Revisiting Weight Initialization of Deep Neural Networks

Maciej Skorski MACIEJ.SKORSKI@UNI.LU
University of Luxembourg

Alessandro Temperoni ALESSANDRO.TEMPERONI@UNI.LU
University of Luxembourg

Martin Theobald MARTIN.THEOBALD@UNI.LU
University of Luxembourg

Editors: Vineeth N Balasubramanian and Ivor Tsang

Abstract

The proper *initialization of weights* is crucial for the effective training and fast convergence of *deep neural networks* (DNNs). Prior work in this area has mostly focused on the principle of *balancing the variance among weights per layer* to maintain stability of (i) the input data propagated forwards through the network, and (ii) the loss gradients propagated backwards, respectively. This prevalent heuristic is however agnostic of dependencies among gradients across the various layers and captures only first-order effects per layer. In this paper, we investigate a *unifying approach*, based on approximating and controlling the *norm of the layers' Hessians*, which both generalizes and explains existing initialization schemes such as *smooth activation functions*, *Dropouts*, and *ReLU*. We empirically demonstrate that tracking the Hessian norm is a useful diagnostic tool which helps to more rigorously initialize weights over a variety of DNN applications, including both (i) shallow networks like simplified GoogleNet using Flowers dataset, and (ii) deep networks like EfficientNet and ResNet using fashion MNIST, CIFAR-10 and Google SVHN for image processing.

1. Introduction

Years of research and practical experience show that parameter initialization is of critical importance for the training *neural networks* (NNs), particularly *deep neural networks* (DNNs), which process their input by stacking several layers of parameterized activation functions. A main challenge remains to determine a good initial "guess" of the parameters: small weights may lead to a vanishing effect of (i) the input data being processed during the forward pass, and (ii) the loss gradients being propagated during the backward pass through the network, respectively. Overly large weights, on the other hand, may (iii) unduly amplify certain dimensions of the input data in the forward pass, and then in turn (iv) strongly penalize those dimensions during the backward pass.

To address the above four points, a variety of common initialization schemes for DNNs have been explored in the literature Arpit et al. (2019); Glorot and Bengio (2010); He et al. (2015); Hendrycks and Gimpel (2016); Hanin and Rolnick (2018); Szegedy et al. (2013); Virmaux and Scaman (2018); Xu et al. (2016) in the past. Since both forward and backward propagation of gradients through a DNN is based on iterative tensor products,

current approaches mostly focus on preserving the *variance among the weights per layer balanced* (i.e., close to 1), which primarily aims to avoid numerical issues (i.e., vanishing or exploding sums of element-wise matrix multiplications). They, however, ignore dependencies among weights across the various layers of a DNN, and are limited to capturing only first-order effects (whereas the landscape of optimization problems strongly depends on second-order effects, e.g., the loss surface curvature). Only recently, the usage of second-order methods, i.e., by explicitly considering the layers' Hessians, has been investigated in the literature Dauphin and Schoenholz (2019); Ghorbani et al. (2019); Sagun et al. (2017). In practice, these approaches however suffer from the high computational overhead of computing the actual Hessian matrix (which is quadratic in the number of weights per layer).

While the source of difficulty is well-understood, there is no universal remedy: the choice of the initialization scheme still is typically studied on a case-by-case basis (depending on the specific architecture and use-case setting) and often under simplifying theoretical assumptions (such as first-order approximations and under strong independence assumptions) Arpit et al. (2019); Glorot and Bengio (2010); Hanin and Rolnick (2018); He et al. (2015); Hendrycks and Gimpel (2016); Xu et al. (2016) . Even for some relatively simple models, we still lack a complete understanding of initialization nuances and instead rely on empirically chosen defaults Kocmi and Bojar (2017).

Contributions. We summarize the contributions of our work as follows.

- We investigate the usage of *second-order methods* to more accurately estimate the global curvature of weights at initialization time of a DNN. Specifically, by developing a novel form of *the chain rule for Hessians* in combination with their factorization into *Jacobian products*, we provide an *efficient approximation scheme* to back-propagate the weights' Hessians through the various layers of the DNN.
- The usage of Hessians for DNN optimization has only recently been investigated in Dauphin and Schoenholz (2019); Ghorbani et al. (2019); Sagun et al. (2017), but no formal proofs for the relationship to existing initialization schemes have been provided so far. We thereby provide a *detailed theoretical justification* for existing initialization schemes based on *smooth activation functions* Glorot and Bengio (2010), *Dropouts* Hendrycks and Gimpel (2016), and *Rectified Linear Unit* (ReLU) Krizhevsky et al. (2012).
- Besides our theoretical results, we provide an *efficient implementation* of our framework in Tensorflow along with *detailed experiments* over a wide range of recently proposed convolutional networks, including EfficientNet Tan and Le (2019) and ResNet He et al. (2016).
- Finally, our suggested approximation scheme will also be of interest for a variety of *further applications* in non-convex optimization (e.g., the Hessian is widely used for adjusting parameters and diagnosing convergence issues Salvatier et al. (2016)).

2. Background & Related Work

Before we review the most popular weight-initialization schemes, we briefly introduce the key concepts and notation we use through the rest of the paper.

Tensor Derivatives. For two *tensors* $y = y_{j_1,\ldots,j_q}$ and $x = x_{i_1,\ldots,i_p}$, of rank q and p respectively, the *derivative* $D = D_x y$ is a tensor of rank $q + p$ with coordinates $D_{j_1,\ldots,j_q,i_1,\ldots,i_p} =$

$\frac{\partial y_{j_1,\ldots,j_q}}{\partial x_{i_1,\ldots,i_p}}$. If $y = f(x)$, where x has shape $[n]$ and y has shape $[m]$, then $D_x y$ is of shape $[m, n]$ and equals to the total derivative of f.

