Revisiting Exploding Gradient: A Ghost That Never Leaves

Kai Hu [1]

Abstract

The exploding gradient problem is one of the main barriers to training deep neural networks. It is widely believed that this problem can be solved by techniques like careful weight initialization and normalization layers. However, we find that exploding gradients still exist in deep plain networks with these techniques applied. Our theory shows that the nonlinearity of activation layers are the source of such exploding gradients, and the gradient of a plain network increases exponentially with the number of nonlinear layers. Based on our theory, we are able to mitigate this gradient problem and train deep plain networks without any skip connection or shortcuts. Our 50-layer plain network, SeqNet50, achieves a 77.1% top-1 validation accuracy on ImageNet. To the best of our knowledge, it is the first plain network that can reach the performance of ResNet50. Codes are available.

1. Introduction

For a long period of time, the vanishing and exploding gradient problem was a major barrier for training large networks. Many methods are proposed to solve this problem: gradient clipping (Allen-Zhu et al., 2019), careful weight initialization (Glorot and Bengio, 2010; He et al., 2015) and most importantly, adding normalization layers (Ioffe and Szegedy, 2015) to the network. People generally believe that the vanishing/exploding gradient has been largely addressed, especially with the help of normalization layers.

However, we check the gradients of layer weights in a plain network (a sequential network without any kinds of skip connections/shortcuts), and find something counter-intuitive: the gradients increase exponentially from deep layers (close to network outputs) to shallow layers (close to network inputs) even the network is carefully initialized and uses batch normalization (BatchNorm). This phenomenon does not happen in residual networks (He et al., 2016). Figure 1 (left) simulates the gradient norm of a plain network (PlainNet20) and a residual network (ResNet20). The architectures of two networks are similar, except that the residual network uses a skip connection every 2 layers. Please refer to Section 2 for implementation details. Recall that the back propagation algorithm derives the gradient of a network from deep layers to shallow layers, we need to read the figure from right to left. For both networks, the gradient norm of the deepest layer are close. As the backward signal passes from deep layers to shallow layers, the gradient norm increases exponentially for the plain network but roughly linearly for the residual network.

Existing studies cannot explain this phenomenon: 1) the gradients in the plain network explode regardless of the activation choice, e.g., ReLU (Nair and Hinton, 2010) or SELU (Klambauer et al., 2017); 2) all layers' weights in the network are properly initialized (He et al. (2015) or Glorot and Bengio (2010)); 3) the batch normalization layer is added after every weight layer, thus the variance of forward signals are stable. We argue that **the nonlinearity property of the activation is the cause of the exploding problem when the network is carefully initialized and used normalization layers**.

To understand this, we analyse the signal variance propagation. A common misunderstanding is that the backward variance propagation is similar to the forward variance propagation (He et al., 2015). It might be true for linear operators, but not for nonlinear activations. A simple intuition: for an operator $y = f(x)$, the forward and backward variance propagation are determined by the function f and its derivative f' respectively, which are not very correlated unless f is linear. A stable forward propagation cannot guarantee a stable backward variance propagation.

We define two ratios to describe how an operator changes the the signal variance during forward and backward (Definition 3.1), and derive the relation between the two ratios. Our conclusion is (informally): if an operator scales the variance of forward signals by k times, it will scale the variance of backward signals by k times given the operator is linear, but more than k times when it is nonlinear. Consider a network, with careful initialization or normalization techniques, the variance of network output and the variance of network input can be close, but in the same time, the variance of input derivatives will be larger than the variance of output derivatives. The increasing speed differs in network architectures as in Figure 1 (left). In Section 3, we show that the increasing speed is exponential for plain network

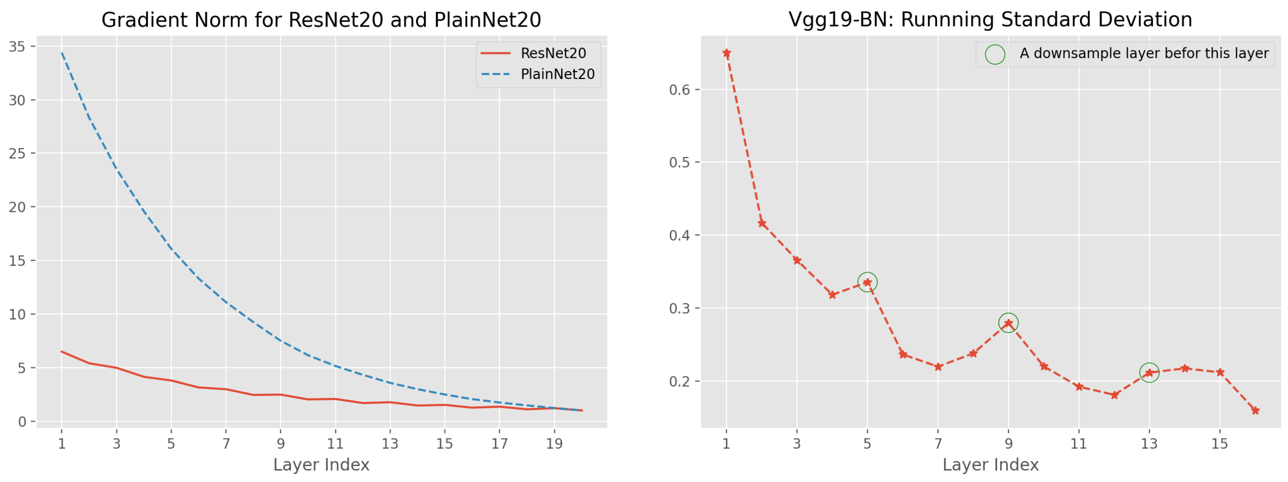


Figure 1: **Left**: The gradient norm of layer weights against layer indexes for a PlainNet20 and a ResNet20 with BatchNorm and standard initialization. **Right**: BatchNorm layers' running variance (square rooted) against layer indexes from VGG19 trained checkpoints. The metrics increase from deep layers to shallow layers.

and roughly linear for residual networks.

With this exploding gradient problem, it is not surprised that deep plain networks are not trainable while residual networks can be very deep. An interesting question would be: if the exploding gradient problem is solved, can deep plain networks match the performance of residual networks? We find that deep plain networks can be trainable by breaking a common practice: the weights of different layers are considered independent and initialized independently, as shown if Figure 2 (left). Our theory shows that by using a tied weight schema as in Figure 2, the increasing speed of backward signal variance can be greatly reduced.

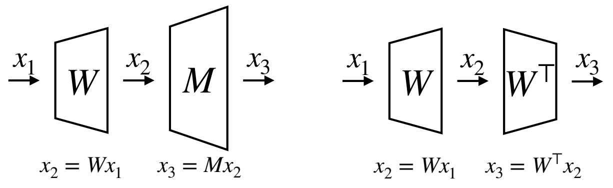


Figure 2: Left: common practice: weights of adjacent layers are initialized independently. Right: basic module of proposed SeqNet. Weights of the second layer are initialized by the transpose of the first layer's weights. The ReLU between two linear layers is not displayed.

In Section 4, we propose the SeqNet, built by stacking the basic module as in Figure 2. SeqNet is a deep plain ReLU network without any kinds of skip connections or shortcuts. We are able to train 50∼100 layers SeqNet with common training settings and match the corresponding ResNet performance on ImageNet classification. To the best of our knowledge, it is the first plain network that achieves ResNet performance on ImageNet in the 100-layer setting. We need to clarify that the purpose of SeqNet is not another strong architecture, but an empirical example for studying the difference between plain and residual networks on non-toy dataset. Investigating SeqNet facilitates the learning theories of very deep neural networks.

Our contributions are threefold: 1) we point out and analyse an exploding gradient problem in plain networks. When the network is well initialised or uses normalization layers, the source of the exploding gradients is the non-linear activations, which is less noticed in previous works. 2) We propose a trainable deep plain network that is initialized at a nonlinear state. Some existing works (Xiao et al., 2018; Qi et al., 2020) also propose such networks, but they largely rely on a linear initialization state. 3) The proposed network is the first to match the validation accuracy of ResNet50 on ImageNet.

2. Evidence of Exploding Gradient

Let $\boldsymbol{x}_i$ be the i-th element of $\boldsymbol{x}$, and $\boldsymbol{W}_{ij}$ be the i-th row, j-th column of $\boldsymbol{W}$. Let Var$[\boldsymbol{x}]$ be the average element variance: $\text{Var}[\boldsymbol{x}] = \dfrac{1}{d}\sum_i \text{Var}[\boldsymbol{x}_i]$. Let $\boldsymbol{x}^{(0)} \in \mathbb{R}^{d_0}$ be the input to neural networks, and $\boldsymbol{x}^{(k)} \in \mathbb{R}^{d_k}$ be the output of k-th layer.

2.1. Simulation evidence

We first go into details for the aforementioned PlainNet20 and ResNet20 networks in Section 1. For PlainNet20, the k-th $(0 < k \leq 20)$ layer is a stack of a fully-connected layer with weights $\boldsymbol{W}^{(k)} \in \mathbb{R}^{d_{k-1}\times d_k}$ initialized by He et al. (2015): $\boldsymbol{W}_{ij}^{(k)} \sim \mathcal{N}(0, 2/d_k)$, a BatchNorm layer $\mathcal{B}^{(k)}$ and a ReLU activation layer $\sigma(x) = \max(x, 0)$:

$$k=0,1,\cdots 19, \;\; \boldsymbol{x}^{(k+1)} = \sigma(\mathcal{B}^{(k)}(\boldsymbol{W}^{(k)}\boldsymbol{x}^{(k)})),.$$

For simplicity, we use the same hidden dimension $d_k = d = 256$ for every k.

The only difference for Residual20 is a skip connection. We follow the Basic Block design (He et al., 2016) and add the skip connection every two layers:

$$\begin{aligned} k=0,2,\cdots 18, \;\; \boldsymbol{x}^{(k+1)} &= \sigma(\mathcal{B}^{(k)}(\boldsymbol{W}^{(k)}\boldsymbol{x}^{(k)})) \\ \boldsymbol{x}^{(k+2)} &= \sigma(\mathcal{B}^{(k)}(\boldsymbol{W}^{(k)}\boldsymbol{x}^{(k+1)}) + \boldsymbol{x}^{(k)}), \end{aligned}$$

We sample a batch of 128 inputs from normal distribution $\boldsymbol{x}_0 \sim \mathcal{N}(0, \mathcal{I}_d)$, and simulate the corresponding derivative on top of the networks with normal distribution $\frac{\partial \ell}{\partial \boldsymbol{x}^{(20)}} \sim \mathcal{N}(0, \frac{1}{d}\mathcal{I}_d)$. We do the back propagation with PyTorch's automatic differentiation to get the gradients of all fully-connected layer weights for two networks. The experiments are repeated 4096 times and we use the average gradient norm to plot the gradient curve. More examples can be found in Appendix A.

2.2. Circumstantial evidence

In the PlainNet, $\boldsymbol{x}^{(k)}$ is activated from the output of a BatchNorm layer, thus should always have a stable variance. The output variance of the fully-connected layer, i.e., $\text{Var}[\boldsymbol{W}^{(k)}\boldsymbol{x}^{(k)}]$ is mainly determined by the magnitude of $\boldsymbol{W}^{(k)}$. With a careful initialization, this term should also be stable. However, if exploding gradient happens, the gradients in shallow layers are much larger than those in deep layers, making larger updates to the weights in shallow layers. The variance term $\text{Var}[\boldsymbol{W}^{(k)}\boldsymbol{x}^{(k)}]$ can be exponential large in the shallow layers of a plain network.

The output variance of the fully-connected layer is recorded by the next BatchNorm layer with the "running variance" parameter, which can be used to verify whether exploding gradients happen in a trained network. We consider the VGG19-BN network (Simonyan and Zisserman, 2015), a classical 19-layer plain convolution network. We check the final model of VGG19 (with BatchNorm) trained on ImageNet classification. The checkpoint is provided by PyTorch[1] (Paszke et al., 2019). Specifically, for a BatchNorm layer with input dimension d, there are d running variances for each input dimension. We use the square root of the average as the metric: $\sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(\text{running\_variance})_i}$.

[1] https://pytorch.org/vision/stable/models.html#torchvision.models.vgg19_bn

Figure 1 (right) shows the metric σ, square rooted average of running variance, in BatchNorm layers from the trained checkpoint (still read the figure from right to left). In the final model, the metric σ increases at most layer indexes except for some layers. The overall trend is that the metric grows exponentially as back propagation goes from deep layers to shallow layers. An exploding gradient explains for this exploding running variance.

