GRADIENT CLIPPING FOR AUTOMATIC HYPERPARAMETER TUNING

Jesse Huang, Isaac Bannerman, Daniel Savidge, Daniel Chen

ABSTRACT

Gradient descent and its variations are a very popular and heavily used optimization algorithm used while training machine learning models. Within gradient descent, hyperparameters such as the step size, or learning rate, greatly affect the performance and convergence of the model and thus have a great impact. One way to determine these hyperparameters is by calculating a hypergradient to determine the optimal parameters from sub-optimal values. However, this method is prone to exploding gradients, and thus cannot handle initially high-valued hyperparmeters. In this study, we enhance the methodology to effectively handle high-valued initial hyperparameter values by employing the gradient clipping technique.

Our approach involves the incorporation of the normed-based averaged gradient clipping into the hyperoptimizer's secondary optimization process. This modification, integrated into existing frameworks, contributes to mitigating the adverse effects of exploding gradients and improving overall system performance. The empirical evaluation, conducted on the MNIST dataset, demonstrates the efficacy of this enhancement, particularly in scenarios where traditional methods struggle with large hyperparameter values.

1 INTRODUCTION AND BACKGROUND

Modern machine learning models heavily rely on optimization algorithms like Stochastic Gradient Descent (SGD) or its variants. Among the crucial hyperparameters, the step size and momentum are pivotal for algorithmic performance. However, determining optimal values for these hyperparameters pose a significant challenge. Building on the work of Almeida et al. (1999), Baydin et al. (2017) proposed that one can apply a second level gradient descent algorithm to help choose an optimum step size. They did this by manually computing the so-called hypergradients that is the derivative of the optimization problem (typically the loss function) with respect to the step size. This methodology proved successful and helped to bring suboptimal step sizes to optimal values that aided convergence.

Quite recently, Chandra et al. (2022), building on the work of Baydin et al. (2017) computed this hypergradients using reverse mode automatic differentiation. This allowed them to extend the work to optimization of parameters such as the momentum coefficient more easily. Lastly, they also demonstrated that their approach can be applied to tall towers of optimizers to make their hyperparameter optimization algorithm more robust to bad initial guess. Despite the great success achieved by Chandra et al. (2022), their methodology of optimizing the hyper gradients is based on gradient descent and is susceptible to typical challenges of gradient descent: divergence due to high initial learning rate. They reported that using relatively high initial guess caused the algorithm to be unstable and fail to converge to an optimum hyperparameter. In this work we seek to rectify this issue by a gradient clipping algorithm.

The structure of this report unfolds as follows: section 2 expounds on the research gap and section 3 describes in detail the gradient clipping method used. Section 4 shows the results of applying the gradient clipping method to various optimizer stack; various work related to hyperparameter optimization is described in section 5. Finally, we describe possible ways of extending this work in section 6 and draw the conclusions in section 7.

2 PROBLEM DESCRIPTION

As discussed in Chandra et al. (2022), a notable limitation of hyperoptimizers is their inability to effectively handle initial hyperparameters that are set excessively high. The reason for this issue is that when hyperparameters are initialized at a higher magnitude, we have exceedingly large values for gradients and thus any subsequent updates will overshoot optimal values for the hyperparameters and prevent the model from converging effectively. Divergence in the early stages of the method prevent the hyperoptimizer from effectively exploring the hyperparameter space and refining the choices for optimal performance.

In the execution of the methodology proposed by Chandra et al. (2022), hyperoptimization is conducted through stacks. To ensure clarity regarding the transmission of hyperparameters within the stack, we propose the following distinction: the optimizer positioned at the commencement of each stack may be regarded as the primary optimizer, while the subsequent hyperoptimizer(s) can be referred to as secondary optimizer(s). For example, for stack Adam / SGD, Adam can be referred to as the primary optimizer, and SGD can be referred to as the secondary optimizer for that stack.

In the examination of the original hyperoptimizer implementation by Chandra et al. (2022), we observe a noteworthy phenomenon concerning the SGD / SGD stack (Chandra et al., 2022), the Adam / SGD stack (Chandra et al., 2022), the AdaGrad / SGD stack (Chandra et al., 2022), and the RMSProp / SGD stack (Chandra et al., 2022). Specifically, when a relatively large learning rate is supplied to the secondary optimizer, the system encounters challenges in effectively learning hyperparameters for the primary optimizer, resulting in suboptimal model fitting to the data. This discrepancy is evident in a corresponding escalation of test error. Our empirical investigations, conducted on the MNIST and CIFAR-10 datasets, reveal a significant increase in test error when the learning rate for the SGD secondary optimizer exceeds certain values, which varies depending on the optimizer stack, neural network architecture, and the task.