Tensor Products. *Contraction* sums over paired indices (axes), thus lowering the rank by 2 (or more when more pairs are specified). For example, contracting positions a and b in x produces the tensor $\sum_{i_a=i_b} x_{i_1\ldots i_a\ldots i_b\ldots i_p}$ with indices $\{i_1,\ldots,i_p\} \setminus \{i_a, i_b\}$, where the dimensions of paired indices should match. A *full tensor product* combines tensors x and y by cross-multiplications $(x \otimes y)_{i_1,\ldots,i_p,j_1,\ldots,j_q} = x_{i_1,\ldots,i_p} \cdot y_{j_1,\ldots,j_p}$, thereby producing a tensor of rank $p + q$. A *tensor dot-product* is the full tensor product followed by contraction of two compatible dimensions. For example, the standard matrix product of $A_{i,j}$ and $B_{k,l}$ is the tensor product followed by contraction of j and k. We denote the dot-product by $\bullet$, thereby omitting the contracted axes when this is clear from the context.

Chain & Product Rules. Tensors obey similar chain and product rules as matrices. Specifically, we have $D_x(A \bullet B) = D_x A \bullet B + A \bullet D_x B$. Also, when $B = f(A(x))$ holds, we have $D_x B = D_A f \bullet D_x(A)$. The contraction is over all dimensions of A which match the arguments of f.

Spectral Norm. For any matrix A, the *singular eigenvalues* are defined as the square roots of the eigenvalues of $A^T A$ (which is square symmetric and positive definite). The *spectral norm* then is the biggest singular eigenvalue of A.

Neural Networks. From an algebraic perspective, we look at a *neural network* (NN) as a chain of mappings of the form

$$z^{(k+1)} = f^{(k)}\left(w^{(k)} \cdot z^{(k)} + b^{(k)}\right)$$

which sequentially processes an *input vector* $x = z^0$ through a number of *layers* $k = 0 \ldots n-1$.

We assume that $z^{(k)}$ are real-valued vectors of shape $[d_k]$, *weights* $w^{(k)}$ are matrices of shape $[d_{k+1}, d_k]$, *biases* $b^{(k)}$ are of shape $[d_{k+1}]$, and $f^{(k)}$ are (possibly non-linear) *activation functions* which are applied element-wisely. The task of learning then is to minimize a given *loss function* $L(z, t)$ where $z = z^n$ is the network output and t is the ground-truth, over the weights $w^0, \ldots, w^{n-1}$. Neural networks are optimized with variants of gradient-descent and weights are initialized randomly. Overly small weights make the learning process slow, while too high weights may cause unstable updates and overshooting issues. Good initialization schemes aim to find a good balance between the two ends.

Initialization Based on Variance Flow Analysis. Glorot and Bengio Glorot and Bengio (2010) proposed a framework which estimates the variance at different layers in order to maintain the aforementioned balance. The approach assumes that the activation functions approximately behave like the identity function around zero, i.e., $f(u) \approx u$ for small u. This has been generalized to other nearly-linear functions in follow-up works He et al. (2015); Xu et al. (2016)). By linearization, we then obtain:

$$z_i^{(k+1)} \approx \sum_{j=1\ldots d_k} w_{i,j}^{(k)} \cdot z_j^{(k)} + b_i^k \tag{1}$$

In the forward pass, we require $\mathbf{Var}[z^{(k+1)}] \approx \mathbf{Var}[z^{(k)}]$ to maintain the magnitude of inputs until the last layer. In the backward pass, we compute the gradients by recursively applying

the chain rule

$$\partial_{z_i^{(k)}} L = \sum_{j=1\ldots d_{k+1}} \partial_{z_j^{(k+1)}} L \cdot \partial_{z_i^{(k)}} z_j^{(k+1)} \tag{2}$$

$$\approx \sum_{j=1\ldots d_{k+1}} \partial_{z_j^{(k+1)}} L \cdot w_{j,i}^{(k)} \tag{3}$$

while we aim to keep their magnitude, i.e., $\mathbf{Var}\left[\partial_{z^{(k-1)}} L\right] \approx \mathbf{Var}\left[\partial_{z^{(k)}} L\right]$.

Looking at Eq. 1 and 2, we see that the weights $w^{(k)}$ interact with the previous layer during the forward pass and with the following layer during the backward pass. The first action is multiplying along the input dimension d_k, while the second action is multiplying along the output dimension d_{k+1}. One can prove that, in general, taking the dot-product with an *independently* centered random matrix along dimension d scales the variance by the factor d Glorot and Bengio (2010); Xu et al. (2016). Thus, to balance the two actions during the forward and backward pass, one usually chooses the $w^{(k)}$ as i.i.d. samples from a normal distribution $N(\mu, \sigma^2)$ with mean $\mu = 0$ and standard deviation

