REVISITING EXPLODING GRADIENT: A GHOST THAT NEVER LEAVES

Anonymous authors
Paper under double-blind review

ABSTRACT

The exploding gradient problem was 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 eve 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.

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 & Bengio, 2010; He et al., 2015) and most importantly, adding normalization layers (Ioffe & Szegedy, 2015) to the network. People generally believe that the vanishing/exploding gradient has been largely addressed, especially with the help of normalization layers. For example, in the residual network (He et al., 2016) paper, authors believed the gradient problem is not the cause of the optimization difficulty of deep plain network.

However, we check the gradients of layer weights in a plain network (a sequential network with batch normalization (Ioffe & Szegedy, 2015), but without any kinds of skip connections/shortcuts), and the gradient of the first layer increases exponentially as the network goes deep. Figure 1 (left) shows this. See Section 2 for details.

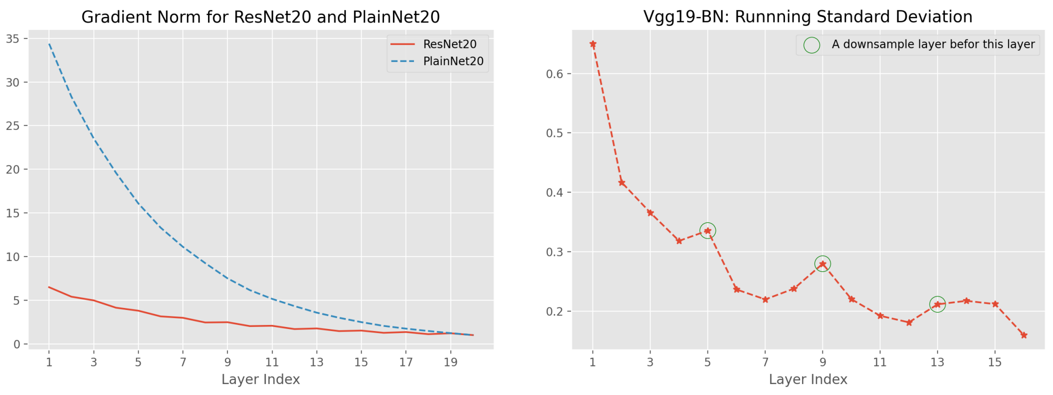

Figure 1: **Left**: The gradient norm of layer weights against layer indexes for a PlainNet20 and a ResNet20 with BatchNorm and standard initialization. **Right**: BatchNorm layers' running variance (square rooted) against layer indexes from VGG19 trained checkpoints.

Existing studies cannot explain this phenomenon: 1) the gradients in the plain network explode regardless of the activation choice, e.g., ReLU (Nair & Hinton, 2010) or SELU (Klambauer et al., 2017); 2) all weights in the network are properly initialized (He et al. (2015) or Glorot & Bengio (2010)); 3) the batch normalization layer is added after every weight layer, thus the forward of the network is 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. An intuitive explanation is the different variance propagation behaviour of activations in forward and backward. For example $\mathrm{ReLU}(x)$ decrease the variance of a Gaussian input by $\frac{1}{2}(1-\frac{1}{\pi})$, but during back propagation, the variance is decrease by $1/2$ when passing ReLU. As this difference accumulates, gradient exploding happens.

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 $\mathrm{Var}[\boldsymbol{x}]$ be the average element variance: $\mathrm{Var}[\boldsymbol{x}] = \dfrac{1}{d}\displaystyle\sum_i \mathrm{Var}[\boldsymbol{x}_i]$. 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. 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): $\boldsymbol{W}^{(k)}_{ij} \sim \mathcal{N}(0, 2/d_k)$, 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 ResNet20 is a skip connection. We follow the Basic Block design (He et al., 2016) and add the skip connection every two layers:

$$k=0,2,\cdots 18,\ \ \boldsymbol{x}^{(k+1)} = \sigma(\mathcal{B}^{(k)}(\boldsymbol{W}^{(k)}\boldsymbol{x}^{(k)})),\quad \boldsymbol{x}^{(k+2)} = \sigma(\mathcal{B}^{(k)}(\boldsymbol{W}^{(k)}\boldsymbol{x}^{(k+1)}) + \boldsymbol{x}^{(k)}).$$

