Mitigating Depth-Induced Challenges via Close-to-linear Neural Networks

Li Ke [1] **Libutti Simone** [1] **Nonino Lara** [1] **Zaera Alvaro** [1]

Abstract

In the context of deep neural networks, we offer an overview of two approaches that have been proposed to tackle the issue of degenerate gradients. The first one consists of a so-called "shaping" of the ReLU activation function, while the second employs the well-known *residual connections* with scaled residual branches. We found that these remedies are indeed successful in making simple but very deep architectures trainable through the role of the extra parameters they introduce in the model. Also, we present the advantages of setting these parameters as trainable rather than constant.

1. Introduction

The problem of exploding and vanishing gradients is one of the main setbacks to the efficiency of deep neural networks. During training, accuracy rapidly deteriorates as learning becomes either extremely slow or the magnitude of the gradients makes computations unfeasible. Analyzing how this issue manifests into simple architectures can yield useful insights into tackling it when dealing with very complex models.

In our attempts to study a solution for this problem, we have been looking at two different suggested techniques that employ simple modifications to the structure of the network, ultimately achieving the goal of making it "more linear". The main idea is in fact balancing the sources of non-linearity in the network while at the same time preserving expressiveness.

1.1. Tailored Activation Transformations (TAT)

We read in (Zhang et al., 2022) that an efficient way of training deep vanilla architectures consists of adding an extra degree of freedom to the typical ReLU activation, where $\text{ReLU}(x) = \max\{x, 0\}$, to obtain what they refer to as Leaky ReLU. Leaky ReLU has been suggested for the first time in (Maas, 2013) to create some light dependency on possible negative inputs, and takes the form

$$\text{LReLU}(x) = \max\{x, 0\} + \alpha \min\{x, 0\}.$$

In (Zhang et al., 2022), however, this function is scaled by a factor $\sqrt{\frac{2}{1+\alpha^2}}$. This decision stems from the Q/C map analysis contained in (Poole et al., 2016). In brief, a feedforward network characterized by input propagation $x^{l+1} = \phi\left(W_l x^l + b_l\right)$ with normally distributed starting weights can be viewed at initialization as a random feature model $f_\theta^l(x) := x^l$ at each layer l. As the width of each layer tends to infinity this induces a computable layer-by-layer deterministic kernel with matrices Σ^l. The aforementioned Q and C maps are defined layer-by-layer and are recursions for, respectively, the diagonal and off-diagonal entries of these matrices. These concepts can be extended to convolutions as well. Now, by scaling the Leaky ReLU by the factor described above, we have obtained a *tailored* ReLU (TReLU) defined as

$$\tilde{\phi}_\alpha(x) = \sqrt{\frac{2}{1+\alpha^2}}\left(\max\{x, 0\} + \alpha \min\{x, 0\}\right) \qquad (1)$$

which presents desirable closed-form expressions and properties for its Q and C maps (see equation 9 of (Zhang et al., 2022)).

Our contribution to this topic relates to the choice of the value of α, which we have treated as a learnable parameter. We have found empirically that this approach achieves better results on deep networks for image classification, with an efficient optimizer, than the static initialization method described in the paper, which we will go over in Subsection 2.1.

1.2. Residual Branch Scaling

The usage of residual branches (He et al., 2015) to stabilize deep convolutional architectures is immensely popular in deep learning. A residual branch in a network is a mapping of the form $x \mapsto x + \mathcal{F}(x)$, where $\mathcal{F}$ is a regular stack of network layers. This technique aims to overcome the problem of vanishing gradients by driving each layer output

[1]D-INFK, ETH Zürich, Switzerland. Correspondence to: Ke Li <keli@student.ethz.ch>, Simone Libutti <slibutti@student.ethz.ch>, Lara Nonino <lnonino@student.ethz.ch>, Alvaro Zaera <azaera@student.ethz.ch>.

Student Project of the Deep Learning Course at ETH Zürich, Switzerland, 2023.

towards linearity with respect to the input. In addition, the introduction of depth-dependent scaling factors to the residual connections has been proved in (Hayou et al., 2021) to be beneficial in stabilizing the gradient.

Similarly to the TAT case, we contribute to this aspect with the addition of an extra trainable parameter on top of the depth-related one, analyzing the effects of adding more freedom to the balancing of the non-linearity sources. It has positive consequences such as slightly better accuracy results and an automatic regularization of the block of layers, resulting in an automatic selection of the depth of the network.

2. Models and Methods

In the following subsections, we will give the details of our contributions to the topics of Section 1 and their implementations. The results obtained with these procedures will be gone over in Section 3 along with meaningful baseline comparisons.