$$\sigma[w^{(k)}] = \sqrt{\frac{2}{d_k + d_{k+1}}}. \tag{4}$$

Variance-based initialization schemes Arpit et al. (2019); Glorot and Bengio (2010); He et al. (2015); Hendrycks and Gimpel (2016); Xu et al. (2016), however, implicitly assume *independence* of weight gradients across layers. This is *not true already in first pass*, since back-propagated gradients depend on weights used during the forward pass and also on the input data. As an example, consider a regression setting with two layers and a linear activation function, such that $L = (z-t)^2$, $z = w_2 w_1 x$. Note that $\partial_z L = 2(z-t) = -2(w_2 w_1 - t)$ here depends on both w_2 and w_1. To see correlations with the input vector, consider a one-dimensional regression $L = (z-t)^2$, $z = wx$. From Eq. 5 in Glorot and Bengio (2010), we should have $\mathbf{Var}[\partial_w L] = \mathbf{Var}[\partial_w z] \cdot \mathbf{Var}[\partial_z L]$ for w with unit variance, but this gives $\mathbf{Var}[2(wx-t)x] = \mathbf{Var}[x] \cdot \mathbf{Var}[2(wx-t)]$. Not only two sides can be a factor away but also the target t can be correlated to the input x. In addition to this lack of correlations, the above kinds of variance analyses provide only *qualitative* insights, since they do not directly connect the variance estimation to the underlying optimization problem. In fact, we cannot get more quantitative insights, such as estimating the step size, from these first-order methods.

Other First-Order Approximations. The stability of the linearization in (2) has also been studied using the *Lipschitz property* as the sensitivity metric Szegedy et al. (2013); Virmaux and Scaman (2018) as well as *mean-field theory* to investigate the dynamic of the forward-pass in for very deep networks Xiao et al. (2018).

Some generalizations to the classical schemes discussed above have been also proposed. For example Balduzzi et al. (2017) generalizes the variance analyses to ResNets, while Bachlechner et al. (2020) introduces extra parameters at the activation layers.

Hessian Approaches to Neural Nets. Considering the Hessian is ubiquitous in the field of optimization Salvatier et al. (2016), and has been also applied in the context of DNNs Dauphin and Schoenholz (2019); Ghorbani et al. (2019); Sagun et al. (2017). Since the exact calculation of the Hessian is not feasible for larger networks (which is quadratic

in the total, usually already large, number of parameters), efficient approximation schemes have been proposed.

Block-diagonal approximations of the Hessian tensor were considered in TONGA (Le Roux et al., 2008) and Martens and Grosse (2015), but were not extensively evaluated (very small networks, not-real world, visual not a quantitative comparison).

Some Hessian characteristics such as the eigenvalues Ghorbani et al. (2019) or the trace Jacot et al. (2020), have been proposed. However these are simplified and do not give insights on the Hessian tensor structure (such as extracting the diagonal or the block) . Furthermore, these approximations are probabilistic with no clear guarantee on the error.

3. Hessian-Stabilizing Initialization

We now present and discuss our suggested weight-initialization scheme by applying a variant of the Hessian chain rule across the (hidden) layers $k = 0 \dots n-1$ of a neural network, which constitutes the main contribution of our work. In general, for training a neural network, variants of gradient-descent are applied in order to update the model parameters w iteratively towards the gradient $g = D_w L$ of the loss function. In order to quantify this decrease, we need to consider the *second-order approximation*

$$L(w - \gamma g) \approx L(w) - \gamma\, g^T \cdot g + \frac{\gamma^2}{2}\, g^T \cdot \mathbf{H} \cdot g$$

where $\cdot$ stands for the matrix (or more generally: tensor) dot product. The maximal step size γ^* guarantees that the decrease equals $\gamma^* = \|\mathbf{H}\|^{-1}$ Goodfellow et al. (2016) where $\|\mathbf{H}\|$ is the Hessian norm, i.e., its maximal eigenvalue. In other words, if we want to train with a constant step size, then we need to control the norm of the Hessian. We therefore propose the following paradigm:

> *Good weight initialization controls the Hessian:* we initialize the weights $w^{(k)}$ such that $\|\mathbf{H}_{w^{(k)}}\| \approx 1$.

Moreover, we only make the following mild assumption about the loss functions, which requires the activation function to be *nearly linear around 0*:

> *Admissible activation functions:* the loss function must satisfy $f(0) = 0$ and $f''(0) = 0$. We note that this condition holds for all standard activation functions, such as linear, sigmoid, tanh or relu.

Finally, our techniques aim to approximate the global curvature of weights up to leading terms. These approximations are accurate under the following mild assumption:

> *Relatively small inputs:* we have $\|z^{(k)}\| \leqslant c$ for all layers k, for a small constant c (e.g., $c = 0.5$).

Note that the latter is necessary to ensure the stability of the forward pass and is implicitly assumed so also in the variance flow analyses.

Before presenting our results, we need to introduce some more notation. Let $F^{(k)} = z^{(k)}$ be the input of the k-th layer. Let $A^{(k)} = D_{u^{(k)}} z^{(k+1)}$ be the derivative of the forward

activation at the k-th layer, with respect to the output before activation $u^{(k)} = w^{(k)} \cdot z^{(k)} + b^{(k)}$. Let $B^{k+1} = D_{z^{(k+1)}} z^{(n)}$ be the output derivative back-propagated to the input of the $(k+1)$-th layer. Let $\mathbf{H}_z = D_z^2 L(z,t)$ be the loss Hessian with respect to the predicted value z. Finally, let $\mathbf{H}_w = D_w^2 L(z^{(n)}, t)$ be the loss Hessian with respect to the weights w.

3.1. Approximation via Hessian Chain Rule

The Hessian of the loss function over its domain is usually very simple and has nice properties. This however changes when a neural network reparameterizes the problem by a complicated dependency of the output z on the weights w. We therefore have to answer the following question: how does the dependency of the network output on the weights affect the loss curvature?