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 $\dfrac{\partial \ell}{\partial \boldsymbol{x}^{(20)}} \sim \mathcal{N}(0, \dfrac{1}{d}\mathcal{I}_d)$. 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. We also provide evidence from a pre-trained weights as shown in Figure 1 (right). See Appendix A for details and and C for more evidences.

3 Theory of Exploding Gradients

Let $\boldsymbol{y} = \pi(\boldsymbol{x})$ be an arbitrary operator in the network. Let $\dfrac{\partial \ell}{\partial \boldsymbol{x}}$ and $\dfrac{\partial \ell}{\partial \boldsymbol{y}}$ be the derivative of the network loss ℓ with respect to the operator input and output: $\dfrac{\partial \ell}{\partial \boldsymbol{x}} = \pi'(\boldsymbol{x})\dfrac{\partial \ell}{\partial \boldsymbol{y}}$. Define the forward and backward scaling ratio $R_f(\pi)$ and $R_b(\pi)$ as

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

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

Proposition B.4 and B.8 shows that the two ratios are roughly equal for carefully initialized linear layers and batch normalization layers. Proposition B.12 shows that $R_f(\pi) \leq R_b(\pi)$ for nonlinear operators, and the equation holds if and only if π is linear. As the result, Corollary B.15 shows that for a network f, $R_b(f)/R_f(f) = \Omega(c^L)$ where $c > 1$ is determined by the nonlinear activation and L is the number of activation layers. Thus gradient exploding would happen in plain networks. See Appendix B for proof details and how residual networks could solve this problem.

4 Conclusion

This paper argues that the long-held idea that batch normalization solves the gradient explosion problem while skip connections is not working on this is not correct and provide empirical and mathematical evidences towards it.

URM Statement

Please iThe authors acknowledge that at least one key author of this work meets the URM criteria of ICLR 2024 Tiny Papers Track.

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*, pp. 342–350. PMLR, 2017.

Herman Chernoff. A note on an inequality involving the normal distribution. *The Annals of Probability*, pp. 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.

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.

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.

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)*, pp. 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*, pp. 972–981, 2017.

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

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.

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

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*, pp. 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.

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 Circumstantial evidence of exploding gradient

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., 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 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 & 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: $\sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(\text{running\_variance})_i}$.

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.

B 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 B.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 B.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 & 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.

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

B.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 B.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 B.4. *Under Assumption B.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.*

Proof of Proposition B.4 Note that the entries of $\boldsymbol{x}$ are independent and identically distributed, let Var$[\boldsymbol{x}_i] = \sigma^2$ for all i. Then Var$[\boldsymbol{x}] = \sigma^2$.

$$\text{Var}[\boldsymbol{y}_i] = \text{Var}[\sum_j \boldsymbol{W}_{ij}\boldsymbol{x}_j] = \sum_j \boldsymbol{W}_{ij}^2 \text{Var}[\boldsymbol{x}_j] = \sigma^2 \sum_j \boldsymbol{W}_{ij}^2.$$

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

Remark B.5. Assumption B.3 is widely used in many existing studies, but not likely to be true in generally. However, Proposition B.4 matches the real case quite well, e.g., He initialization. We also verified this in Figure 2.

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 Var$[\frac{\partial \ell}{\partial x}]$ and 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 Var$[\frac{\partial \ell}{\partial x}]$:

Assumption B.6. For any neuron in the network, the forward signal x and the backward signal $\frac{\partial \ell}{\partial x}$ are independent.