3. Theory of Exploding Gradients

Proof strategy. For every operator (e.g., linear/activation layers) in a network, we study how it scales the forward and backward signals. We define two scaling ratios:

Definition 3.1. Let $\boldsymbol{y} = \pi(\boldsymbol{x})$ be an arbitrary operator in the network with input $\boldsymbol{x} \in \mathbb{R}^d$ and output $\boldsymbol{y} \in \mathbb{R}^D$. Let $\frac{\partial \ell}{\partial \boldsymbol{x}}$ and $\frac{\partial \ell}{\partial \boldsymbol{y}}$ be the derivative of the network loss ℓ with respect to the operator input and output: $\frac{\partial \ell}{\partial \boldsymbol{x}} = \pi'(\boldsymbol{x})\frac{\partial \ell}{\partial \boldsymbol{y}}$. Denote the forward scaling ratio $R_f(\pi)$ and backward scaling ratio $R_b(\pi)$ by

$$R_f(\pi) = \frac{\text{Var}[\boldsymbol{y}]}{\text{Var}[\boldsymbol{x}]}, \quad R_b(\pi) = \text{Var}[\frac{\partial \ell}{\partial \boldsymbol{x}}]/\text{Var}[\frac{\partial \ell}{\partial \boldsymbol{y}}],$$

where the variances are computed over the distribution of the input $\boldsymbol{x}$ and the output derivative $\frac{\partial \ell}{\partial \boldsymbol{y}}$.

Remark 3.2. Some existing work on variance propagation computes the variance over the distribution of the inputs **and the network parameters** (He et al., 2015; Zhang et al., 2019; De and Smith, 2020), which we think is not proper. The network we study or verify is always an initialized network with fixed parameters. The randomness of parameters should not be used for computing the expectation and variance. In our setting, the variance is computed with respect to the input distribution and is a function of the fixed parameters, if any. Then the distribution of the parameters can be used to estimate the computed variance.

The two ratios describes how signal variance flows during network forward and backward.

3.1. Scaling ratios of common operators

We first discuss two linear operators: fully-connected layers and BatchNorm layers (large enough batch size). Convolution layers are a special case of fully-connected layers. The results for linear operators may not be new, however we still state them here for completeness.

Assumption 3.3. For the studied operator, dimensions of the input to $\boldsymbol{x}$, or dimensions of the output derivative $\frac{\partial \ell}{\partial \boldsymbol{y}}$, are independent and identically distributed.

Proposition 3.4. *Under Assumption 3.3, for a linear layer $\pi_{\boldsymbol{W}}$ with weights $\boldsymbol{W} \in \mathbb{R}^{D\times d}$: $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$, the forward and backward scaling ratios are $R_f(\pi_{\boldsymbol{W}}) = \frac{\|\boldsymbol{W}\|_F^2}{D}, R_b(\pi_{\boldsymbol{W}}) = \frac{\|\boldsymbol{W}\|_F^2}{d}$, where $\|\boldsymbol{W}\|_F$ denotes the Frobenius norm.*

Remark 3.5. Assumption 3.3 is widely used in many existing studies, but not likely to be true in generally. However, Proposition 3.4 matches the real case quite well, e.g., He initialization. We also verified this in Appendix A.1.

BatchNorm is done for each dimension of the input independently. Thus it is equivalent to analysis one-dimensional data. For a batch of data $\{x_i\}_{i=1}^B$, let $\mu_x = \frac{1}{B}\sum_{i=1}^B x_i, \sigma_x^2 = \frac{1}{B}\sum_{i=1}^B (x_i - \mu_x)^2$ denote the batch mean and batch variance respectively, the output of the BatchNorm is $y_i = \frac{x_i - \mu_x}{\sigma_x}$. Let $\mathrm{Var}[\frac{\partial \ell}{\partial x}]$ and $\mathrm{Var}[\frac{\partial \ell}{\partial y}]$ denotes the sample variance of $\{\frac{\partial \ell}{\partial x_i}\}_{i=1}^B$ and $\{\frac{\partial \ell}{\partial y_i}\}_{i=1}^B$ respectively. We need the following assumption and corollary to further analyse $\mathrm{Var}[\frac{\partial \ell}{\partial x}]$:

Assumption 3.6. For any neuron in the network, the forward signal x and the backward signal $\frac{\partial \ell}{\partial x}$ are independent.

Corollary 3.7. *Under Assumption 3.6, any neuron before the last BatchNorm layer in a network has zero expectation derivative, i.e.,* $\mathbb{E}\frac{\partial \ell}{\partial x} = 0$.

Proposition 3.8. *Under Assumption 3.6, for a BatchNorm layer $\mathcal{B}$, the forward scaling ratio is $R_f(\mathcal{B}) = \frac{1}{\sigma^2}$ where σ^2 is the input batch variance. The backward scaling ratio is $R_b(\mathcal{B}) = \frac{1}{\sigma^2}(1 - \frac{z}{B})$ where B is the batch size and z is a positive random variable that $\mathbb{E}z = 1$, Var$[z] = 2$.*

With a properly large batch size, the forward and backward scaling ratios are approximately the same for BatchNorm layers. Thus the gradient exploding problem is not likely to happen is a plain network that only consists fully-connected operators and BatchNorm operators.

Remark 3.9. Assumption 3.6 is widely used (Yang and Schoenholz, 2017). In fact our proof only need non-correlation. Even if the network inputs and targets are correlated, non-linearity will reduce the correlation greatly between a neuron and its derivative in a deep network.

Nonlinear operators We study element-wise activations that operate on each neuron independently. Similarly, we just consider one dimensional data.

Assumption 3.10. Let g be the element-wise nonlinear activation with input $x \in \mathbb{R}$ and output $y \in \mathbb{R}$: $y = g(x)$. Assume g is absolutely continuous and $\mathrm{Var}[g(x)]$ exists.

For activation g, the forward scaling ratio is $R_f(g) = \mathrm{Var}[g(x)]/\mathrm{Var}[x]$, and the backward scaling ratio is $R_b(g) = \mathrm{Var}[\frac{\partial \ell}{\partial x}]/\mathrm{Var}[\frac{\partial \ell}{\partial y}]$. Recall assumption 3.6 and Corollary 3.7, we have:

$$\mathrm{Var}[\frac{\partial \ell}{\partial x}] = \mathrm{Var}[\frac{\partial \ell}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}x}] = \mathbb{E}\left[(\frac{\partial \ell}{\partial y})^2[g'(x)]^2\right]$$
$$= \mathbb{E}\left[\frac{\partial \ell}{\partial y}^2\right]\mathbb{E}[g'(x)^2] = \mathrm{Var}[\frac{\partial \ell}{\partial y}]\mathbb{E}[g'(x)^2].$$

The backward scaling ratio is $\mathbb{E}[g'(x)^2]$. Both two scaling ratios are related to the input distribution. We discuss with different assumptions on the input distribution.

Assumption 3.11. The input to the activation is Gaussian.

Many existing studies (Klambauer et al., 2017) make this assumption. Chernoff (1981) shows that:

Proposition 3.12. *Under Assumption 3.10 and 3.11, Var$[g(x)]/$Var$[x] \le \mathbb{E}[g'(x)^2]$. The equality holds if and only if $g(x) = ax + b$ for some a and b.*

With proposition 3.12, the backward scaling ratio is larger than the forward scaling ratio for nonlinear activations. Specifically, for ReLU activation $\sigma(x) = \max(x, 0), R_f(\sigma) = \frac{1}{2}(1 - \frac{1}{\pi}), R_g(\sigma) = \frac{1}{2}$. For ReLU activation, we can use weaker assumptions:

Assumption 3.13. The input to the activation layer follows a symmetric distribution centered at zero: $p(x) = p(-x)$ where $p(x)$ denotes the density function of x.

Without loss of generality, we can let $\mathrm{Var}[x] = 1$ for ReLU since $\sigma(x) = s \cdot \sigma(x/s)$ where s denotes the standard deviation of x and $\mathrm{Var}[x/s] = 1$.

Proposition 3.14. *Under Assumption 3.13 and Var$[x] = 1$, Var$[\sigma(x)] \le \mathbb{E}[\sigma'(x)^2] - \frac{1}{4}E[|x|]$.*

Proposition 3.14 describes the state of activation layers in the "linear-BN-ReLU" architecture at initialization precisely. The inputs to ReLU is standardized by the BatchNorm. Recall that the forward and backward scaling ratios are $\mathrm{Var}[\sigma(x)]$ and $\mathbb{E}[\sigma'(x)^2]$ respectively in this case. The backward scaling ratio is strictly larger than the forward scaling ratio in a "linear-BN-ReLU" plain network at initialization. To summarize, under some assumptions, the forward scaling ratio R_f and the backward scaling rate R_b are approximately the same for linear operators. However for the nonlinear activations, $R_b(g) \ge R_f(g) + \epsilon$ for some $\epsilon > 0$. Note that the two ratios are positive and bounded for a single operator in a real network, it is equivalent to say $R_b(g) \ge c \cdot R_f(g)$ for some $c > 1$.

Corollary 3.15. *Suppose a plain network is a stack of linear layers, batch normalization layers and nonlinear activations. Let L denote the number of activation layers in the network and $c = \min_g \frac{R_b(g)}{R_f(g)}$ for every nonlinear activation g in the network. Let $R_f(\pi)$ and $R_b(\pi)$ be the forward and backward scaling ratios of the entire network respectively, $R_b(\pi)/R_f(\pi) = \Omega(c^L)$.*

Careful initialization or normalization methods make the forward signals' variance stable in the plain network, i.e., $R_f(\pi)$ is bounded, However, the backward scaling ratio $R_b(\pi)$ is exponential large, thus the first layer derivative is exponential larger than the output derivative.

Remark 3.16. Although the two ratios are not equal for fully-connected operators as in proposition 3.4, it does not lead to gradient exploding in practice. The ratio between the maximum hidden layer dimension and the minimum hidden layer dimension is bounded and independent with the network depth.

In Figure 1 (right), the running variance decreases every time passing a downsample (max-pooling) layer, which is different than the curve trend. We can have an informal explanation using our framework in Appendix C.1.

3.2. How do ResNets solve the problem

As in Figure 1 (left), the gradient of ResNets grow slowly as backward signals pass from deep layers to shallow layers, without the problem of exploding gradient. In fact, the vanilla ResNets also has the problem of exploding gradient[2] caused in another reason discussed by Zhang et al. (2019). A zero-gamma initialization (Goyal et al., 2017) or a $1/\sqrt{L}$ initialization (Balduzzi et al., 2017) are used solve the problem stated by Zhang et al. (2019). Interestingly, this method also solve the problem stated in this paper.

A ResNet is a stack of multiple residual blocks. We analyse the two scaling ratios of a residual block $\boldsymbol{y} = \boldsymbol{x} + \mathcal{F}(\boldsymbol{x})$, where $\mathcal{F}$ is the residual branch. The residual branch is typically a $2 \sim 3$-layer plain network. The $1/\sqrt{L}$ initialization use $1/\sqrt{L}$ to multiply the residual branch[3] where L is the number of residual blocks: $\boldsymbol{y} = \boldsymbol{x} + \frac{1}{\sqrt{L}}\mathcal{F}(\boldsymbol{x})$.

Proposition 3.17. *Let $R_f(\mathcal{F}), R_b(\mathcal{F}), R_f(\mathcal{R}), R_b(\mathcal{R})$ denote the forward and backward scaling ratios of the* ***residual branch****, the forward and backward scaling ratios of the* ***residual block*** *respectively. We have: $R_f(\mathcal{R}) = 1 + \frac{1}{L}R_f(\mathcal{F}), \quad R_b(\mathcal{R}) = 1 + \frac{1}{L}R_b(\mathcal{F})$.*

[2]See Appendix C.3 for the details.

[3]The $1/\sqrt{L}$ initialization actually uses $1/\sqrt{L}$ to initialize the weight of the last BatchNorm layer in the residual branch. However it is equivalent to write in this form to analyse the variance.

Although $R_b(\mathcal{F})$ and $R_f(\mathcal{F})$ are not equal due to nonlinearity, note that the residual branch is a shallow network with $2 \sim 3$ layers, both two ratios are in the same order and can be considered bounded. When the ResNet is deep, both $R_b(\mathcal{F}) - 1$ and $R_f(\mathcal{F}) - 1$ are $\mathcal{O}(\frac{1}{L})$ order. When considering the entire network, the two ratios are both $\mathcal{O}\left((1 + \frac{1}{L})^L\right)$ order which is independent with the network depth.

4. SeqNet without skip connections

In this section, we try to explore methods to solve the exploding gradient problem and make deep plain networks trainable. The proposed SeqNet may provide intuition for the theoretical explanation of the better performance of ResNets than plain networks.