In general, if $z = z(w)$ is a reparameterization, then

$$\underbrace{D_w^2 L(z(w),t)}_{\text{reparameterized Hessian}} = D_z^2 L(z,t) \underbrace{\bullet D_w z(w) \bullet D_w z(w)}_{\text{linearization effect}} + D_z L(z,t) \underbrace{\bullet D_w^2 z(w)}_{\text{curvature effect}} \tag{5}$$

where bullets denote tensor dot-products along the appropriate dimensions. This is more subtle than back-propagation of first derivatives, because both first- and second-order effects have to be captured. The main contribution of this work thus is the following result, which in its essence states that, usually, the curvature effect contributes less than the linearization effect[1].

Theorem 1 (Approximated Hessian chain rule for neural networks) *With notation as above, the loss Hessian $\mathbf{H}_{w^{(k)}}$ with respect to the weights $w^{(k)}$ satisfies*

$$\mathbf{H}_{w^{(k)}}[g,g] \approx v^T \cdot \mathbf{H}_z \cdot v \quad \text{with} \quad v = B^{(k)} \cdot A^{(k)} \cdot g \cdot F^{(k)} \tag{6}$$

where products are standard matrix products. More precisely, the approximation holds up to a third-order *error term $\sim f''' c^3 \cdot \|g\|^2$ where f''' is the bound on the third derivative of the activation functions and c is the bound on the inputs x. The leading term then is of order $\sim c^2 \cdot \|g\|^2$.*

We observe the following important properties.

Remark 2 (Beyond block-diagonal Hessian) *The approximation above computes the main blocks of the Hessian, namely Hessians with respect to each layer. In practice, the cross-layer components are of smaller order and can be omitted Botev et al. (2017). However our approximation extends to this case by using two different v's instead of one.*

Remark 3 (Low computational complexity) *Computing the Hessian approximation is of cost comparable to back-propagation. The only Hessian we need is the final loss/output Hessian, which is usually small (K^2 for the classification of K classes).*

Remark 4 (Beyond MLP model) *We formulated the result for densely-connected networks but the approximation holds in general with $v = D_{w^{(k)}} z^{(n)} \bullet g$ (as we will see in empirical evaluation).*

1. Proofs are provided as part of the Supplementary Material submitted along with this paper.

Remark 5 (Perfect approximation for ReLU networks) *We have exact equality for activations with $f'' = 0$ such as variants of ReLU (see Section 4.3).*

Remark 6 (Good approximation up to leading terms) *Regardless of the activation function, the error term is of smaller order (under our assumption of bounded inputs).*

Theorem 1 involves a hidden constant which we demonstrate to be small in practice (see our experiments in Section 5).

3.2. Approximation via Jacobian Products

From the previous subsection, we are left with the linearization effect of the chain rule, which can be further factored. This reduces the problem of controlling the *products of the hidden layers' Jacobians.*

Theorem 7 (Hessian factorized into Jacobians) *Up to third-order terms in $z^{(i)}$, we can factorize v from Theorem 1 into*

$$v \approx \mathbf{J}^{(n-1)} \cdot \ldots \mathbf{J}^{(k+1)} \cdot A \cdot g \cdot \mathbf{J}^{(k-1)} \cdot \ldots \mathbf{J}^{(0)} \cdot z^{(0)} \tag{7}$$

where $\mathbf{J}^k = D_{z^{(k)}} z^{(k+1)}$ is the derivative of the output with respect to the input at the k-th layer. In particular, the Hessian's dominant eigenvalue scales by a factor of at most $\|v\|^2$, where it holds that:

$$\|v\| \leqslant \| \underbrace{\mathbf{J}^{(n-1)} \cdot \ldots \cdot \mathbf{J}^{(k+1)}}_{\textit{backward product}} \| \cdot \|A\| \cdot \| \underbrace{\mathbf{J}^{(k-1)} \cdot \ldots \cdot \mathbf{J}^{(0)}}_{\textit{forward product}} \| \cdot \|z^{(0)}\| \tag{8}$$

The norm of the matrix product $\|J_k \ldots J_1\|$ then is computed as the maximum of the vector norm $\|J_k \cdots J_1 \cdot v\|$ over vectors v with unit norm. Given this result, a good weight initialization thus aims to make the backward and forward products having a norm close to 1.

Remark 8 (Connection to products of random matrices) *Note that our problem closely resembles the problem of* random matrix products *Kargin et al. (2010). This is because Jacobians for smooth activation functions are simply random-weight matrices.*

Remark 9 (Connection to spectral norms) *Further, it is possible to estimate the product of random matrices by the product of their spectral norms. In particular, the spectral norm of a random $m \times n$ matrix with zero-mean and unit-variance entries is $\frac{1}{\sqrt{m}+\sqrt{n}}$ on average Silverstein (1994). For the Gaussian case, this can be found precisely by Wishart matrices Edelman (1988). This however is overly pessimistic for long products, for similar reasons as overestimating of Lipschitz constants Szegedy et al. (2013); Virmaux and Scaman (2018).*

4. Relationship to Existing Initialization Schemes

We next discuss the relationship of our Hessian-based weight initialization scheme to a number of previous schemes, namely *smooth activations* Glorot and Bengio (2010), *dropout* Hendrycks and Gimpel (2016), and *ReLU* Krizhevsky et al. (2012).

4.1. Smooth Activations

We first formulate the following lemma.

Lemma 10 (Dot-product by random matrices) *Let w be a random matrix of shape $[n, m]$, with zero-mean entries and a variance of σ^2. Let z, z' be independent vectors of shape $[m]$ and $[n]$, respectively. It then holds that:*

$$\mathbf{E}\|w \cdot z\|^2 = n\sigma^2 \cdot \mathbf{E}\|z\|^2 \tag{9}$$

$$\mathbf{E}\|z' \cdot w\|^2 = m\sigma^2 \cdot \mathbf{E}\|z'\|^2 \tag{10}$$

Using this, we can estimate the growth of the Jacobian products in Theorem 7 as follows.