Corollary B.7. *Under Assumption B.6, any neuron before the last BatchNorm layer in a network has zero expectation derivative, i.e., $\mathbb{E}\frac{\partial \ell}{\partial x} = 0$.*

Proof of Corollary B.7 Let $\{x_i\}_{i=1}^B$ is a batch of data with batch size B and y_i is the normalized results:

$$\mu_x = \frac{1}{B}\sum_{i=1}^B x_i, \quad \sigma_x = \sqrt{\frac{1}{B}\sum_{i=1}^B (x_i - \mu_x)}, \quad y_i = \frac{x_i - \mu_x}{\sigma_x}$$

The backward derivative for the batch normalization layer is: $\frac{\partial \ell}{\partial x_i} = \frac{1}{\sigma_x}[\frac{\partial \ell}{\partial y_i} - \frac{y_i}{B}\sum_{j=1}^B \frac{\partial \ell}{\partial y_j} y_j - \frac{1}{B}\sum_{j=1}^B \frac{\partial \ell}{\partial y_j}]$. Refer to Ioffe & Szegedy (2015) for the proof of this result. Note that $\sum_{i=1}^B y_i = 0$, we have:

$$
\begin{aligned}
\sum_{i=1}^{B} \frac{\partial \ell}{\partial x_i} &= \frac{1}{\sigma_x}[\sum_{i=1}^{B} \frac{\partial \ell}{\partial y_i} - \sum_{i=1}^{B} \frac{y_i}{B} \sum_{j=1}^{B} \frac{\partial \ell}{\partial y_j} y_j - \sum_{i=1}^{B} \frac{1}{B} \sum_{j=1}^{B} \frac{\partial \ell}{\partial y_j}] \\
&= \frac{1}{\sigma_x}[\sum_{i=1}^{B} \frac{\partial \ell}{\partial y_i} - \frac{1}{B} \left(\sum_{i=1}^{B} y_i \right) \left(\sum_{j=1}^{B} \frac{\partial \ell}{\partial y_j} y_j \right) - \sum_{j=1}^{B} \frac{\partial \ell}{\partial y_j}] \\
&= \frac{1}{\sigma_x}[\sum_{i=1}^{B} \frac{\partial \ell}{\partial y_i} - \sum_{j=1}^{B} \frac{\partial \ell}{\partial y_j}] = 0,
\end{aligned}
$$

which means the sample mean of the batch normalization layer input derivative is 0, thus $\mathbb{E} \frac{\partial \ell}{\partial x} = 0$ 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 $\mathbb{E}\left[\frac{\partial \ell}{\partial y}\right] = 0$ 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.

Proposition B.8. *Under Assumption B.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 $\mathbb{E}z = 1, Var[z] = 2$.*

Proof of Proposition B.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 $\mathbb{E}z = 1, \text{Var}[z] = 2$.

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 B.9. Assumption B.6 is widely used (Yang & 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 B.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 Var$[g(x)]$ exists.

For activation g, the forward scaling ratio is $R_f(g) =$ Var$[g(x)]/$Var$[x]$, and the backward scaling ratio is $R_b(g) = \text{Var}[\frac{\partial \ell}{\partial x}]/\text{Var}[\frac{\partial \ell}{\partial y}]$. Recall assumption B.6 and Corollary B.7, we have:

$$\begin{aligned}\text{Var}[\frac{\partial \ell}{\partial x}] &= \text{Var}[\frac{\partial \ell}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}x}] = \mathbb{E}\left[(\frac{\partial \ell}{\partial y})^2[g'(x)]^2\right] \quad \text{(because} \frac{\partial \ell}{\partial x} \text{has a zero mean)} \\ &= \mathbb{E}\left[\frac{\partial \ell}{\partial y}^2\right]\mathbb{E}[g'(x)^2] = \text{Var}[\frac{\partial \ell}{\partial y}]\mathbb{E}[g'(x)^2].\end{aligned}$$

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 B.11. The input to the activation is Gaussian.

Many existing studies (Klambauer et al., 2017) make this assumption.