4.1. Breaking our assumptions

A plain network without exploding gradients sounds contradictory to our theory. However most propositions in Section 3 rely heavily on assumption 3.3, which is a rough approximation. We can design a module where the real distribution is far from this assumption. For example, let $\boldsymbol{x} \in \mathbb{R}^d$ be a vector variable with all independent dimensions. Any projection of $\boldsymbol{x}$ to a much higher dimension $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} \in \mathbb{R}^D$ cannot be dimension-wise independent[4].

Initialized as linear functions A simple situation that assumption 3.3 does not hold is that a neuron and its negative exist at the same time. Specifically, for weight $\boldsymbol{W} \in \mathbb{R}^{D\times d}$ $(D \geq d)$, let $\widetilde{\boldsymbol{W}} = \begin{bmatrix} \boldsymbol{W} \\ -\boldsymbol{W} \end{bmatrix} \in \mathbb{R}^{2D\times d}$. assumption 3.3 cannot hold for $\widetilde{\boldsymbol{y}} = \widetilde{\boldsymbol{W}}\boldsymbol{x}$. Next define $\widetilde{\boldsymbol{M}} = [\boldsymbol{M}, -\boldsymbol{M}] \in \mathbb{R}^{d\times 2D}$ for some matrix $\boldsymbol{M} \in \mathbb{R}^{d\times D}$. We construct a two-layer ReLU network $h(\boldsymbol{x}) = \widetilde{\boldsymbol{M}}\text{ReLU}(\widetilde{\boldsymbol{W}}\boldsymbol{x}) = \boldsymbol{N}\boldsymbol{W}\boldsymbol{x}$, which is a linear function.

The idea of initializing as linear functions comes from Balduzzi et al. (2017) that uses concatenated ReLU (Shang et al., 2016), but we propose it from a different motivation. However, this method (also some existing work (Qi et al., 2020) to be discussed in Section 6) may give us a wrong feeling that a linear (or close linear) initialization state is the key for making deep plain network trainable. We show it is not necessary with a nonlinear design.

Initialized with weight tying Let $\boldsymbol{W}, \boldsymbol{M} \in \mathbb{R}^{D\times d}$ $(D \geq d)$ are initialized from $\mathcal{N}(0, 1/D)$. We study the different forward and backward behaviours of two single-hidden-

[4]Breaking our assumptions is a necessary but insufficient condition to make deep plain network trainable

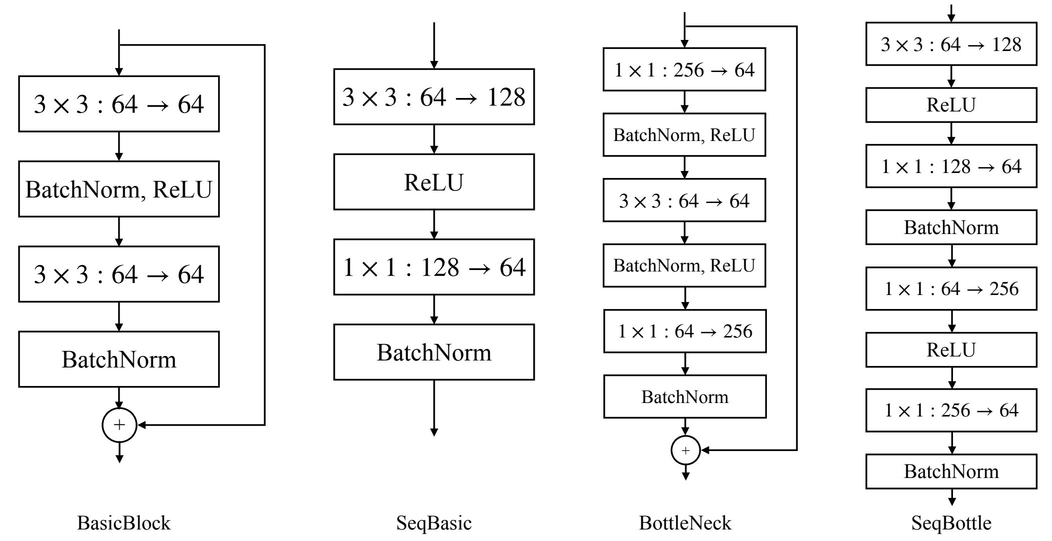


Figure 3: Examples for the ResNet building blocks and the SeqNet counterpart. $k \times k : d \to D$ indicates a convolution layer with input dimension d, output dimension D, kernel size k-by-k.

layer networks $h(\boldsymbol{x}) = \boldsymbol{M}^\top \text{ReLU}(\boldsymbol{W}\boldsymbol{x})$ and $g(\boldsymbol{x}) = \boldsymbol{W}^\top \text{ReLU}(\boldsymbol{W}\boldsymbol{x})$.

Proposition 4.1. *(informal) Assume the network inputs and the derivatives on top of the two networks are i.i.d. Gaussian:* $\boldsymbol{x} \sim \mathcal{N}(0, \mathcal{I}_d)$ *and* $\frac{\partial \ell_h}{\partial h}, \frac{\partial \ell_g}{\partial g} \sim \mathcal{N}(0, \mathcal{I}_d)$. *Let* $k = D/d$, *we have:* $R_f(h) \approx \frac{2}{k}(1 - \frac{1}{\pi}) \quad R_b(h) \approx \frac{2}{k}; \quad R_f(g) \approx 1 + \frac{2}{k}(1 - \frac{1}{\pi}) \quad R_b(g) \approx 1 + \frac{2}{k}$.

Stacking the network h to get a deep plain network (even with BatchNorm), the gradient will increase exponentially with a rate of $R_b(h)/R_f(h) = \frac{\pi}{\pi - 1}$. However, the gradient exploding rate would be much smaller for the network g if $k = D/d \geq 2$. It is possible that a deep plain network of stacking the network g is trainable. However it is not guaranteed and needs to be verified by experiments. Our analysis focuses on the variance propagation of initialized network, and cannot describe or guarantee the network after training, which is the major limitation of our work.

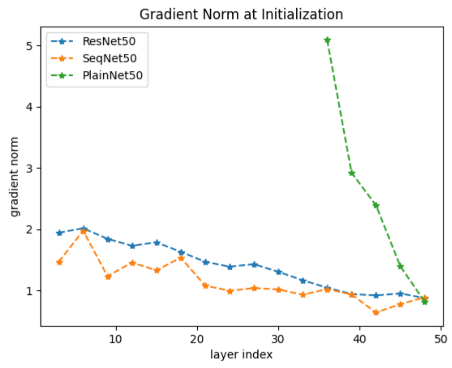
4.2. SeqNet without skip connections

In this section, we introduce the proposed SeqNet by stacking the mentioned network g.

Fully-connected Network The basic module is as stated in Section 3.1. Let d be the input dimension, and $k \geq 2$ be a network hyper-parameter. The basic module is a stack of 1) a fully connected layer with input and output dimension d and kd respectively; 2) a ReLU activation; 3) a fully connected layer with input and output dimension kd and d respectively and 4) a batch normalization layer. Let $\boldsymbol{W} \in \mathbb{R}^{kd \times d}, \boldsymbol{M} \in \mathbb{R}^{d \times kd}$ denote the fully connected layers' weights, the formula is $\boldsymbol{y} = \mathcal{B}(\boldsymbol{M}^\top \text{ReLU}(\boldsymbol{W}\boldsymbol{x}))$. Initialize the element $\boldsymbol{W}$ with i.i.d. normal distribution $\boldsymbol{W}_{ij} \sim \mathcal{N}(0, \frac{2}{kd})$. The weight $\boldsymbol{M}$ is initialized as $\boldsymbol{W}^\top$.

Convolution Network Following the same ideas for fully-connected network, we make minimal changes to the ResNet architectures (He et al., 2016). The network consists of the first 7-by-7 kernel size convolution layer, four stages using a unified building block, a global pooling layer, and the last classification layer. Our modifications are on the building block, i.e., **BasicBlock** and **BottleNeck**.

BasicBlock has two convolution layers. Suppose the input dimension is d, the two convolution layers are of input dimension d, output dimension d and kernel size 3-by-3. We replace them with one convolution layer with input dimension d, output dimension $2d$, kernel size 3-by-3, and another convolution layer with input dimension $2d$, output dimension d, kernel size 1-by-1. Figure 3 shows the examples for the two building blocks. We only initialize the "center" of the convolution weights and leave other weights as zero. Taking the 3-by-3 convolution in Figure 3 **SeqBasic** as an example, its weight is a 4-D tensor $\boldsymbol{W} \in \mathbb{R}^{128 \times 64 \times 3 \times 3}$. The "center" of the weight is $\boldsymbol{W}[:, :, 2, 2]$, a 128-by-64 matrix. For 1-by-1 convolution, the "center" of the weight is itself. For the **SeqBasic** block, we can initialize the "center" of the weight for the two convolution layers using the three methods mentioned for fully-connected layers. Parameters at other positions are initialized as zero. **SeqBottle** is the counterpart for **BottleNeck**, and built on **SeqBasic**. We move the first 1-by-1 convolution in **BottleNeck** after the third 1-by-1 convolu-



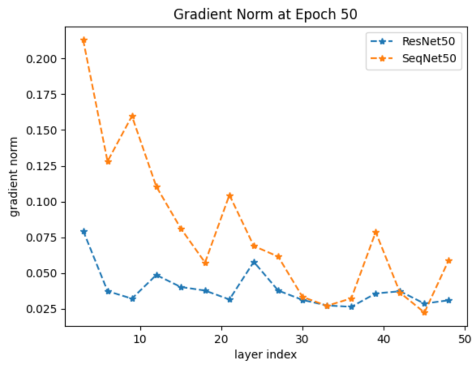


Figure 4: **Left**: The gradient norm of three models against layer index at initialization using ImageNet data. **Right**: The gradient norm of ResNet50 and SeqNet50 against layer index during training (at Epoch 50).

tion. It forms a similar basic module as discussed for fully-connected network. Then we add these two layers to the top of **SeqBasic**, and get the **SeqBottle** building block. The initialization of **SeqBottle** is the same as **SeqBasic**.

We replace **BasicBlock** in ResNet18 and ResNet34 with **SeqBasic** to get SeqNet18 and SeqNet34, and replace **BottleNeck** in ResNet50 and ResNet101 with **SeqBottle** to get SeqNet50 and SeqNet101.

5. Experiments

ImageNet Classification. We experiment in the ImageNet classification dataset (Deng et al., 2009). We use the conventional training settings without any training tricks. Please refer to Appendix B.1 for details. Figure 4 shows the gradient norm of three models at initialization. PlainNet50 simply removes the skip connection in ResNet50. At the initial state, both ResNet50 and SeqNet50 have stable gradient during back propagation, while the gradient of PlainNet50 increases quickly. The right figure shows the gradient norm of ResNet50 and PlainNet50 during training. Both two curves are stable generally stable, but the gradient norm increases faster for SeqNet50.

Comparison with ResNets and other plain networks. As shown in Table 1, our methods are significantly better than existing plain networks in all layer settings and able to match the performance of ResNets. It is overfitting that makes SeqNet50/101 worse than ResNets. If any block of the plain network does not learn well (maybe because of unlucky initialization or noise), the final output results cannot be good. However the results of ResNets behaves like ensembles of all residual blocks (Veit et al., 2016), making ResNets more robust. The reason why the performance of SeqNet101 is worse than SeqNet50 is possibly the performance degradation problem discussed in (He et al., 2016; Balduzzi et al., 2017) as the training accuracy of SeqNet101 is also lower than SeqNet50. The gradient norm of SeqNet101 is in a normal range. Due to page limit, please refer to the Appendix B.2 for the ablation studies and Appendix B.3 for learning curves of SeqNet.

Object Detection and Segmentation. We adopt Mask R-CNN (He et al., 2017) with a Feature Pyramid Network (FPN) as the detection model, and compare the SeqNet backbone with the ResNet50 backbone with BatchNorm (BN) or GroupNorm (GN) (Wu and He, 2018). The models are pre-trained on ImageNet classification, fine-tuned on the COCO train2017 set and evaluated on the COCO val2017 set. We report the standard COCO metrics of Average Precision (AP), AP50, and AP75 , for bounding box detection (AP^{bbox}) and instance segmentation (AP^{mask}). As shown in Table 2, SeqNet model's performance is comparable to the ResNet50 models, demonstrating that our model has good feature transfer abilities. (2X or 3X indicate the 180k or 270k iterations training).