Corollary 11 (Smooth activations Glorot and Bengio (2010)) *Consider activation functions such that $f'(0) = 1$. Then $\mathbf{J}^{(k)} \approx w^{(k)}$ (up to leading terms) and the norm of the forward product is stable when*

$$\mathbf{Var}[w^{(k)}] = \frac{1}{d_{k+1}}, \tag{11}$$

while the norm of the backward product is stable when

$$\mathbf{Var}[w^{(k)}] = \frac{1}{d_k}. \tag{12}$$

As a compromise, we can choose $\mathbf{Var}[w^{(k)}] = \frac{2}{d_{k+1}+d_k}$.

Note that we exploit the fact that (up to leading terms) Jacobians of smooth activation functions are independent from any other components.

4.2. Dropouts

Dropouts (i.e., inactive neurons) can be described by a *randomized function* f_p which, for a certain dropout probability p, multiplies the input by $B_{1-p} \cdot \frac{1}{1-p}$, where B_{1-p} is a Bernoulli random variable with parameter $1-p$. The Jacobian then is precisely given by:

$$\mathbf{J} = w = \mathsf{diag}(B_1, \ldots, B_d), \quad B_i \sim \mathsf{Bern}(1-p) \tag{13}$$

When multiplying from left or right, this scales the norm square by $(1-p)^{-2} \cdot \mathbf{E}[\mathsf{Bern}(1-p)]^2 = 1-p$. Thus, we obtain the following corollary.

Corollary 12 (Initialization for dropout) *Let $1-p$ be the* keep rate *of a dropout. Let σ^2 be the initialization variance without dropouts, then it should be corrected as:*

$$\sigma' = \sigma/\sqrt{1-p} \tag{14}$$

This corresponds to the analysis in Hendrycks and Gimpel (2016), except that they suggested a different correction factor for the back-propagation phase.

4.3. ReLU

Rectified Linear Unit (ReLU) Krizhevsky et al. (2012) is a non-linear activation function given by $f(u) = \max(u, 0)$. Consider a layer such that $z' = f(u)$, $u = w \cdot z$, where w is zero-centered with a variance of σ^2 and again of shape $[n, m]$, while z is of shape m. We then have $\mathbf{J} = D_z z' = \mathsf{diag}(f'(u)) \cdot w$.

For the forward product, we therefore consider $\mathbf{J} \cdot z = \mathsf{diag}(f'(u)) \cdot u$. This scales the norm of u by $\frac{1}{2}$ when u is symmetric and zero-centered, which is true when also w is symmetric and zero-centered. The norm of z is thus changed by $\frac{n\sigma^2}{2}$.

For the backward product, on the other hand, we have to consider $v \cdot J \cdot \mathbf{J}$, where J is the Jacobian product for the subsequent layers and possibly depends on u. However, if the next layer is initialized with i.i.d. samples, the output distribution only depends on the number of active neurons $r = \#\{i : u_i = 1\}$. Conditioned on this information, the following layers are independent from $\mathbf{J}$. Given r, the squared backward product norm thus changes by the factor $r/n \cdot m\sigma^2$. Since $\mathbf{E}[r] = n/2$, the scaling factor is $\frac{m\sigma^2}{2}$.

Corollary 13 (ReLU intialization He et al. (2015)) *The initialization variance σ^2 in the presence of ReLU should thus be corrected as:*

$$\sigma' = \frac{\sigma}{\sqrt{2}} \tag{15}$$

We remark that similar techniques can also be used to derive formulas for weighted ReLU Arpit et al. (2019).

5. Experiments

We conducted the following experiments to support our theoretical findings stated in the previous two sections. To cover a broad and diverse range of experiments, we trained our models on the MNIST, FashionMNIST, CIFAR10, Google SVHN and Flowers image datasets.

Implementation. All models were coded in Python using the `Tensorflow 2.4` A. et al. (2015) library.

Hessian Calculation. Hessian calculations are not currently well supported by the Tensorflow API, even in its most recent release `2.4.0`. The default implementation under `tf.hessians` does not work with fused operations, including certain loss functions such as the sparse cross-entropy used in classification GitHub (2019a); moreover, it does not support evaluating Hessian products without explicitly creating the whole Hessian which quickly leads to out-of-memory issues; batch mode is also not supported. The parallel computation of components is supported for Jacobians only as of very recently Agarwal (2019), but also does not work when composing higher-order derivatives GitHub (2019b). When implementing our approximation, we thus resort to a hybrid solution by expressing Hessians as a composition of sequential gradients which are followed by a parallelized computation of the Jacobians. We also take advantage of the fast Hessian-vector algorithm.

5.1. Correlations among Loss and Layer Gradients

As previously argued, we expect the gradient of the loss with respect to the output $\frac{\partial L}{\partial z^{(n)}}$ and the layer-to-layer gradients $\frac{\partial z^{(i)}}{z^{(i-1)}}$ to be correlated. We therefore prepared an experiment to demonstrate these correlations based on a simple 3-layer NN with Glorot's initialization scheme over the MNIST dataset. We re-ran the initialization a large number of times and used *Pearson's r correlation test*to estimate the dependencies, each using different random seeds. Our findings confirm (i) *very significant correlations* among components of the loss gradient, as well as (ii) *significant correlations* between the loss/output gradient and the output/layer gradients. The experiment is summarized in Table 1, for more details we refer to the supplementary material.

