# Vanishing Gradients: alternative solutions to ReLU-like functions and Initialization

Alberto Novati

July 10, 2022

## 1 Introduction

The issue of the Vanishing Gradient and its resolution have been critical factors for the development of Deep Learning. Nowadays, one just inserts ReLU-like activation functions in the hidden layers of the network and chooses a default, optimized, weight's initialization and it's good to go for most applications.
The problem is clear: activation functions which saturate contribute negatively to the back-propagation process of gradients' computation, making gradients shrink in the deep layers, thus making the learning algorithm stuck in sub-optimal points of the parameters' space, hindering the whole learning process. This results in a very slow convergence, or a full plateau of the training accuracy.

Even though the aforementioned solutions found in recent years seem to be effective, the theoretical idea of finding a solution to the Vanishing Gradients problem without getting rid of the $\sigma$-like functions seemed intriguing to explore and might even bring some benefits on the table compared to the standard ReLU + Optimized initialization procedure.
In this document, the paper from Ven and Lederer *"Regularization and Reparametrization: Avoid Vanishing Gradients in Sigmoid-Type Networks"* is explored and some computational experiments are reproduced, with slight variations, in order to check whether the solutions proposed are valid substitutions to the standard ways of proceeding.

## 2 The theory: two Vanishing Gradients regimes

The issue of the Vanishing Gradients problems stems directly from the neural networks' structure and the way each layer influences directly the next one, and subsequently all the successive layers until the output is reached.
This "influence" is what enables the application of the chain-rule to the network and is at the core of the back-propagation equations, which is the mathematical procedure that allows the computation of the loss-function gradients relative to each parameter of the network.
Back-Propagation theory for fully-connected feed-forward neural networks tell us that the gradient relative to a parameter that connects neuron $j$ of the layer $l$ to neuron $k$ of the layer $l-1$ is

$$\frac{\partial J}{\partial \omega_{jk}^{n[l]}} = \Delta_j^{[l]} a_k^{[l-1]} = \left(\sum_{k=1}^{n[l+1]} \omega_{kj}^{[l+1]} \Delta_k^{[l+1]}\right) \phi'(z_j^{[l]}) a_k^{[l-1]} \tag{1}$$

where the apex [...] indicates the considered layer, $z_j$ the input of neuron $j$, $a_k = \phi(z_k)$ the activation function of neuron $k$, and $\Delta_k$ the error of neuron $k$.

To be more clear about what the contributions to the gradient are, we can consider a deep network made of just a single neuron per layer. In that case, the general equation 1 becomes, after full expansion of all $\Delta$s

$$\frac{\partial J}{\partial \omega^{[l]}} = \frac{\partial J}{\partial \hat{y}} g'(z^{[L]}) \left[\prod_{k=l+1}^{L} \omega^{[k]}\right] \left[\prod_{k=l}^{L-1} \phi'(a^{[k-1]}\omega^{[k]})\right] a^{[l-1]} \tag{2}$$