6. Related Work and Discussion

Theories for residual networks and plain networks. Veit et al. (2016) show ResNets behaves like ensembles of shallow networks. Hardt and Ma (2016) prove ResNets with ReLU activations have universal finite-sample expressivity. Orhan and Pitkow (2017) hypothese that skip-connections improve performance by breaking symmetries. Balduzzi et al. (2017) suggest that the "shattered gradients problem" might be the reason for training difficulties in deep plain networks. Zaeemzadeh et al. (2020) prove that skip connections facilitate preserving the norm of the gradient, and lead to stable back-propagation. Hanin and Rolnick (2018) provide some theoretical results in predicting when networks are able to start training. Isometry is another popular motivation for analyse the signal propagation (Saxe et al., 2014; Philipp et al., 2017; Pennington et al., 2017). Tarnowski et al. (2019) demonstrate that dynami-

Model	18-layer	34-layer	50-layer	101-layer
ResNet (He et al., 2016)	69.8%	73.3%	77.4%	78.8%
DiracNet(et al, 2017)	70.4%	72.8%	NA	NA
ISONet(Qi et al., 2020)	68.1%	70.9%	71.2%	71.0%
SeqNet (Ours)	72.4%	75.7%	77.1%	76.7%

Table 1: Comparison with ResNets and other plain networks on ImageNet validation.

Model	AP^{bbox}	AP^{bbox}_{50}	AP^{bbox}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}
ResNet50 BN 2X	38.6	59.8	42.1	34.5	56.4	36.3
ResNet50 GN 2X	40.3	61.0	44.0	35.7	57.9	37.7
SeqNet50 2X	39.8	59.5	43.1	35.3	57.4	37.8
ResNet50GN 3X	40.8	61.6	44.4	36.1	58.5	38.2
SeqNet50 3X	40.6	60.7	44.9	36.0	58.2	38.5

Table 2: Detection and segmentation results in COCO using Mask R-CNN and FPN.

cal isometry is achievable irrespective of the activations in ResNets.However, these work implicitly ignore the gradient exploding problem, which is the first obstacle for training. Another tool to study this problem is mean field theory (Poole et al., 2016; Schoenholz et al., 2016), which requires the where the network to be very wide. Although our SeqNets also require this assumption (Proposition 4.1), understanding the bad trainability of plain networks do not need this assumption (Proposition 3.14).

Other Attempts to train deep plain networks. et al (2017) propose to train plain networks up to 34 layers. However their idea is to simulating identical kernels and initial the convolution weights as an intensity matrix. More generally, orthogonality is widely used in deep networks (Bansal et al., 2018; Wang et al., 2020). Xiao et al. (2018) show that orthogonality enables deep ConvNets trainable. However, the model requires the activation layer to be linear around 0 (tanh). The performance is far below the baseline even on CIFAR dataset. By using isometry, Qi et al. (2020) propose a deep isometric network without skip connections or normalisation with a Shifted ReLU, making the activation linear around 0. These models require special initialization methods and take the advantage of initialized as linear operations. However, our models use Gaussian distribution and every building block in SeqNet is not linear at initialization.

Insights from our work We first need to emphasize again that there is not many significant advantages of training plain networks, and the propose of SeqNet is not another strong architecture for practice. However, our work can provide new insights for many phenomena. Firstly, although residual network generally has better performance than the VGG network on a wide range of tasks, it is not as good as VGG network on the style transfer task. The reason behind this is still an open question. As we analyse, the gradient exploding problem is more significant on the VGG network, thus the derivative over the inputs for VGG network is much larger that that of residual network (Wang et al., 2021). A small change over the input can lead to large change over the output for VGG network, make it earlier to transfer an image. Another recent example is the self-supervised vision transformers (MoCo V3) (Chen et al., 2021). The authors find the instability of shallow layers and choose to freeze these layers during training. Although we cannot provide a better solution, we can explain this phenomenon. Although residual architecture can solve the exploding gradient problem, the entire network as a whole is still nonlinear, especially after training. The layers' weight gradient is larger than deeper layers. If the network is not very stable, the instability is more likely to happen in shallow layers. Finally, thanks to the performance on non-toy datasets and simple training settings, SeqNets can be used as a convincing example to empirically study the difference of learning dynamics between residual and plain networks, e.g., what does skip connection solves besides the exploding gradient problem.

7. Conclusion

In this paper, we re-visit the exploding gradient problem, and conclude that nonlinear activation may be the source of explosive gradients. The gradient norm of a plain network is of exponential order to the number of nonlinear layers, making plain networks not trainable. Based on our theory, we propose SeqNet, a plain network that does not have the exploding gradient problem. Without skip connections, SeqNet can match the performance of ResNet counterpart on image classification and object detection tasks. Studying SeqNet may help better understand ResNets.

References

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. On the convergence rate of training recurrent neural networks. In *Advances in Neural Information Processing Systems*, 2019.

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 *International Conference on Machine Learning*, pages 342–350. PMLR, 2017.

Nitin Bansal, Xiaohan Chen, and Zhangyang Wang. Can we gain more from orthogonality regularizations in training deep cnns? *arXiv preprint arXiv:1810.09102*, 2018.

Xinlei Chen, Saining Xie, and Kaiming He. An empirical study of training self-supervised vision transformers. *arXiv preprint arXiv:2104.02057*, 2021.

Herman Chernoff. A note on an inequality involving the normal distribution. *The Annals of Probability*, pages 533–535, 1981.

Soham De and Sam Smith. Batch normalization biases residual blocks towards the identity function in deep networks. *Advances in Neural Information Processing Systems*, 33, 2020.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Zagoruyko et al. Diracnets: Training very deep neural networks without skip-connections. *arXiv preprint arXiv:1706.00388*, 2017.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Boris Hanin and David Rolnick. How to start training: the effect of initialization and architecture. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 569–579, 2018.

Moritz Hardt and Tengyu Ma. Identity matters in deep learning. *arXiv preprint arXiv:1611.04231*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *IEEE International Conference on Computer Vision*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

Zhen Huang, Tim Ng, Leo Liu, Henry Mason, Xiaodan Zhuang, and Daben Liu. Sndcnn: Self-normalizing deep cnns with scaled exponential linear units for speech recognition. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6854–6858. IEEE, 2020.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, 2015.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Proceedings of the 31st international conference on neural information processing systems*, pages 972–981, 2017.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

A Emin Orhan and Xaq Pitkow. Skip connections eliminate singularities. *arXiv preprint arXiv:1701.09175*, 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 2019.

Jeffrey Pennington, Samuel S Schoenholz, and Surya Ganguli. Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 4788–4798, 2017.

George Philipp, Dawn Song, and Jaime G Carbonell. The exploding gradient problem demystified-definition, prevalence, impact, origin, tradeoffs, and solutions. *arXiv preprint arXiv:1712.05577*, 2017.

Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. *Advances in neural information processing systems*, 29: 3360–3368, 2016.

PyTorch. Pytorch, official image models implementation. https://github.com/pytorch/vision/tree/master/references/, 2020.

Haozhi Qi, Chong You, Xiaolong Wang, Yi Ma, and Jitendra Malik. Deep isometric learning for visual recognition. In *International Conference on Machine Learning*, pages 7824–7835. PMLR, 2020.

Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *International Conference on Learning Representations*, 2014.

Samuel S Schoenholz, Justin Gilmer, Surya Ganguli, and Jascha Sohl-Dickstein. Deep information propagation. *International Conference on Learning Representations, ICLR 2017*, 2016.

Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. Understanding and improving convolutional neural networks via concatenated rectified linear units. In *international conference on machine learning*, pages 2217–2225. PMLR, 2016.

Jie Shao, Kai Hu, Changhu Wang, Xiangyang Xue, and Bhiksha Raj. Is normalization indispensable for training deep neural network? *Advances in Neural Information Processing Systems*, 33, 2020.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

Wojciech Tarnowski, Piotr Warchol, Stanislaw Jastrzebski, Jacek Tabor, and Maciej Nowak. Dynamical isometry is achieved in residual networks in a universal way for any activation function. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2221–2230. PMLR, 2019.

Andreas Veit, Michael Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. *arXiv preprint arXiv:1605.06431*, 2016.

Jiayun Wang, Yubei Chen, Rudrasis Chakraborty, and Stella X Yu. Orthogonal convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11505–11515, 2020.

Pei Wang, Yijun Li, and Nuno Vasconcelos. Rethinking and improving the robustness of image style transfer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 124–133, 2021.

Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19, 2018.

Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks. In *International Conference on Machine Learning*, pages 5393–5402. PMLR, 2018.

Greg Yang and Samuel S Schoenholz. Mean field residual networks: on the edge of chaos. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 2865–2873, 2017.

Greg Yang, Jeffrey Pennington, Vinay Rao, Jascha Sohl-Dickstein, and Samuel S Schoenholz. A mean field theory of batch normalization. *arXiv preprint arXiv:1902.08129*, 2019.

Alireza Zaeemzadeh, Nazanin Rahnavard, and Mubarak Shah. Norm-preservation: Why residual networks can become extremely deep? *IEEE transactions on pattern analysis and machine intelligence*, 2020.

Hongyi Zhang, Yann N. Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

A. Numerical simulations of exploding gradient

In Section 2 we show that plain ReLU networks with batch normalization have the exploding gradient problem. Now we show that neither the type of activation layers nor the existence of batch normalization layers is the key of this problem. The existence of nonlinear activation layers is the *culprit*. We still use the plain network as in Section 2, but replace the activation layers with SELU [5] or Tanh[6]. We also study the case when batch normalization layers are removed from the network. We use normal distribution $\mathcal{N}(0, 1/d), \mathcal{N}(0, 2/d), \mathcal{N}(0, 2/d)$ to initialize the layer weights of SELU, Tanh and ReLU networks respectively so that the forward pass is stable. As shown in Figure 5, different activations have different exponential rate, but they all have the exploding gradient problems regardless of the type of activation layers nor the existence of batch normalization layers.

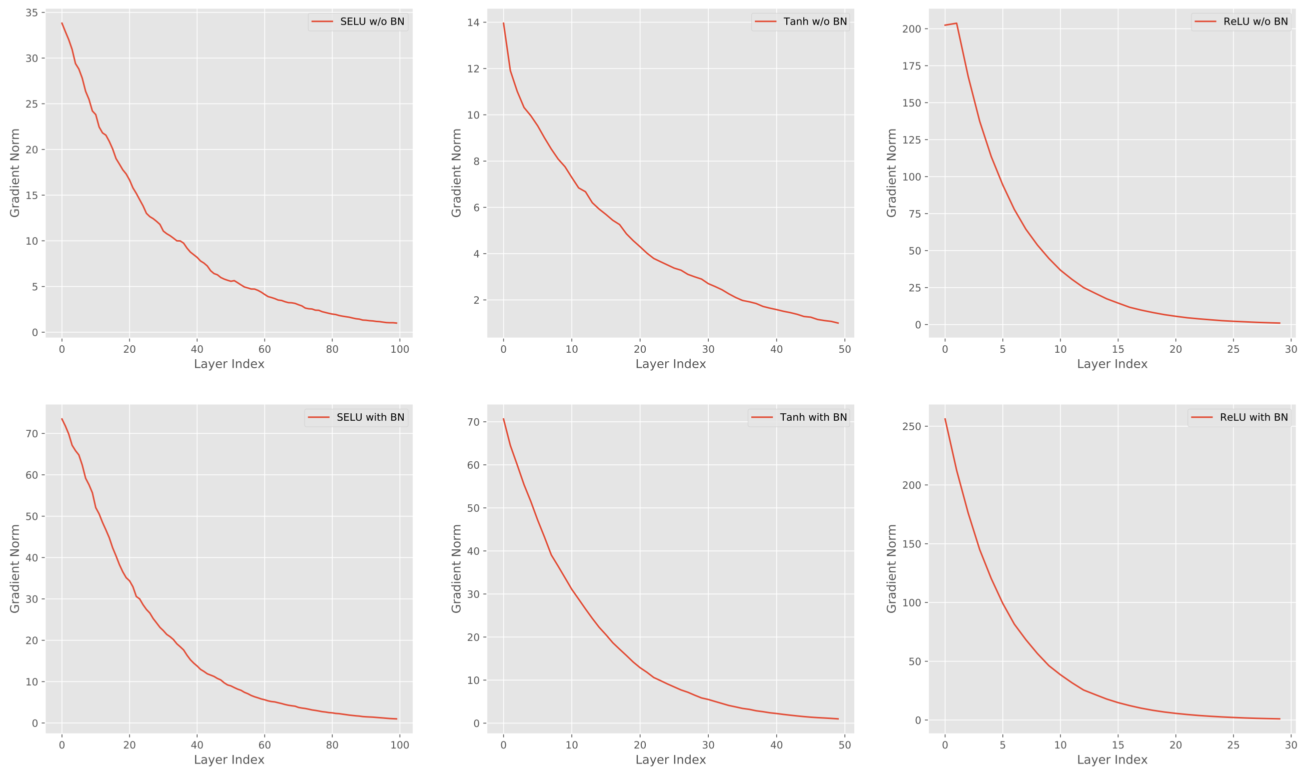


Figure 5: The gradient norm against layer index for SELU, Tanh, and ReLU with/without batch normalization layers. Different activations have different exponential rate.