#Samples	Tested Gradients	Dependencies	Test Result
10^4	Loss/Output (diff. components)	10/10 times	avg. p-value $\approx 10^{-5}$ (strong evidence)
10^5	Loss/Output6 vs. Output6/Output5	9/10 times	avg. p-value $\approx 2 \cdot 10^{-2}$ (strong evidence)

Table 1: Correlations among the components of the back-propagation equation for a simple 3-layer NN. Dependencies tested with *Pearson's r* at 95% significance, using 10 seeds each.

5.2. Hessian at Initialization and Convergence

5.2.1. Improving Convergence by Hessian-driven Initialization

In this experiment, we compare a) the initialization approach using Theorem 1, picking the weights according to the Hessian values, and b) other standard initialization schemes.

The model is a dense 3-layer network on MNIST, with layers of $128, 128$ and 10 output units, respectively. We train for two epochs, using SGD with a step size of 0.01. To summarize, our approach gives much better results for the loss after 2 training epochs, as depicted in Table 2.

Activation	Initialization	Loss after 2 Epochs
relu	Hessian-driven stddev (Theorem 1)	0.181294
ReLu	He Normal	0.307550
ReLu	Orthogonal	0.373200
tanh	Glorot	0.356311
tanh	Orthogonal	0.375280

Table 2: Our initialization approach compared with other methods. The architecture is a 3-layer network trained on MNIST data.

To make the experiment more challenging, we trained a model with ResNet18 on Cifar-10 with 10 output units on the last Dense layer. The more detailed results of this setup, including the estimated Hessians and the loss over time, are presented in Figure 1. The initial

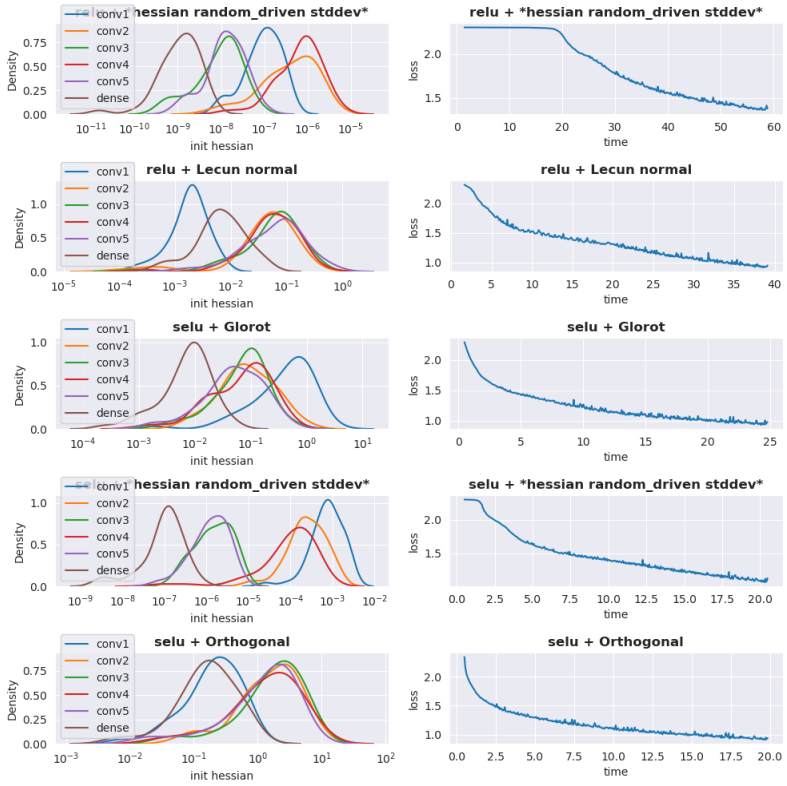

Figure 1: Hessian at initialization compared with the training loss progress.

Hessian values are evaluated on the normalized gradients, and weights are re-randomized to obtain the density plots. Compared to our approach, other schemes are under-estimating the weights.

5.2.2. Diagnosing Convergence by Hessian-driven Initialization

In this experiment, we compare a) the initialization approach using Theorem 1, picking the weights according to the Hessian values, and b) different standard deviations.

The architecture is ResNet18 on Cifar-10 with 10 output units on the last Dense layer. We train for 10 epochs, using SGD with step size of 0.01, activation function set to selu (except for the last Dense layer), initializer set to random normal distribution (with different standard deviations). The estimation of the Hessians is evaluated on 5 convolutional layers and on the last dense layer. It is possible to look at the Hessian values (with on information about the loss function) and make the right choice on which standard deviation to use. This is possible because the Hessian carries with itself second order information which captures the complexity of the loss landscape. To summarize, our approach gives a diagnostic tool which provides a guidance on how to tune the initialization schemes and allows faster convergence. More detailed results of this setup, including the estimated Hessians and the loss after different steps, are presented in Figure 2. The Hessian values explode or vanish for

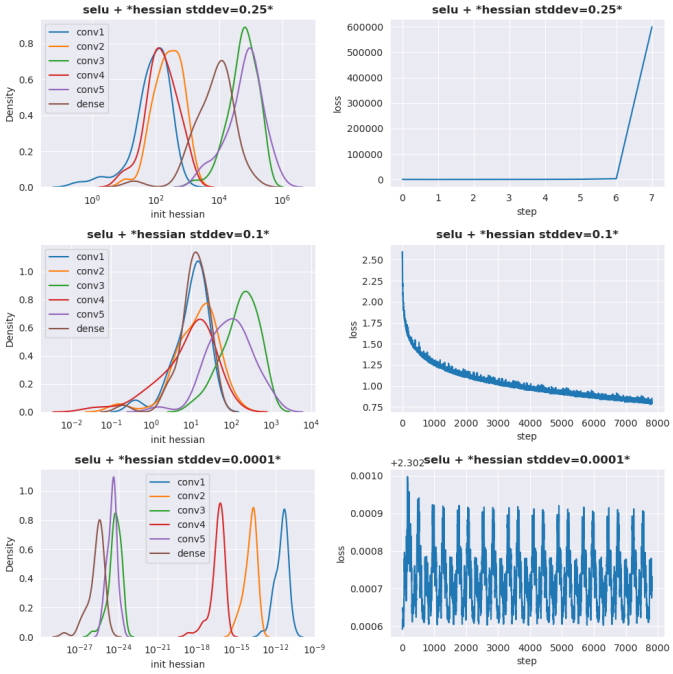

Figure 2: Hessian at initialization (distributions under initialization randomness) compared with the training loss progress.