Proposition B.12. *Under Assumption B.10 and B.11, Var*$[g(x)]/$*Var*$[x] \leq \mathbb{E}[g'(x)^2]$. *The equality holds if and only if* $g(x) = ax + b$ *for some* a *and* b.

Please refer to Chernoff (1981) for the proof of Proposition B.12. With proposition B.12, the backward scaling ratio is larger than the forward scaling ratio for nonlinear activations. Specifically, for ReLU activation $\sigma(x) = \max(x, 0), R_f(\sigma) = \frac{1}{2}(1 - \frac{1}{\pi}), R_g(\sigma) = \frac{1}{2}$.

Some people may find the Gaussian assumption is too strong. For ReLU activation, we can use weaker assumptions:

Assumption B.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 Var$[x] = 1$ for ReLU since $\sigma(x) = s \cdot \sigma(x/s)$ where s denotes the standard deviation of x and Var$[x/s] = 1$.

Proposition B.14. *Under Assumption B.13 and Var*$[x] = 1$, *Var*$[\sigma(x)] \leq \mathbb{E}[\sigma'(x)^2] - \frac{1}{8}[\mathbb{E}(|x|)]^2$.

Proof of Proposition B.14 Since the distribution of x is symmetric centered at zero, we have

- $\mathbb{E}[\sigma^2(x)] = \mathbb{E}[x^2|x > 0] = \frac{1}{2}\mathbb{E}[x^2] = \frac{1}{2}$
- $\mathbb{E}[\sigma'(x)^2] = \mathbb{E}[1|x > 0] = P(x > 0) = \frac{1}{2}$
- $\mathbb{E}[\sigma(x)] = \mathbb{E}[|x|\big|x > 0] = \frac{1}{2}\mathbb{E}(|x|)$

Thus

$$\begin{aligned}\text{Var}[\sigma(x)] &= \mathbb{E}[\sigma^2(x)] - (\mathbb{E}[\sigma(x)])^2 \\ &= \frac{1}{2} - (\frac{1}{2}\mathbb{E}(|x|))^2 = \frac{1}{2} - \frac{1}{4}[\mathbb{E}(|x|)]^2 \\ &= \mathbb{E}[\sigma'(x)^2] - \frac{1}{4}[\mathbb{E}(|x|)]^2 \\ &\leq \mathbb{E}[\sigma'(x)^2] - \frac{1}{8}[\mathbb{E}(|x|)]^2\end{aligned}$$

Proposition B.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 $\text{Var}[\sigma(x)]$ 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, $R_b(g) \geq R_f(g) + \epsilon$ for some $\epsilon = \frac{1}{8}[\mathbb{E}(|x|)]^2 > 0$. Note that the two ratios are positive and bounded for a single operator in a real network, it is equivalent to say $R_b(g) \geq c \cdot R_f(g)$ for some $c > 1$.

Corollary B.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 B.16. Although the two ratios are not equal for fully-connected operators as in proposition B.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. 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.

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 2 how close the gradient norm curve matches the exponential function. Although some of our assumptions may not be the real case, our conclusions still match the read case quite well.

B.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 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