2.1. TAT

The approach described in (Zhang et al., 2022) to choose the α parameter for $\tilde{\phi}_\alpha$ in the context of MLPs aims to achieve the condition $\mathcal{C}_f(0) = \eta$. Here $\mathcal{C}_f$ is a function defined by composing the C map of the network f as many times as the number of layers in it (and can thus be considered a "global" C map as opposed to the layer-by-layer ones) and $0 \leq \eta \leq 1$ is a hyperparameter. By Theorem 1 stated in the paper, η balances the trade-off between a well-behaved $\mathcal{C}_f$ and the linearity of $\tilde{\phi}_\alpha$, and we set it to 0.9 as suggested.

To follow this schema we have plotted the graph of $\mathcal{C}_f(0) - \eta$ in the case of 100 layers as a function of α (Figure 1), noting that it takes value 0 twice in the interval $[0, 2.5]$ at points α_1 and α_2. Considering that the zeros of this function are exactly the values of α for which $\mathcal{C}_f(0) = \eta$, we used `scipy`'s `fsolve` routine to numerically compute said zeros with initial guesses within this range. The choice of plotting for a depth of 100 is indicative: the function rapidly converges to this shape even for lower depths.

2.1.1. Architectural and Training Details

Using $\alpha = \alpha_1$, we have trained feedforward classifiers with depths of 50, 100 and 200 and a constant width of 100, and then trained the same networks by letting α become a trainable parameter.

Training was carried out for 5 epochs on the MNIST digit dataset after appropriate image vectorization with a batch size of 256 samples. To clarify on the loss function optimization method, we have used ADAM with ad-hoc learning rates as specified in Table 1 with a 10-class cross-entropy

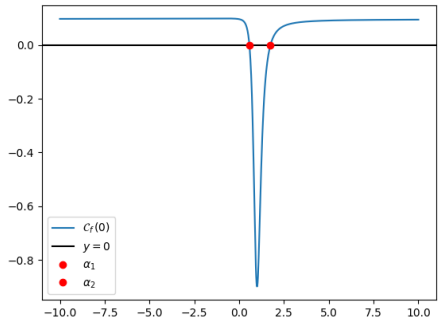


Figure 1. Graph of $\mathcal{C}_f(0) - \eta$ as a function of α for a 100 layer MLP, with zeros highlighted

function. It is important to point out that in all experiments it has been deemed necessary to make use of Batch Normalization as described in (Ioffe & Szegedy, 2015), but the quality of the results is not infringed. Indeed, using Batch Normalization without TATs leads to models that are either untrainable or perform notably worse than their TAT counterparts. Concerning parameter initialization, we found that the PyTorch default $\mathcal{U}[-\frac{1}{\sqrt{d_{in}}}, \frac{1}{\sqrt{d_{in}}}]$ for both weights and biases works better, for our architectures, than the normal and constant-0 initializations suggested in (Zhang et al., 2022). When trainable, α is initialized to 1.0.

Depth	ADAM Learning Rate	
	Base Parameters	α
50	10^{-3}	10^{-2}
100	10^{-3}	10^{-2}
200	10^{-4}	10^{-2}

Table 1. ADAM Learning Rates for 50, 100 and 200 layer MLPs for TAT testing

The specifics of the model just described give rise to the simple architecture represented in Figure 2.

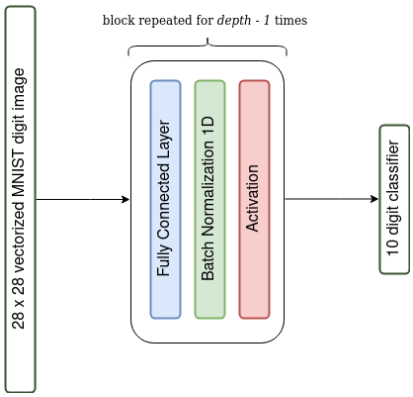


Figure 2. MLP architecture for TAT testing

2.2. Residual Connections

The experiments using residual branch scaling consisted of the application to every intermediate layer l of the mapping given by

$$x^{l+1} = x^l + \frac{\beta}{\sqrt{L}}\mathcal{F}(\text{ReLU}(x^l)),$$

where $\mathcal{F}$ is the network layer, β is a parameter and L is the actual network depth. The goal of the $\sqrt{L}$ parameter is to add a depth-related scaling factor to ensure more stability by enforcing mappings closer to the identity when the depth grows. In addition, we add the β parameter to refine the scaling. In our experiments, we compared the performance using constant and trainable values of β. For the trainable case, we compare the effect of using a global β that is trained for all the layers and using layer-individual ones, giving more independence to the scaling of each block of the network.