bad standard deviation whereas for good standard deviations the density plots are almost aligned (concentrated between 10^0 and 10^1).

5.3. Good Initializers Nearly Stabilize the Hessians

In this experiment, we demonstrate that under popular initialization schemes Hessian norms are relatively small at the initialization and during training, also when estimated on batches. However, often for some of the layers Hessian norms are *significantly* smaller than 1, which signals that there is still room for improvement (as discussed in the previous section) for these schemes.

To illustrate this claim, we use the LeNet5 architecture on the CIFAR10 dataset (Figure 3) and EfficientNet architecture for other datasets: FashionMNIST Xiao et al. (2017) and SVHN (Figure 4), using tanh as activation functions and Glorot initialization. The reported Hessians' values are evaluated on normalized gradients, as before.

5.4. Error in Hessian Approximation

Theorem 1 shows that, up to terms of smaller orders, the curvature effect can be neglected. Below, we experimentally verify that the constant under the $O()$ term is indeed small.

We estimated a) the relative error in Theorem 1 at the initialization, using several initialization restarts, as shown in Figure 5 and b) the relative error in Theorem 1 during the training (computed on the batch), shown in Figure 6 For the first experiment, we used

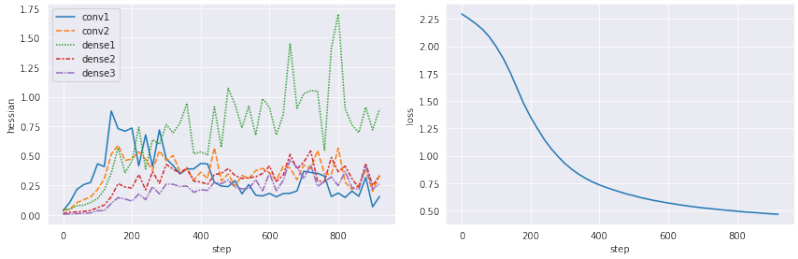

Figure 3: Hessian and loss values during training (batch estimates).

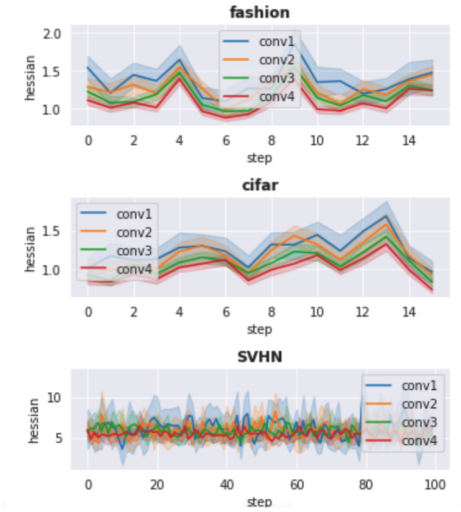

Figure 4: Hessian norm during training.

both ResNet and EfficientNet architectures on MNIST, CIFAR10 and Flowers dataset. For the second experiment, we used ResNet, EfficientNet and a simplified version of GoogleNet on CIFAR10. We trained the models for 5 epochs. For both experiments we sampled 5 layers (4 convolutional and 1 dense) to show how the hessian changes (wrt the error and the iterations), the models are initialized with selu activation function and with random distribution. The Hessians were evaluated on the corresponding gradients.

We find that the approximation is accurate at initialization, and works remarkably well during training on small batches (the relative error smaller than a small constant ensures that we capture the right order of magnitude).

6. Conclusions

We discussed how to approximate Hessians of loss functions for neural networks, and devised how to use them to gain better insights into weight initialization. The main theoretical finding is that our approach more rigorously explains a wide variety of existing initialization schemes, and is able to capture second-order effects. Besides our theoretical results, we provide a detailed empirical validation of these ideas, which demonstrates that (i) con-

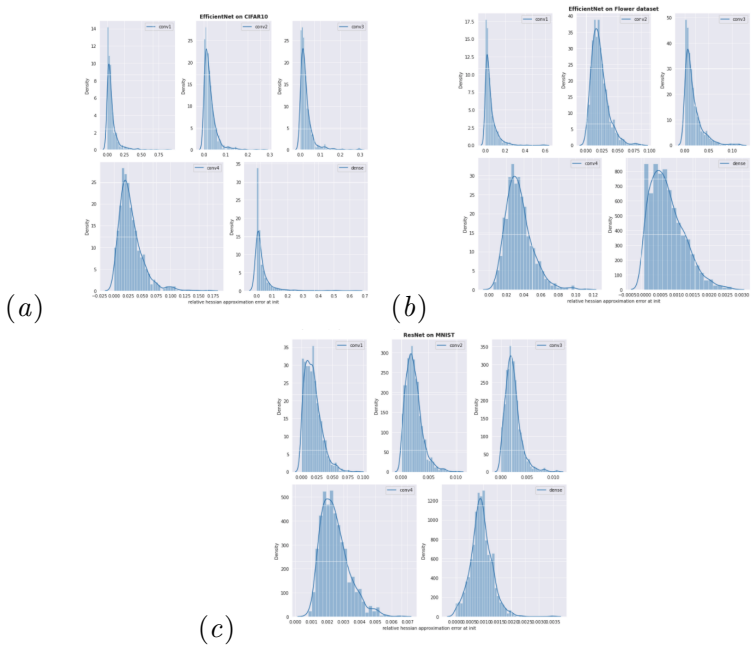

Figure 5: Our Hessian approximation at initialization.