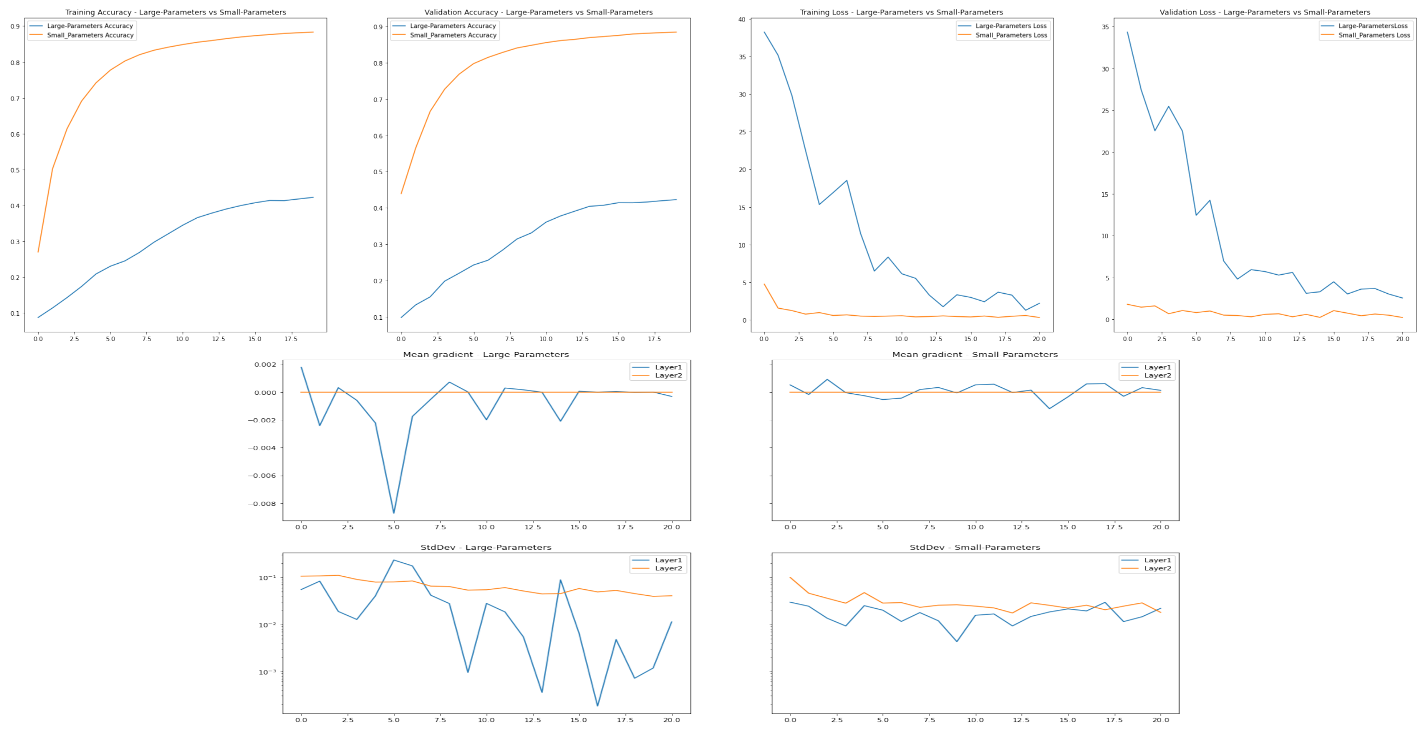

Figure 1: Comparison of the training and validation accuracy (y-axis) curves (top-left panels) of Large Parameters initialization and an optimized parameters initialization, at each epoch (x-axis). Evolution of the loss function in training and validation (top-right panels) at each epoch. Gradients' mean and standard deviation evolution in time for the first and last layer of the network, in the Large Parameters (bottom-left panels) and optimized (bottom-right panels) models. Notice low variability of the gradients in the first layer of the Large Parameters model, showing inability to consistently update weights.

The product of the derivatives $\phi'$ is the one which makes the Vanishing Gradients possible: if we assume to be using a $\sigma$-like function, it takes even just one very large $\omega$ to saturate the function, bringing the whole product down. Since all the $\phi'$s are $< 1$ and $\phi'(a\omega) \cdot \omega \to 0$ for $\omega \to \infty$ for any $a$, the other terms cannot counter-balance, not even the product of the $\omega_s$. This makes the gradient plummet. This is the **Large Parameters** regime.
Proposed solution: *Regularization* is demonstrated to be a mathematically valid solution for this issue, as the added term to the loss-function increases with larger parameters, counter-balancing the product of $\phi'$s.

There is another way in which the gradient can do gown to zero: if *many* not-so-small (due to not-so-large parameters) terms $\phi' < 1$ multiply by each other. There are many of these terms when the network has many layers. This is the **Many Layers (Deep)** regime. Proposed solution: *Rescaled activation functions* such as $g(z) = \alpha\phi(z/\alpha)$ is demonstrated to be a mathematically valid solution for this issue, as the rescaling parameter $\alpha$ increments the upper bound for the gradients and makes it higher the deeper the network.
Of course the two effects can co-exist in large networks.

## 3 The practice: proposed solutions applied to the MNIST dataset

The computational experiments have been conducted in a structured and linear way so that regularization or activation function would be the only tweaked parameters, everything else has been kept constant. For details about the fixed structure of the experiments, functions and hyperparameters used, see *Appendix*.