We can see that SELU has the smallest exponential rate and ReLU has the largest exponential rate. As we analysed in Section e, this exponential rate is predictable (if can we assume the hidden layer distribution is asymptotic normal). The exponential rate for activation σ is approximately:

$$X \sim \mathcal{N}(0, 1), \quad r(\sigma) = \frac{\mathbb{E}[\sigma'(X)]^2}{\text{Var}[\sigma(X)]} \tag{1}$$

The exponential rates are 1.072, 1.178, and 1.467 for SELU, Tanh, and ReLU respectively. Huang et al. (2020) train a 50-layer plain network without skip connections by using SELU, but provides no explanations about why such a deep plain network is trainable. Our work can provide some insights, and the maximum trainable depth (of plain networks) is larger for SELU than that for ReLU. Yang et al. (2019) also find the exploding gradients in plain networks, but ascribe it to batch normalization layers. Figure 5 shows that it may not be the case. However, batch normalization layers indeed increase the

[5] https://pytorch.org/docs/stable/generated/torch.nn.SELU.html
[6] https://pytorch.org/docs/stable/generated/torch.nn.Tanh.html

exponential rates. This is because batch normalization layers rescale the hidden distributions to the most non-linear regime of the activations.

A.1. Verification of the propositions

According to our propositions, the exponential rate of the exploding gradient for the network defined above should only determined on the types of the activations, specifically the square root of $r(\sigma)$ defined in Equation 1 (since we are plotting the gradient norm not the square of gradient norm). For ReLU activation, the exponential rate is $\alpha = \sqrt{\frac{\pi}{\pi-1}}$. Figure 6 how close does the gradient norm curve match the exponential function. Although some of our assumptions may not be the real case, our conclusions still match the read case quite well.

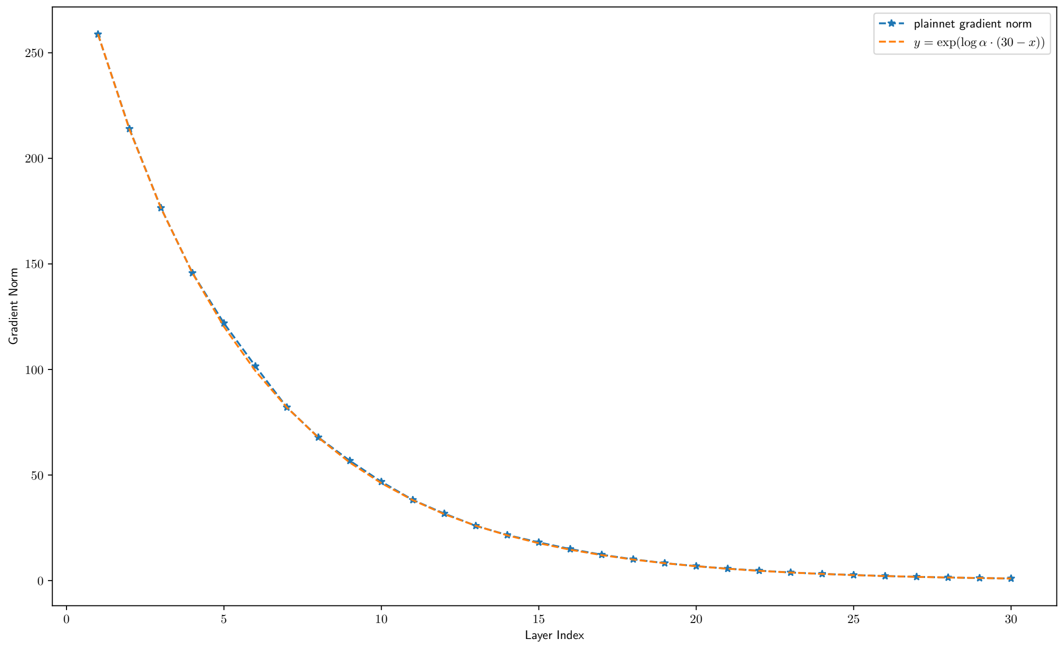


Figure 6: The actual gradient norm and the predicting gradient norm.

B. ImageNet Classification Experiments

We experiment in the ImageNet classification dataset. The dataset contains 1.28 M training images and 50k validation images that are labeled with 1000 categories.

B.1. Implementation details.

All models are trained using SGD with weight decay 0.0001, momentum 0.9 and batch size 256. W adopt standard data augmentations as in PyTorch (2020). The top-1 classification accuracy on the validation set is reported. All results are averaged over 5 runs. We use cosine learning rate decay with a gradual warmup (Goyal et al., 2017) applied for the first 5 epochs. For ResNets, we use an initial learning rate of 0.1. For SeqNets, we use an initial learning rate of 0.1 for the last classification layer, but need a smaller learning rate for the convolution layers since these is no skip connection. We grid search the learning rate from $\{0.01, 0.02, 0.04, 0.08\}$ for SeqNet34, and find that 0.04 works best. We use this learning rate (0.04) to guide the learning rate selections for other SeqNets.

- The number of layers in SeqNet18 is half of SeqNet34, we choose the learning rate 0.08.
- The number of layers in SeqNet50 is twice of SeqNet34, we choose the learning rate 0.02.
- The number of layers in SeqNet101 is twice of SeqNet50, we initially choose the learning rate 0.01, but the convergence is slow. We find the reason is that SeqNet101's weights are not uniformly distributed. The number of building blocks for the 4 stage of SeqNet34, SeqNet50 (also ResNet34 and ResNet50) are 3, 4, 6, 3 respectively. However, for SeqNet101, they are 3, 4, 23, 3 respectively. The added 50 layers are all located in the third stage. We choose to keep learning rate 0.02 except for the third stage, and use a learning rate of $0.02 \times \dfrac{6}{23}$ for the third stage in SeqNet101. The new learning rate strategy brings 0.4% accuracy improvement for SeqNet101.

B.2. Ablation Study.

Our baseline is SeqNet50, replacing ResNet50's *BottleNeck* with *SeqBottle*, and trained with the orthogonal regularization. Table 3 shows the following ablation studies.

Initialization We compare the three initialization methods discussed for the SeqNet module:

L-Init Randomly initialize $\boldsymbol{R} \in \mathbb{R}^{\frac{kd}{2} \times d}$ as orthogonal $\boldsymbol{R}^\top \boldsymbol{R} = \mathcal{I}_d$. Let $\boldsymbol{W} = \sqrt{2}\begin{bmatrix} \boldsymbol{R} \\ -\boldsymbol{R} \end{bmatrix}$ and $\boldsymbol{M} = \boldsymbol{R}^\top$. ($\sqrt{2}$ is a compensate for ReLU (He et al., 2015))

O-Init Randomly initialize $\boldsymbol{R} \in \mathbb{R}^{kd \times d}$ as orthogonal $\boldsymbol{R}^\top \boldsymbol{R} = \mathcal{I}_d$. Let $\boldsymbol{W} = \sqrt{2}\boldsymbol{R}, \boldsymbol{M} = \boldsymbol{W}^\top$.

R-Init Initialize the element $\boldsymbol{W}$ with i.i.d. normal distribution $\boldsymbol{W}_{ij} \sim \mathcal{N}(0, \frac{2}{kd})$. Let $\boldsymbol{M} = \boldsymbol{W}^\top$.

R-Init is the method in our main paper. As shown in Table 3 (a), R-Init is better than O-Init and R-Init, but with a very limit margin, which supports our argument. Moreover, if we use two random matrices to initialize $\boldsymbol{W}$ and $\boldsymbol{M}$ in $\boldsymbol{y} = \boldsymbol{M}\text{ReLU}(\boldsymbol{W}\boldsymbol{x})$ without no weight tying (the "looks linear" initialization proposed by Balduzzi et al. (2017)), the network cannot reach a validation accuracy higher than 70%.

Basic Module As we analyse in Section 4, a bias before the ReLU activation is not necessary, i.e., $\boldsymbol{y} = \boldsymbol{W}^\top\text{ReLU}(\boldsymbol{W}\boldsymbol{x})$. However, adding a bias or a BatchNorm layer before the activation layer generally helps training. In Table 3 (b), we test whether using no bias is reasonable, and find that using a bias or BatchNorm is slightly worse. However, adding a BatchNorm layer slightly improve the performance of SeqNet101 by 0.1%. We think it is marginal thus not show this in the table.

Position for 3-by-3 convolution As shown in Figure 3, we put the 3-by-3 convolution as the first layer in both *SeqBasic* and *SeqBottle*. It would be interesting to see whether placing the 3-by-3 convolution somewhere would influence the model performance. We consider two extra cases: 1) Second Layer: using 3-by-3 convolution as the second layer and 1-by-1 convolution for the other three layers. 2) Third layer: move the current third and fourth layer in front of the the 3-by-3 convolution, i.e., $1 \times 1(64 \to 256); 1 \times 1(256 \to 64); 3 \times 3(64 \to 128); 1 \times 1(128 \to 64)$. We find that putting 3-by-3 convolution as the second layer is not good since it will make 3-by-3 convolution away from activation function. However, putting it as the third layer does not make much difference.

Table 3: Ablation Study on ImageNet with SeqNet50 backbone

(a) Initialization

Method	Accuracy
R-Init	**77.1%**
O-Init	77.0%
L-Init	77.0%

(b) Bias before ReLU

Method	Accuracy
No Bias	**77.1%**
Use Bias	77.0%
Use BatchNorm	76.8%

(c) Position for 3-by-3 conv

Method	Accuracy
First layer	**77.1%**
Second layer	76.7%
Third Layer	77.0%

Orthogonal Regularization While the SeqNets alone are trainable, we find that a widely used orthogonal regularization (Xiao et al., 2018; Bansal et al., 2018; Wang et al., 2020; Qi et al., 2020) can improve the performance a lot. For a weight $\boldsymbol{W} \in \mathbb{R}^{D\times d}$, let $\Sigma = \boldsymbol{W}^\top \boldsymbol{W}$ if $D \geq d$, and $\Sigma = \boldsymbol{W}\boldsymbol{W}^\top$ if $D < d$. A regularization loss is added to the original loss function for back propagation: $n = \min\{d, D\}, \quad \ell(W) = \eta \| \frac{n}{\text{trace}(\Sigma)} \Sigma - \mathcal{I}_n \|_F^2$

We further study how orthogonal regularization impact the training of SeqNet with different layers. As shown in 4, although SeqNet itself is trainable, orthogonal regularization can provide significant improvement. The improvement is much larger than the gain of orthogonal regularization on ResNets (Bansal et al., 2018), which may indicate unknown different properties of SeqNets and ResNets.

#layers	- OrthReg	+ OrthReg	Improvement
34	73.7%	75.3%	+1.6%
50	75.7%	77.1%	+1.4%
101	75.6%	76.7%	+1.1%

Table 4: Ablation on orthogonal regularization. Validation accuracies on ImageNet. Higher is better.

Tying Conv2	Tying Conv4	Accuracy
Yes	Yes	76.7%
Yes	No	**77.1%**
No	No	76.9%

Table 5: Ablation on weight tying. SeqNet50 validation accuracies on ImageNet.

The last ablation study is about weight tying. At initialization, the weights between a ReLU activation are initialized with the matrix (with a transpose difference). Whether tying (sharing) these weights remain a question. We name the convolution layers in the *SeqBottle* as Conv1, 2, 3, and 4 from shallow to deep. Table 5 shows the results of tying two 1-by-1 convolution layers. Tying (or not tying) does not always indicate performance improvement.

