
<td>Gaussian</td>
<td>id</td>
<td><strong>0.06%</strong></td>
<td>0.096%</td>
</tr>
<tr>
<td>SeLU</td>
<td>layer</td>
<td>Gaussian</td>
<td>id</td>
<td><strong>1.55%</strong></td>
<td><strong>1.55%</strong></td>
</tr>
</tbody>
</table>

Table 2: Training classification error for popular architectures trained on CIFAR10. In the ‘Normalization’ column, ‘layer’ refers to layer normalization, ‘batch’ refers to batch normalization and ‘none’ refers to an absence of a normalization layer. In the ‘Matrix type’ column, ‘Gaussian’ refers to matrices where each entry is drawn from an independent Gaussian distribution with mean zero and a standard deviation that is constant across all entries. ‘looks-linear’ refers to the looks-linear initialization scheme proposed by [Balduzzi et al., 2017] and expounded in section 9.8. In the ‘Skip type’ column, ‘identity’ refers to identity skip connections and ‘none’ refers to an absence of skip connections. In the two rightmost columns, we show the training classification error achieved when using a single step size and when using a custom step size for each layer. For further methodological details, see section 9.9. For a detailed breakdown of these results, see figures 2, 4, 5 and 6.
Figure 4: Key metrics for architectures with collapsing domain trained on CIFAR10. The top left graph shows the estimated optimal relative update size in each layer according to the algorithm described in section 9.9.3. Remaining graphs show results obtained from training with a single step as this achieved lower error than training with layer-wise step sizes (see table 2). The top two rows are equivalent to graphs in figure 2. The bottom row shows pre-activation standard deviation and pre-activation sign diversity (see section 6 for explanation) of the highest nonlinearity layer as training progresses.
Figure 5: Key metrics for ResNet architectures trained on CIFAR10. The top left graph shows the estimated optimal relative update size in each layer according to the algorithm described in section 9.9.3. Remaining graphs show results obtained from training with either those step sizes or a single step size, whichever achieved a lower error (see table 2). The top two rows are equivalent to graphs in figure 2. The bottom row shows pre-activation standard deviation and pre-activation sign diversity (see section 2 for explanation) of the highest nonlinearity layer as training progresses.
Figure 6: Key metrics for ReLU-based architectures with looks-linear initialization trained on CIFAR10. The top left graph shows the estimated optimal relative update size in each layer according to the algorithm described in section 9.9.3. Remaining graphs show results obtained from training with either those step sizes or a single step size, whichever achieved a lower error (see table 2). The top two rows are equivalent to graphs in figure 2. The bottom row shows pre-activation standard deviation and pre-activation sign diversity (see section 5 for explanation) of the highest nonlinearity layer as training progresses.
9.2 Discussion

9.2.1 Exploding Gradients vs. Other Metrics - Related Work

So far, we have discussed exploding gradients and collapsing domains. In this section, we review related metrics and concepts from literature.

We build on the work of Balduzzi et al. (2017), who introduced the concept of gradient shattering. This states that in deep networks, gradients with respect to nearby points become more and more uncorrelated with depth. This is very similar to saying that the gradient is only informative in a smaller and smaller region around the point at which it is taken. This is precisely what happens when gradients explode and also, as we argue in section 6, when the domain collapses. Therefore, the exploding gradient problem and collapsing domain problem can be viewed as a further specification of the shattering gradient problem rather than as a counter-theory or independent phenomenon.

We extend the work of Balduzzi et al. (2017) in several important ways. First, as mentioned in the introduction, we correct the mistaken claim that the exploding gradient problem “has been largely overcome”. Second, by using effective depth we make a rigorous argument as to why exploding gradients cause hardness of training. While Balduzzi et al. (2017) point out that shattering gradients interfere with theoretical guarantees that exist for various optimization algorithms, they do not provide a definitive argument as to why shattering gradients are in fact a problem. Third, our analysis extends beyond ReLU networks.

We also build on the work of Raghu et al. (2017). They showed that both trajectories and small perturbations, when propagated forward, can increase exponentially in size. However, they do not distinguish too important cases: (i) an explosion that is simply due to an increase in the scale of forward activations and (ii) an explosion that is due to an increase in the gradient relative to forward activations. We are careful to make this distinction and focus only on case (ii). Since this is arguably the more interesting case, we believe the insights generated in our paper are more robust.

Saxe et al. (2014) investigated another important pathology of very deep networks: the divergence of singular values in multi-layer Jacobians. As layer-wise Jacobians are multiplied, the variances of their singular values compound. This leads to the direction of the gradient being determined by the dominant eigenvectors of the multi-layer Jacobian rather than the label, which slows down training considerably.

In their seminal paper, Ioffe & Szegedy (2015) motivated batch normalization with the argument that changes to the distribution of intermediate representations, which they term ‘covariate shift’, are pathological and need to be combated. This argument was then picked up by e.g. Salimans & Kingma (2016) and Chunjie et al. (2017) to motivate similar normalization schemes. We are not aware of any rigorous definition of the ‘covariate shift’ concept nor do we understand why it is undesirable. After all, isn’t the very point of training deep networks to have each layer change the function it computes, to which other layers co-adapt, to which then other layers co-adapt and so on? Having each layer fine-tune its weights in response to shifts in other layers seems to be the very mechanism by which deep networks achieve high accuracy.

A classical notion of trainability in optimization theory is the conditioning of the Hessian. This can also deteriorate with depth. Recently, Luo (2017) introduced an architecture that ameliorates this pathology in an effective and computationally tractable way via iterative numerical methods and matrix decomposition. Matrix decomposition has also been used by e.g. Arjovsky et al. (2016) to maintain orthogonality of recurrent weight matrices. Maybe such techniques could also be used to reduce the divergence of singular values of the layer-Jacobian during training.

9.2.2 Exploding and Vanishing Gradients in RNNs

Exploding gradients and their counterpart, vanishing gradients, have been studied mostly in the context of on RNNs (e.g. Pascanu et al. (2013); Bengio et al. (1994)). It is important to note that the problem as it arises in RNNs is similar but also different from the exploding gradient problem in feedforward networks. The goal in RNNs is often to absorb information early on and store that information through many time steps and sometimes indefinitely. In the classical RNN architecture, signals acquired early would be subjected to a non-orthogonal transformation at every time step which lead to all the negative consequences described in this paper. LSTMs...
huber (1997) and GRUs Cho et al. (2014), which are the most popular solutions to exploding /
vanishing gradients in RNNs, solve this problem by simply leaving each neuron that is considered
part of the latent state completely unmodified from time step to time step unless new information is
received that is pertinent to that specific neuron. This solution does not apply in feedforward net-
works, because it is the very goal of each layer to modify the signal productively. Hence, managing
exploding gradients in feedforward networks is arguably more difficult.

Nevertheless, there is similarity between LSTM and the orthogonal initial state because both elimi-
nate non-orthogonality “as much as possible”. LSTM eliminates non-orthogonality completely from
time step to time step whereas in the orthogonal initial state, non-orthogonality is eliminated only
from the initial function. Again, viewing feedforward networks as ensembles of shallower networks,
orthogonal initial functions ensure that information extracted from each ensemble member does not
have to pass through non-orthogonal transformations needlessly. This is precisely what LSTM does.

9.2.3 ON THE DIFFICULTY OF ASSESSING COLLAPSING DOMAINS

It is difficult to assess or measure the degree to which the domain collapses in a given network, for
reasons outlined below.

First, there is an unlimited number of ways the domain might collapse. In the tanh architecture
we studied, the domain collapses onto the origin. In linear MLPs, the domain collapses onto the
line through the dominant eigenvector of the product of weight matrices, but never collapses onto a
single point. In ReLU, the domain collapses onto a ray from the origin.

Second, even if the volume of the domain collapses at a high rate, this collapse does not necessarily
translate into degenerate forward activations. In proposition 9, we show that the volume of the
domain under LOlayer-ReLU collapses at a very high rate. Comparing this with table 1, we find that
the standard deviation of activations of LOlayer-ReLU does not collapse nearly as quickly. This is
because in LOlayer-ReLU, the domain of forward activations is a union of spaces, many of which
have lower dimensionality than, in this case, the hypersphere. These spaces are not captured by
the volume measure. Consider the extreme case where the volume of the domain is zero. Then the
domain could still be composed of say, 1-dimensional curves. As long as the domain was composed
of sufficiently many of these curves, it could still be very expressive. The case where successive
layers have different dimensionality, which is not covered in propositions 6 and 9, also raises the
question of how to compare the sizes of domains of different dimensionality.

Third, even in architectures like LOlayer-tanh which provably do not suffer from a collapsing do-
main (proposition 6), we do not necessarily know whether almost all the probability mass of the
input distribution collapses onto one or more highly localized modes and if so how quickly. Such
a concentration of probability can be indistinguishable from a collapsing domain. Consider the
LOlayer-tanh architecture where weights have been initialized to very high values. Then the tanh
layers cause almost all the probability mass to collapse onto the corners of the hypercube. Also, con-
sider LOlayer-leaky ReLU with \( c_{neg} = 10^{-100} \). This does not have a collapsing domain (proposition
6) yet is practically indistinguishable from LOlayer-ReLU, which has a quickly collapsing domain
by volume (proposition 9). The difference between the two architectures is that while the domain
of LOlayer-ReLU is composed mainly of spaces of lower dimensionality, LOlayer-leaky ReLU as-
signs high probability on thin high-dimensional slices around those low-dimensional spaces and
very small but non-zero probability elsewhere.

9.2.4 ON CHOOSING NETWORK ARCHITECTURES

What architecture is best? Can we use the metrics presented in this paper to decide which non-
linearity, which normalization scheme and which initial weight scale is best? For example, when
comparing SeLU and layer-tanh, we find that neither seems to suffer from a collapsing domain
but layer-tanh’s gradients explode more quickly. Does that mean that SeLU is superior? Unfortu-
nately, it is difficult to make such judgments. The first reason is because of the difficulty of assessing
whether the domain collapses and if so how quickly and in what manner, as outlined in section 9.2.3.
The second reason is that we do not have a sufficient understanding of representational capacity.

Consider a linear MLP with weight matrices initialized to be orthogonal. It exhibits neither ex-
plooding gradients nor a collapsing domain in its initial state and only a slowly collapsing domain
as training progresses. Based on those metrics, it appears to be the perfect architecture. However, its representational capacity is very low. In general, we have no way of judging representational capacity. We might take an educated guess that e.g. leaky ReLU with $c_{\text{neg}} = 0.99$ has a low representational capacity. However, it is not clear why, say, a layer-tanh MLP with 10 linear layers could not have the same representational capacity as a SeLU MLP with 100 linear layers. If that were the case, relatively speaking, SeLU’s gradients would in fact be exploding much more quickly.

Further, while it is generally accepted that depth can be very helpful for modelling certain function, there is certainly no one-to-one correspondence with error. In table 2 and figure 6 (center left), we find that the looks-linear initialization achieves training errors as low or lower than ResNet while achieving an effective depth comparable to vanilla networks. The absence of non-orthogonal initial functions may make each of these relatively shallow sub-networks more powerful.

What gradient is best? $GSC(1, L)$ indicates the relative responsiveness of the prediction layer with respect to changes in the input layer. Of course, the goal in deep learning, at least within a prediction framework, is to model some ground truth function $t$ that maps data inputs to true labels. That function has itself a GSC at each input location $x$ that measures the relative responsiveness of $t(x)$ to changes in $x$. If the network was to perfectly represent the ground truth function, the GSCs would also have to match up. If, on the other hand, the GSC of the network differs significant from that of $t$, the network is not fitting $t$ well. This suggests that in fact, the “best” value of the GSC is one that matches that of the ground truth. If the GSC of the network is too low, we may experience underfitting. If the GSC of the network is too high, we may experience overfitting.

It is worth noting that the GSC measures the responsiveness of a higher layer with respect to a random change in a lower layer. However, if the change to the lower layer is aligned with the dominant eigenvectors of the Jacobian, the impact on the higher layer can be much greater. In the extreme case that $J^L_1$ has only one non-zero singular value, the relative responsiveness of the prediction is $\sqrt{d}$ times the GSC if the change to the input is aligned with the corresponding eigenvector. Therefore a network with, say, $GSC(L, 1) = 1$ can still be capable of modelling a function that is highly sensitive to certain features of the input.

Is the number of layers really important? ResNet was regarded as a milestone for enabling the training of very deep networks without a decline in performance. We argue that it is not challenging to train networks of arbitrary depth without a decline in performance, though this is not necessarily desirable. Consider a network with an orthogonal initial state such as a ReLU MLP with looks-linear initialization. Assume that at depth $L$ it can be trained to achieve some error with step size $s$. We conjecture that for any $k > 1$, the same architecture can be trained to achieve a similar error level at depth $kL$ by using step size $\frac{s}{k}$. In such a scenario, the effective depth would be bounded as $k$ increases. While we are far from arguing that effective depth as used in this paper is the “true” measure of depth, it may be a more realistic criterion than the apparent compositional depth for assessing what kind of functions a network can model in practice.

An orthogonal initial state only goes so far. An orthogonal initial state reduces gradients via dilution, which allows for relatively larger updates, which enables increased growth of residual functions, which allows for greater effective depth. However, as residual functions grow, dilution decreases, so the gradient increases, so updates must shrink, so the growth of residual functions slows, so the growth of effective depth slows.

In other words, for the network to become deeper, it needs to be shallow.

Therefore, while training from an orthogonal initial state can increase effective depth, we expect this effect to be limited. Additional techniques could be required to learn functions which require a compositional representation beyond this limit.

9.2.5 Future work

We plan to address convolutional layers and normalization layers with trainable bias and variance parameters. We expect all results in this paper to have analogues that apply to networks containing these layers. While the technical bookkeeping would be more complex, our analysis is not fundamentally dependent on fully-connected layers.
We are also interested in investigating the ability of deep networks to have a significantly differing gradient scale from one data input to the next; or to learn highly structured gradients. For example, given an image of a dog standing in a meadow, we might desire a high gradient with respect to pixels signifying e.g. facial features of the dog but a low gradient with respect to pixels that make up the meadow, and a uniformly low gradient given an image of a meadow. Such gradients would be very valuable not just in modelling real world functions more accurately, but in making the output of neural networks more explainable and avoiding susceptibility to adversarial attacks.

9.3 Further terminology and notation

- \( x \) and \( y \) are generally used to refer to the components of a datapoint. Then, we have \((x,y) \in D\).
- \( X \) refers to a vector of dimension \( d \), i.e. the same dimension as the \( x \) component of datapoints. Similarly, \( Y \) refers to an element of the domain of possible labels. We call \( X \) a ‘data input’ and \( Y \) a ‘label input’.
- \( F_l \) refers to a vector of dimension \( d_l \), i.e. the same dimension as \( f_1 \).
- We write \( f_l(\theta,x) \) as a short form of \( f_l(\theta, f_{l+1}(...f_L(\theta_L, x))..) \). Sometimes, we omit \( x \) and / or \( \theta \). In that case, \( x \) and / or \( \theta \) remains implicit. We write \( f_l(\theta, X) \) analogously to \( f_l(\theta,x) \).
- We write \( f_l(f_k) \) as a short form of \( f_l(\theta, f_{l+1}(...f_{k-1}(\theta_{k-1}, f_k)..)) \) where \( x \) (or \( X \)) and often \( \theta \) remain implicit. We write \( f_l(F_k) \) analogously.
- We use \( f_{L+1}, i_{L+1} \) and \( F_{L+1} \) interchangeably with \( x \) or \( X \).
- We say a random vector is ‘radially symmetric’ if its length is independent of its orientation.
- We say a random matrix is ‘Gaussian initialized’ if its entries are independent Gaussian random variables with mean zero and the standard deviation of all entries is the same.
- We say an \( m \times n \) random matrix is ‘orthogonally initialized’ if it is a fixed multiple of an \( m \times n \) submatrix of a \( \max(m,n) \times \max(m,n) \) uniformly random orthogonal matrix.
- We use parentheses \((\)\) to denote vector and matrix elements, i.e. \( A(3,4) \) is the fourth element in the third row of the matrix \( A \).

9.4 Effective depth: details

9.4.1 Formal definition

Let a ‘gradient-based algorithm’ for training a mutable parameter vector \( \theta \) from an initial value \( \theta^{(0)} \) for a network \( f \) be defined as a black box that is able to query the gradient \( \frac{df(\theta,X,Y)}{d\theta} \) at arbitrary query points \((X,Y)\) but only at the current value of the mutable parameter vector \( \theta \). It is able to generate updates \( \Delta \theta \) which are added to the mutable parameter vector \( \theta \). Let the sequence of updates be denoted as \( \Delta \theta^{(1)}, \Delta \theta^{(2)}, \ldots \). We define the successive states of \( \theta \) recursively as \( \theta^{(t)} = \theta^{(t-1)} + \Delta \theta^{(t)} \).