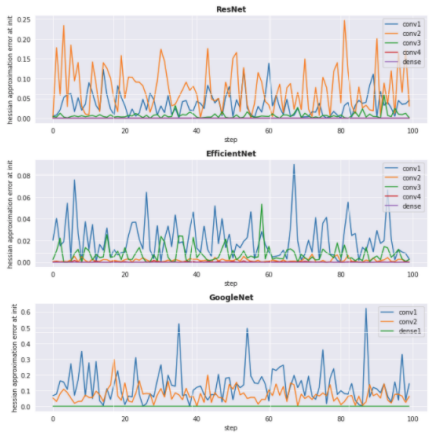

Figure 6: Our Hessian approximation during training.

sidering the Hessian norm leads to faster convergence of the learning loss than existing initialization schemes under various activation functions, and (ii) our approximation of the layers' Hessians via Jacobian products and the chain rule allows for fast (comparable to backpropagation time) computations without a noticeable impact on the learning loss.

Acknowledgments

We thank the NVIDIA AI Technology Center (NVAITC) for the fruitful discussion.

References

Martín A. et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

Ashish Agarwal. Static automatic batching in TensorFlow. In *ICML*, pages 92–101, 2019.

Devansh Arpit, Víctor Campos, and Yoshua Bengio. How to Initialize your Network? Robust Initialization for WeightNorm & ResNets. In *NeurIPS*, pages 10900–10909, 2019.

Thomas Bachlechner, Bodhisattwa Prasad Majumder, Huanru Henry Mao, Garrison W. Cottrell, and Julian J. McAuley. ReZero is All You Need: Fast Convergence at Large Depth. *CoRR*, abs/2003.04887, 2020.

David Balduzzi, Marcus Frean, Lennox Leary, J. P. Lewis, Kurt Wan-Duo Ma, and Brian McWilliams. The Shattered Gradients Problem: If resnets are the answer, then what is the question? In *ICML*, pages 342–350, 2017.

Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical Gauss-Newton optimisation for deep learning. In *ICML*, pages 557–565. PMLR, 2017.

Yann N. Dauphin and Samuel S. Schoenholz. MetaInit: Initializing learning by learning to initialize. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *NeurIPS*, pages 12624–12636, 2019.

Alan Edelman. Eigenvalues and condition numbers of random matrices. *SIAM Journal on Matrix Analysis and Applications*, 9(4):543–560, 1988.

Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. An investigation into neural net optimization via Hessian eigenvalue density. In *ICML*, pages 2232–2241. PMLR, 2019.

GitHub, 2019a. https://github.com/tensorflow/tensorflow/issues/5876.

GitHub, 2019b. https://github.com/tensorflow/tensorflow/issues/675.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, pages 249–256, 2010.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Boris Hanin and David Rolnick. How to Start Training: The Effect of Initialization and Architecture. In *NeurIPS*, pages 569–579, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *ICCV*, pages 1026–1034, 2015. doi: 10.1109/ICCV.2015.123.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

Dan Hendrycks and Kevin Gimpel. Adjusting for dropout variance in batch normalization and weight initialization. *CoRR*, abs/1607.02488, 2016.

Arthur Jacot, Franck Gabriel, and Clément Hongler. The asymptotic spectrum of the Hessian of DNN throughout training. In *ICLR*. OpenReview.net, 2020.

Vladislav Kargin et al. Products of random matrices: Dimension and growth in norm. *The Annals of Applied Probability*, 20(3):890–906, 2010.

Tom Kocmi and Ondrej Bojar. An Exploration of Word Embedding Initialization in Deep-Learning Tasks. In *ICON*, pages 56–64, 2017.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*, pages 1106–1114, 2012.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *ICML*, pages 2408–2417. PMLR, 2015.

Levent Sagun, Utku Evci, V. Ugur Güney, Yann N. Dauphin, and Léon Bottou. Empirical analysis of the Hessian of over-parametrized neural networks. *CoRR*, abs/1706.04454, 2017.

John Salvatier, Thomas V. Wiecki, and Christopher Fonnesbeck. Probabilistic programming in Python using PyMC3. *PeerJ Comput. Sci.*, 2:e55, 2016. doi: 10.7717/peerj-cs.55.

Jack W Silverstein. The spectral radii and norms of large dimensional non-central random matrices. *Stochastic Models*, 10(3):525–532, 1994.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013.

Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages 6105–6114, 2019.

Aladin Virmaux and Kevin Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. In *NeurIPS*, pages 3839–3848, 2018.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *CoRR*, abs/1708.07747, 2017.

Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel S. Schoenholz, and Jeffrey Pennington. Dynamical Isometry and a Mean Field Theory of CNNs: How to Train 10, 000-Layer Vanilla Convolutional Neural Networks. In *ICML*, pages 5389–5398, 2018.

Bing Xu, Ruitong Huang, and Mu Li. Revise Saturated Activation Functions. *CoRR*, abs/1602.05980, 2016.