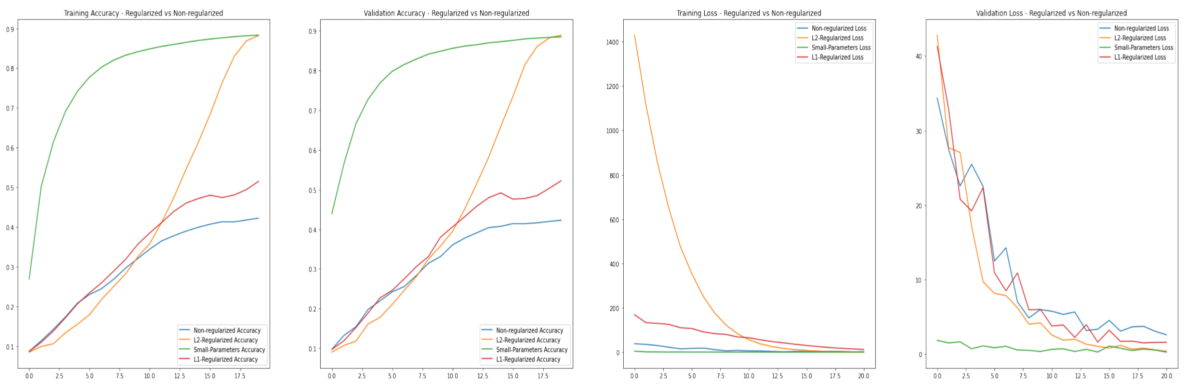

Figure 2: Performance comparison between regularized, unregularized and optimized models. The L2-regularized model reaches optimal performance after few epochs.

### 3.1 The Large Parameters Regime results

The network used in this case is a shallow network with 8 neurons in the hidden layer. This choice has been made in order to minimize the effects of the Many Layer regime and isolate the effects brought on by the large parameters choice.
The Large Parameters regime initialization $N(\omega; \mu = 0, \sigma^2 = 10)$ has been compared with a more optimal initialization $N(\omega; \mu = 0, \sigma^2 = 1)$ to verify that the Vanishing Gradients issue occurs in the former and does not in the latter, as in figure 1.

Now, the regularization L2 and LASSO are applied separately to the model in the Large Parameters regime with regularization parameter $r = 0.01$. As can be seen from fig.2, L2-regularized model performs significantly better than the unregularized one, reaching in very few training epochs the performance of the optimized-initialization model. LASSO does not show the same efficacy, although it still provides a decent improvement over no regularization. In order to find the best regularization parameter $r$, training has been repeated for 10 values of $r \in [10^{-3}, 10^2]$ finding that, indeed, the one providing best performance was around $r_{opt} \approx 0.01$, and that for values higher than 0.2 the regularization performs more poorly than the unregularized model.

A super-Large Parameter regime has been also explored $N(\omega; \mu = 0, \sigma^2 = 20)$ in order to see how strongly the regularization is able to impact an "extreme" case of bad weights' initialization. By the end of the 20 epochs, L2 is able to reach the unregularized case: in the ealry epochs, the gradients show greater variability compared to the unregularized case, making it possible to improve and catch up, eventually surpassing it, thanks to the higher variance in the first layer's gradients [*see "mnist_LargeParameters" notebook*].

Finally, the last trial consists in using a Uniform Large Parameter regime $U[-10, 10]$ as used in the paper. In this case, the L2-regularized model outperforms completely the unregularized and L1-models, as the regularization allows, again, for grater variance in the first layer, making it possible for the learning process to keep exploring the parameter-space in search of a minimum [*see "mnist_LargeParameters" notebook*].

The same pipeline has been then repeated with the CNN and CNN-paper (CNN used in the paper's experiments): in this cases, as shown in fig.3, it is even more clear how the regularization impacts the variance of gradients, which otherwise drops to approximately zero mean for the first convolutional layers.
To check whether the proposed solution's performance can be in any way compared to the standards of Vanishing Gradients workarounds, the regularized model with $N(\omega; \mu = 0, \sigma^2 = 10)$ has been compared to a ReLU model with the same parameters' initialization and a ReLU with the optimized Glorot Uniform initialization. The results are promising, as shown in fig.4

### 3.2 The Many Layers Regime results

The network used in this case is a deep network with 20 hidden layers and 8 neurons each. The same pipeline has been repeated, showing in this case too how the Vanishing Gradients phenomena impacts the learning process. The results are similar to the ones shown in figure 1. In order to try and solve