For simplicity, assume the algorithm is deterministic.

In a residual network defined according to equation \[2\] we can write the gradient with respect to a parameter sub-vector as \( \frac{df(\theta,X,Y)}{d\theta_l} = \frac{df_0(\theta_0,X,Y)}{d\theta_1} \ldots \frac{df_{l-1}(\theta_{l-1},X,Y)}{d\theta_l} \frac{df_l(\theta_l,X,Y)}{d\theta_l} \). Multiplying this out, we obtain \( g_l-1 \) terms. We call a term ‘\( \lambda \)-residual’ if it contains \( \lambda \) or more Jacobians of residual functions. Let \( \text{res}_1(\theta,X,Y) \) be the sum of all \( \lambda \)-residual terms in \( \frac{df(\theta,X,Y)}{d\theta_l} \).

Now consider two scenarios. In scenario (1), when the algorithm queries the gradient, it receives \( \{ \frac{df(\theta_0,X,Y)}{d\theta_1}, \frac{df(\theta_1,X,Y)}{d\theta_2}, \ldots, \frac{df(\theta_{l-1},X,Y)}{d\theta_l} \} \) i.e. the “regular” gradient. In scenario (2), it receives \( \{ \frac{df(\theta_0,X,Y)}{d\theta_1} - \text{res}_1(\theta,X,Y), \frac{df(\theta_1,X,Y)}{d\theta_2} - \text{res}_2(\theta,X,Y), \ldots, \frac{df(\theta_{l-1},X,Y)}{d\theta_l} - \text{res}_l(\theta,X,Y) \} \), i.e. a version of the gradient where all \( \lambda \)-residual terms are removed. Let the parameter vector attain states \( \theta^{(1)}, \theta^{(2)}, \ldots \) in scenario (1) and \( \theta^{(1,\lambda)}, \theta^{(2,\lambda)}, \ldots \) in scenario (2). Then we say the ‘\( \lambda \)-contribution’ at time \( t \) is \( \theta^{(t)} - \theta^{(t,\lambda)} \). Finally, we say the ‘effective depth’ at time \( t \) with threshold \( h \) is the largest \( \lambda \) such that there exists an \( l \) with \( \| \theta^{(t)} - \theta^{(t,\lambda)} \|_2 \geq h \).
There is no objective value for the threshold $h$. In practice, we find that the $\lambda$-contribution decreases quickly when $\lambda$ is increased beyond a certain point. Hence, the exact value of $h$ is not important when comparing different networks by effective depth.

The impact that the shift $\theta_i^{(t)} - \theta_i^{(t,\lambda)}$ has on the output of the network is influenced by the scale of $\theta_i^{(t)}$ as well as $GSC(l, 0)$. If those values vary enormously between layers, it may be advisable to set different thresholds for different layers.

### 9.4.2 Computational estimate

Unfortunately, computing the effective depth measure is intractable as it would require computing exponentially many gradient terms. In this section, we explain how we estimate effective depth in our experiments.

In this paper, we train networks only by stochastic gradient descent with either a single step size for all layers or a custom step size for each layer. Our algorithm for computing effective depth assumes this training algorithm.

**Vanilla networks** For the rest of this section we assume that the subscript $l$ ranges only over parametrized layers, which in our case means linear layers with no trainable bias parameters. Also, we assume that the network is expressed as a residual network as in equation 2.

Let $B$ be the batch size, let $c_i^{(t)}$ be the step size used at layer $l$ for the $t'$th update and let $((X^{(t,1)}, Y^{(t,1)}), (X^{(t,2)}, Y^{(t,2)}), \ldots, (X^{(t,b)}, Y^{(t,b)}))$ be the batch of query points used to compute the $t'$th update. Then SGD computes

$$
\Delta \theta_i^{(t)} = c_i^{(t)} \sum_b \frac{df_0(\theta^{(t-1)}, X^{(t,b)}, Y^{(t,b)})}{d\theta_i^{(t-1)}}
$$

$$
\theta_i^{(t)} = \theta_i^{(t-1)} + \Delta \theta_i^{(t)}
$$

For any update $t$ and query point $b$, we estimate its $\lambda$-contribution at layer $l$ as follows.

```
arr := [1];
for k = 0 to l - 1 do
    size = size(arr);
    arr.push_back(arr[size - 1] * ||r_k||_op);
    for i = size - 1 to 1 do
        arr[i] = arr[i] * \frac{||df_i^{(t-1)}||_2}{||df_i^{(t-1)}||_2} + arr[i - 1] * ||r_k||_op;
    end
    arr[0] = arr[0] * \frac{||df_0||_2}{||df_0||_2};
end
out = 0;
for i = \lambda to size(arr) - 1 do
    out = out + arr[\lambda];
end
return out * c_i^{(t)} * ||f_{t-1}||_2;
```

For unparametrized layers, $||r_k||_op$ is set to zero. For linear layers, it is the operator norm of the residual weight matrix. The final estimate of the length of the $\lambda$-contribution at layer $l$ for the entire training period is then simply the sum of the lengths of the estimated $\lambda$-contributions over all time points and query points.

The core assumption here is that applying the Jacobian of the initial function of a given layer will increase the lengths of all terms approximately equally, no matter how many residual Jacobians they contain. In other words, we assume that in $\lambda$-residual terms, the large singular values of Jacobians
do not compound disproportionately compared to other terms. This is similar to the core assumption in theorem [1] in section [9.6.1].

We conservatively bound the impact of the initial Jacobian with the impact of the layer Jacobian, i.e.

\[
\frac{\|\tilde{d}_{k+1}\|_2}{\|\tilde{d}_k\|_2}.
\]

We use \(\|r_k\|_{op}\) as a conservative estimate on how a residual Jacobian will increase the length of a term.

We use the sum of the lengths of all \(\lambda\)-residual terms in a batch as a conservative bound on the length of the \(\lambda\)-contribution of the batch. In essence, we assume that all \(\lambda\)-residual terms have the same orientation.

Finally, we use the sum of the lengths of the \(\lambda\)-contributions within each update as an estimate of the length of the total \(\lambda\)-contribution of the entire training period. On the one hand, this is conservative as we implicitly assume that the \(\lambda\)-contributions of each batch have the same orientation. On the other hand, we ignore indirect effects that \(\lambda\)-contributions in early batches have on the trajectory of the parameter value and hence on \(\lambda\)-contributions of later batches. Since we are ultimately interested in effective depth, we are ok with our estimate of the length of the \(\lambda\)-contribution to only be accurate if the \(\lambda\)-contribution is small. In that case, these indirect higher-order effects should be negligible.

Overall, we expect that our estimate of the effective depth is larger than its actual value. This is bolstered by the robustness of some of our trained networks to Taylor expansion (see figure 2).

**ResNet**  For ResNet architectures, we need to tweak our estimate of effective depth to take into account skip connections. Below, we detail how the variable \(arr\) is modified as it crosses a skip connection / residual block. We write \(f_n(f_m) = s_n(f_m) + \rho_n(f_m)\), where \(f_n\) is the layer at which the skip connection terminates, \(f_m\) is the layer at which the skip connection begins, \(s_n\) is the function computed by the skip connection and \(\rho_n(f_m) = \rho_n(f_{m+1}(..f_{m-1}(f_m)..))\) is the function computed by the residual block. We write \(f_k = i_k + r_k\) for \(n + 1 \leq k \leq m - 1\) and \(\rho_n = i_n + r_n\), i.e. we break down each layer in the residual block into an initial function and a residual function.

```python
arr_copy = arr;

for k = n to m - 1 do
    size = size(arr);
    arr.push_back(arr[size - 1] * ||r_k||_op);
    for i = size - 1 to 1 do
        arr[i] = arr[i] * \(\frac{\|d_{lk}\|_2}{\|d_{lk}\|_2}\) + arr[i - 1] * ||r_k||_op;
    end
    arr[0] = arr[0] * \(\frac{\|d_{0k}\|_2}{\|d_{0k}\|_2}\);
end

for i = 0 to size(arr_copy) - 1 do
    arr[i] = arr_copy[i] * \(\frac{\|d_{lk}\|_2 - \|d_{lk}\|_2}{\|d_{lk}\|_2}\);
end
```

This construction does not quite align with equation [2] as we break down each layer in the residual block into initial and residual function rather than the entire block. We made this choice so that our estimate for ResNets would be as comparable to our estimate for vanilla networks as possible.

The combined effect of the skip connection and the initial functions of the residual block is approximated by the effect of the entire block, i.e. \(\frac{\|d_{0m}\|_2}{\|d_{0m}\|_2}\). We must subtract the impact of the initial functions accumulated while passing through the residual block, i.e. \(-\frac{\|d_{0m}\|_2}{\|d_{0m}\|_2}\).

The impact of the residual functions in the block is, correctly, unaffected by the skip connection and bounded by the operator norm, as before.
9.4.3 DISCUSSION

The effective depth measure has several limitations.

One can train a linear MLP to have effective depth much larger than 1, but the result will still be equivalent to a depth 1 network.

Consider the following training algorithm: first randomly re-sample the weights, then apply gradient descent. Clearly, this algorithm is equivalent to just running gradient descent in any meaningful sense. However, the re-sampling of weights nonetheless blows up the residual functions so as to significantly increase effective depth.

The effective depth measure is very susceptible to the initial step size. In our experiments, we found that starting off with unnecessarily large step sizes, even if those step sizes were later reduced, lead to worse outcomes. However, because of the inflating impact on the residual function, the effective depth would be much higher nonetheless.

Effective depth may change depending on how layers are defined. In a ReLU MLP, for example, instead of considering a linear transformation and the following ReLU operation as different layers, we may define them to be part of the same layer. While the function computed by the network and the course of gradient-based training do not depend on such redefinition, effective depth can be susceptible to such changes.

9.5 PROPOSITIONS AND PROOFS

9.5.1 PROPOSITION

Proposition 1. Given:

- a neural network \( f \) of nominal depth \( L \)
- an initial parameter value \( \theta^{(0)} \)
- a mutable parameter value \( \theta \) that can take values in some closed, bounded domain \( \Theta \)
- a dataset \( D \) of datapoints \((x, y)\)
- a closed, bounded domain \( \mathcal{D} \) of possible query points \((X, Y)\)
- a function \( ||\cdot|| \) from matrices to the reals that has \( c||\cdot|| = ||c|| \) and \( ||\cdot|| \geq 0 \)
- some deterministic algorithm that is able to query gradients of \( f \) at the current parameter value and at query points in \( \mathcal{D} \) and that is able to apply updates \( \Delta \theta \) to the parameter value
- constant \( r' \)

Assume:

- Running the algorithm on \( f \) with \( \theta \) initialized to \( \theta^{(0)} \) for a certain number of updates \( T \)
  causes \( \theta \) to attain a value \( \hat{\theta} \) at which \( f \) attains some error value \( E_{\text{final}} \) on \( D \).
- At every triplet \((\theta, x, y)\) where the algorithm evaluates the gradient of \( f \), we have \( ||J^l_k|| \neq 0 \) and \( ||T^l_k|| \neq 0 \) for all \( 0 \leq l \leq k \leq L \).

Then we can specify some other neural network \( f' \) and some other initial parameter value \( \theta'^{(0)} \) such that the following claims hold:

1. \( f' \) has nominal depth \( L \).
2. The algorithm can be used to compute \( T \) updates by querying gradients of \( f' \) at the current parameter value and at query points in \( \mathcal{D} \) which cause \( \theta \) to attain a value \( \hat{\theta}' \) where \( f' \) attains error \( E_{\text{final}} \) on \( D \) and makes the same predictions as \( f(\hat{\theta}) \) on \( D \).
3. At every triplet \((\theta, X, Y)\) where the algorithm queries the gradient of \(f'\), we have \(||T^l_k||| \geq r^{k-l}|| for all \(0 \leq l \leq k \leq L\) and \(||J^l_k||| \geq r^{k-l}|| for all \(0 \leq l \leq k \leq L\) except \((k, l) = (1, 0)\).

Proof. Since \(\Theta\) and \(D\) are closed and bounded, so is \(\Theta \times D\). Therefore for all \(0 \leq l \leq k \leq L\), both \(||T^l_k|||\) and \(||J^l_k|||\) attain their infimum on that domain if it exists. \(||.||\) is non-negative, so the infimum exists. \(||.||\) is non-zero on the domain, so the infimum, and therefore the minimum, is positive. Since \(f\) has finite depth, there is an \(r\) such that for all tuplets \((\theta, X, Y, k, l)\), we have \(||J^l_k||| \geq r^{k-l}\) and \(||T^l_k||| \geq r^{k-l}\). Let \(R = \frac{r^l}{r^l} \).

Now, we define \(f'\) via its layer functions.

\[
\begin{align*}
    f'_0 &= f_0 \\
    f'_l(\theta_1, F_2) &= f_1(R\theta_1, R^2F_2) \\
    f'_l(\theta, F_{l+1}) &= R^{-1}f_l(R\theta_l, R^{l+1}F_{l+1}) \text{ for } 2 \leq l \leq L - 1 \\
    f'_L(\theta_L, X) &= R^{-L}f_L(R^L\theta_L, X)
\end{align*}
\]

\(f\) and \(f'\) clearly have the same nominal depth, so claim (1) holds. Given any vector \(v\) with \(L\) sub-vectors, define the transformation \(R(v)\) as \(R(v)_l = R^lv_l\). Finally, we set \(\theta'^{(0)} := R^{-1}(\theta^{(0)})\).

We use the algorithm to train \(f'\) as follows. Whenever the algorithm queries some gradient value \(\frac{df'}{d\theta}\), we instead submit to it the value \(R^{-1}(\frac{df'}{d\theta})\). Whenever the algorithm wants to apply an update \(\Delta \theta\) to the parameter, we instead apply \(R^{-1}(\Delta \theta)\). Let \(S^{(t)}\), \(0 \leq t \leq T\) be the state of the system after applying \(t\) updates to \(\theta\) under this training procedure. Let \(S^{(t)}\), \(0 \leq t \leq T\) be the state of the system after applying \(t\) updates to \(\theta\) when the algorithm is run on \(f\). Then the following invariances hold.

A \( \theta'^{(t)} = R^{-1}(\theta^{(t)})\), where \(\theta^{(t)}\) is the value of \(\theta\) under \(S^{(t)}\) and \(\theta'^{(t)}\) is the value of \(\theta\) under \(S^{(t)}\).

B \( f \) makes the same predictions and attains the same error on \(D\) under \(S^{(t)}\) as \(f' \) under \(S^{(t)}\).

C Any state the algorithm maintains is equal under both \(S^{(t)}\) and \(S'^{(t)}\).

We will show these by induction. At time \(t = 0\), we have \(\theta'^{(0)} = R^{-1}(\theta^{(0)})\) as chosen, so (A) holds. It is easy to check that (B) follows from (A). Since the algorithm has thus far not received any inputs, (C) also holds.

Now for the induction step. Assuming that \(\theta'^{(t)} = R^{-1}(\theta^{(t)})\), it is easy to check that \(\frac{df'}{d\theta} = R(\frac{df}{d\theta})\). Therefore, whenever the algorithm queries a gradient of \(f'\), it will receive \(R^{-1}(\frac{df'}{d\theta}) = R^{-1}(R(\frac{df}{d\theta})) = \frac{df}{d\theta}\). Therefore, the algorithm receives the same inputs under both \(S^{(t)}\) and \(S'^{(t)}\). Since the internal state of the algorithm is also the same, and the algorithm is deterministic, the update returned by the algorithm is also the same and so is the internal state after the update is returned, which completes the induction step for (C). Because the algorithm returns the same update in both cases, after the prescribed post-processing of the update under \(f'\), we have \(\Delta \theta'^{(t)} = R^{-1}(\Delta \theta^{(t)})\). Therefore \(\theta'^{(t+1)} = \theta'^{(t)} + \Delta \theta'^{(t)} = R^{-1}(\theta^{(t)}) + R^{-1}(\Delta \theta^{(t)}) = R^{-1}(\theta^{(t)} + \Delta \theta^{(t)}) = R^{-1}(\theta^{(t+1)})\). This completes the induction step for (A) and again, (B) follows easily from (A).

(B) implies directly that claim (2) holds. Finally, we note that for any gradient query during the training procedure, we have \(||T^l_k||| = ||\frac{df}{d\theta}||| = ||R^{-k-l}d\theta||| \geq r^{k-l}|| and unless \((k, l) = (1, 0)\) we have \(||J^l_k||| = ||\frac{df}{d\theta}||| = ||R^{-k-l}d\theta||| \geq r^{k-l}||. Therefore, claim (3) also holds, which completes the proof.
Proposition 2. Let $f$ be a function such that its Jacobians have non-zero norms. Consider a small change $\epsilon u$ in the input $X$ at some point $(\theta, X, Y)$.