2.2.1. Architectural and Training Details

The model is almost identical to Figure 2, the differences being the addition of a residual connection for each block and the omission of Batch Normalization, which turned out to be unnecessary to achieve meaningful results. The values of β used were 1, 0.5, 0.1, 0.01, and 0.001. The training procedure is also very similar to the description of Subsubsection 2.1.1, but the learning rate for Adam was kept 10^{-3} for all depths. Also, some experiments have been run for more than 5 epochs, as described in Section 3.

Extra experiments were also performed by replacing the linear layers with convolutional layers with kernels of size 3×3 and 12 input and output channels, in order to verify if the results obtained for the MLPs held with a different kind of main layer.

3. Results

In this section we go over the outcomes of our experiments. Note that we only included an example plot for what concerns the applications of our methods to CNNs (Subsection 3.3) to avoid clutter and preserve space for our descriptions. The remaining graphical representations of our results can be viewed in the Appendix [A].

3.1. TAT for MLPs

To evaluate the performance of the TAT method we will consider as a baseline our network with standard ReLU activations. We have computed the weight and bias gradient norms (intended as $\|\text{vec}(\partial\mathcal{L}/\partial\mathbf{W})\|_2$ and $\|\text{vec}(\partial\mathcal{L}/\partial\mathbf{b})\|_2$) of the 50, 100 and 200 layer ReLU MLPs over the training steps, and their test accuracy over the epochs [Figure A.1]. The networks start with untreatable gradient magnitudes, and despite exponential decay of the gradient norms the accuracy remains poor for the 50-layer case and there is virtually no learning for the 100 and 200-layer cases, where the accuracy never surpasses that of a random guess.

We then computed the gradient norms and accuracy for the case of TReLU with static α, chosen as the lesser value of those obtained following the process described in Subsection 2.1 [Figure A.2]. The networks achieve at all depths an accuracy superior to 90% and the gradients of the parameters are under control at all epochs. Accuracy also increases at a much greater speed in the 50-layer case.

Finally, we ran the experiments using TReLU with trainable α [Figure A.3]. Not only does this reduce the gradient norms even further (albeit introducing slightly more variance in the bias ones) but it also improves accuracy at each depth, with the 200-layer network reaching 96%. In each case, it is interesting to note that the values of α at the end of the training routine stabilize around the interval $[-1.5, 1.5]$ with few outliers.

3.2. Residual Connections for MLPs

Residual connections proved to be clearly effective in solving the baseline issues under diverse settings. The method works well for all values of β tested, with test accuracy reaching 96% in the 100-layer case [Figure A.4]. The best results were obtained with $\beta = 0.5$, so this is the value that was fixed for the following experiments. Consult Figure A.5 for the results obtained with this value for all depths considered.

With initial β set to 0.5, we compared the global and layer-individual training settings for the parameter. As a sample we analyzed the performance for a 100-layer network, which is slightly better for the layer-individual case but does not bear any significant difference with the global one. [Figure A.6]

Given the similar results using trainable and constant β, we repeated the experiment for 100 epochs for all depths, in order to analyze other consequences of using layer-individual β parameters. For the three considered depths, training showed no signs of depth-related issues except for some vanishing gradients in the last epochs, when performance could not be improved. The accuracy achieved was above 98%. [Figure A.7]

We have observed an interesting behavior in the evolution of the β parameters. During training, for many layers, the respective β decayed towards zero (e.g. layers 38 and 95 in Figure A.7). We decided to test the performance of the model after replacing the blocks with a low β with an identity mapping, letting the input flow through unaltered. The threshold used to consider a β low is dynamically set as the 10% of the maximum β present. For depths of 50, 100 and

200, the number of layers dropped was respectively 24, 75 and 73. The threshold for these layers was between 0.11 and 0.14.

The test accuracy ignoring these layers was identical, with an even lower test loss. It could be said that the training process automatically regularizes the β parameters, progressively ignoring unnecessary blocks of the network. This is a very interesting result, as it seems that the addition of layer-individual trainable β parameters allows the network to automatically select an ideal depth. For the 50 and 100-layer cases, in fact, the automatic depth selected by the network was very similar ($50-24 = 26$ and $100-75 = 25$). On the downside, the 200-layer case shows that this automatic depth selection does not ideally scale to larger depths, where the automatic depth selected was $200 - 73 = 127$. In any case, this regularization ability is impressive, given the fact that this behavior is not explicitly encouraged in the loss function.