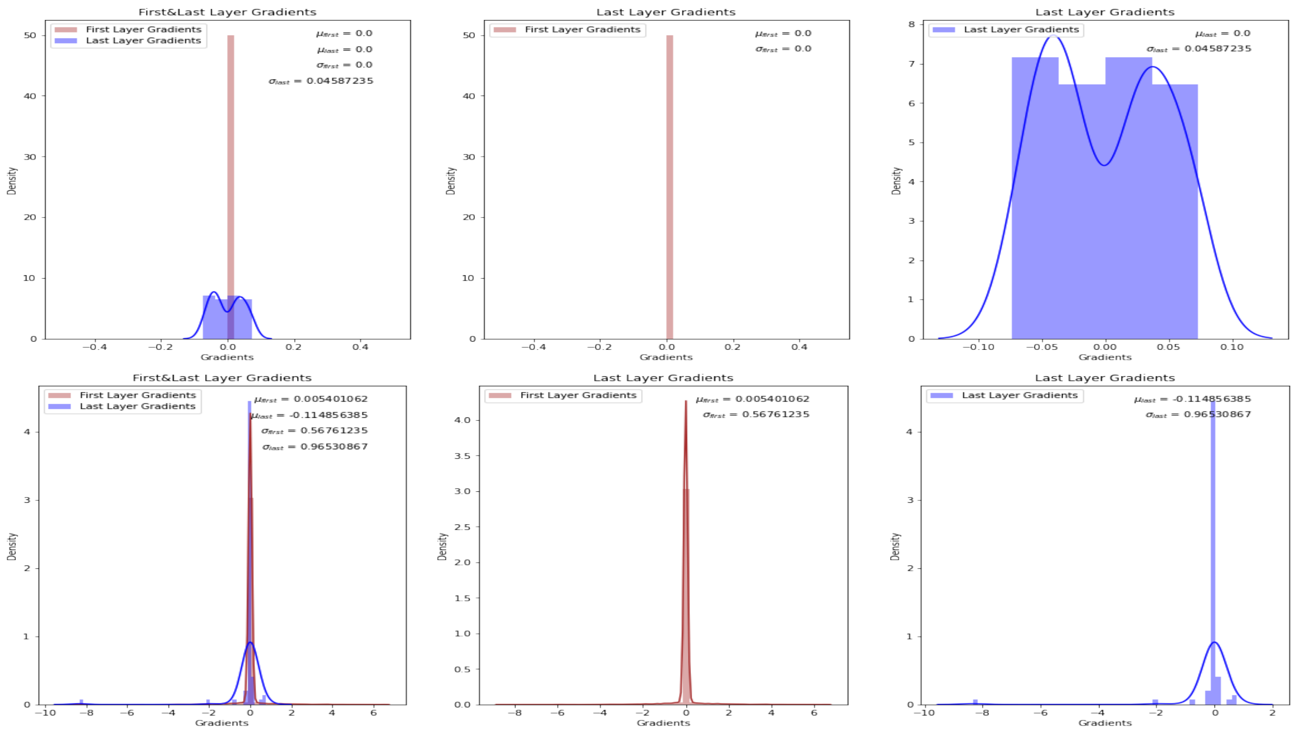

Figure 3: Comparison of the gradients distribution in the CNN network, in the first and last layer of the network in the last epoch of training. The unregularized model (top panel) shows a first layer distribution heavily peaked around zero with very low variance, in high contrast with the last layer distribution which is fairly wide. On the another hand, in the L2-regularized model the two distributions are almost comparable in width, showing more capacity for learning.

