Efficient Hessian-based DNN Optimization via Chain-Rule Approximation

Alessandro Temperoni*
alessandro.temperoni@uni.lu
University of Luxembourg
Esch-sur-Alzette, Luxembourg

Mauro Dalle Lucca Tosi
mauro.dalleluccatosi@uni.lu
University of Luxembourg
Esch-sur-Alzette, Luxembourg

Martin Theobald
martin.theobald@uni.lu
University of Luxembourg
Esch-sur-Alzette, Luxembourg

ABSTRACT

Learning non use-case specific models has been shown to be a challenging task in Deep Learning (DL). Hyperparameter tuning requires long training sessions that have to be restarted any time the network or the dataset changes and are not affordable by most stakeholders in industry and research. Many attempts have been made to justify and understand the source of the use-case specificity that distinguishes DL problems. To this date, *second-order optimization methods* have been partially shown to be effective in some cases but have not been sufficiently investigated in the context of learning and optimization.

In this work, we present a *chain rule for the efficient approximation of the Hessian matrix* (i.e., the second-order derivatives) of the weights across the layers of a Deep Neural Network (DNN). We show the application of our approach for weight optimization during DNN training, as we believe that this is a step that particularly suffers from the enormous variety of the optimizers provided by state-of-the-art libraries such as Keras and PyTorch. We demonstrate—both theoretically and empirically—the improved accuracy of our approximation technique and that the Hessian is a useful diagnostic tool which helps to more rigorously optimize training. Our preliminary experiments prove the efficiency as well as the improved convergence of our approach which both are crucial aspects for DNN training.

ACM Reference Format:
Alessandro Temperoni, Mauro Dalle Lucca Tosi, and Martin Theobald. 2023. Efficient Hessian-based DNN Optimization via Chain-Rule Approximation. In *6th Joint International Conference on Data Science & Management of Data (10th ACM IKDD CODS and 28th COMAD) (CODS-COMAD 2023), January 4–7, 2023, Mumbai, India.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3570991.3571042

1 INTRODUCTION

Deep Neural Networks (DNNs) have been successfully employed in a broad range of different fields to solve diverse challenging tasks. Meanwhile, the optimization of a DNN has become a field of research by itself, since classical techniques used for optimization, which are widely implemented in other fields, often turn out to be either ineffective or inefficient (or both) for DNNs due to the non-convex nature and the high dimensionality of such problems [2]. First-order methods (mostly SGD [5] and its variants) have become the standard for training DNN models. However, the full landscape of optimization problems strongly depends also on second-order effects (i.e., the curvature of the loss function), and therefore first-order methods are not the best solution as they tend to ignore dependencies among weights across the various layers of a DNN. Second-order derivatives are a valuable solution to better capture the loss curvature, but their prohibitive computational effort remains a strong limitation for the application of such alternatives to real-world problems.

In this work, we investigate the usage of a novel chain rule [6] for the approximation of the Hessian matrix across the weight layers of a DNN, thereby also leveraging second-order derivatives. We apply this strategy to improve the optimizer's update rule (either based on SGD [5] or ADAM [4]). Specifically, by plugging our chain rule into the back-propagation step of the optimizer, we allow the the gradients' Hessians to be propagated through the various layers of the DNN. In our preliminary experiments, we compare our approach against SGD and the well-known ADAM optimizer. We show that, when employing our chain rule on a text-based (Word2Vec) usecase, we are able to converge to a smaller value of the *validation loss*. In addition, our optimizer is less prone to *overfitting*, which is the problem of the majority of optimizers (especially for "non difficult" problems).

2 METHODOLOGY

Before presenting our approximated chain rule, we briefly introduce the notation which we are going to use throughout the paper. We look at a DNN as a chain of mappings of the form

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

which sequentially process an *input vector* $x = z^{(0)}$ through a number of *layers* $k = 0 \ldots n-1$. We assume that the $z^{(k)}$ 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 $w^{(0)},\ldots,w^{(n-1)}$. Now, we are ready to discuss our main contribution; in general, if $z = z(w)$ is a reparameterization, then by the chain rule for tensors [6] we have

$$\underbrace{D_w^2 L(z(w),t)}_{\text{reparameterized Hessian}} = \underbrace{D_z^2 L(z,t) \bullet D_w z(w) \bullet D_w z(w)}_{\text{linearization effect}} + D_z L(z,t) \underbrace{\bullet D_w^2 z(w)}_{\text{curvature effect}} \quad (1)$$

*Doctoral Candicate at University of Luxembourg

CODS-COMAD 2023, January 4–7, 2023, Mumbai, India

ACM ISBN 978-1-4503-9797-1/23/01.
https://doi.org/10.1145/3570991.3571042

where the bullets denote tensor dot-products. This result implies that the curvature effect contributes less than the linearization effect [6]. This is important because it allows to optimize DNNs using second-order derivatives which capture the complexity of the loss landscape. As such, we are able to approximate the Hessian with respect to the loss function in an efficient manner and with comparable computational effort to calculating the SGD. Without this contribution, which constitutes our main result, the calculation of the Hessian would be infeasible as the usual size of DNNs are of millions of parameters and consequently the weight matrices are huge (the exact calculation of each layer's Hessian is quadratic in the number of its weights).