**Proof.** We use $L \Sigma R^T$ to denote the singular value decomposition and $s_i$ to denote singular values.

\[
Q_u \frac{||f_i(f_k + \epsilon u) - f_i(f_k)||_2}{||f_k + \epsilon u - f_k||_2} = Q_u \frac{||f_i(f_k + \epsilon u) - f_i(f_k)||_2}{\epsilon ||f_i(f_k)||_2} \\
= \frac{||f_k||_2}{\epsilon ||f_i(f_k)||_2} Q_u ||f_i(f_k + \epsilon u) - f_i(f_k)||_2 \\
= \frac{||f_k||_2}{\epsilon ||f_i(f_k)||_2} Q_u ||f_i(f_k) + \epsilon \mathcal{J}_f u + O(\epsilon^2) - f_i(f_k)||_2 \\
\approx \frac{||f_k||_2}{||f_i(f_k)||_2} Q_u ||\mathcal{J}_f u||_2 \\
= \frac{||f_k||_2}{||f_i(f_k)||_2} Q_u ||L \Sigma R^T u||_2
\]
we have

Further, if \( \theta \) is connected linear layer without trainable bias parameters and \( \epsilon u \) weight matrix. Similarly, we will use \( \Sigma \) singular values.

Proof. Throughout this derivation, we will use \( \theta \) to refer to both the parameter sub-vector and the weight matrix. Consider a small multiple of a random variable drawn from \( \mathcal{U} \) that is a fully-differentiable linear layer without trainable bias parameters and \( \epsilon u \) weight matrix. Similarly, we will use \( \Sigma \) singular values.

When we take the limit as \( \epsilon \) goes to 0, \( O(\epsilon^2) \) terms become 0, and so the only \( \approx \) sign in this derivation becomes \( = \), and therefore the GSC is equal to the desired expression.

\[ \theta \]

9.5.3 Proposition [3]

Proposition [3] Let \( \mathcal{U} \) be the uniform distribution over the hypersphere. Assume \( f_k \) is a fully-connected linear layer without trainable bias parameters and \( \theta_k \) contains the entries of the weight matrix. At any point \( (\theta, X, Y) \) where \( f \) is differentiable, \( GSC(k, l) = \frac{||f_k||_2}{||f_k||_2} \) measures the quadratic expectation \( \mathbb{Q} \) of the relative size of the change in the value of \( f_k \) in response to a change in \( \theta_k \) that is a small multiple of a random variable drawn from \( \mathcal{U} \).

Equivalently, \( GSC(k, l) ||\theta_k||_q ||f_k||_q = \lim_{\epsilon \to 0} \mathbb{Q}_{\epsilon \rightarrow 0} \frac{||f_k(\theta_k + \epsilon u, f_k) - f_k(\theta_k, f_k)||_q}{||\theta_k + \epsilon u - \theta_k||_q} \)

Further, if \( \theta_k \) is random and

- all entries of \( \theta_k \) have the same quadratic expectation
- all products of two different entries of \( \theta_k \) have an expectation of 0
- \( \theta_k \) is radially symmetric

we have \( \mathbb{Q}_{\theta_k}^{-1} ||\theta_k||_q ||f_k||_q = 1 \).

Proof. Throughout this derivation, we will use \( \theta_k \) to refer to both the parameter sub-vector and the weight matrix. Similarly, we will use \( \epsilon u \) to refer to both a perturbation of the parameter sub-vector and of the weight matrix. We use \( \Sigma \) to denote the singular value decomposition and \( s_1 \) to denote singular values.

\[ \frac{||f_k(\theta_k + \epsilon u, f_k) - f_k(\theta_k, f_k)||_q}{||\theta_k + \epsilon u - \theta_k||_q} \]
$\|\theta_k\|_2^2 \frac{Q_u}{\epsilon}\|f_1(f_k(\theta_k + \epsilon u, f_{k+1})) - f_1(f_k(\theta_k, f_{k+1}))\|_2$

$= \|\theta_k\|_2^2 \frac{Q_u}{\epsilon}\|f_1((\theta_k + \epsilon u) f_{k+1}) - f_1(\theta_k f_{k+1})\|_2$

$= \|\theta_k\|_2^2 \frac{Q_u}{\epsilon}\|f_1(\theta_k f_{k+1} + \epsilon u f_{k+1}) - f_1(\theta_k f_{k+1})\|_2$

$= \|\theta_k\|_2^2 \frac{Q_u}{\epsilon}\|f_1(\theta_k f_{k+1}) + \mathcal{J}_k^i \epsilon u f_{k+1} + O(\epsilon^2) - f_1(\theta_k f_{k+1})\|_2$

$\approx \|\theta_k\|_2^2 \frac{Q_u}{\epsilon}\|\mathcal{J}_k^i \epsilon u f_{k+1}\|_2$

$= \|\theta_k\|_2^2 \frac{Q_u}{\epsilon}\|\mathcal{J}_k^i u f_{k+1}\|_2$

$= \|\theta_k\|_2^2 \frac{Q_u}{\epsilon}\|L\Sigma R^T u f_{k+1}\|_2$

$= \|\theta_k\|_2^2 \frac{Q_u}{\epsilon}\|\Sigma u f_{k+1}\|_2$

$= \|\theta_k\|_2^2 \frac{Q_u}{\epsilon}\|\left( \sum_{i=1}^{\min(d_k, d_l)} \sum_{j=1}^{d_{k+1}} s_{i,j} u(i, j) f_{k+1}(j) \right)^2 \|_2$

$= \|\theta_k\|_2^2 \frac{Q_u}{\epsilon}\|\mathbb{E}_u \left( \sum_{i=1}^{\min(d_k, d_l)} \sum_{j=1}^{d_{k+1}} s_{i,j} u(i, j) f_{k+1}(j) \right)^2 \|_2$

$= \|\theta_k\|_2^2 \frac{Q_u}{\epsilon}\|\mathbb{E}_u \left( \sum_{i=1}^{\min(d_k, d_l)} \sum_{j=1}^{d_{k+1}} \sum_{m=1}^{d_{k+1}} s_{i,j}^2 f_{k+1}(j) f_{k+1}(m) \mathbb{E}_u u(i, j) u(i, m) \right) \|_2$

$= \|\theta_k\|_2^2 \frac{Q_u}{\epsilon}\|\mathbb{E}_u \left( \sum_{i=1}^{\min(d_k, d_l)} \sum_{j=1}^{d_{k+1}} \sum_{m=1}^{d_{k+1}} s_{i,j}^2 f_{k+1}(j) f_{k+1}(m) \mathbb{E}_u u(i, j) \right) \|_2$

$= \|\theta_k\|_2^2 \frac{Q_u}{\epsilon}\|\sum_{i=1}^{\min(d_k, d_l)} \sum_{j=1}^{d_{k+1}} s_{i,j}^2 f_{k+1}(j) f_{k+1}(m) \mathbb{E}_u u(i, j) \|_2$

$= \|\theta_k\|_2^2 \frac{Q_u}{\epsilon}\|\sum_{i=1}^{\min(d_k, d_l)} \sum_{j=1}^{d_{k+1}} s_{i,j}^2 f_{k+1}(j) \|_2^2 \mathbb{E}_u u(i, j) \|_2$

$= \|\theta_k\|_2^2 \frac{Q_u}{\epsilon}\\frac{1}{d_k} \sum_{i=1}^{\min(d_k, d_l)} \sum_{j=1}^{d_{k+1}} s_{i,j}^2 f_{k+1}(j) \|_2^2$

$= \|\theta_k\|_2^2 \|\mathcal{J}_k^i\|_2 \|f_{k+1}\|_2$

$= GSC(k, l) \|\theta_k\|_2 \|f_{k+1}\|_2$
When we take the limit as $\epsilon$ goes to 0, $O(\epsilon^2)$ terms become 0, and so the only $\approx$ sign in this derivation becomes $=$, and therefore the desired equality.

Further, assume that $\theta_k$ is random and fulfills the conditions stated. Under those conditions, $\theta_k$ is the product of a random scalar length variable $\ell$ and an independent random vector orientation variable $\theta'_k$ of unit length. Then for all $1 \leq i \leq d_k$ and $1 \leq j \leq d_{k+1}$, we have $E\theta_k(i, j)^2 = E\ell^2 \theta'_k(i, j)^2 = E\ell^2 E\theta'_k(i, j)^2$ and so $Q \theta'_k(i, j) = Q \theta'_k(i, j) = \frac{Q \theta_k(i, j)}{\ell}$. And so since all entries of $\theta_k$ have the same quadratic expectation, all entries in $\theta'_k$ have the same quadratic expectation. Further, $1 = ||\theta'_k||_2 = Q||\theta'_k||_2 = \sqrt{E \sum_{i,j} \theta'_k(i, j)^2} = \sqrt{d_k d_{k+1} E \theta'_k(i, j)^2}$. So $Q \theta'_k(i, j)$, so the quadratic expectation of each entry of $\theta'_k$ is $Q \theta'_k(i, j)$. Further, for all $1 \leq i_1, i_2 \leq d_k$ and $1 \leq j_1, j_2 \leq d_{k+1}$ with $(i_1, j_1) \neq (i_2, j_2)$, we have $0 = E \theta_k(i_1, j_1) \theta_k(i_2, j_2) = E \theta'_k(i_1, j_1) \theta'_k(i_2, j_2) \ell = E \ell^2 E \theta'_k(i_1, j_1) \theta'_k(i_2, j_2) \ell$, so the expectation of the product of two different entries of $\theta'_k$ is 0.

Then, we have:

\[
Q \theta'_k - \frac{||\theta_k||_2 ||f_{k+1}||_2}{\sqrt{d_{k+1}}} = \frac{||f_{k+1}||_2}{\sqrt{d_{k+1}}} Q^{-1} \frac{||\theta_k||_2}{||f_k||_2} \frac{||f_{k+1}||_2}{\sqrt{d_{k+1}}} Q^{-1} \frac{||\theta_k||_2}{||f_k||_2} \frac{||f_{k+1}||_2}{\sqrt{d_{k+1}}} E \frac{||\theta_k f_{k+1}||_2^2}{||\theta_k||_2^2} = \frac{||f_{k+1}||_2}{\sqrt{d_{k+1}}} E \frac{||\theta_k f_{k+1}||_2^2}{||\theta_k||_2^2} = \frac{||f_{k+1}||_2}{\sqrt{d_{k+1}}} E \frac{||\theta_k f_{k+1}||_2^2}{||\theta_k||_2^2} \frac{||f_{k+1}||_2}{\sqrt{d_{k+1}}} \sum_i \sum_j \theta'_k(i, j) f_{k+1}(j)^2 = \frac{||f_{k+1}||_2}{\sqrt{d_{k+1}}} \sum_i \sum_j \theta'_k(i, j) f_{k+1}(j)^2 \frac{||f_{k+1}||_2}{\sqrt{d_{k+1}}} \sum_j f_{k+1}(j)^2 \frac{||f_{k+1}||_2}{\sqrt{d_{k+1}}} \sum_i Q \theta'_k(i, j)^2 = \frac{||f_{k+1}||_2}{\sqrt{d_{k+1}}} \sum_j f_{k+1}(j)^2 \frac{||f_{k+1}||_2}{\sqrt{d_{k+1}}} \sum_i (Q \theta'_k(i, j))^2 = \frac{||f_{k+1}||_2}{\sqrt{d_{k+1}}} \sum_j f_{k+1}(j)^2 \frac{||f_{k+1}||_2}{\sqrt{d_{k+1}}} \sum_i (Q \theta'_k(i, j))^2
\]
\[
\frac{||f_{k+1}||_2}{\sqrt{d_{k+1}}} = \frac{\sum_j f_{k+1}(j)^2 \sum_i \frac{1}{d_k d_{k+1}}}{\sqrt{||f_{k+1}||_2^2 d_k \frac{1}{d_k d_{k+1}}}} = \frac{1}{\sqrt{L}} \cdot \sqrt{d_{k+1}} = 1
\]

\(\blacksquare\)

The conditions stated in the proposition for the random parameter sub-vector \(\theta_k\) are fulfilled, for example, if the corresponding weight matrix is either Gaussian initialized or orthogonally initialized. Therefore, the most popular initialization strategies for weight matrix initialization are covered by this proposition.

9.5.4 Proposition

Proposition\(\textsuperscript{4}\) Given:

- some network \(f\) of nominal depth \(L\)
- some parameter value \(\theta = (\theta_1, ..., \theta_L)\)
- constants \(c_0, ..., c_L\) and \(\gamma_1, ..., \gamma_L\)
- a network \(f'\) of nominal depth \(L\) defined via its layer functions as follows.

\[
\begin{align*}
    f'_0(y, F_1) &= c_0 f_0(y) \frac{1}{c_1} F_1 \\
    f'_l(\theta_l, F_{l+1}) &= c_l f_l(\gamma_l \theta_l, \frac{1}{\gamma_{l+1}} F_{l+1}) \quad \text{for } 1 \leq l \leq L - 1 \\
    f'_L(\theta_L, X) &= c_L f_L(\gamma_L \theta_L, X)
\end{align*}
\]

- a parameter value \(\theta' = (\theta'_1, ..., \theta'_L)\) defined via \(\theta'_l = \frac{1}{\gamma_l} \theta_l\)

Then for all tuples \((\theta, \theta', X, Y)\) where \(f(\theta, X, Y)\) and \(f'(\theta', X, Y)\) are differentiable, \(GSC(k, l, f, \theta, X, Y) = GSC(k, l, f', \theta', X, Y)\).

Proof. We have \(||f'_l||_2 = c_l ||f_l||_2\) for \(0 \leq l \leq L\) and we have \(||J'_k||_{qm} = \|\frac{\partial}{\partial \theta} J'_k\|_{qm} = \|\frac{\partial}{\partial \theta} J'_k\|_{qm} = \frac{c_k}{c_k} ||J'_k||_{qm}\) for \(0 \leq l \leq k \leq L\), so \(GSC(k, l) = \frac{||J'_k||_{qm} ||f'_k||_2}{||f'_k||_2} = \frac{c_k}{c_k} ||J'_k||_{qm} ||f'_k||_2 = GSC(k, l)\) for \(0 \leq l \leq k \leq L\) as required. \(\blacksquare\)

Here, we consider general multiplicative rescalings provided they do not change the function represented by the network. To ensure this, each layer function must compensate for the factor introduced by the previous layer as well as for the rescaling of the parameter. Not all network transformations that are intended to control the scale of forward activations fall under this proposition. Changing the scale of weights in a tanh or Selu MLP or adding normalization layers is not covered. These changes can have a drastic impact on the mathematical properties of the network, as shown throughout the paper. On the other hand, changing the scale of weights in a ReLU architecture or any architecture with a normalization layer is covered by the proposition.

9.5.5 Proposition

Proposition\(\textsuperscript{5}\) Assuming the approximate decomposability of the norm of the product of Jacobians, i.e. \(\|J_{l+1}^{t+1} J_{l+2}^{t+1} \cdots J_k^{t+1}\|_{qm} \approx \|J_{l+1}^{t+1}\|_{qm} \|J_{l+2}^{t+1}\|_{qm} \cdots \|J_k^{t+1}\|_{qm}\), we have \(GSC(k, l) \approx GSC(k, k - 1) GSC(k - 1, k - 2) \cdots GSC(l + 1, l)\).
Lemma 1. Let this point on the ray be $x$. Proposition 6. The dimension of the hypersphere be $d$ intersected with the hypersphere, yields a single point. We will show this via a series of claims. Let $\sigma$ that has $\sigma(0) = 0$, (ii) multiplication with a full-rank matrix and (iii) length-only layer normalization is bijective.

Proof. We will prove this by showing that the inverse image of any point under such an endomorphism is a single point. Assume its inverse image contained two points $y$ and $z$. Then for $f(i) = x(i)$ and $f(z(i)) = x(i)$ and so $\sigma(y(i)) = \sigma(z(i))$. But $y \neq z$, so there exists an $i$ such that $y(i) \neq z(i)$. So there exist two different values $y(i)$ and $z(i)$ at which $\sigma$ returns the same result. But $\sigma$ is strictly monotonic. Contradiction.

Claim 1: If a point on this ray has an inverse image, that inverse image is a single point.

Claim 2: If two points $x_1$ and $x_2$ on the ray have inverse images $y_1$ and $y_2$ and $x_1$ is closer to the origin than $x_2$, then for $1 \leq i \leq d$, we have $|y_1(i)| \leq |y_2(i)|$.