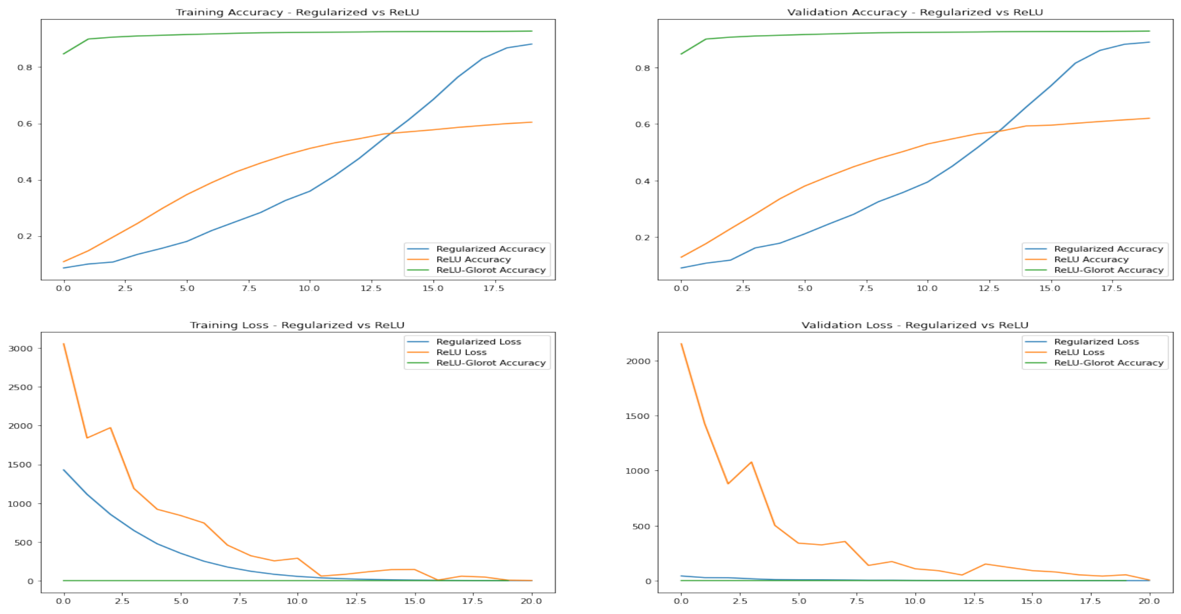

Figure 4: Performance comparison between L2-regularized and ReLU models. L2-regularized model catches up with the ReLU-Glorot Uniform performance

this issue, a rescaled activation function $g = \alpha \tanh(z/\alpha)$ with $\alpha = 2$ is applied to all layers. The results are not satisfactory: the rescaled activation function does not seem to be influencing the issue in any way, performing almost identically to the unscaled model as shown in fig.5.

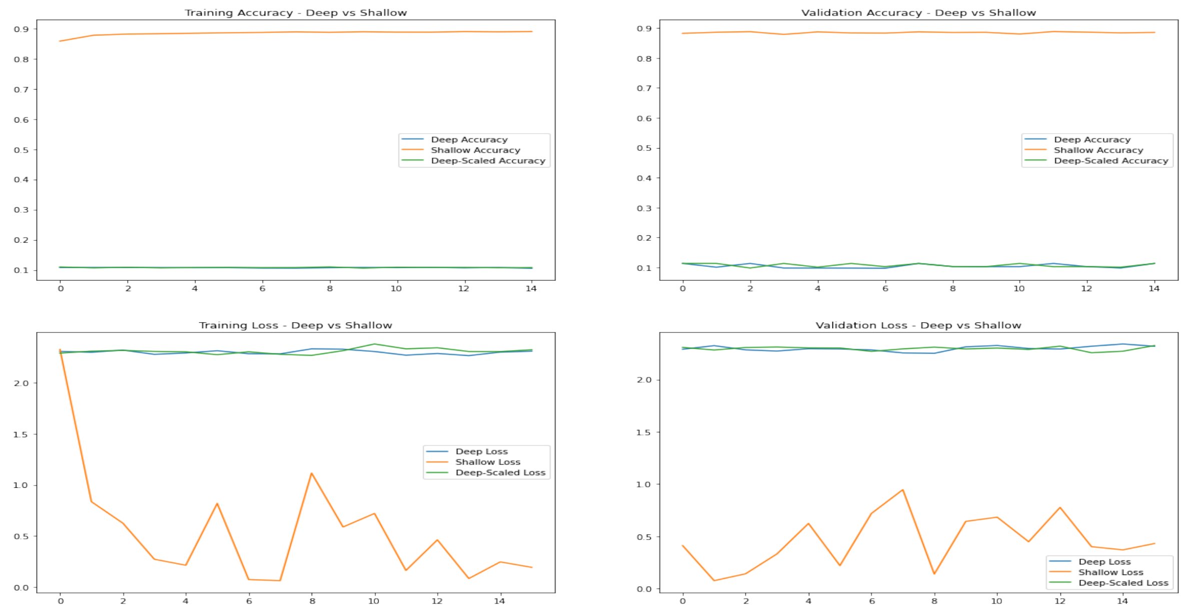

Figure 5: Performance comparison between scaled, unscaled and shallow network models. The scaled and unscaled models overlap, whereas the shallow model performs optimally as it does not suffer from the Vanishing Gradient issue.