3 PRELIMINARY EXPERIMENTS

We plug our Hessian approximation into a new update rule based on the well-established ADAM [4] optimizer available in both Keras (based on TensorFlow) and PyTorch.

Experimental setting. We trained a Word2Vec[1] model using the Tiny Shakespeare dataset[3] in three settings: (1) using SGD; (2) using the standard ADAM optimizer; (3) using ADAM adapted with our novel Hessian approximation. We used TensorFlow [1] to conduct our experiments. All sources are available on GitLab[2].

The Tiny Shakespeare dataset [3] is composed of 40,000 lines of varied Shakespeare plays. We chose it due to its limited size, which should impact the convergence of such a high-dimensional problem and make it more challenging. Thus, this limitation should accentuate the difference of how different optimizers estimate the global curvature of the loss landscape. We used 85% of the dataset for training and 15% for validation of the model. For the SGD setting, we used `learning_rate = 0.1`. Regarding the ADAM based settings, we used mostly the same hyperparameters on both of them: `CategoricalCrossentropy` to calculate the loss, `mini_batch_size = 512`, `learning_rate = 0.001`, `beta_1 = 0.99` and `beta_2 = 0.999`. Differently, we used `epsilon = 1e-4` for our setting, while we used `epsilon = 1e-7` for standard ADAM. The increase in the `epsilon` for our setting was simply to avoid gradient vanishing, which seems to happen at different values due to the Hessian approximation. We note that all the values of the other hyperparameters are the standard ones for ADAM when using TensorFlow.

Preliminary Results. Figure 1 illustrates the results of our experiments. We can compare the difference of the *validation loss* when using different optimizers, with respect to the number of epochs used to train the Word2Vec models. There are two results that we would like to highlight: (1) our ADAM adaptation achieved a smaller validation loss during the whole training; (2) our ADAM adaptation was impacted less by overfitting than ADAM and SGD.

Considering the computational time to train the Word2Vec model, our approach took 38% longer than ADAM and was over 4 times faster than SGD. It took 229s to reach minimum validation loss (3 epochs), while ADAM took 165s (3 epochs), and SGD took 999s (333 epochs). Those preliminary results are indications that the Hessian approximation may improve standard ADAM convergence in exchange of some computational efficiency. However, we note that the complexity of using our Hessian approximation still is orders of magnitude smaller than using the exact Hessian. Even so, it is still possible to observe the benefits of the second-order derivatives when comparing SGD, standard ADAM, and our approach with respect to the validation loss curves.

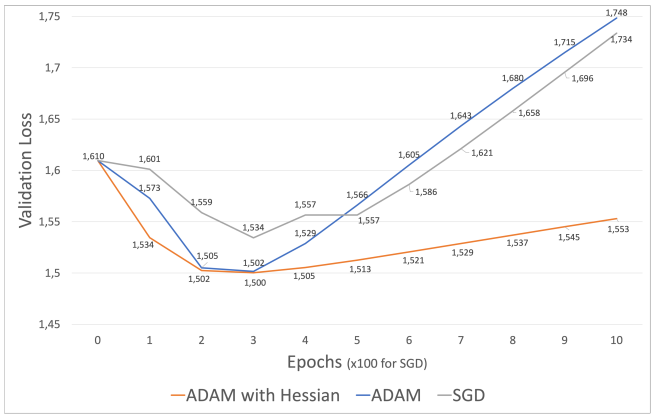

Figure 1: Comparison of SGD, standard ADAM, and our ADAM adapted with the Hessian approximation.

4 CONCLUSIONS

We discussed how to approximate the calculation of the Hessian matrix of loss functions for DNNs, presented a new chain rule applied to the optimizer's update rule, and devised on how to use it to gain better insights into training. Besides our theoretical results, we provide a preliminary empirical validation of these ideas, which demonstrates that (i) the Hessian can be used to improve the model convergence, and (ii) our approximation of the chain rule allows for fast computations without a noticeable impact on the training time. As our empirical evaluation confirmed our theoretical findings, in the future, we aim to apply our approximated chain rule to more challenging datasets and to formalize an optimized version of ADAM's update rule.

ACKNOWLEDGMENTS

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

This work is partially funded by the Luxembourg National Research Fund under the PRIDE programme (PRIDE17/12252781).

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: a system for Large-Scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*. 265–283.
[2] Prateek Jain and Purushottam Kar. 2017. Non-convex Optimization for Machine Learning. *Foundations and Trends in Machine Learning* (2017).
[3] Andrej Karpathy. 2015. char-rnn. https://github.com/karpathy/char-rnn.
[4] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2015).
[5] Herbert E. Robbins. 1951. A Stochastic Approximation Method. *Annals of Mathematical Statistics* (1951).
[6] Maciej Skorski, Alessandro Temperoni, and Martin Theobald. 2021. Revisiting Weight Initialization of Deep Neural Networks.. In *Proceedings of Machine Learning Research*, Neil Lawrence (Ed.), Vol. 157. 16 pages.

[1] https://www.tensorflow.org/tutorials/text/word2vec
[2] https://gitlab.uni.lu/atemperoni/hessian_optimizer