For $1 \leq i \leq d$, $\sigma$ attains $x_2(i)$ at $y_2(i)$ and 0 at 0. Since $\sigma$ is strictly monotonic, continuous and $0 \leq |x_1(i)| \leq |x_2(i)|$ and $x_1(i)$ and $x_2(i)$ have the same sign, $\sigma$ attains $x_1(i)$ at a point between 0 and $y_2(i)$. Hence, $|y_1(i)| \leq |y_2(i)|$ as required.

Claim 3: The function $f$ that assigns to each point on the ray that has an inverse image the length of that inverse image is strictly increasing in the direction away from the origin.

Take any two points on the ray $x_1$ and $x_2$ that have inverse images $y_1$ and $y_2$ where $x_1$ is closer to the origin. By the previous claim, for $1 \leq i \leq d$, we have $|y_1(i)| \leq |y_2(i)|$ and therefore $|y_1(i)| \leq |y_2(i)|$ and therefore $f(x_1) \leq f(x_2)$. Assume $f(x_1) = f(x_2)$. Then we must have $|y_1(i)| = |y_2(i)|$ for $1 \leq i \leq d$. Since $\sigma$ is strictly monotonic and $\sigma(0) = 0$, $\sigma$ either preserves the sign of all inputs or reverses the sign of all inputs. Since $x_1(i)$ and $x_2(i)$ have the same sign, so do $y_1(i)$ and $y_2(i)$. So $y_1(i) = y_2(i)$, so $y_1 = y_2$, so the forward images of $y_1$ and $y_2$ are the same, so $x_1 = x_2$. Contradiction. So $f(x_1) \neq f(x_2)$, so $f(x_1) < f(x_2)$.

Claim 4: The function $f$ that assigns to each point on the ray that has an inverse image the length of that inverse image is continuous.

\[ GSC(k, l) = \left\| J_k^{\sigma} \right\|_{qm} \| f_k \|_2 \]

\[ = \left\| J_{i+1}^{\sigma} J_{i+2} \cdots J_{k-1}^{\sigma} \right\|_{qm} \| f_k \|_2 \]

\[ = \left\| J_{i+1}^{\sigma} J_{i+2} \cdots J_{k-1}^{\sigma} \right\|_{qm} \| f_k \|_2 \]

\[ = \left\| J_{i+1}^{\sigma} J_{i+2} \cdots J_{k-1}^{\sigma} \right\|_{qm} \| f_k \|_2 \]

\[ = GSC(k, k - 1) GSC(k - 1, k - 2) \cdots GSC(l + 1, l) \]

\[ \square \]

9.5.6 Proposition. Any endomorphism on the hypersphere composed of (i) a strictly monotonic, continuous nonlinearity $\sigma$ that has $\sigma(0) = 0$, (ii) multiplication with a full-rank matrix and (iii) length-only layer normalization is bijective.
Since $f$ is only defined on a 1-dimensional space, it is enough to show left-continuity and right-continuity.

Part 1: left-continuity. Take a sequence of points on the ray with inverse images that approach some point $x_{\text{lim}}$ on the ray from the left and assume that $x_{\text{lim}}$ also has an inverse image $y_{\text{lim}}$. Then we need to show that the length of $y_{\text{lim}}$ is the limit of the lengths of the inverse images of the sequence. It is enough to show this for the monotonic re-ordering of the sequence. Let that monotonic re-ordering be $x_n$. Then we have $x_n \to x_{\text{lim}}$ and $||x_n||$ increases. By claim 2, for $1 \leq i \leq d$, $|y_n(i)|$ is an increasing sequence. This means that $y_{\text{lim}}(i)$ either converges or it is an unbounded sequence. If the latter is true, then it will exceed $|y_{\text{lim}}(i)|$. But since $x_{\text{lim}}$ is at least as large as all $x_n$, again by claim 2 we must have $|y_{\text{lim}}(i)| \geq |y_n(i)|$. Contradiction. So $|y_n(i)|$ converges. Since $\sigma$ is strictly monotonic and $\sigma(0)$, $\sigma$ either preserves the sign of all values or it reverses the sign of all values. Since the $x_n(i)$ all have the same sign because the $x_n$ are on a ray, the $y_n(i)$ all have the same sign and so since $|y_n(i)|$ converges, $y_n(i)$ converges. Let its limit be $y_{\text{lim}}(i)$. Because $\sigma$ is continuous, its value at $y_{\text{lim}}(i)$ is the limit of $\sigma(y_n(i))$. But that is $y_{\text{lim}}(i)$. So if $y_{\text{lim}}$ is the vector made up of the $y_{\text{lim}}(i)$, it is the inverse image of $x_{\text{lim}}$, i.e. $y_{\text{lim}}' = y_{\text{lim}}$. Since $||y_n||_2$ is also a continuous function, $||y_n||_2 \to ||y_{\text{lim}}||_2$ and so $||y_n||_2 \to ||y_{\text{lim}}||_2$ and so $f(x_n) \to f(x)$ as required.

Part 2: right-continuity. This case is analogous. We have a decreasing sequence $x_n$ and so decreasing sequences $|y_n(i)|$ and so convergent sequences $y_n(i)$ with a limit $y_{\text{lim}}'$ that is equal to $y_{\text{lim}}$ and so $f(x_n) \to f(x)$ as required.

Claim 5: The co-domain of the function $f$ that assigns to each point on the ray that has an inverse image the length of that inverse image is the positive reals.

We argue by contradiction. Assume the co-domain of $f$ is not the positive reals. Then the set $S$ of positive reals not attained by $f$ is non-empty. Let $s$ be the infimum of $S$.

Case 1: $s = 0$. Since $\sigma$ is strictly monotonic with $\sigma(0) = 0$, there exists an $\epsilon > 0$ such as $\sigma$ attains all values in the interval $[-\epsilon, \epsilon]$. So all points on the ray for which all components have absolute value less than $\epsilon$ have an inverse image, so there exists an $\epsilon' > 0$ such that all points on ray with length in the interval $(0, \epsilon']$ have an inverse image, so $f$ is defined there. Further, we extend the domain of $f$ to include the origin, we have $f(0) = 0$. But by the same argument used for claim 4, $f$ is then right-continuous at 0. Since $f$ is also continuous and defined on $(0, \epsilon']$, it attains all values in the interval $[0, f(\epsilon')]$. Specifically, if $f$ is restricted to its original domain (the ray), it still attains all values in $[0, f(\epsilon')]$. So the infimum of $S$ is at least $f(\epsilon')$. Contradiction.

Case 2: $s > 0$ and $s \in S$. Then there exists a sequence $x_n$ of points on the ray such that $f(x_n) \to s$ and $f(x_n)$ is strictly increasing. By claim 3, $||x_n||_2$ is strictly increasing. Let the inverse images of the $x_n$ be $y_n$. By claim 2, $|y_n(i)|$ is increasing for $1 \leq i \leq d$. Since $|y_n(i)| \leq ||y_n||_2 < s$, $|y_n(i)|$ is bounded from above, so it converges. As $\sigma$ is strictly monotonic and $\sigma(0) = 0$, $\sigma$ either preserves the sign or reverses it for all inputs. So since the $x_n(i)$ all have the same sign, so do the $y_n(i)$. So $y_n(i)$ converges. Let this limit be $y_{\text{lim}}(i)$. Since for $1 \leq i \leq d, y_n(i) \to y_{\text{lim}}(i)$, we have $y_n \to y_{\text{lim}}$. Since $\sigma$ is continuous, the forward image of $y_{\text{lim}}$ is the limit of forward images of the $y_n$. Since the forward images of the $y_n$ lie on the ray, so does their limit. Hence, the forward image of $y_{\text{lim}}$ lies on the ray. Call it $x_{\text{lim}}$. Since length is also continuous, $s = \lim_{n \to \inf} ||y_n||_2 = ||y_{\text{lim}}||_2$. So $f(x_{\text{lim}}) = s$ so $s \notin S$. Contradiction.

Case 3: $s > 0$ and $s \notin S$. Then there exists a point $x$ on the ray for which $f(x) = s$. Let its inverse image be $y$. Let $I$ be the set of indices $i$ with $1 \leq i \leq d$ and $x(i) \neq 0$. For $i \in I$, let $\sigma_{\text{max}}(i) = \sigma(2y(i))$. Since $\sigma$ has $\sigma(0) = 0$ and it is strictly monotonic, we have $|\sigma(2y(i))| > |\sigma(y(i))| > 0$ and so $|\sigma_{\text{max}}(i)| > |x(i)| > 0$ and also $\sigma_{\text{max}}(i)$ and $x(i)$ have the same sign. Let $C = \min_{i \in I} \sigma_{\text{max}}(i)$. Take some vector $y'$ that can vary. Since $\sigma$ is continuous, as $y'(i)$ varies between $y(i)$ and $2y(i)$, it attains all values between $x(i)$ and $\sigma_{\text{max}}(i)$. So for all $1 \leq c \leq C$, we can set $y'(i)$ to some value $y_c(i)$ such that $\sigma(y_c(i)) = cx(i)$. So the vector $y_c$ that has the aforementioned $y_c(i)$ components for $i \in I$ and has zero components for $i \notin I$ is the inverse image of $cx$. So $f$ is defined on $cx$ for $1 \leq c \leq C$. Let $s' := f(Cx)$. By claim 3, $f$ is strictly increasing so $s' > s$. By claim 4, $f$ is continuous. So between $x$ and $Cx$, $f$ takes all values between and including $s$ and $s'$. Also, by the original definition of $s$, $f$ attains all positive real values less than $s$. So $f$ attains all positive real values less than $s'$. But $s$ was defined to be the infimum of positive real values that $f$ does not attain. Contradiction.

Claim 6: The inverse image of the ray intersects the hypersphere in a single point.
By claim 5, there is a point on the ray that has an inverse image of length $r$. By claim 3, there is exactly one such point. Therefore, the inverse image of the ray contains exactly one point of length $r$, so it intersects the hypersphere in exactly one point, as required.

The proposition also applies if each neuron in the nonlinearity layer uses a different nonlinearity $\sigma_i$ as long as it fulfills the stated conditions.

9.5.7 Proposition

We say a random matrix $A$ is ‘orthosymmetric’ if for any vector $v$, $\frac{Av}{||Av||_2}$ is uniformly distributed on the hypersphere.

**Proposition** For a linear MLP with orthosymmetric weight matrices that are independent of each other and no trainable bias parameters, we have $Q_{\theta}^{-1}GSC(k, l) = 1$ for all $k$ and $l$.

**Proof.** Denote the weight matrix of the $l$’th layer by $\theta_l$ and define $\theta_l^k = \theta_l \theta_{l+1} \ldots \theta_k$. Because the last step in computing $f_k$ is multiplication with an orthosymmetric random matrix, $\frac{f_k}{||f_k||_2}$ is uniformly distributed on the hypersphere. Hence we can write $f_k$ as a product between a random vector $u$ that is uniformly distributed on the hypersphere and a scalar random variable $\ell(u)$. Further, by assumption, $u$ and $\ell(u)$ are independent of $\theta_l^k$. Then we have

$$Q_{\theta}^{-1}GSC(k, l) = Q_{\theta}^{-1} \frac{||J_l^k||_{qm}||f_k||_2}{||f_l||_2}$$

$$= (Q_{\theta} \frac{||f_l||_2}{||J_l^k||_{qm}||f_k||_2})^{-1}$$

$$= (Q_{\theta} \frac{||\theta_l^k\ell(u)||_2}{||\theta_l^k||_{qm}||\ell(u)||_2})^{-1}$$

$$= (Q_{\theta} \frac{||\theta_l^k||_{qm}}{||\theta_l^k||_{qm}})^{-1}$$

$$= (Q_{\theta} \frac{Q_{\theta}^{-1}||u||_2}{||\theta_l^k||_{qm}})^{-1}$$

$$= (Q_{\theta} \frac{Q_{\theta}^{-1}||u||_2}{||\theta_l^k||_{qm}})^{-1}$$

$$= 1$$

The transformation of $Q_{\theta}||\theta_l^k||_{qm}$ to $||\theta_l^k||_{qm}$ is analogous to the transformation from $Q_{\theta}||J_l^k||_{qm}$ to $||J_l^k||_{qm}$ in the proof of proposition.

The weight matrices are orthosymmetric if they are either Gaussian initialized or orthogonally initialized. Hence, both popular initialization schemes are covered by the proposition.

9.5.8 Proposition

We say a random matrix $A$ is ‘orthosymmetric’ if for any vector $v$, $\frac{Av}{||Av||_2}$ is uniformly distributed on the hypersphere.

We say a nonlinearity $\sigma$ is ‘input-bound’ if either $\sigma(a) = c_{pos}a$ or $\sigma(a) = c_{neg}a$, where the decision about which of the two cases applies depends only on the data input $X$ of the network and not on the parameter $\theta$. 

The weight matrices are orthosymmetric if they are either Gaussian initialized or orthogonally initialized. Hence, both popular initialization schemes are covered by the proposition.
Proposition 9. For any MLP composed of layers which are themselves composed of (i) an input-bound nonlinearity layer and (ii) multiplication with an orthosymmetric random matrix where these matrices are independent, \( \mathbb{P}_\theta \text{GSC}(k, l) = 1 \) for all \( k \) and \( l \).

Proof. For each input \( X \), each layer is equivalent to the product of an orthosymmetric matrix and a diagonal matrix. Such a product is itself orthosymmetric. Therefore proposition 7 applies.

An input-bound nonlinearity is an idealized version of a self-similar nonlinearity. In this idealized version, the dependence on the parameter \( \theta \) of whether each nonlinearity neuron multiplies its pre-activation by \( c_{\text{pos}} \) or \( c_{\text{neg}} \) is obscured. Of course, in practice, whether \( c_{\text{pos}} \) or \( c_{\text{neg}} \) is multiplied depends on the parameter because the pre-activation depends on the parameter.

The assumption that the nonlinearity is input-bound is strong. Nevertheless, we believe the weight matrices and the activations patterns may be “independent enough” in practice that this proposition is still illustrative of why MLPs with self-similar nonlinearities do not exhibit exploding gradients in our experiments.

The weight matrices are orthosymmetric if they are either Gaussian initialized or orthogonally initialized. Hence, both popular initialization schemes are covered by the proposition.

9.5.9 Proposition 9

We say a random matrix \( A \) is ‘orthosymmetric’ if for any vector \( v \), \( \frac{Av}{||Av||_2} \) is uniformly distributed on the hypersphere.

Proposition 9. Consider an MLP composed of layers which are endomorphisms on the \( d \)-dimensional hypersphere composed of (i) a ReLU nonlinearity, (ii) multiplication with a full-rank, orthosymmetric matrix and (iii) length-only layer normalization. Assume the weight matrices are independent of each other. For any vector \( u \) on the hypersphere, the probability that there exists a vector \( X \) on the hypersphere such that \( f_l(X) = u \) is \( 2^{d(l-L-1)} \).

Proof. We will prove this by induction from the lowest to the highest layer. Take some vector \( u \). We will start by looking at its inverse image under \( f_L(X) \). The inverse image of \( u \) under length-only layer normalization is a ray from the origin not including the origin. The inverse image of that under a full-rank, orthosymmetric linear transformation is ray from the origin not including the origin with a uniformly distributed orientation. Regarding the inverse image of that ray under the ReLU layer, there are three cases. If the ray does not lie in the positive \( d \)-dimensional quadrant, its inverse image is empty. The probability of this case is \( 1 - 2^{-d} \). If it lies in the positive \( d \)-dimensional quadrant but not in its boundary, its inverse image is itself. The probability of this case is \( 2^{-d} \). Finally, if it lies in the boundary of the positive \( d \)-dimensional quadrant, its inverse image is a multi-dimensional subspace. The probability of this case is 0. Only the second case has positive probability and an inverse image which intersects the hypersphere. So the total probability of the inverse image of the ray intersecting the hypersphere, which is the domain of \( X \), is \( 2^{-d} \), which is \( 2^{d/(L-1)} \), as required. Also note that if the inverse image of the ray intersects the hypersphere, it does so in a single point.

Let’s look at the induction step. By the same argument as before, the probability that a vector \( u \) has an inverse image under \( f_{l-1}(f_l) \) is \( 2^{-d} \) and if it has an inverse image, almost surely it’s a single point. Let that point be \( u' \). By the induction hypothesis, the probability that \( u' \) has an inverse image under \( f_l(X) \) is \( 2^{d(l-1-L)} \). Further, note that the event that \( u \) has an inverse image under \( f_{l-1}(f_l) \) and the event that \( u' \) has an inverse image under \( f_l(X) \) are independent, because they depend on different weight matrices. Therefore, the probability that \( u \) has an inverse image under \( f_{l-1}(X) \) is \( 2^{-d-2d(l-1-L)} = 2^{d((l-1-1)-L)} \), as required.