3.3. Results using CNNs

To verify our results for CNNs we considered as a baseline a convolutional network of depth 100 using ReLU, and tested our best settings from Subsection 3.1 (TReLU with trainable α) and Subsection 3.2 (residual connections with trainable per-layer β, initialized to 0.5). Results can be found in Figure 3.

The baseline architecture presented an instant problem of vanishing gradients, which was efficiently solved by our approaches. The TReLU architecture reached the highest accuracy, being the only experiment that surpassed the 99% accuracy, while the residual method reached a maximum of 98.7%. However, the residual method showed faster convergence and training time and more stable training in terms of accuracy. The automatic regularization of the β parameters observed in Subsection 3.2 was also present in the convolutional case, where, in the final layers, the same accuracy was obtained by removing 53 layers with $\beta <= 0.4$ (resulting in an "effective depth" of $100 - 53 = 47$ layers).

4. Discussion

TATs and residual branch scaling have shown to be effective in blunting the effects of exploding and vanishing gradients in deep architectures, and our contributions have brought significant benefits. The role of our learnable TReLU in making de facto untrainable networks reach high accuracy could be refined and employed in modern architectures, and the same holds for the application of the scaled residual branches, which are more stable and make the use of batch normalization unnecessary. In addition, the automatic regularization that we have witnessed during training, which highlights what can be considered a form of "effective depth" of the network, can be meaningful in the context of larger models, since if developed it could lead to the identification of unnecessary complexity.

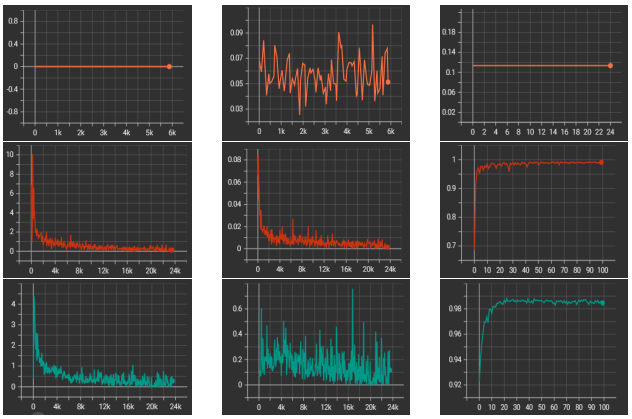


Figure 3. Parameter gradient norms and accuracy for CNN with 100 layers using baseline network with ReLU (25 epochs), trainable TReLU activations with BatchNorm (100 epochs) and residual connections with layer-individual trainable β (100 epochs).

It must be noted that this approach has been mainly empirical and the way in which it could be generalized to different architectures and tasks remains object of speculation, as well as the exact steps that should be taken to allow both methods to flourish in a more elaborate setting. We must also note that we have not obtained relevant information regarding the problem of rank collapse, and as a consequence no investigation could be made on possible ways to mitigate it, this due to the fact that the issue did not manifest in the analyzed architectures.

5. Summary

Throughout this work, we have empirically analyzed the effect that TATs and residual branch scaling have on mitigating the negative effects to which the gradients are subject in deep networks. For our experiments we have considered as a baseline feedforward classifiers with Batch Normalization and ReLU activation function, and convolutional classifiers without Batch Normalization and ReLU activation function. The effectiveness of both methods has been verified on these architectures, and the residual connection method has been considered to be more robust for the task. In addition, we have shown the benefits that can be obtained with by including trainable scaling parameters to both methods, which improve the performance and lead to an automatic regularization that results in an ideal depth selection in the residual connections case.

References

Hayou, S., Clerico, E., He, B., Deligiannidis, G., Doucet, A., and Rousseau, J. Stable resnet, 2021.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition, 2015.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Bach, F. and Blei, D. (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/ioffe15.html.

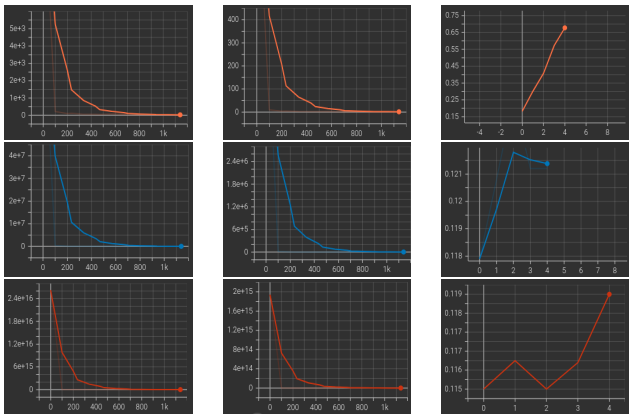
Maas, A. L. Rectifier nonlinearities improve neural network acoustic models. 2013. URL https://api.semanticscholar.org/CorpusID:16489696.