B.3. Training dynamics of SeqNet and ResNets

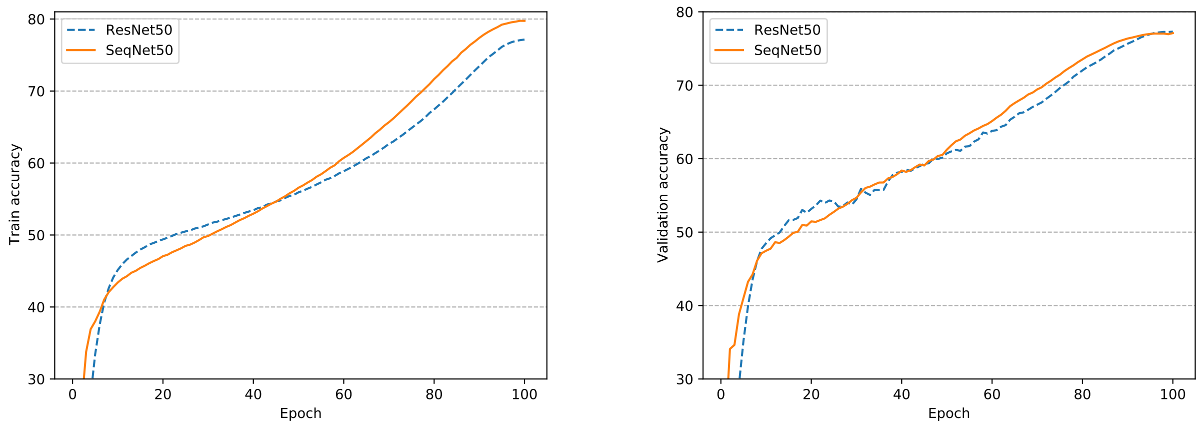


Figure 7: Training (left) and validation (right) accuracy(%) on ImageNet using 50-layer network

Figure 7 and Figure 8 show the training and validation curves of ResNets and the corresponding SeqNets. We can find that SeqNets converge faster than ResNets at the very beginning, but converge slower after 10 epochs. Thus our learning rates might be smaller than the optimal learning rates. The train accuracy gap between SeqNets and ResNets decreases in the middle of training, showing that the loss landscape of SeqNets is in a good condition during training. However, the final train accuracy of SeqNet50 is higher than ResNet50 while the final train accuracy of SeqNet101 is lower than ResNet101, showing that SeqNet101 may still suffer from some uncertain model degradation. The validation accuracy of SeqNet101 is also lower than SeqNet50 and ResNet101.

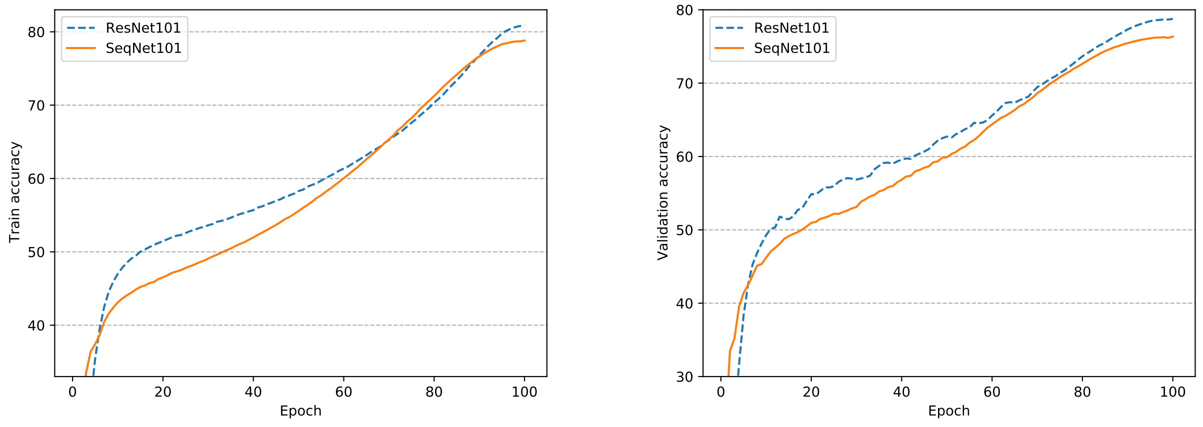


Figure 8: Training (left) and validation (right) accuracy(%) on ImageNet using 101-layer network

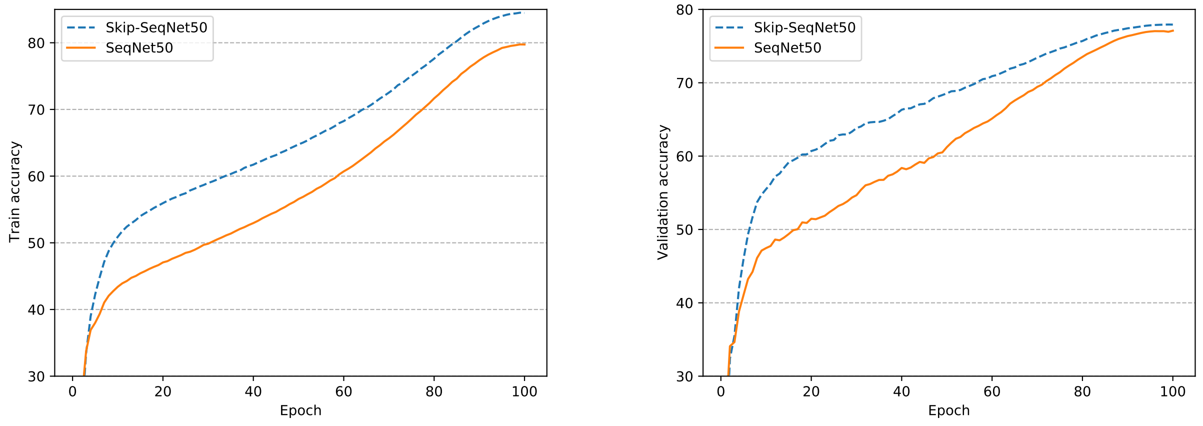


Figure 9: Training (left) and validation (right) accuracy(%) on ImageNet using 50-layer network

One might be curious, what if we add skip connections back to SeqNets? We do the experiments on SeqNet50. We add the ResNet-like skip connections to SeqNet50, and the validation accuracy is 77.9%, which improves SeqNet50 by 0.8%. Figure 9 shows the training and validation curves of SeqNet50 and the skip-connected version, Skip-SeqNet50. Skip-SeqNet50 is completely better than SeqNet50, showing that skip connections are still the most effective components for training deep neural network.

C. Discussion

C.1. Decreasing variance in the VGG figure.

Figure 1 (right) in the main paper can be seem as a record of the exploding gradient magnitude. However the metric σ decreases at layer 5, 9 and 13 (green circles), which means the derivative variance decreases at these layers.

The max-pooling layer (kernel size 2, stride 2) right before each of the mentioned layers could be the reason. Max-pooling operation is nonlinear but does not apply to our previous analysis since it is not an element-wise operation. It takes as input some neighbor pixels in the same feature map $y = \max(x_1, x_2, x_3, x_4)$, which are obviously not independent. Thus we can only give an informal intuition with an imprecise assumption.

We assume x_1, x_2, x_3, x_4 are independent standard Gaussian distributions. Under this assumption, easy to know the backward scaling ratio for the max pooling operation is $\frac{1}{4}$, and the forward scaling ratio is 0.492 by numerical simulation. The backward scaling ratio is smaller than the forward scaling ratio, making the derivative variance decrease.

C.2. No previous observation of the exploding gradient phenomenon.

He et al. (2016) argue that the degradation problem of plain networks is unlikely to be caused by vanishing gradients by verifying that the backward gradients exhibit healthy norms with BN. We argue that BatchNorm cannot solve, but conceal this problem.

We consider one weight $\boldsymbol{w}$ in the the shallow layer. At initialization, the gradient of the weight $\boldsymbol{g}$ is exponentially large $\|\boldsymbol{g}\| = \mathcal{O}(c^L)$. After one step of gradient decent update $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \boldsymbol{g}$, the weight norm is in the same order of the gradient norm $\|\boldsymbol{w}\| = \mathcal{O}(\eta c^L)$. In the next step forward, the variance of this layer's output (also the input to the next BatchNorm layer) also grows large: $\sigma^2 = \mathcal{O}(\eta^2 c^{2L})$. Recall proposition 3.8, the BatchNorm layer will scale the variance of the backward signals by $1/\sigma^2$, which canceled the exponentially large backward signals. Thus we are not likely to observe the large gradient after the first several updates. Please note that the degradation problem still exists. The weight has a very large norm while its gradient is normal, thus the convergence would be exponentially slow.

C.3. Exploding gradient in vanilla residual network

The vanilla residual network does not use special initialization for the last batch normalization layer, thus also has the exploding gradient problem. This is known in the literature, and can also be concluded from our theory. The exploding rate is smaller than plain network: $R_b(\mathcal{R})/R_f(\mathcal{R}) = (1 + R_b(\mathcal{F}))/(1 + R_f(\mathcal{F})) < R_b(\mathcal{F})/R_f(\mathcal{F})$. We can observe this by increasing network depth in the experiment of Figure 1 (left), as shown in Figure 10.

C.4. Non-centered input for ReLU

During training, the affine operator in the BatchNorm layer may learn a non-zero bias b, and the input to the next activation layer is not zero-centered. If the bias is small, the inequality still holds if we can make some assumption on the neurons' tail distribution:

Assumption C.1. Let x be the output of the normalization operator in the BatchNorm layer, and assumption 3.13 holds for x. Further assume there exists a function h defined on the real line such that $|\frac{p(x+b) - p(x)}{b}| \leq h(x)$ and $\int_0^\infty u^2 h(u) du < \infty$. Let b denote the bias of the affine operator in the BatchNorm layer that $\mathcal{O}(b^2)$ terms can be ignored.

assumption C.1 is mild: hidden neurons are generally considered to be bounded, which would satisfy assumption C.1. If the bias is small, we still have:

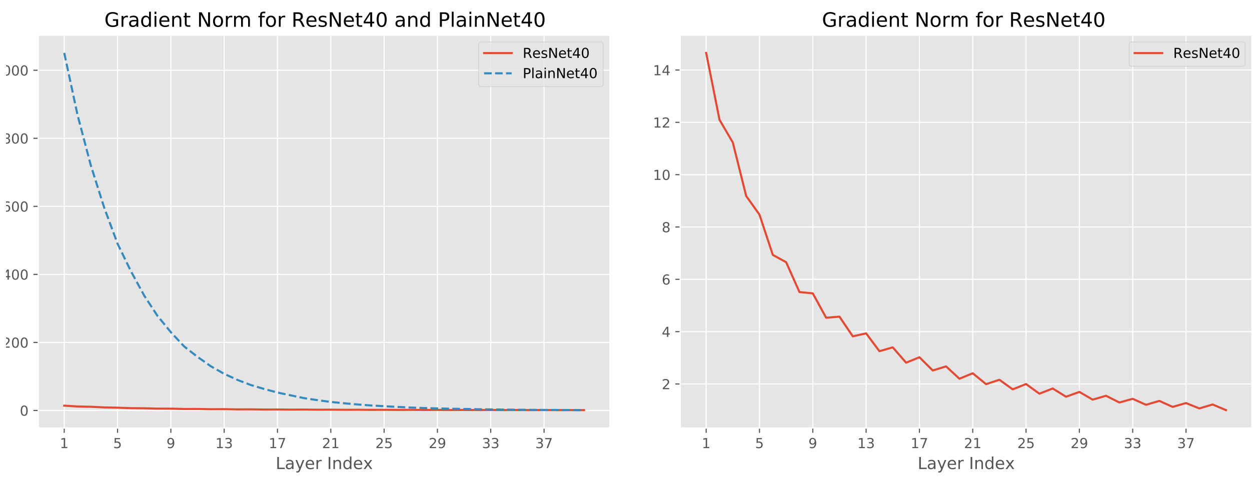


Figure 10: The gradient norm of ResNet40 and PlainNet40.

Proposition C.2. *Let $y = x + b$ be the output of the BatchNorm layer (also input to the next activation):*

$$Var[\sigma(y)] \le \mathbb{E}[\sigma'(y)^2] - \frac{1}{4}E[|x|] + \left(\frac{1}{2}\mathbb{E}[|x|] - p(0)\right) b.$$

Further if $|b| \le \dfrac{\mathbb{E}[|x|]}{4\,|\mathbb{E}[|x|] - 2p(0)|}$, from proposition C.2, we have $\mathrm{Var}[\sigma(y)] \le \mathbb{E}[\sigma'(y)^2] - \dfrac{1}{8}E[|x|]$.

D. Missing proofs in the main paper

Proof of Proposition 3.4 Note that the entries of $\boldsymbol{x}$ are independent and identically distributed, let $\mathrm{Var}[\boldsymbol{x}_i] = \sigma^2$ for all i. Then $\mathrm{Var}[\boldsymbol{x}] = \sigma^2$.

$$\mathrm{Var}[\boldsymbol{y}_i] = \mathrm{Var}[\sum_j \boldsymbol{W}_{ij}\boldsymbol{x_j}] = \sum_j \boldsymbol{W}_{ij}^2 \mathrm{Var}[\boldsymbol{x_j}] = \sigma^2 \sum_j \boldsymbol{W}_{ij}^2.$$

Then $\mathrm{Var}[\boldsymbol{y}] = \dfrac{1}{D}\sum_i \mathrm{Var}[\boldsymbol{y}_i] = \dfrac{\sigma^2}{D}\sum_i\sum_j \boldsymbol{W}_{ij}^2 = \dfrac{\|W\|_F^2}{D}\sigma^2 = \dfrac{\|W\|_F^2}{D}\mathrm{Var}[\boldsymbol{x}]$. The backward pass is similar.