If the probability that any given \( u \) has an inverse image is \( 2^{d(L-1)} \) as \( \theta \) varies, then the joint probability that \( u \) has an inverse image as \( \theta \) varies and \( u \) is uniformly distributed on the hypersphere is also \( 2^{d(L-1)} \). For any given \( \theta \), the probability that a uniformly random \( u \) has an inverse image can be interpreted as the volume of the domain. Hence, in expectation over \( \theta \), that volume is \( 2^{d(L-1)} \).
The weight matrices are twice orthosymmetric if they are either Gaussian initialized or orthogonally initialized. Hence, both popular initialization schemes are covered by the proposition.

9.5.10 Proposition

We say a random function \( f_b \) is ‘\( k \)-diluted in expectation’ with respect to random vector \( v \) if there exists a random matrix \( S_b \) and a random function \( \rho_b \) such that \( f_b(v) = S_b v + \rho_b(v) \) and \( \mathbb{Q} \| S_b v \|_2 = k \).

We say a random function \( \rho(v) \) is ‘scale-symmetric decomposable’ (SSD) if it can be written as \( u_\rho \ell_\rho \left( \frac{1}{\|v\|_2} \right) \|v\|_2 \), where \( \ell_\rho \) is a random scalar function and \( u_\rho \) is uniformly distributed on the hypersphere and independent of both \( \ell_\rho \) and \( v \).

We say a random matrix \( S \) is ‘scale-symmetric decomposable’ (SSD) if \( S v \), when viewed as a function of the vector \( v \) is SSD.

Proposition. Let \( u \) be a uniformly distributed unit length vector. Given random functions \( f_b \), \( 1 \leq b \leq B \), which are \( k_b \)-diluted in expectation with respect to \( u \), a matrix \( S_b \), that is either SSD or a multiple of the identity and an SSD random function \( \rho_b := f_b - S_b \) where all the \( S_b \) and \( \rho_b \) are independent, \( f_1 \circ f_2 \circ \ldots \circ f_B \) is \( \left( \prod_b (1 + \frac{1}{k_b}) \right)^{-\frac{1}{2}} \)-diluted in expectation with respect to \( u \).

Proof. Let \( U \) be the uniform distribution over unit length vectors and \( u \sim U \). We will proceed by induction over \( B \), where the induction hypothesis includes the following claims.

1. \( f_1 \circ f_2 \circ \ldots \circ f_B \) is \( \left( \prod_b (1 + \frac{1}{k_b}) \right) - 1 \)^{-\frac{1}{2}}-diluted in expectation.

2. \( \mathbb{Q} \| f_1 \circ f_2 \circ \ldots \circ f_B(u) \|_2 = \sqrt{\mathbb{Q} \| S_1 S_2 \ldots S_B u \|_2^2} + (\mathbb{Q} \| f_1 \circ f_2 \circ \ldots \circ f_B(u) - S_1 S_2 \ldots S_B u \|_2^2} \)

3. \( S_1 S_2 \ldots S_B u \), \( f_1 \circ f_2 \circ \ldots \circ f_B(u) \) and \( f_1 \circ f_2 \circ \ldots \circ f_B(u) - S_1 S_2 \ldots S_B u \) are all radially symmetric

Let’s start by looking at the case \( B = 1 \). Claim (1) follows directly from the conditions of the proposition. We have:

\[
\mathbb{Q} \| f_1(u) \|_2 = \mathbb{Q} \| S_1 u + \rho_1(u) \|_2 = \mathbb{Q} \sqrt{\mathbb{E} (S_1 u + \rho_1(u)) \cdot (S_1 u + \rho_1(u))} = \sqrt{\mathbb{E} (S_1 u + \rho_1(u)) \cdot (S_1 u + \rho_1(u))}
\]

\[
= \sqrt{\mathbb{E} \rho_1(u) \rho_1(u)} = \sqrt{\mathbb{E} \rho_1(u) \rho_1(u)}
\]

\[
= \sqrt{\mathbb{E} \rho_1(u) \rho_1(u) + 2 \mathbb{E} (\rho_1(u)) \rho_1(u) + 2 \mathbb{E} (\rho_1(u)) \rho_1(u)}
\]

\[
= \sqrt{\mathbb{E} \rho_1(u) \rho_1(u) + 2 \mathbb{E} (\rho_1(u)) \rho_1(u)} + 2 \mathbb{E} (\rho_1(u)) \rho_1(u)
\]

\[
= \sqrt{\mathbb{E} \rho_1(u) \rho_1(u) + 2 \mathbb{E} (\rho_1(u)) \rho_1(u)}
\]

\[
= \sqrt{\mathbb{E} (\rho_1(u) \rho_1(u))} = \sqrt{\mathbb{E} \rho_1(u) \rho_1(u)}
\]

This is claim (2). For any \( u \), \( \rho_1(u) \) is radially symmetric because \( \rho_1 \) is SSD. If \( S_1 \) is SSD, \( S_1 u \) is also radially symmetric for arbitrary \( u \). If \( S_1 \) is a multiple of the identity, \( S_1 u \) is radially symmetric because \( u \) is radially symmetric. In either case, \( S_1 u \) is radially symmetric. Because the orientation of \( \rho_1(u) \) is governed only by \( u_\rho \), which is independent of both \( u \) and \( S_1 \), the orientations of \( S_1 u \) and
\( \rho_1(u) \) are independent. But the sum of two radially symmetric random variables with independent orientations is itself radially symmetric, so \( f_1(u) \) is also radially symmetric. This yields claim (3).

Now for the induction step. Set \( B \) to some value and also define \( k_o := \left( \prod_{k=2}^{B} \left( 1 + \frac{1}{k^2} \right) \right)^{-\frac{1}{2}}. \)
\( S_o := S_2S_3\ldots S_B, f_o := f_2 \circ \ldots \circ f_B \) and \( \rho_o := f_o - S_o. \) Then the induction hypothesis yields that \( f_o \) is \( k_o \)-diluted in expectation, that \( Q||f_o(u)||_2 = \sqrt{Q||S_o u||_2 + Q||\rho_o(u)||_2} \) and that \( f_o(u), \rho_o(u) \) and \( S_o u \) are radially symmetric. This implies that \( \frac{f_o(u)}{||f_o(u)||_2}, \frac{\rho_o(u)}{||\rho_o(u)||_2} \) and \( \frac{S_o u}{||S_o u||_2} \) are uniformly distributed unit length vectors. Define \( c_o := Q||S_o u||_2. \) Then \( k_o \)-dilution in expectation implies \( Q||\rho_o(u)||_2 = \frac{c_o}{k_o} \) and hence \( Q||f_o(u)||_2 = \sqrt{1 + \frac{c_o}{k_o}}. \) Similarly, let \( c_1 := Q||S_1 u||_2 \) and then \( Q||\rho_1(u)||_2 = \frac{c_1}{k_1} \) and analogously to the \( B = 1 \) case we have \( Q||f_1(u)||_2 = \sqrt{(Q||S_1 u||_2)^2 + Q||f_1(u) - S_1 u||_2^2} = \sqrt{c_1^2 + (\frac{c_1}{k_1})^2} = 1 + \frac{c_1}{k_1}. \) We have

\[
\begin{align*}
Q||\rho_1(f_o(u))||_2 &= \sqrt{\mathbb{E}_{\rho_1(f_o(u)).\rho_1(f_o(u))} } \\
&= \sqrt{\mathbb{E}_{u\rho_1, \ell_1, f_o(u)} \frac{f_o(u)}{||f_o(u)||_2} \cdot \frac{f_o(u)}{||f_o(u)||_2} \cdot \frac{f_o(u)}{||f_o(u)||_2} \\
&= \sqrt{\mathbb{E}_{||f_o(u)||_2, \frac{f_o(u)}{||f_o(u)||_2}} \cdot \frac{f_o(u)}{||f_o(u)||_2} \cdot \frac{f_o(u)}{||f_o(u)||_2} \\
&= \sqrt{(1 + \frac{1}{k_1^2})c_1^2 \mathbb{E}_{||f_o(u)||_2} \cdot \frac{f_o(u)}{||f_o(u)||_2} \\
&= \sqrt{1 + \frac{1}{k_1^2} \frac{c_1}{k_1}}
\end{align*}
\]

If \( S_1 \) is SSD, analogously, we have \( Q||S_1 S_o u||_2 = c_o c_1, \) \( Q||S_1 \rho_0(u)||_2 = \frac{c_o}{k_o} c_1 \) and \( Q||S_1 f_o(u)||_2 = \sqrt{1 + \frac{c_1}{k_1}} c_1 \). If \( S_1 \) is a multiple of the identity, by the definition of \( c_1 \), it is \( c_1 \) times the identity. Therefore \( Q||S_1 S_o u||_2 = Q||c_1 S_o u||_2 = c_o c_1, \) \( Q||S_1 \rho_0(u)||_2 = Q||c_1 \rho_0(u)||_2 = \frac{c_1}{k_1} \) and \( Q||S_1 f_o(u)||_2 = Q||\rho_1(u)||_2 = \sqrt{1 + \frac{c_1}{k_1}} c_1 \) also hold. Then we have

\[
\begin{align*}
Q||f_1(f_o(u)) - S_1 S_o u||_2 \\
Q||S_1(f_o(u)) + \rho_1(f_o(u)) - S_1 S_o u||_2 \\
Q||S_1(S_o u + \rho_o(u)) + \rho_1(f_o(u)) - S_1 S_o u||_2 \\
Q||S_1 \rho_o(u) + \rho_1(f_o(u))||_2 \\
&= \sqrt{\mathbb{E}(S_1 \rho_o(u)) \cdot (S_1 \rho_o(u)) + \mathbb{E}(S_1 \rho_o(u)) \cdot \rho_1(f_o(u)) + \mathbb{E}(f_o(u)) \cdot \rho_1(f_o(u))} \\
&= \sqrt{\mathbb{E}(\ldots + 2 \mathbb{E}(S_1 \rho_o(u)) \cdot \frac{f_o(u)}{||f_o(u)||_2}) \cdot ||f_o(u)||_2 + \mathbb{E}(\ldots)} \\
&= \sqrt{\mathbb{E}(\ldots + 2 \mathbb{E}(f_o(u), \rho_o(u), S_o, \ell_1) \cdot (S_1 \rho_o(u)) \cdot (\ell_1 \cdot \frac{f_o(u)}{||f_o(u)||_2}) \cdot ||f_o(u)||_2 \cdot \mathbb{E}_{u\rho_1, \ell_1} \cdot \mathbb{E}(\ldots))}
\end{align*}
\]
\[ \begin{align*}
&= \sqrt{\mathbb{E}(S_1 \rho_0(u)).(S_1 \rho_0(u)) + 0 + \mathbb{E}\rho_1(f_0(u)).\rho_1(f_0(u))} \\
&= \sqrt{(Q||S_1 \rho_0(u)||^2 + (Q||\rho_1(f_0(u)||^2)^2} \\
&= \sqrt{\left(c_o c_1 \frac{1}{k_o^2} + \left(1 + \frac{1}{k_o^2}\right) \frac{1}{k_1^2}\right)^2} \\
&= c_o c_1 \sqrt{-1 + \left(1 + \frac{1}{k_0^2}\right) \left(1 + \frac{1}{k_1^2}\right)}
\end{align*} \]

And analogously we have

\[ \begin{align*}
&\frac{Q||f_1(f_0(u))||^2}{Q||f_1(f_0(u)) - S_1 S_o u||^2} = \frac{c_o c_1 - \frac{1}{k_0^2} \sqrt{-1 + \left(1 + \frac{1}{k_0^2}\right) \left(1 + \frac{1}{k_1^2}\right)}}{c_o c_1 \sqrt{-1 + \left(1 + \frac{1}{k_0^2}\right) \left(1 + \frac{1}{k_1^2}\right)}} = (-1 + \left(1 + \frac{1}{k_0^2}\right) \left(1 + \frac{1}{k_1^2}\right))^{-\frac{1}{2}}.
\end{align*} \]

Substituting back \(k_o, S_o\), and \(f_o\), we obtain claim (1). Claim (2), when substituting in \(k_o, S_o\) and \(f_o\) becomes \(Q||f_1(f_0(u))||^2 = \sqrt{(Q||S_1 S_o u||^2 + (Q||f_1(f_0(u)) - S_1 S_o u||^2)^2}\). Substituting the identities we obtained results in \(c_o c_1 \sqrt{(1 + \frac{1}{k_0^2}) \left(1 + \frac{1}{k_1^2}\right)} = \sqrt{(c_o c_1)^2 + \left(c_o c_1 \sqrt{-1 + \left(1 + \frac{1}{k_0^2}\right) \left(1 + \frac{1}{k_1^2}\right)}\right)^2}\), which is true, so we have claim (2).

Consider \(S_1 S_2..S_B u = S_1 S_o u\). We know \(S_o u\) is radially symmetric by the induction hypothesis, so if \(S_1\) is a multiple of the identity, so is \(S_1 S_o u\). If \(S_1\) is SSD, then \(S_1 S_o u\) is radially symmetric for any value of \(S_o u\). In either case, \(S_1 S_o u\) is radially symmetric.

Consider \(f_1 f_2..f_B u = f_1 f_o u\). We know \(f_o u\) is radially symmetric by the induction hypothesis. We also have \(f_1 f_o u = S_1 f_o u + \rho_1(f_o u)\). We just showed \(S_1 f_o u\) is radially symmetric. Because \(\rho_1\) is SSD, \(\rho_1(f_o u)\) is radially symmetric with an orientation independent of that of \(S_1 f_o u\) because it is governed only by \(u_{\rho_1}\). The sum of two radially symmetric random variables with independent orientation is itself radially symmetric, so \(f_1 f_o u\) is radially symmetric.

Finally, consider \(f_1 f_2..f_B u - S_1 S_2..S_B u = S_1 \rho_0(u) + \rho_1(f_o(u))\). We know \(\rho_0(u)\) is radially symmetric by the induction hypothesis so as before, \(S_1 \rho_0(u)\) is radially symmetric. And again, \(\rho_1(f_o(u))\) is radially symmetric with independent orientation, so the sum \(f_1 f_2..f_B u - S_1 S_2..S_B u\) is radially symmetric.

So we also have claim (3). This completes the proof.

\(\square\)

A Gaussian initialized matrix and an orthogonally initialized matrix are both SSD. Therefore the condition that the \(S_3\) are either the identity or SSD is fulfilled in the ResNets we study in this paper in their randomly initialized state.
Unfortunately, those ResNets do not quite fulfill the SSD condition on $\rho_b$, but they come close. The last operation of each $\rho_b$ used in the ResNets we study is multiplication with an SSD matrix, because the last layer in each residual block is a linear layer. This means the orientation of $\rho_b$ is indeed governed by an independent, uniform unit length vector as required. However, because the first operation of each $\rho_b$ is normalization, the length of the incoming vector $||v||_2$ is erased by $\rho_b$ instead of multiplied to the final result as in an SSD function. However, because the inputs to each $\rho_b$ are random and high-dimensional, we do not expect their lengths to vary much, especially if the original inputs to the network have themselves been normalized. So the loss of the information of the length of the incoming vector should not cause the overall behavior to change significantly.

It is possible to construct a ResNet architecture that fulfills all conditions of this proposition. All we need to do is (i) use length-only layer normalization as the normalization layer and (ii) before the result computed by the residual block is added onto the skip connection, we must multiply to it the constants that were divided out by L0ayer. The architectures we study in this paper seem “close enough” to this “perfect” architectures that we do not expect fundamentally different behavior.

Finally, note that this proposition applies only in expectation over randomly initialized matrices. As long as those matrices are high-dimensional, we expect it to apply approximately to specific realizations of those matrices as well.

9.6 Theorems and Proofs

9.6.1 Theorem 1: Exploding Gradients Limit Depth

See section 9.4 for the formal definition of effective depth and related concepts.

Consider some MLP $f$ with nominal depth $L$ and layers $f_l$, $1 \leq l \leq L$. Let its compositional depth be $N$ and its linear layers be $f_l$, $1 \leq n \leq N$ where $l_1 < l_2 < \ldots < l_N$. Let each linear layer be the sum of an unparametrized initial function $i_{l_n}$ and a parametrized residual function $r_{l_n}(\theta_{l_n})$. $i_{l_n}$ represents multiplication with the initial weight matrix and is used interchangeably to denote that initial weight matrix. $r_{l_n}(\theta_{l_n})$ represents multiplication with the residual weight matrix and is used interchangeably with that residual weight matrix. The parameter sub-vector $\theta_{l_n}$ contains the entries of the residual weight matrix.