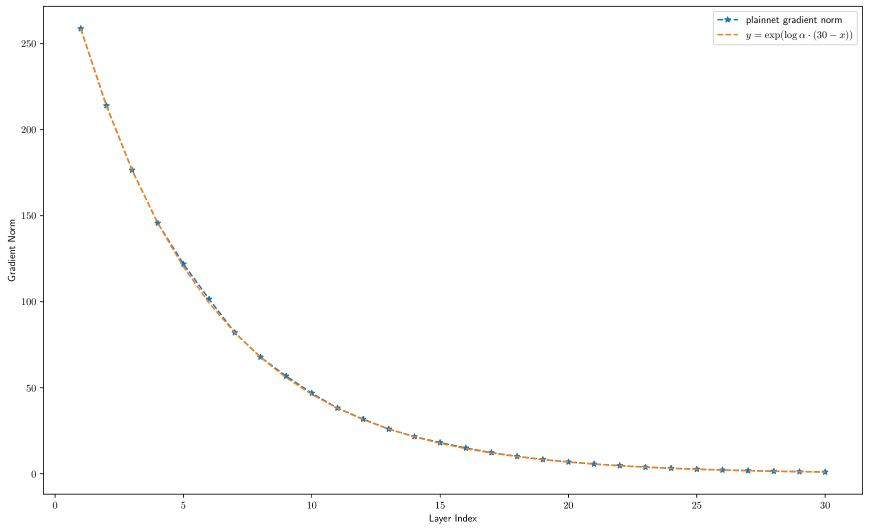

Figure 2: The actual gradient norm and the predicting gradient norm.

network. The $1/\sqrt{L}$ initialization use $1/\sqrt{L}$ to multiply the residual branch[2] where L is the number of residual blocks: $\boldsymbol{y} = \boldsymbol{x} + \frac{1}{\sqrt{L}}\mathcal{F}(\boldsymbol{x})$.

Proposition B.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: $R_f(\mathcal{R}) = 1 + \frac{1}{L}R_f(\mathcal{F}), \quad R_b(\mathcal{R}) = 1 + \frac{1}{L}R_b(\mathcal{F})$.*

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.

C 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 [3] or Tanh[4]. 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 3, 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.

[2]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.

[3]`https://pytorch.org/docs/stable/generated/torch.nn.SELU.html`

[4]`https://pytorch.org/docs/stable/generated/torch.nn.Tanh.html`

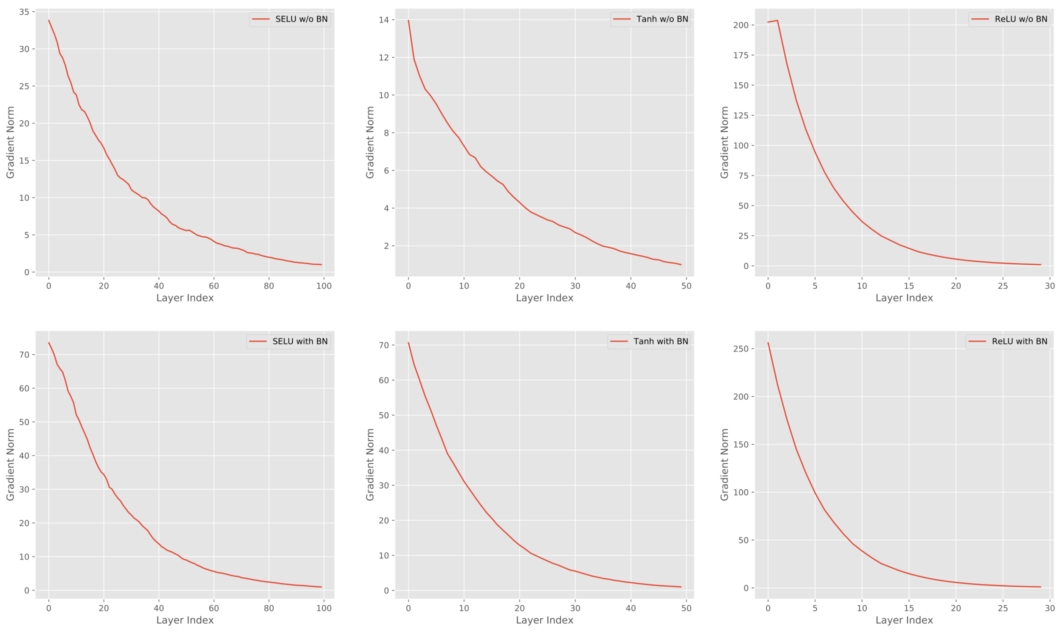

Figure 3: The gradient norm against layer index for SELU, Tanh, and ReLU with/without batch normalization layers. Different activations have different exponential rate.

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 3 shows that it may not be the case. However, batch normalization layers indeed increase the exponential rates. This is because batch normalization layers rescale the hidden distributions to the most non-linear regime of the activations.