Proof of Corollary 3.7 The backward derivative for the batch normalization layer is (following the notation in the main paper): $\dfrac{\partial \ell}{\partial x_i} = \dfrac{1}{\sigma_x}[\dfrac{\partial \ell}{\partial y_i} - \dfrac{y_i}{B}\sum_{j=1}^{B}\dfrac{\partial \ell}{\partial y_j}y_j - \dfrac{1}{B}\sum_{j=1}^{B}\dfrac{\partial \ell}{\partial y_j}]$. Refer to Ioffe and Szegedy (2015) for the proof of this result. Note that $\sum_{i=1}^{B} y_i = 0$, we have $\sum_{i=1}^{B}\dfrac{\partial \ell}{\partial x_i} = 0$, which means the sample mean of the batch normalization layer input derivative is 0 and $\mathbb{E}\dfrac{\partial \ell}{\partial x} = 0$ where x is the input to the batch normalization layer.

Let $y = f(x)$ be the previous layer of the last batch normalization layer, i.e., the output of f is the input of the last batch normalization layer. We have $\mathbb{E}\left[\dfrac{\partial \ell}{\partial y}\right] = 0$ since y is the input to the batch normalization layer. Further $\mathbb{E}\left[\frac{\partial \ell}{\partial x}\right] = \mathbb{E}\left[\frac{\partial \ell}{\partial y}f'(x)\right] = \mathbb{E}\left[\frac{\partial \ell}{\partial y}\right]\mathbb{E}\left[f'(x)\right] = 0$ during assumption 2.

We can continue to prove for all layers before the last batch normalization recursively.

Proof of Proposition 3.8 Recall the backward for the batch normalization layer: $\frac{\partial \ell}{\partial x_i} = \frac{1}{\sigma_x}[\frac{\partial \ell}{\partial y_i} - \frac{y_i}{B}\sum_{j=1}^{B}\frac{\partial \ell}{\partial y_j}y_j - \frac{1}{B}\sum_{j=1}^{B}\frac{\partial \ell}{\partial y_j}]$ and $\sum_{i=1}^{B}\frac{\partial \ell}{\partial x_i} = 0$, we have:

$$\begin{aligned}
\text{Var}[\frac{\partial \ell}{\partial x}] &= \frac{1}{B}\sum_{i=1}^{B}\left(\frac{\partial \ell}{\partial x_i}\right)^2 = \frac{1}{B\sigma_x^2}\sum_{i=1}^{B}\left[(\frac{\partial \ell}{\partial y_i} - \frac{1}{B}\sum_{j=1}^{B}\frac{\partial \ell}{\partial y_j}) - \frac{y_i}{B}\sum_{j=1}^{B}\frac{\partial \ell}{\partial y_j}y_j\right]^2 \\
&= \frac{1}{B\sigma_x^2}\left[\sum_{i=1}^{B}(\frac{\partial \ell}{\partial y_i} - \frac{1}{B}\sum_{j=1}^{B}\frac{\partial \ell}{\partial y_j})^2 - \frac{1}{B}(\sum_{j=1}^{B}\frac{\partial \ell}{\partial y_j}y_j)^2\right] \\
&= \frac{1}{\sigma_x^2}\left[\text{Var}[\frac{\partial l}{\partial y}] - \frac{1}{B^2}(\sum_{j=1}^{B}\frac{\partial \ell}{\partial y_j}y_j)^2\right]
\end{aligned}$$

Let $s_i = \frac{\partial \ell}{\partial y_i}/\sqrt{\text{Var}[\frac{\partial l}{\partial y}]}$, we have $\text{Var}[\frac{\partial \ell}{\partial x}] = \frac{1}{\sigma_x^2}\text{Var}[\frac{\partial \ell}{\partial y}](1 - \frac{1}{B^2}(\sum_{j=1}^{B}s_i y_i)^2)$ where s_i and y_i are independent and $\mathbb{E}s_i = \mathbb{E}y_i = 0, \mathbb{E}s_i^2 = \mathbb{E}y_i^2 = 0$. Let $z = \frac{1}{B}(\sum_{j=1}^{B}s_i y_i)^2$. Expand the equation, we have $\mathbb{E}[z] = 1, \mathbb{E}[z^2] = 3 + \mathcal{O}(\frac{1}{B})$, and $\text{Var}[z] \approx 2$. Thus the backward scaling ratio is $R_b(\mathcal{B}) = \frac{1}{\sigma^2}(1 - \frac{z}{B})$ where B is the batch size and z is a positive random variable that $\mathbb{E}z = 1, \text{Var}[z] = 2$.

Please refer to Chernoff (1981) for the proof of proposition 3.

Proof of Proposition 3.14 We know $\text{Var}[\sigma(x)] = \mathbb{E}[\sigma^2(x)] - (\mathbb{E}[\sigma(x)])^2$. Since the distribution of x is symmetric centered at zero, $\mathbb{E}[\sigma^2(x)] = \frac{1}{2}\mathbb{E}[x^2] = \frac{1}{2}$, $\mathbb{E}[\sigma'(x)^2] = P(x > 0) = \frac{1}{2}$, and $\mathbb{E}[\sigma(x)] = \frac{1}{2}\mathbb{E}[|x|]$. Thus $\text{Var}[\sigma(x)] \leq \mathbb{E}[\sigma'(x)^2] - \frac{1}{4}E[|x|]$.

Proof of Proposition 3.17 Consider a residual block $\boldsymbol{y} = \boldsymbol{x} + \mathcal{F}(\boldsymbol{x})$, Shao et al. (2020) prove that $\text{Var}[\boldsymbol{y}] = \text{Var}[\boldsymbol{x}] + \text{Var}[\mathcal{F}(\boldsymbol{x})] + \mathcal{O}(\frac{1}{d})$ where d is the dimension of $\boldsymbol{x}$. Then the equation for forward pass is trivial given the definition. The backward pass is proved similarly.

Proof of Proposition 4.1 The four ratios are all functions of the initialized weights. We use the weight distribution to estimate the expectation of the ratios, which indicates the approximation of these ratios at different initialization time following the Gaussian distribution. The proof sketch is to first derive the ratios as functions of the weights, and then do some simple statistics to get the expectation. We only give the proofs of $R_f(g)$ and $R_b(g)$. The proofs of $R_f(h)$ and $R_b(h)$ are similar but much simpler.

Proof of $R_f(g)$: $g(\boldsymbol{x}) = \boldsymbol{W}^\top\text{ReLU}(\boldsymbol{W}\boldsymbol{x}) = \frac{1}{2}(\boldsymbol{W}^\top\boldsymbol{W}\boldsymbol{x} + \boldsymbol{W}^\top|\boldsymbol{W}\boldsymbol{x}|)$. Thus the second order momentum of $g(\boldsymbol{x})$ is:

$$\begin{aligned}
\mathbb{E}[g(\boldsymbol{x})g(\boldsymbol{x})^\top] &= \frac{1}{4}\mathbb{E}_{\boldsymbol{x}}\left[(\boldsymbol{W}^\top\boldsymbol{W}\boldsymbol{x} + \boldsymbol{W}^\top|\boldsymbol{W}\boldsymbol{x}|)(\boldsymbol{W}^\top\boldsymbol{W}\boldsymbol{x} + \boldsymbol{W}^\top|\boldsymbol{W}\boldsymbol{x}|)^\top\right] \\
&= \frac{1}{4}\mathbb{E}_{\boldsymbol{x}}\boldsymbol{W}^\top\boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^\top\boldsymbol{W}^\top\boldsymbol{W} + \frac{1}{4}\mathbb{E}_{\boldsymbol{x}}\boldsymbol{W}^\top|\boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^\top\boldsymbol{W}^\top|\boldsymbol{W} + \frac{1}{4}\mathbb{E}_{\boldsymbol{x}}\boldsymbol{W}^\top|\boldsymbol{W}\boldsymbol{x}|x^\top\boldsymbol{W}^\top\boldsymbol{W} + \frac{1}{4}\mathbb{E}_{\boldsymbol{x}}\boldsymbol{W}^\top\boldsymbol{W}\boldsymbol{x}|\boldsymbol{x}^\top\boldsymbol{W}^\top|\boldsymbol{W} \\
&= \frac{1}{4}\mathbb{E}_{\boldsymbol{x}}\boldsymbol{W}^\top\boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^\top\boldsymbol{W}^\top\boldsymbol{W} + \frac{1}{4}\mathbb{E}_{\boldsymbol{x}}\boldsymbol{W}^\top|\boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^\top\boldsymbol{W}^\top|\boldsymbol{W} \\
&= \frac{1}{4}\boldsymbol{W}^\top\boldsymbol{W}\boldsymbol{W}^\top\boldsymbol{W} + \frac{1}{4}\boldsymbol{W}^\top\mathbb{E}_{\boldsymbol{x}}\left[|\boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^\top\boldsymbol{W}^\top|\right]\boldsymbol{W}
\end{aligned} \tag{2}$$

Since $x \sim \mathcal{N}(0, 2/D)$, the last two terms of Cov$[g(\boldsymbol{x})]$ is zero. Let $\boldsymbol{M} = \mathbb{E}_{\boldsymbol{x}}\left[|\boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^\top\boldsymbol{W}^\top|\right]$ and $\boldsymbol{w}_i$ be the i-th row of $\boldsymbol{W}$, we have $\boldsymbol{M}_{ij} = \mathbb{E}_{\boldsymbol{x}}|\boldsymbol{w}_i^\top\boldsymbol{x}\boldsymbol{w}_j^\top\boldsymbol{x}|$. If $i = j$, $\boldsymbol{M}_{ii} = \mathbb{E}_{\boldsymbol{x}}|\boldsymbol{w}_i^\top\boldsymbol{x}\boldsymbol{w}_i^\top\boldsymbol{x}| = \mathbb{E}_{\boldsymbol{x}}(\boldsymbol{w}_i^\top\boldsymbol{x})^2 = \boldsymbol{w}_i^\top\mathbb{E}_{\boldsymbol{x}}\boldsymbol{x}\boldsymbol{x}^\top\boldsymbol{w}_i = \boldsymbol{w}_i^\top\boldsymbol{w}_i$.

Now we consider the case $i \neq j$. Let $\boldsymbol{u} = (\|w_i\|, 0, \cdots, 0)^\top$, there exists an orthogonal matrix $\boldsymbol{R}$ such that $\boldsymbol{w}_i^\top\boldsymbol{R} = \boldsymbol{u}^\top$. Let $\boldsymbol{y} = \boldsymbol{R}^\top\boldsymbol{x}, \boldsymbol{v} = \boldsymbol{R}\boldsymbol{w}_j$, we know $\boldsymbol{y}$ is also $\mathcal{N}(0,1)$ and has has a one-one mapping with $\boldsymbol{x}$. We have

$$\boldsymbol{M}_{ij} = \mathbb{E}_{\boldsymbol{x}}|\boldsymbol{w}_i^\top\boldsymbol{R}\boldsymbol{R}^\top\boldsymbol{x}\boldsymbol{w}_j^\top\boldsymbol{R}\boldsymbol{R}^\top\boldsymbol{x}| = \mathbb{E}_{\boldsymbol{x}}|\boldsymbol{u}^\top\boldsymbol{y}\boldsymbol{v}^\top\boldsymbol{y}| = \mathbb{E}_{\boldsymbol{y}}|\boldsymbol{u}^\top\boldsymbol{y}\boldsymbol{v}^\top\boldsymbol{y}| = \|\boldsymbol{w}_i\|\mathbb{E}_{\boldsymbol{y}}\left|\boldsymbol{y}_1\sum_{k=1}^{d}\boldsymbol{v}_k\boldsymbol{y}_k\right|.$$