Let an $N$-trace $\phi_N$ be a subset of $\{1, \ldots, N\}$. Let $\Phi_N$ be the set of all possible $N$-traces and let $\Phi_N^\lambda$ be the set of all $N$-traces of size $\lambda$ or more. We define the ‘gradient term’ $G(\phi_N, f, \theta, X, Y)$ := $J_0J_1\ldots J_{N+1}$ if layer $k$ is not a linear layer, $J_k = r_{l_n}(\theta_{l_n})$ if layer $k$ corresponds to linear layer $l_n$ and $n \in \phi_N$, and $J_k$ if layer $k$ corresponds to linear layer $l_n$ and $n \not\in \phi_N$.

Let $res_N^\lambda(f, \theta, X, Y) := \sum_{\lambda=1}^{\min(N-1, \lambda)} \sum_{\phi_N-1 \in \Phi_N-1} G(\phi_N-1, f, \theta, X, Y)$.

**Theorem 1** Consider an MLP $f$ as defined above with all parameter sub-vectors initialized to $\theta_{l_n}^{(0)} = 0$. Take some set of possible query points $D$. Let each of the parameter sub-vectors be updated with a sequence of updates $\Delta \theta_{l_n}^{(t)}$ such that $\theta_{l_n}^{(t)} = \theta_{l_n}^{(t-1)} + \Delta \theta_{l_n}^{(t)}$ with $1 \leq t \leq T$. Let $alg$ be a fixed function and $\Lambda$ an integer. Further assume:

1. $\Delta \theta_{l_n}^{(t)} = alg\left(\frac{df(\theta^{(t-1)}, X^{(t)}, Y^{(t)})}{d\theta_{l_n}^{(t-1)}}\right)$ for some $(X^{(t)}, Y^{(t)}) \in D$

2. there exist constants $r$ and $c$ such that $\frac{||\Delta \theta_{l_n}^{(t)}||_2}{||\theta_{l_n}^{(t)}||_2} \leq \frac{1}{c\sqrt{n}}$ for all $n$ and $t$

3. there exists a constant $c' \geq 1$ such that

$$\frac{||alg\left(\frac{df(\theta^{(t-1)}, X^{(t)}, Y^{(t)})}{d\theta_{l_n}^{(t-1)}}\right) - alg\left(\frac{df(\theta^{(t-1)}, X^{(t)}, Y^{(t)})}{d\theta_{l_n}^{(t-1)}} - res_N^\lambda(f, \theta^{(t-1)}, X^{(t)}, Y^{(t)})\right)||_2}{||alg\left(\frac{df(\theta^{(t-1)}, X^{(t)}, Y^{(t)})}{d\theta_{l_n}^{(t-1)}}\right)||_2} \leq c' \sum_{\lambda=1}^{\min(N-1, \lambda)} \sum_{n \in \phi_N-1} \prod_{\phi_N-1 \in \Phi_N-1} \frac{||r_{l_n}(\theta_{l_n}^{(t)})||_2}{||\theta_{l_n}^{(t)}||_2}$$

43
Proof. For all $T' \leq T$, we have $||\theta_i^{(t')}||_2 = ||\sum_{t=1}^{T'} \Delta \theta_i^{(t)}||_2 \leq \sum_{t=1}^{T'} ||\Delta \theta_i^{(t)}||_2 \leq \frac{T'}{cr^n}||i_n||_2$. Then we have

$$1 \leq \frac{1}{||i_n||_2} \sum_{t=1}^{T} ||\text{alg}(\frac{df(\theta^{(t-1)}, X(t), Y(t))}{d\theta^{(t-1)}_i}) - \text{alg}(\frac{df(\theta^{(t-1)}, X(t), Y(t))}{d\theta^{(t-1)}_i}) - res^\lambda(f, \theta^{(t-1)}, X(t), Y(t))||_2 \leq h$$

Then for all $N$ we have

$$\frac{1}{||i_n||_2} \sum_{t=1}^{T} ||\text{alg}(\frac{df(\theta^{(t-1)}, X(t), Y(t))}{d\theta^{(t-1)}_i}) - \text{alg}(\frac{df(\theta^{(t-1)}, X(t), Y(t))}{d\theta^{(t-1)}_i}) - res^\lambda(f, \theta^{(t-1)}, X(t), Y(t))||_2 \leq h$$

Let $K(\lambda, n)$ be the number of ways to choose $\lambda$ distinct positive integers such that their sum is $n$. Clearly, $K(\lambda, n) = 0$ for $n < \frac{\lambda(\lambda+1)}{2}$. For $n \geq \frac{\lambda(\lambda+1)}{2}$, the largest number that can be chosen is $n - \frac{\lambda(\lambda+1)}{2} + \lambda$ and so $K(\lambda, n) \leq (n - \frac{\lambda(\lambda+1)}{2} + \lambda)^\lambda = (n - \frac{\lambda(\lambda-1)}{2})^\lambda$. So we have

$$\sum_{\lambda=0}^{\infty} \frac{(\frac{T}{c})^\lambda}{r^n} \sum_{\phi_n \in \Phi_n} \prod_{\phi \in \Phi_n} \frac{1}{r^n}$$
Under review as a conference paper at ICLR 2018

\[
< \frac{Tc'}{cF^2N} \sum_{\lambda=1}^{\infty} \left( \frac{T}{c} \right)^{\lambda} r^{-\frac{\lambda(\lambda-1)}{2}} \lambda^\lambda (\ln r)^{-\lambda}
\]

\[
\leq c' \sum_{\lambda=1}^{\infty} \left( \frac{T}{c} \right)^{\lambda+1} r^{-\frac{\lambda(\lambda-1)}{2}} \lambda^\lambda (\ln r)^{-\lambda}
\]

\[
\leq c' \sum_{\lambda=1}^{\infty} \left( \frac{ch}{2} \left( \frac{\ln r}{r} \right)^3 \frac{r}{c} \right)^{\lambda+1} r^{-\frac{\lambda(\lambda-1)}{2}} \lambda^\lambda (\ln r)^{-\lambda}
\]

\[
< \sum_{\lambda=1}^{\infty} h \frac{\lambda+1}{\lambda} \left( \frac{1}{2} \right)^{\lambda+1} (\frac{1}{16})^{\lambda+1} (\ln r)^{2\lambda+3} r^{-\frac{\lambda^2}{4} \lambda^\lambda}
\]

\[
< \sum_{\lambda=1}^{\infty} h \frac{\lambda+1}{\lambda} \left( \frac{1}{2} \right)^{\lambda+1}
\]

\[
< h
\]

We can interpret this theorem by viewing alg as representing the gradient-based algorithm and the quantity that is ultimately bounded by \( h \) as an approximation of the relative \( \lambda \)-contribution at layer \( l_N \) until time \( T \).

Let’s first look at this approximation. It is the sum the lengths of the \( \lambda \)-contributions at each update. On the one hand, this is a conservative approximation as we essentially assume that all these \( \lambda \)-contribution have the same orientation. On the other hand, we ignore indirect effects that \( \lambda \)-contributions in early updates have on the trajectory of the parameter value and hence on \( \lambda \)-contributions of later updates. Since we are ultimately interested in effective depth, we are ok with our estimate of the length of the \( \lambda \)-contribution to only be accurate if the \( \lambda \)-contribution is small. In that case, these indirect higher-order effects should be negligible.

Now, we analyze the four conditions in turn.

Condition (1) states that the algorithm computes the update. For convenience, we write the algorithm as a deterministic function of the gradient of the layer for which the update is computed. We can easily extend the proof to algorithms that use the gradients of other layers, past gradients and as well randomness if we adjust condition (3) appropriately. Also for convenience, we assume a batch size of 1. We can apply to proof to larger batch sizes, for example, by having alg use past gradients and setting the majority of updates to zero.

Condition (2) reflects the argument from section 4.3 that the area around the current parameter value in which the gradient is reflective of the function is bounded by a hypersphere of relative radius \( \frac{1}{\sqrt{\|l_n\|_2}} \). Note that we divide the size of the update by the weight matrix in the initialized state \( \epsilon \). This reflects the general observation that the largest useful update size decreases in practice when training a deep network. Therefore, we can bound all updates by the largest useful size of the first update.

The strongest condition is condition (3). It can be understood as making two distinct assertions.

Firstly, ignoring the alg() function, it bounds the length of the sum of the \( \Lambda \)-residual terms. In essence, it requires that on average, the size of these terms is “what one would expect” given the \( \ell_2 \) norm of the initial and residual weight matrices up to some constant \( c' \). In other words, we assume that in \( \lambda \)-residual terms, the large singular values of layer-wise Jacobians do not compound disproportionately compared to other terms. This assertion is also conservative in the sense that we implicitly assume that all \( \lambda \)-residual terms have the same orientation.

Secondly, it asserts that alg() is “relatively Lipschitz” over the gradient. This is fulfilled e.g. for SGD and SGD with custom layer-wise step sizes as used in our experiments. It is fulfilled by SGD with momentum as long as \( \epsilon \) is sufficiently small and the momentum term is bounded below. In theory, it is not fulfilled by RMSprop or Adam as gradients on individual weight matrix entries can be “scaled up” arbitrarily. In practice, the \( \epsilon \) regularization term used in the denominator prevents this, although this is rarely necessary.
Finally, condition (4) states that the training time is limited, from which we then derive that the λ-contribution is limited, which then implies effective depth is limited. Importantly, the bound on \( T \) is exponential in \( \Lambda \) and independent of both \( L \) and \( N \). Note that we did not attempt to make the bound tight. As it stands, unfortunately, the bound has no practical value and would indicate that networks can be trained to far greater depth than is possible in practice.

9.6.2 Theorem 2: Why gradients explode

**Theorem 2** Consider a neural network \( f \) with random parameter \( \theta \) composed of layer functions \( f_i \) that are surjective, continuously differentiable endomorphisms on the \( d \)-dimensional hypersphere. Let \( \mathcal{J}_l := \mathcal{J}_{l+1} \). Let \( S \) be the hypersphere and let \( \mathcal{S} \) be the uniform distribution on it. Let \( F_l, 1 \leq l \leq L, \) be random vectors independent of \( \theta \) where \( F_l \sim \mathcal{S} \). Assume:

1. The \( \theta_l \) are independent of each other.
2. Each Jacobian \( \mathcal{J}_l(\theta_l, F_{l+1}) \) has \( d - 1 \) nonzero singular values which, as \( F_{l+1} \) and \( \theta_l \) vary, are independent and drawn from some distribution \( P_l \).
3. There exist some \( \epsilon > 0 \) and \( \delta > 0 \) such that \( \mathbb{P}_{F_l} \left( \|p_l\| - \mathbb{E}(\|p_l\|) > \epsilon \right) > \delta \) for all \( l \).
4. \( f_l(\theta_l, F_{l+1}) \) is a uniformly distributed vector on the hypersphere.
5. For any unit length vector \( u, \mathcal{J}_l(\theta_l, F_{l+1})u \) can be written as \( \ell_l(\theta_l, F_{l+1}, u)u_l \). Here, \( \ell_l \) is random scalar independent of both \( u_l \) and \( f_l(\theta_l, F_{l+1}) \). \( u_l \) conditioned on \( f_l(\theta_l, F_{l+1}) \) is uniformly distributed in the space of unit length vectors orthogonal to \( f_l(\theta_l, F_{l+1}) \).

Let \( X \sim S \) be a random vector independent of \( \theta \). Then setting \( r(\delta, \epsilon) := \sqrt{1 + \delta^2} \) we have

\[
\mathbb{Q}_{\theta, X} \text{GSC}(k, l, f, \theta, X) \geq \sqrt{\frac{d - 1}{d}} r(\delta, \epsilon)^{k-l}.
\]

**Proof.** The hypersphere is a \( d \)-1-dimensional subspace in \( \mathbb{R}^d \). Hence, any endomorphism \( f_l \) on that subspace will have Jacobians with at least one zero singular value with right eigenvector equal to the normal of the subspace at the input and the left eigenvector equal to the normal of the subspace at the output. Since the subspace is the unit hypersphere, the normal at the input is the input and the normal at the output is the output. By assumption, the Jacobian has no other zero singular values. Let the singular values of the Jacobian be \( s_1, \ldots, s_{d-1}, s_d \). WLOG we set \( s_d = 0 \).

Throughout this proof, we use \( \text{det} (\mathcal{J}_l) \) to denote the product of singular values of the Jacobian excluding the zero singular value, i.e. \( \text{det} (\mathcal{J}_l) = \prod_{i=1}^{d-1} s_i \). We use \( \text{||\mathcal{J}_l||}_{qm} \) to denote the quadratic mean of the singular values excluding the zero singular value, i.e. \( \text{||\mathcal{J}_l||}_{qm} = \sqrt{\frac{\sum_{i=1}^{d-1} s_i^2}{d-1}} \).

As \( f_l \) is surjective, for fixed \( \theta_l \), we have by integration by substitution

\[
\int_S \left| \text{det} (\mathcal{J}_l(\theta_l, F_{l+1})) \right| dF_{l+1} \leq \int_S 1 \leq \int_S \left| \text{det} (\mathcal{J}_l(\theta_l, F_{l+1})) \right| dF_{l+1}.
\]

But we also have

\[
\int_S 1 = \mathbb{E}_{F_{l+1}} \left| \text{det} (\mathcal{J}_l(\theta_l, F_{l+1})) \right| = \mathbb{E}_{F_{l+1}} \left| \text{det} (\mathcal{J}_l(\theta_l, F_{l+1})) \right| \text{ and so } \mathbb{E}_{F_{l+1}} \left| \text{det} (\mathcal{J}_l(\theta_l, F_{l+1})) \right| \geq 1 \text{ and so } \mathbb{E}_{F_{l+1}, \theta_l} \left| \prod_{i=1}^{d-1} s_i \right| = \mathbb{E}_{F_{l+1}, \theta_l} \prod_{i=1}^{d-1} |s_i| \geq 1.
\]

But the nonzero singular values are assumed to be independent by condition (2). So

\[
\mathbb{E}_{F_{l+1}, \theta_l} \left| f_{l+1} \prod_{i=1}^{d-1} |s_i| \right| = (\mathbb{E}_{F_{l+1}, \theta_l} |s_i|)^{d-1} \text{ for } 1 \leq i \leq d - 1. \text{ So } (\mathbb{E}_{F_{l+1}, \theta_l} |s_i|)^{d-1} \geq 1 \text{ and so } \mathbb{E}_{F_{l+1}, \theta_l} |s_i| \geq 1 \text{ for } 1 \leq i \leq d - 1.
\]

Similarly, we have

\[
\mathbb{Q}_{F_{l+1}, \theta_l} \left| \mathcal{J}_l(\theta_l, F_{l+1}) \right|_{qm} \geq \mathbb{Q}_{F_{l+1}, \theta_l} \left( \sqrt{\frac{\sum_{i=1}^{d-1} s_i^2}{d-1}} \right) = \frac{1}{\sqrt{\mathbb{E}_{F_{l+1}, \theta_l} \sum_{i=1}^{d-1} s_i^2}}.
\]
of all layer functions is the hypersphere, we have 

\[ Q_f \text{ in the space of unit length vectors orthogonal to } \theta. \]

Under review as a conference paper at ICLR 2018

The last identity uses condition (3). Let \( \theta^l_k := (\theta_1, \theta_{l+1}, \ldots, \theta_{k-1}) \).

Now, we will prove the following claim by induction:

(A) \( f_l(\theta^l_k, X) \) is a uniformly distributed vector on the hypersphere independent of \( \theta^l_1 \).

Since \( X \sim S \), by condition (4), we have \( f_l(\theta^l_k, X) \sim S \). Also, since \( X \) is independent of \( \theta^l_1 \) and \( \theta_L \) is independent of \( \theta^l_1, f_L(\theta^l_k, X) \) is independent of \( \theta^l_1 \), as required.

Now for the induction step. Assume (A) is true for some \( l \). Then by the induction hypothesis, \( f_l(\theta^l_k, X) \sim S \) and \( f_l(\theta^l_k, X) \) is independent of \( \theta^l_1 \) and thus independent of \( \theta^l_{l-1} \). Then by condition (4), \( f_{l-1}(\theta^l_{l-1}, X) = f_{l-1}(\theta_{l-1}, f_l(\theta^l_k, X)) \sim S \). Also, since both \( \theta_{l-1} \) and \( f_l(\theta^l_k, X) \) are independent of \( \theta^l_1 \), so is \( f_{l-1}(\theta^l_{l-1}, X) \). This completes the induction step.

Analogously, we have claim (A2): \( f_l(\theta^l_k, F_k) \) is a uniformly distributed vector on the hypersphere independent of \( \theta^l_1 \).

Now we will prove the following claim by induction on \( k - l \).