As done in the case of Large Parameters, an optimal rescaling parameter $\alpha_{opt}$ has been looked for but they all perform similarly in a range of $\alpha \in [1.2, 5]$, overlapping with the unscaled deep model. The pipeline has been repeated for CNN and CNN-paper with the same results, thus the problem does not seem to be model-dependent.

## 4 Conclusions

The quality of the results obtained are mixed. On the one hand, the Large parameters regime has been successfully solved by imposing regularization over the parameters of the model, in particular L2-regularization performs almost as well as a ReLU model with optimized initial parameters, even though it takes some time before the model reaches convergence. Thus, the latter still seems to be more convenient, but the regularization definitely makes $\sigma$-like functions viable in the scope of neural networks.
On the other hand, the issue of many layers stacking together in the back-propagation gradients' computation has not given any satisfactory result. Comparing the unscaled model with the scaled one, it can be noticed that they are in fact almost identical, raising the hypothesis that there could be a code problem: it is as if the custom training loop used is not able to evaluate correctly the gradient using the custom scaled function. One possible solution that will be attempted is the implementation of ad-hoc activation layer into the model rather than passing the custom activation function from the outside. In this way, TensorFlow might be able to evaluate gradients correctly.
It would have been interesting to see how combining both regularization and rescaling in a $\sigma$-like model would perform compared to a fully optimized network: from the paper, tanh performs better than ReLU in the hypothesis of sub-optimal parameter initialization, even when ReLU is regularized.
Ultimately, these tests have been of great interest from a theoretical point of view, however the question about which activation function performs better, and in which cases it does so, remains open, giving rise to a follow-up question: if something seems to be performing well, and better than anything else, on a variety of contexts, namely, ReLU-like functions combined with proper initialization, why should we try to restore sigmoid functions other than for a mathematical interest in finding alternative ways to overcome the Vanishing Gradient effect?

# 5 Appendix

## 5.1 Computational Experiments: general notes

The computational experiments have been conducted in a structured and linear way, as a series of subsequent questions arising from previous steps. Some initial, general notes are mentioned here:

- Datasets: pipelines are executed for both MNIST and CIFAR-10 (the one used in the paper) datasets, in order to compare directly the results with the ones from the paper.
- PCA: dimensionality reduction has been applied to reduce computational times. The number of features used accounts for approximately 20% - 30% of the initial number of features, while preserving 95% of the variance.
- Networks used: the experiments have been conducted using three types of networks, to test the proposed resolutions on different configuration and see if they perform any different. These are
  1. MLP network
  2. CNN network
  3. CNN-paper network (the same network used in the paper to perform the experiments)
- Custom Training: the implementation of training has been done through a custom function that uses the Gradient Tape to perform automatic differentiation, rather than using the black-box train method of Keras' models. This has been done in order to retrieve gradients after each training epoch. It added a layer of technical complexity to the problem: for it was necessary to add everything by hand (eg. losses had to be added by hand to the loss function).
- Constant Specifics of Training: to maintain coherence over the different experiments, some hyper-parameters and functions are maintained constant throughout all experiments. This way, the changes applied from one experiment to another are isolated and any variation in performance can be traced back the specific change applied. These constants are:
  - Optimizer: Adam(learning_rate = 0.001, as in the paper). Faster convergence, allowed for clear visual distinction between good and bad performance due optimal and sub-optimal parameters/layers regimes.
  - Loss: Sparse Categorical Cross Entropy. Multi-class classification problem. The Sparse version is chosen as the datasets have a label-encoding, rather than one-hot encoding.
  - Metric of performance: Accuracy. The goal of the experiment is to measure performance in the sense of ”How many samples the model is able to correctly recognize”.
  - Activation function: $g = \tanh$. The same $\sigma$-like function used in the paper.
  - epochs: 15, single seed. Due to computational times, the best number of epochs to allow enough convergence time. For the same reason, all tests have been done with a single seed=1 for all Parameters initializations.
  - batch size: 32.

  Other values have been tried before-hand, with no significant differences; thus, the most optimal choices have been made in order to optimize everything leaving only the Large Parameters or the Many Layers regimes as the only issue present in the model. In this way every change added to the model was only affecting the Vanishing Gradient problem, while everything else was working properly from the beginning.