The term in the expectation can be written into two terms: $\boldsymbol{y}_1\sum_{k=1}^{d}\boldsymbol{v}_k\boldsymbol{y}_k = \boldsymbol{v}_1\boldsymbol{y}_1^2 + \boldsymbol{y}_1\sum_{k>1}\boldsymbol{v}_k\boldsymbol{y}_{\boldsymbol{k}}$. Recall that $\max\{|a| - |b|, |b| - |a|\} \leq |a+b| \leq |a| + |b|$, we have:

$$\begin{aligned}\mathbb{E}_{\boldsymbol{y}}\left|\boldsymbol{y}_1\sum_{k=1}^{d}\boldsymbol{v}_k\boldsymbol{y}_k\right| &\leq \mathbb{E}_{\boldsymbol{y}}|\boldsymbol{v}_1\boldsymbol{y}_1^2| + \mathbb{E}_{\boldsymbol{y}}|\boldsymbol{y}_1\sum_{k>1}\boldsymbol{v}_k\boldsymbol{y}_k| = |\boldsymbol{v}_1|\mathbb{E}_{\boldsymbol{y}}|\boldsymbol{y}_1^2| + \mathbb{E}_{\boldsymbol{y}}|\boldsymbol{y}_1|\mathbb{E}_{\boldsymbol{y}}|\sum_{k>1}\boldsymbol{v}_k\boldsymbol{y}_k| \\ &= |\boldsymbol{v}_1| + \frac{2}{\pi}\sqrt{\sum_{k>1}\boldsymbol{v}_k^2} \leq |\boldsymbol{v}_1| + \frac{2}{\pi}\|\boldsymbol{v}\|\end{aligned}$$

$$\begin{aligned}\mathbb{E}_{\boldsymbol{y}}\left|\boldsymbol{y}_1\sum_{k=1}^{d}\boldsymbol{v}_k\boldsymbol{y}_k\right| &\geq -\mathbb{E}_{\boldsymbol{y}}|\boldsymbol{v}_1\boldsymbol{y}_1^2| + \mathbb{E}_{\boldsymbol{y}}|\boldsymbol{y}_1\sum_{k>1}\boldsymbol{v}_k\boldsymbol{y}_k| = -|\boldsymbol{v}_1|\mathbb{E}_{\boldsymbol{y}}|\boldsymbol{y}_1^2| + \mathbb{E}_{\boldsymbol{y}}|\boldsymbol{y}_1|\mathbb{E}_{\boldsymbol{y}}|\sum_{k>1}\boldsymbol{v}_k\boldsymbol{y}_k| \\ &= -|\boldsymbol{v}_1| + \frac{2}{\pi}\sqrt{\sum_{k>1}\boldsymbol{v}_k^2} \geq -|\boldsymbol{v}_1| + \frac{2}{\pi}\sqrt{\sum_{k=1}^{d}\boldsymbol{v}_k^2} - \frac{2}{\pi}|\boldsymbol{v}_1| \text{(Pythagorean inequality)}\end{aligned}$$

Thus the expectation term is very close to $\frac{2}{\pi}\|\boldsymbol{v}\|$ (also $\frac{2}{\pi}\|\boldsymbol{w}_j\|$):

$$\left|\mathbb{E}_{\boldsymbol{y}}|\boldsymbol{y}_1\sum_{k=1}^{d}\boldsymbol{v}_k\boldsymbol{y}_k| - \frac{2}{\pi}\|\boldsymbol{v}\|\right| \leq 2|\boldsymbol{v}_1|.$$

We can write $\mathbb{E}_{\boldsymbol{y}}|\boldsymbol{y}_1\sum_{k=1}^{d}\boldsymbol{v}_k\boldsymbol{y}_k|$ as $\frac{2}{\pi}\|\boldsymbol{w}_j\| + \epsilon_j$ where $\epsilon_j^2 \leq c/D$ for some constant c with high probability, and $\boldsymbol{M}_{ij} = \frac{2}{\pi}\|\boldsymbol{w}_i\|\boldsymbol{w}_j\| + \epsilon_{ij}$ where $\epsilon_{ij}^2 \leq cd/D^2$ for some constant c with high probability.

Now we compute the trace of $\mathbb{E}[g(\boldsymbol{x})g(\boldsymbol{x})^\top]$. First we have tr$(\boldsymbol{W}^\top\boldsymbol{W}\boldsymbol{W}^\top\boldsymbol{W}) = \|\boldsymbol{W}^\top\boldsymbol{W}\|_F^2 = \sum_{i=1}^{d}\sum_{j=1}^{d}(\sum_{k=1}^{D}\boldsymbol{w}_{ki}\boldsymbol{w}_{kj})^2$.

The trace of the second term is

$$\begin{aligned}\text{tr}(\boldsymbol{W}^\top\boldsymbol{M}\boldsymbol{W}) &= \sum_{i=1}^{d}\sum_{l=1}^{D}\sum_{k=1}^{D}\boldsymbol{m}_{kl}\boldsymbol{w}_{ki}\boldsymbol{w}_{li} \\ &= \sum_{i=1}^{d}\sum_{k=1}^{D}\boldsymbol{m}_{kk}\boldsymbol{w}_{ki}^2 + \sum_{i=1}^{d}\sum_{k\neq l}\boldsymbol{m}_{kl}\boldsymbol{w}_{ki}\boldsymbol{w}_{li} \\ &= \sum_{i=1}^{d}\sum_{k=1}^{D}\|\boldsymbol{w}_k\|^2\boldsymbol{w}_{ki}^2 + \sum_{i=1}^{d}\sum_{k\neq l}(\frac{2}{\pi}\|\boldsymbol{w}_k\|\|\boldsymbol{w}_l\| + \epsilon_{kl})\boldsymbol{w}_{ki}\boldsymbol{w}_{li} \\ &= \sum_{i=1}^{d}\sum_{k=1}^{D}(\sum_{j=1}^{d}\boldsymbol{w}_{kj}^2)\boldsymbol{w}_{ki}^2 + \sum_{i=1}^{d}\sum_{k\neq l}(\frac{2}{\pi}\|\boldsymbol{w}_k\|\|\boldsymbol{w}_l\| + \epsilon_{kl})\boldsymbol{w}_{ki}\boldsymbol{w}_{li}\end{aligned}$$

Estimate it using the distribution of the weights.

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{W}}\text{tr}(\boldsymbol{W}^\top\boldsymbol{W}\boldsymbol{W}^\top\boldsymbol{W}) &= \mathbb{E}_{\boldsymbol{W}}\sum_{i=1}^{d}\sum_{j=1}^{d}(\sum_{k=1}^{D}\boldsymbol{w}_{ki}\boldsymbol{w}_{kj})^2 \\
&= \mathbb{E}_{\boldsymbol{W}}\sum_{i=1}^{d}(\sum_{k=1}^{D}\boldsymbol{w}_{ki}^2)^2 + \mathbb{E}_{\boldsymbol{W}}\sum_{i\neq j}^{d}(\sum_{k=1}^{D}\boldsymbol{w}_{ki}\boldsymbol{w}_{kj})^2 \\
&= \sum_{i=1}^{d}\left([\mathbb{E}_{\boldsymbol{W}}(\sum_{k=1}^{D}\boldsymbol{w}_{ki}^2)]^2 + \text{Var}_{\boldsymbol{W}}(\sum_{k=1}^{D}\boldsymbol{w}_{ki}^2)]\right) + d(d-1)D\text{Var}_{\boldsymbol{W}}[\boldsymbol{w}_{ki}\boldsymbol{w}_{kj}] \\
&= d(4+4/D) + d(d-1)D\cdot 4/D^2 = 4d(1+d/D)
\end{aligned}$$

Similarly, $\text{tr}(\boldsymbol{W}^\top\boldsymbol{M}\boldsymbol{W}) = 4d^2/D$.

$$\text{tr}(\mathbb{E}[g(\boldsymbol{x})g(\boldsymbol{x})^\top]) = \frac{1}{4}\text{tr}[\mathbb{E}_{\boldsymbol{W}}\text{tr}(\boldsymbol{W}^\top\boldsymbol{W}\boldsymbol{W}^\top\boldsymbol{W}) + \text{tr}(\boldsymbol{W}^\top\boldsymbol{M}\boldsymbol{W})] = d(1+2d/D)$$

Next we consider the trace of $\mathbb{E}[g(\boldsymbol{x})]\mathbb{E}[g(\boldsymbol{x})]^\top$. Since $\boldsymbol{x}\sim\mathcal{N}(0,I_d), \boldsymbol{W}\boldsymbol{x}\sim\mathcal{N}(0,[WW]^\top)$. Thus $\mathbb{E}\text{ReLU}(\boldsymbol{W}\boldsymbol{x}) = \sqrt{\frac{2}{\pi}\text{diag}(\boldsymbol{W}\boldsymbol{W}^\top)}$ and $\mathbb{E}[g(\boldsymbol{x})] = \sqrt{\frac{2}{\pi}\boldsymbol{W}\text{diag}(\boldsymbol{W}\boldsymbol{W}^\top)}$. The the trace of $\mathbb{E}[g(\boldsymbol{x})]\mathbb{E}[g(\boldsymbol{x})]^\top$ is give by:

$$\mathbb{E}\frac{2}{\pi}\text{tr}[\boldsymbol{W}\text{diag}(\boldsymbol{W}\boldsymbol{W}^\top)\boldsymbol{W}^\top] = \mathbb{E}\frac{2}{\pi}\sum_{i=1}^{d}\sum_{k=1}^{D}\boldsymbol{w}_{ik}^2\sum_{t=1}^{d}\boldsymbol{w}_{kt}^2 = \frac{2d^2}{D\pi}$$

. The estimation of $R_f(g)$ is:

$$\mathbb{E}R_f(g) = \frac{1}{d}[\text{tr}(\mathbb{E}[g(\boldsymbol{x})g(\boldsymbol{x})^\top]) - \mathbb{E}[g(\boldsymbol{x})]\mathbb{E}[g(\boldsymbol{x})]^\top] = 1 + 2d/D\cdot(1-\frac{1}{\pi}).$$

Proof of $R_b(g)$: Let $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x}, \boldsymbol{a} = \text{ReLU}(\boldsymbol{z}), \boldsymbol{y} = \boldsymbol{W}^\top\boldsymbol{a}$. Let $\mathbb{I}(x) = (x>0)$ be element-wise for vectors. We have $\frac{\partial\ell}{\partial\boldsymbol{x}} = \boldsymbol{W}^\top\text{diag}(\mathbb{I}[\boldsymbol{z}])\boldsymbol{W}\frac{\partial\ell}{\partial\boldsymbol{y}}$.

Since $\boldsymbol{z}$ (comes from $\boldsymbol{x}$) and $\frac{\partial\ell}{\partial\boldsymbol{y}}$ (with zero mean) are independent, $\mathbb{E}[\frac{\partial\ell}{\partial\boldsymbol{x}}]=0, \text{Cov}[\frac{\partial\ell}{\partial\boldsymbol{x}}]=\mathbb{E}[\frac{\partial\ell}{\partial\boldsymbol{x}}\frac{\partial\ell}{\partial\boldsymbol{x}}^\top]$.

Recall that $\frac{\partial\ell}{\partial\boldsymbol{y}}\sim\mathcal{N}(0,\mathcal{I}_d)$, $\text{Cov}[\frac{\partial\ell}{\partial\boldsymbol{x}}] = \mathbb{E}\left[\boldsymbol{W}^\top\text{diag}(\mathbb{I}[\boldsymbol{z}])\boldsymbol{W}\boldsymbol{W}^\top\text{diag}(\mathbb{I}[\boldsymbol{z}])\boldsymbol{W}\right]$. Let $\delta_i(i\in[D])$ be *i.i.d.* Bernoulli with parameter $p=0.5$ and $\Lambda = \text{diag}(\delta_1,\cdots,\delta_D)$.

Recall $\text{Var}[\frac{\partial\ell}{\partial\boldsymbol{x}}]=\frac{1}{d}\text{trace}(\text{Cov}[\frac{\partial\ell}{\partial\boldsymbol{x}}])$ and $\text{Var}[\frac{\partial\ell}{\partial\boldsymbol{y}}]=1$, we have

$$R_b(g)=\text{Var}[\frac{\partial\ell}{\partial\boldsymbol{x}}]/\text{Var}[\frac{\partial\ell}{\partial\boldsymbol{y}}]=\frac{1}{d}\text{trace}(\mathbb{E}_\Lambda\left[\boldsymbol{W}^\top\Lambda\boldsymbol{W}\boldsymbol{W}^\top\Lambda\boldsymbol{W}\right]).$$

Let w_{ij} be element i,j of $\boldsymbol{W}$, we have:

$$\begin{aligned}
R_b(f) &= \frac{1}{d}\mathbb{E}_\Lambda\sum_{s=1}^{d}\sum_{t=1}^{d}\sum_{i=1}^{D}\sum_{j=1}^{D}\delta_i\delta_j w_{si}w_{ti}w_{tj}w_{sj} \\
&= \frac{1}{2d}\sum_{s=1}^{d}\sum_{t=1}^{d}\sum_{i=1}^{D}w_{si}^2w_{ti}^2 + \frac{1}{4d}\sum_{s=1}^{d}\sum_{t=1}^{d}\sum_{i\neq j}w_{si}w_{ti}w_{tj}w_{sj}.
\end{aligned}$$

Now we estimate the range of $R_b(g)$ given the initialization $w_{ij}\sim\mathcal{N}(0,2/D)$. This is similar to that of $R_f(g)$ and we directly give the results: $\mathbb{E}\left[R_b(f)\right] = 1+\frac{2d}{D}+\frac{3}{D}, \text{Var}\left[R_b(f)\right] < \frac{8}{D^2}$.