(B) For any unit length vector \( u \), \( J^l_k(\theta^l_k, F_k) u \) can be written as \( \ell_k^l(\theta^l_k, F_k, u) u \). Here, \( \ell_k^l \) is a random scalar independent of both \( u \) and \( f_l(\theta^l_k, F_k) \), \( u \), conditioned on \( f_l(\theta^l_k, F_k) \), is uniformly distributed in the space of unit length vectors orthogonal to \( f_l(\theta^l_k, F_k) \).

The case \( k = l + 1 \) is equivalent to condition (5). For the induction step, consider

\[
J^l_k(\theta^l_k, F_k) u = J^l_{l+1}(\theta_l, f_{l+1}(\theta^l_{k+1}, F_k)) J^l_{l+1}(\theta^l_{k+1}, F_k) u \\
= J^l_{l+1}(\theta_l, f_{l+1}(\theta^l_{k+1}, F_k)) \ell^l_{l+1}(\theta^l_{k+1}, F_k, u) u_{l+1} \\
= \ell^l_{l+1}(\theta^l_{k+1}, F_k, u) J^l_{l+1}(\theta_l, f_{l+1}(\theta^l_{k+1}, F_k)) u_{l+1} \\
= \ell^l_{l+1}(\theta^l_{k+1}, F_k, u) \ell^l_{l+1}(\theta_l, f_{l+1}, u_{l+1}) u_l
\]

Here, we use claim (A2) and the induction hypothesis twice. Let \( \ell^l_k(\theta^l_k, F_k, u) := \ell^l_{l+1}(\theta^l_{k+1}, F_k, u) \ell^l_{l+1}(\theta_l, f_{l+1}, u_{l+1}) \). Then \( J^l_k(\theta^l_k, F_k) u \) has the required decomposition. From when we used claim (B) the first time, we obtained that \( \ell^l_{l+1} \) is independent of \( f_{l+1} \) and \( u_{l+1} \), and therefore of \( f_l(\theta_l, f_{l+1}) \) and \( J^l_{l+1}(\theta_l, f_{l+1}) \), and therefore of \( u_l \). From when we used claim (B) the second time, we obtain that \( \ell^l_{l+1} \) is independent of \( u_l \) and \( f_l \). Therefore, both \( \ell^l_{l+1} \) and \( \ell^l_{l+1} \) are independent of both \( u_l \) and \( f_l \), and then so is \( \ell^l_k \). From using claim (B) the second time we also obtained that conditioned on \( f_l \), \( u_l \) is a uniformly distributed among unit vectors orthogonal to \( f_l \).

Therefore, the decomposition \( J^l_k(\theta^l_k, F_k) u = \ell^l_k u_l \) fulfills the required conditions, so the induction step is complete.

Now we will prove the main claim by induction on \( k - l \).

We will begin with the case \( k = l + 1 \). We have \( Q_{\theta, X} GSC(l + 1, l, f, \theta, X) = Q_{\theta, X} \sqrt{\frac{d-1}{d-1} ||J^l_{l+1}||_{q_m} ||f_{l+1}||_{2}} \). (Note that the \( q_m \) norm was redefined.) Because the domain of all layer functions is the hypersphere, we have \( ||f_{l+1}||_2 = ||f_l||_2 \) and so \( Q_{\theta, X} GSC(l + 1, l, f, \theta, X) = \sqrt{\frac{d-1}{d} Q_{\theta, X} ||J^l_{l+1}||_{q_m}} \). But by claim (A), \( f_{l+1}(\theta^l_{l+1}, X) \sim S \) and \( f_{l+1}(\theta^l_{l+1}, X) \) is independent of \( \theta_l \). So \( Q_{\theta, X} ||J^l_{l+1}||_{q_m} \geq \sqrt{1 + \delta c^2} \), so \( Q_{\theta, X} GSC(l + 1, l, f, \theta, X) \geq 1 + \frac{\sqrt{d-1}}{d} \delta c^2 \).

47
Now for the induction step. Let $U$ be the uniform distribution over the unit hypersphere. As before, we have $Q_{\theta,X}GSC(k, l, f, \theta, X) = \sqrt{\frac{d-1}{d} Q_{\theta,X} \| \mathcal{J}_k^l \|_{q_m}}$ and so

$$Q_{\theta,X}GSC(k, l, f, \theta, X) = \sqrt{\frac{d-1}{d} Q_{\theta,X} \| \mathcal{J}_{l+1}^k \|_{q_m}}$$

$$= \sqrt{\frac{d-1}{d} Q_{\theta,X,u \sim U \sim \mathcal{S}} \| \mathcal{J}_{l+1}^k \|_{2}}$$

$$= \sqrt{\frac{d-1}{d} Q_{\theta,X,u \sim U \sim \mathcal{S}} \| \mathcal{J}_{l+1}^k(\theta, f_{l+1}(\theta_{k+1}^l, X)) \|_{2}}$$

Then, $f_k(\theta_k^l, X)$ is uniform among unit vectors orthogonal to $f_{l+1}$. Let's look at the first term $Q_{\theta,X}GSC(k, l, f, \theta, X)$. We have

$$Q_{\theta,X}GSC(k, l, f, \theta, X) = \sqrt{\frac{d-1}{d} Q_{\theta,X,u \sim U \sim \mathcal{S}} \| \mathcal{J}_{l+1}^k \|_{2}}$$

$$= \sqrt{\frac{d-1}{d} Q_{\theta,X,u \sim U \sim \mathcal{S}} \| \mathcal{J}_{l+1}^k \|_{2}}$$

$$= \sqrt{\frac{d-1}{d} Q_{\theta,X,u \sim U \sim \mathcal{S}} \| \mathcal{J}_{l+1}^k(\theta, f_{l+1}(\theta_{k+1}^l, X)) \|_{2}}$$

Let's look at the first term $Q_{\theta,X}GSC(k, l, f, \theta, X)$. We have

$$Q_{\theta,X}GSC(k, l, f, \theta, X) = \sqrt{\frac{d-1}{d} Q_{\theta,X,u \sim U \sim \mathcal{S}} \| \mathcal{J}_{l+1}^k \|_{2}}$$

$$= \sqrt{\frac{d-1}{d} Q_{\theta,X,u \sim U \sim \mathcal{S}} \| \mathcal{J}_{l+1}^k \|_{2}}$$

$$= \sqrt{\frac{d-1}{d} Q_{\theta,X,u \sim U \sim \mathcal{S}} \| \mathcal{J}_{l+1}^k \|_{2}}$$

$$= \sqrt{\frac{d-1}{d} Q_{\theta,X,u \sim U \sim \mathcal{S}} \| \mathcal{J}_{l+1}^k \|_{2}}$$

The last line comes from the induction hypothesis.

Now, let's look at the second term $Q_{\theta,X}GSC(k, l, f, \theta, X)$. We have

$$Q_{\theta,X}GSC(k, l, f, \theta, X) = \sqrt{\frac{d-1}{d} Q_{\theta,X,u \sim U \sim \mathcal{S}} \| \mathcal{J}_{l+1}^k \|_{2}}$$

$$= \sqrt{\frac{d-1}{d} Q_{\theta,X,u \sim U \sim \mathcal{S}} \| \mathcal{J}_{l+1}^k \|_{2}}$$

$$= \sqrt{\frac{d-1}{d} Q_{\theta,X,u \sim U \sim \mathcal{S}} \| \mathcal{J}_{l+1}^k \|_{2}}$$

Putting those results together, we obtain

$$Q_{\theta,X}GSC(k, l, f, \theta, X) = \sqrt{\frac{d-1}{d} Q_{\theta,X,u \sim U \sim \mathcal{S}} \| \mathcal{J}_{l+1}^k \|_{2}}$$

$$= \sqrt{\frac{d-1}{d} Q_{\theta,X,u \sim U \sim \mathcal{S}} \| \mathcal{J}_{l+1}^k \|_{2}}$$

$$= \sqrt{\frac{d-1}{d} Q_{\theta,X,u \sim U \sim \mathcal{S}} \| \mathcal{J}_{l+1}^k \|_{2}}$$

This is the desired claim.
Let’s look at the conditions.

Continuous differentiability of the layer functions was assumed for convenience. The proof can be extended to almost sure continuous differentiability, which is the practical scenario.

Condition (1) is standard for randomly initialized weight matrices.

Conditions (4) and (5) are both fulfilled for either Gaussian initialized matrices or orthogonally initialized matrices, if the last two operations of each layer function are multiplication with a weight matrix and length-only layer normalization.

If the weight matrix is orthogonally initialized, this is easy to see, because the linear transformation and the normalization operation commute. If we exchange those two operations then the last operation applied to \(F\) and \(u\) is the orthogonal transformation, which decouples both their orientations from the length of \(u\) as well as decoupling one orientation from the other up to preserving their angle. Finally, note that the Jacobian always maps \(u\) to a vector in the left-null space of the Jacobian. But that space is orthogonal to the image of \(F\), and hence the two are orthogonal as required.

If the weight matrix is Gaussian initialized, note that the product of a Gaussian initialized matrix and an orthogonally initialized matrix is Gaussian initialized. Hence, we can insert an additional orthogonally initialized matrix and then proceed with the previous argument to show that conditions (4) and (5) are fulfilled.

After applying a linear transformation with one of the two initializations, conditions (4) and (5) hold except for the length of \(f_i\). Hence, even if length-only layer normalization is not used, we expect (4) and (5) to hold approximately in practice.

As far as we can tell, conditions (2) and (3) are not fulfilled in practice. They are both used to derive from unit determinants a greater than unit \(qm\) norm. As long this implications holds for practical layer functions, (2) and (3) are not necessary.

### 9.6.3 Theorem 3: Skip Connections reduce the gradient

**Theorem 3.** Let \(g\) and \(u\) be random vectors. Consider a function \(f\) that is \(k\)-dilated with respect to \(u\), a matrix \(S\) and a differentiable function \(\rho\). Let \(R(v)\) be the Jacobian of \(\rho\) at input \(v\). Let \(r := \frac{Q}{gR(u)}\). Assume that \(E(Su).\rho(u) = 0\) and that \(E(gR(u)).(gS) = 0\). Also assume that \(\frac{Q||gR(u) + gS||_2||u||_2}{Q||Su||_2||g||_2} = 1\). Then \(\frac{Q||gR(u) + gS||_2||u||_2}{Q||\rho(u) + Su||_2||g||_2} = 1 + \sqrt{\frac{r-1}{k+1}} + O((r-1)^2)\).

**Proof.** We have

\[
\frac{Q||gR(u) + gS||_2||u||_2}{Q||\rho(u) + Su||_2||g||_2} = \frac{Q\sqrt{||gR(u) + gS||_2||u||_2}}{Q\sqrt{||\rho(u) + Su||_2||g||_2}} = \frac{\sqrt{E||gR(u) + gS||_2||u||_2}}{\sqrt{E||\rho(u) + Su||_2||g||_2}} = \frac{\sqrt{E(gR(u)).(gR(u)) + 2(gS).gR(u) + (gS).gS)}|u||_2}{\sqrt{E(\rho(u)).\rho(u) + 2(Su).\rho(u) + (Su).Su)}|g||_2} = \frac{\sqrt{E(gR(u)).(gR(u)) + (gS).gS)}|u||_2}{\sqrt{E(\rho(u)).\rho(u) + (Su).Su)}|g||_2} = \frac{(Q||gR(u)||_2)^2 + (Q||gS||_2)^2||u||_2}{(Q||\rho(u)||_2)^2 + (Q||Su||_2)^2||g||_2}
\]
be viewed as the growth of the GSC. Similarly, 

\[ Q = \text{the other assumption is} \]

However, we also do not see a cause for such a correlation to specifically exist. 

\[ S = a \text{ statement cannot be made because the gradient has a complex dependence both on} \]

the initialization is absorbed into the expectation. Unfortunately, for the backward direction, such 

linear layer of the residual block is Gaussian / orthogonally initialized and if the randomness of 

direction, this is true if either the skip connection is Gaussian / orthogonally initialized or the last 

residual block and that the same is true for the gradient flowing through them. For the forward 

g as the incoming gradient. 

This theorem can be interpreted by viewing \( u \) as the incoming activation of residual block \( b \) and 
g as the incoming gradient. \( r \) represents a type of expectation over the ratio \( \frac{GSC(b+1,0)}{GSC(b,0)} = 

\frac{\|\sum_{k=1}^{2}q_{k}||u||_{2}||g||_{2}}{\|\sum_{k=1}^{2}q_{k}||u||_{2}||g||_{2}} = \frac{\|\sum_{k=1}^{2}q_{k}||u||_{2}||g||_{2}}{\|\sum_{k=1}^{2}q_{k}||u||_{2}||g||_{2}} \), \( gS \) represents the growth of the GSC after the skip connection has been added. 

The key assumptions are \( \mathbb{E}(Su) \cdot (\rho(u)) = 0 \) and \( \mathbb{E}(gR(u)) \cdot (gS) = 0 \). In plain language, we assume 

that the function computed by the skip connection is uncorrelated to the function computed by the 

residual block and that the same is true for the gradient flowing through them. For the forward 

direction, this is true if either the skip connection is Gaussian / orthogonally initialized or the last 

linear layer of the residual block is Gaussian / orthogonally initialized and if the randomness of 

the initialization is absorbed into the expectation. Unfortunately, for the backward direction, such 

a statement cannot be made because the gradient has a complex dependence both on \( S \) and \( R \). 

However, we also do not see a cause for such a correlation to specifically exist. 

The other assumption is \( \frac{\|q_{0}||gR(u) + gS||_{2}||u||_{2}}{\|q_{0}||u||_{2}||g||_{2}} = 1 \). This is true if \( S \) is an orthogonal matrix and so specifically 

if \( S \) is the identity matrix. If \( S \) is Gaussian / orthogonally initialized, this is true if the randomness 

of the initialization is absorbed into the \( q \) terms. Even if \( S \) is a “reasonably” well-conditioned 

fixed matrix, as long the orientations of \( u \) and \( g \) are “sufficiently random”, we expect this assumption 

to hold at least approximately. 

An implicit assumption made is that the distribution of the incoming gradient \( g \) is unaffected by the 

addition of the skip connection, which is of course not quite true in practice. The addition of 

the skip connection also has an indirect effect on the distribution and scale of the gradient as it flows 

further towards the input layer.

\[ r = \sqrt{\frac{\|\sum_{k=1}^{2}q_{k}||u||_{2}||g||_{2}}{\|\sum_{k=1}^{2}q_{k}||u||_{2}||g||_{2}}} = \frac{\|\sum_{k=1}^{2}q_{k}||u||_{2}||g||_{2}}{\|\sum_{k=1}^{2}q_{k}||u||_{2}||g||_{2}} \]

This is true if \( S \) is an orthogonal matrix and so specifically if \( S \) is the identity matrix. If \( S \) is Gaussian / orthogonally initialized, this is true if the randomness 
of the initialization is absorbed into the \( q \) terms. Even if \( S \) is a “reasonably” well-conditioned 
fixed matrix, as long the orientations of \( u \) and \( g \) are “sufficiently random”, we expect this assumption 
to hold at least approximately. 

An implicit assumption made is that the distribution of the incoming gradient \( g \) is unaffected by the 
addition of the skip connection, which is of course not quite true in practice. The addition of 
the skip connection also has an indirect effect on the distribution and scale of the gradient as it flows 
further towards the input layer.
We state the theorem for differentiable \( \rho \), though almost sure differentiability is of course sufficient in practice.

The experiments in figure 3 bear out the theory discussed here.

### 9.7 Taylor approximation of a neural network

We define the first-order Taylor approximation \( T_l \) of the bottom layers up to layer \( l \) recursively. Write \( i_l(x) \) as the short form of \( i_t(i_{t+1}(\ldots i_L(x)\ldots)) \). Then

\[
T_L(\theta, x) = f_L(\theta_L, x)
\]

\[
T_l(\theta, x) = i_t(T_{t+1}(\theta, x)) + r_t(\theta_t, i_t(1)) + \sum_{k=l+1}^{L} \frac{d r_t(\theta_t, i_t+1(x))}{d i_k(x)} r_k(\theta_k, i_{k+1}(x)) \text{ for } l < L
\]

The maximum number of parametrized residual functions composed in \( T_l \) is 2. Otherwise, only addition and composition with fixed functions is used. Hence, the compositional depth of \( T_l \) is 2. Hence, the network \( f_{\text{Taylor}(l)} := f_0(y, f_1(\ldots f_{l-1}(T_l(x))\ldots)) \) has compositional depth \( l + 1 \).