Poole, B., Lahiri, S., Raghu, M., Sohl-Dickstein, J., and Ganguli, S. Exponential expressivity in deep neural networks through transient chaos, 2016.

Zhang, G., Botev, A., and Martens, J. Deep learning without shortcuts: Shaping the kernel with tailored rectifiers, 2022.

A. Appendix: Graphical Results for MLP Experiments

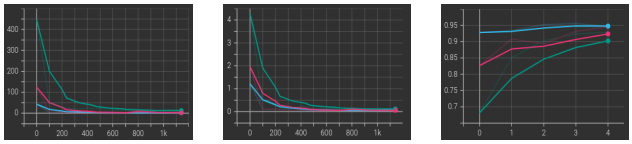
A.1. Baseline



Weight gradients | Bias gradients | Test accuracy

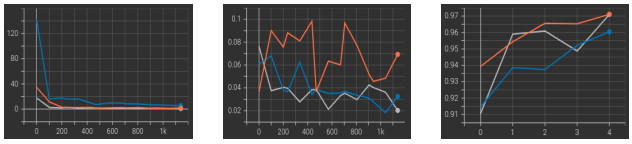
Figure A.1. Parameter gradient norms and accuracy for baseline depths 50, 100 and 200 with ReLU, with separate graphs for each depth to avoid scaling issues.

A.2. TAT



Weight gradients | Bias gradients | Test accuracy

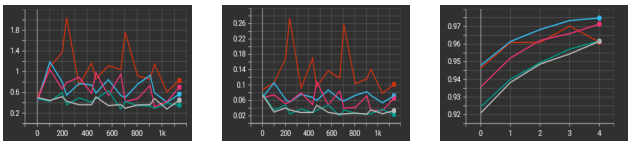
Figure A.2. Parameter gradient norms and accuracy for baseline depths 50, 100 and 200 with static TReLU, setting $\alpha = \alpha_1$.



Weight gradients | Bias gradients | Test accuracy

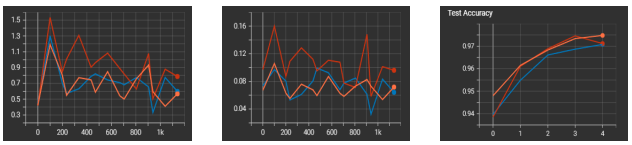
Figure A.3. Parameter gradient norms and accuracy for baseline depths 50, 100 and 200 with trainable TReLU.

A.3. Residual Connections



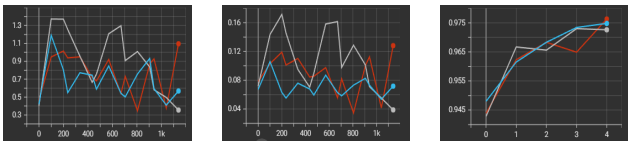
Weight gradients | Bias gradients | Test accuracy

Figure A.4. Parameter gradient norms and accuracy for MLP with 100 layers and residual connections with constant β =0.001, 0.01, 0.1, 0.5 and 1.0.



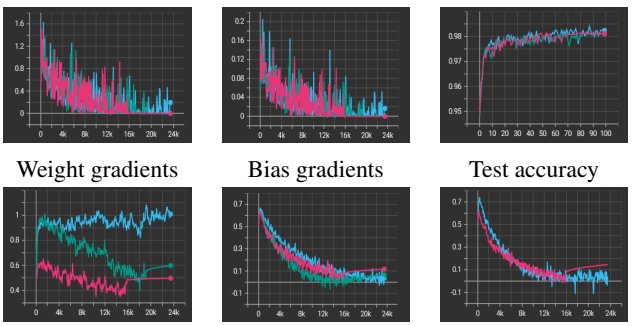
Weight gradients | Bias gradients | Test accuracy

Figure A.5. Parameter gradient norms and accuracy for MLP with 50, 100 and 200 layers using residual connections with constant $\beta = 0.5$.



Weight gradients | Bias gradients | Test accuracy

Figure A.6. Parameter gradient norms and accuracy for MLP with 100 layers and residual connections with initial $\beta = 0.5$. (Constant, Trainable Global and Trainable Layer-Individual).



Weight gradients | Bias gradients | Test accuracy

β Parameter of Hidden Layer #4 | β Parameter of Hidden Layer #38 | β Parameter of Hidden Layer #95

Figure A.7. Parameter gradient norms and accuracy for MLP with 50, 100 and 200 layers for 100 epochs. Architectures use residual connections with trainable layer-individual β with initial value set to 0.5. The second row shows the evolution of the β parameters of three different layers.