### 9.8 Looks-linear initialization

Looks-linear initialization (‘LLI’) of ReLU MLPs achieve an orthogonal initial state. Consider a ReLU MLP with some number of linear layers and a ReLU layer between each pair of linear layers. LLI initializes the weight matrix of the lowest linear layer differently from the weight matrix of the highest linear layer and differently from the weight matrices of the intermediate linear layers. Let a weight matrix \( W \) have dimension \( m \times n \), where \( n \) is the dimension of the incoming vector and \( m \) is the dimension of the linear layer itself. Also, we require that the dimension of all ReLU layers, and thus the dimension of all linear layers except the highest linear layer, is even. Then the weight matrices are initialized as follows:

- **Lowest linear layer:** Draw a uniformly random orthogonal matrix \( W' \) of dimension \( \max(\frac{m}{2}, n) \times \max(\frac{m}{2}, n) \). Then, for all \( 1 \leq i \leq \frac{m}{2} \) and \( 1 \leq j \leq n \), set \( W(2i, j) = \max(\sqrt{n}, 1)W'(i, j) \) and \( W(2i + 1, j) = -\max(\sqrt{n}, 1)W'(i, j) \).

- **Highest linear layer:** Draw a uniformly random orthogonal matrix \( W' \) of dimension \( \max(m, \frac{n}{2}) \times \max(m, \frac{n}{2}) \). Then, for all \( 1 \leq i \leq m \) and \( 1 \leq j \leq \frac{n}{2} \), set \( W(i, 2j) = \max(\sqrt{m}, 1)W'(i, j) \) and \( W(i, 2j + 1) = -\max(\sqrt{m}, 1)W'(i, j) \).

- **Intermediate linear layers:** Draw a uniformly random orthogonal matrix \( W' \) of dimension \( \max(\frac{m}{2}, \frac{n}{2}) \times \max(\frac{m}{2}, \frac{n}{2}) \). Then, for all \( 1 \leq i \leq \frac{m}{2} \) and \( 1 \leq j \leq \frac{n}{2} \), set \( W(2i, 2j) = \max(\sqrt{m}, 1)W'(i, j) \), \( W(2i + 1, 2j) = -\max(\sqrt{m}, 1)W'(i, j) \), \( W(2i, 2j + 1) = -\max(\sqrt{m}, 1)W'(i, j) \) and \( W(2i + 1, 2j + 1) = \max(\sqrt{m}, 1)W'(i, j) \).

Under LLI, pairs of neighboring ReLU neurons are grouped together to effectively compute the identity function. The incoming signal is split between ReLU neurons of even and odd indeces. Each of the two groups preserves half the signal, which are then ‘stitched together’ in the next linear layer only to be re-divided in a different way to pass through the next ReLU layer.

It is not immediately clear that LLI achieves an orthogonal initial state. To see this, insert after each ReLU layer \( f_l \) two additional unparametrized layers \( f'_l \) and \( f''_l \). The first of these layers has width \( d_l \) and computes the difference of neighboring pairs of neurons in the ReLU layer, i.e. for \( 1 \leq i \leq \frac{d_l}{2} \), \( f'_l(i) = f_l(2i) - f_l(2i + 1) \). The second of these layers has width \( d_l \) and spreads the values of \( f'_l \) back out across two neurons: \( f''_l(2i) = f'_l(i) \) if \( f'_l(i) > 0 \) and \( f''_l(2i) = 0 \) otherwise for \( 1 \leq i \leq \frac{d_l}{2} \); and \( f''(2i + 1) = -f''(i) \) if \( f'_l(i) < 0 \) and \( f''_l(2i + 1) = 0 \) otherwise for \( 1 \leq i \leq \frac{d_l}{2} \). Adding those two layers after each ReLU layer does not alter the function the network computes, and we find that \( f'_{l-1} \) is an orthogonal transformation of \( f'_l \).
9.9 EXPERIMENTAL DETAILS

9.9.1 ARCHITECTURES USED

**Vanilla networks without skip connections**  All networks are MLPs composed of only fully-connected linear layers and unparametrized layers. The following types of layers are used.

- **linear layer**: $f_t(\theta_t, f_{t+1}) = W_t f_{t+1}$ where the entries of the weight matrix $W_t$ are the entries of the parameter sub-vector $\theta_t$. Trainable bias parameters are not used.
- **ReLU layer**: $f_t(f_{t+1}) = \sigma_{\text{ReLU}}(f_{t+1})$, where the scalar function $\sigma_{\text{ReLU}}$ is applied element-wise as indicated by .() We have $\sigma_{\text{ReLU}}(a) = a$ if $a \geq 0$ and $\sigma_{\text{ReLU}}(a) = 0$ if $a < 0$.
- **tanh layer**: $f_t(f_{t+1}) = \sigma_{\text{tanh}}(f_{t+1})$, where $\sigma_{\text{tanh}}(a) = \tanh(a)$.
- **SeLU layer**: $f_t(f_{t+1}) = \sigma_{\text{SeLU}}(f_{t+1})$. We have $\sigma_{\text{SeLU}}(a) = c_{\text{pos}} a$ if $a \geq 0$ and $\sigma_{\text{SeLU}}(a) = c_{\text{neg}} (e^a - 1)$ if $a < 0$. We set $c_{\text{pos}} = 1.0507$ and $c_{\text{neg}} = 1.0507 \times 1.6733$ as suggested by Klambauer et al. [2017].
- **batch normalization layer**: $f_t(f_{t+1}) = \frac{f_{t+1} - \mu}{\sigma}$, where $\mu$ is the component-wise mean of $f_{t+1}$ over the current batch and $\sigma$ is the componentwise standard deviation of $f_{t+1}$ over the current batch.
- **layer normalization layer**: $f_t(f_{t+1}) = \frac{f_{t+1} - \mu}{\sigma}$, where $\mu$ is mean of the entries of $f_{t+1}$ and $\sigma$ is the standard deviation of the entries of $f_{t+1}$.
- **length-only layer normalization layer**: $f_t(f_{t+1}) = \frac{f_{t+1} - \mu}{\sigma_{\text{qm}}}$, where $\sigma_{\text{qm}}$ is the quadratic mean of the entries of $f_{t+1}$.
- **dot product error layer**: $f_0(f_1, y) = f_1 : y$
- **softmax layer**: $f_t(f_{t+1})(i) = \frac{e^{f_{t+1}(i)}}{\sum_i e^{f_{t+1}(i)}}$
- **kl error layer**: $f_0(f_1, y) = \ln f_1(y)$ where $y$ is an integer class label and $f_1$ has one entry per class.

Note that normalization layers (batch normalization, layer normalization or length-only layer normalization) do not use trainable bias and variance parameters.

A network of compositional depth $N$ contains $N$ linear layers and $N - 1$ nonlinearity layers (ReLU, tanh or SeLU) inserted between those linear layers. If the network uses normalization layers, one normalization layer is inserted after each linear layer. For Gaussian noise experiments, the error layer is the dot product error layer. For CIFAR10 experiments, a softmax layer is inserted above the last linear or normalization layer and the error layer is a kl error layer.

For Gaussian noise experiments, data inputs as well as predictions and labels have dimension 100. We used a compositional depth of 50. We generally used a uniform width of 100 throughout the network. However, we also ran experiments where the width of all layers from the first linear layer to the layer before the last linear layer had width 200. We also ran experiments where linear layers alternated in width between 200 and 100. For CIFAR10 experiments, data inputs have dimension 3072 and predictions have dimension 10. We use a compositional depth of 51. The first linear layer transforms the width to 100 and the last linear layer transformed the width to 10.

The following initialization schemes for the weight matrices are used.

- **Gaussian**: Each entry of the weight matrix is drawn as an independent Gaussian with mean 0. The variance of this Gaussian is one over the dimension of the incoming vector except when the weight matrix follows a ReLU layer. In that case, the variance of the Gaussian is two over the dimension of the incoming vector.
- **Orthogonal**: The weight matrix is a uniformly random orthogonal matrix. Note that this initialization scheme is only used for square matrices.
- **looks-linear**: See section 9.8
ResNet  In all cases, the first layer is a linear layer. After that, there are 25 skip connections. Each
skip connection bypasses a block of 6 layers: a normalization layer, a nonlinearity layer, a linear
layer, another normalization layer, another nonlinearity layer, and another linear layer. Above the
last skip connection, a final normalization layer is inserted, followed by softmax (CIFAR10 only)
and then the error layer. For Gaussian noise experiments, we use a constant width of 100. For
CIFAR10, the first linear layer transforms the width from 3072 to 100, and the last skip connection
as well as the last linear layer in the last residual block transform the width from 100 to 10.

Skip connections are identity skip connections, except the last skip connection in CIFAR10 experi-
ments that is responsible for reducing the width. There, the skip connection multiplies its incoming
value by a fixed $10 \times 100$ submatrix of a $100 \times 100$ orthogonal matrix. For Gaussian noise experi-
ments, we also conducted some experiments where skip connections used random matrices where
each entry is drawn from an independent Gaussian with mean 0 and the variance being one over the
dimension of the incoming vector.

9.9.2 Protocol for Gaussian noise experiments

For Gaussian noise experiments, both inputs and labels are 100-dimensional vectors were each entry
is drawn from an independent Gaussian with mean 0 and variance $\frac{1}{100}$. We normalized the input
vectors to have length 10. We drew 100 independent datasets of size 10,000.

For each dataset and each architecture we studied (see table 1 for the full list), we computed both the
forward activations and the gradient for each datapoint. For architectures with batch normalization,
we also conducted some experiments where skip connections used random matrices where
each entry is drawn from an independent Gaussian with mean 0 and the variance being one over the
dimension of the incoming vector.

9.9.3 Protocol for CIFAR10 Experiments

For CIFAR10 experiments, we preprocessed each feature to have zero mean and unit variance. We
used the training set of 50,000 datapoints and disregarded the test set. We used batches of size 1,000
except for the vanilla batch-ReLU architecture with Gaussian initialization, for which we used a
batch size of 50,000. (See section 4.5 for the explanation.)

We trained each architecture we studied (see table 2 for the full list) with SGD in two ways. First,
with a single step size for all layers. Second, with a custom step size for each layer.

Single step size  We perform a grid search over the following starting step sizes:
{1e5, 3e4, 1e4, 3e3, ..., 1e \(-4), 3e \(-5), 1e \(-5)\). For each of those 21 starting step sizes, we train
the network until the end-of-epoch training classification error has not decreased for 5 consecutive
epochs. Once that point is reached, the step size is divided by 3 and training continues. Once the
end-of-epoch training classification error has again not decreased for 5 epochs, the step size is di-
vided by 3 again. This process is repeated until training terminates. Termination occurs either after
500 epochs or after the step size is divided 11 times, whichever comes first. The starting step size
is 500 epochs or after the step size is divided 11 times, whichever comes first.
that obtains the lowest final training classification error is selected as the representative step size for which results are presented in the paper.

**Custom step sizes** In this scenario, we use a different starting step size for each layer. After those step sizes are computed, smoothed and scaled as described in section 9.9.4, we train the pre-trained network with those step sizes. As before, periodically, we divide all step sizes jointly by 3. As before, training is terminated after 11 divisions or when 500 epochs are reached, whichever comes first.

We compute the following metrics:

- Largest relative update size for each layer induced by the estimated optimal step size during the epoch where that optimal step size was estimated. See section 9.9.4 for details.
- Effective depth throughout training: see section 9.4.2 for details. λ-contributions are accumulated from batch to batch.
- Training classification error at the end of each epoch.
- Training classification error when compositional depth is reduced via Taylor expansion after training: see section 9.7 for details.
- GSC, pre-activation standard deviation and pre-activation sign diversity: for details, see the end of section 9.9.2. Note that the expectations over the dataset were computed by maintaining exponential running averages across batches.
- Operator norms of residual weight matrices after training.

See table 2 and figures 2, 4, 5 and 6 for results.

### 9.9.4 Selecting custom step sizes

We estimated the optimal step size for each linear layer under SGD for our CIFAR10 experiments. This turned out to be more difficult than expected. In the following, we describe the algorithm we used. It has five stages.

**Pre-training** We started by pre-training the network. We selected a set of linear layers in the network that we suspected would require similar step sizes. In exploding architectures (vanilla batch-ReLU with Gaussian initialization, vanilla layer-tanh, vanilla batch-tanh, SeLU), we chose the second highest linear layer through the sixth highest linear layer for pretraining, i.e. 5 linear layers in total. We expected these layers to require a similar step size because they are close the output and the weight matrices have the same dimensionality. For vanilla ReLU, vanilla layer-ReLU, vanilla tanh and looks-linear initialization, we chose the second lowest linear layer through the second highest linear layer (i.e. 49 linear layers in total) because the weight matrices have the same dimensionality. Finally, for ResNet, we chose the second lowest through the third highest linear layer (i.e. 48 linear layers in total), because the blocks those layers are in have the same dimensionality.

We then trained those layers with a step size that did not cause a single relative update size of more than 0.01 (exploding architectures) or 0.001 (other architectures) for any of the pre-trained layers or any batch. We chose small step sizes for pre-training to ensure that pre-training would not impact effective depth. We pre-trained until the training classification error reached 85%, but at least for one epoch and at most for 10 epochs. The exact pre-training step size was chosen via grid search over a grid with multiplicative spacing of 3. The step size chosen was based on which step size reached the 85% threshold the fastest. Ties were broken by which step size achieved the lowest error.

**Selection** In the selection phase, we train each linear layer one after the other for one epoch while freezing the other layers. After each layer is trained, the change to the parameter caused by that epoch of training is undone before the next layer is trained. For each layer, we chose a step size via grid search over a grid with multiplicative spacing 1.5. The step size that achieved the lowest training classification error after the epoch was selected. Only step sizes that did not cause relative update sizes of 0.1 or higher were considered, to prevent weight instability.

Now we can explain the need for pre-training. Without pre-training, the selection phase yields very noisy and seemingly random outcomes for many architectures. This is because it was often best to
use a large step size to jump from one random point in parameter space to the next, hoping to hit a configuration at the end of the epoch where the error was, say, 88%. Since we used a tight spacing of step sizes, for most layers, there was at least one excessively large step size that achieved this spurious “success”. Since we only trained a single layer out of 51 for a single epoch, the error of the “correct” step size after pre-training often did not reach, say, 88%. When we trained the network for 500 epochs with those noisy estimates, we obtained very high end-of-training errors.

Pre-training ensures that training with an excessively high step size causes the error to exceed 85% again. Therefore, those step sizes are punished and step sizes that ultimately lead to a much better end-of-training error are selected.

**Clipping**  Even though pre-training was used, for some architectures, it was still beneficial to add the following restriction: as we consider larger and larger step sizes during grid search, as soon as we find a step size for which the error is at least 0.1% higher than for the current best step size, the search is terminated. Clipping is capable of further eliminating outliers and was used if and only it improved the end-of-training error. It was used for vanilla tanh, ResNet layer-tanh and looks-linear layer-ReLU.

For each linear layer, the largest relative update size induced by the step size obtained for that layer after the clipping phase (or after the selection phase if clipping was not used) during the epoch of training conducted in the selection phase is shown in the top left graphs in figures 2, 4, 5 and 6.

**Smoothing**  In this stage, we built a mini-regression dataset of \((X, Y)\) points as follows. For each \(X\) from 1 to 51, we include the point \((X, Y)\) where \(Y\) is the largest relative update size the step size selected for linear layer \(X\) after clipping induced during the epoch of training in the selection phase. We then fit a line via least-squares regression on that dataset in log scale. For each \(X\), we thus obtain a smoothed value \(Y'\). The ratio \(\frac{Y'}{Y}\) was multiplied to the step size obtained for each layer at the end of the clipping phase.

We added this phase because we found that the end-of-training error could still be significantly improved by reducing noise among the layer-wise step sizes in this way.

**Scaling**  Finally, we jointly scale all layer-wise step sizes with a single constant. That value is chosen as in the selection phase by trying a small constant, training for one epoch, rewinding that epoch, multiplying that constant by 1.5, rinse, repeat. Again, that process was terminated once any layer experiences an update of relative size at least 0.1. This stage is necessary because the size of the update on the entire parameter vector when all layers are trained jointly is \(\approx \sqrt{51}\) times larger than when only single layers are trained as in the selection phase. Hence, a scaling constant less than 1 is usually needed to compensate. Again, some architectures benefited from using clipping, where we terminated the scaling constant search as soon as one exhibited an error more than 0.1% above the current best scaling constant. Vanilla tanh, vanilla layer-tanh, ResNet layer-tanh and looks-linear layer-ReLU used this clipping.

Formally, for each architecture, we trained three networks to completion. One using no clipping, one using only clipping during the scaling phase, and using the clipping phase as well as clipping during the scaling phase. Whichever of these three networks had the lowest end-of-training error was selected for presentation in the paper. To compare, for single step size training, we compared 21 end-of-training error values.