3 SOLUTION

In this study, we focus on alleviating the divergent behavior induced by substantial learning rates for large learning rates in SGD secondary optimizer for various two optimizer stacks. To address this, we introduced normed-based averaged gradient clipping, a technique outlined by Pascanu et al. (2013) for the SGD secondary optimizer, aiming to mitigate the potential adverse effects of exploding gradients in the secondary optimizer, which corresponds to the gradient of the hyperparameters in the primary optimizer. We found that we could remove the threshold hyperparameter as described in the gradient clipping algorithm in Pascanu et al. (2013) for the clipping. As suggested by Pascanu et al. (2013), an average of the norms of the gradient over a reasonably large number of updates for the threshold can empirically result in convergence.

Therefore, instead of a hyperparameter for the threshold, we used the cumulative average of the gradient norm across t steps with the norm of the unclipped gradient at step t, with each step of the optimization algorithm across a minibatch during model training. Then, the gradient norm average at step t is recomputed with the clipped gradient and used for the clipping at the next step. For each step in the optimization algorithm, the norm-based gradient clipping algorithm utilized is as follows:

if $t = 1$ **then**
  $\theta_t \leftarrow ||\mathbf{g}_t||_2$
else
  $\theta_t \leftarrow \frac{(\theta_{t-1} * (t-1)) + ||\mathbf{g}_t||_2}{t}$
end if
$\mathbf{g}'_t \leftarrow \mathbf{g}_t * \min\left(1, \frac{\theta_t}{||\mathbf{g}_t||_2}\right)$
$\mathbf{g}_t \leftarrow \mathbf{g}'_t$
$\theta'_t \leftarrow \frac{(\theta_{t-1} * (t-1)) + ||\mathbf{g}'_t||_2}{t}$
$\theta_t \leftarrow \theta'_t$

where $\mathbf{g}_t$ denotes the unclipped gradient of a hyperparameter at step t, $\mathbf{g}'_t$ denotes the clipped gradient of a hyperparameter at step t, θ_t denotes the cumulative average of the gradient norms at the t-th step computed with the unclipped gradient at step t, and θ'_t denotes the cumulative average of the gradient norms at the t-th step computed with the clipped gradient at step t.

This enhancement was seamlessly integrated into the existing implementation proposed by Pascanu et al. (2013), contributing to overall system improvement. As an addition to the original source code, the clipping algorithm was added as a feature for the SGD class, the class that corresponds to the SGD optimizer. Gradient norm tracking of the hyperparameter gradients was also added as a feature for the optimizer classes so that those values can be debugged.

Importantly, our implementation of normed averaged gradient clipping is designed to be user-friendly, with no numerical hyperparameter, with the threshold for gradient clipping computed automatically with a running average. Instead, the method contains only one boolean hyperparameter: whether to turn it on or off. The client code can enable or disable this feature for the respective optimizer using a boolean flag. We documented and compared the test errors for the SGD / SGD stack, the Adam / SGD stack, the AdaGrad / SGD stack, and the RMSProp / SGD stack with and without the application of gradient clipping.

In evaluating the method proposed by Chandra et al. (2022), it becomes evident that their approach is upheld primarily through empirical evidence rather than formal theoretical guarantees. Their work demonstrates practical improvements in robustness and performance of hypergradient calculation, with a focus on reduced sensitivity to initial hyperparameters. In contrast, our algorithm incorporates gradient clipping, a technique that offers distinct theoretical strengths. In our implementation of gradient clipping, we are able to limit the size of the calculated hypergradients to a manageable range, which ensures the updates to our model parameters remain small and controlled. This leads to improved training stability and can help in achieving more consistent convergence which mitigates the risk of exploding gradients, particularly under the challenging conditions presented by large initial hyperparameter values.

4 Experimental Design and Results

For our results, in order to see the effects of gradient clipping with the large hyperparameter problem as described by Chandra et al. (2022) for their hyperoptimizer method, we chose the MNIST and CIFAR-10 datasets, used the same MLP neural network architecture as implemented for that paper's results for the MNIST dataset, used a CNN architecture for the CIFAR-10 dataset, used the same automatic differentiation method (the only change in terms of the optimizer is use of the proposed gradient clipping methodology), and followed a similar experimental design. Statistics were computed by training and testing the neural networks with the respective optimizer stacks for 3 trials and taking the average.

4.1 Optimizer Stacks and Gradient Clipping with MNIST

For the MNIST task, we used a fully-connected neural network with a 128-neuron hidden layer and tanh for the activation functions, with a standard 256 batch size (Chandra et al., 2022). We tested the effect of the proposed gradient clipping methodology on test error for the following optimizer stacks used as the optimizer of the neural network: SGD / SGD, Adam / SGD, AdaGrad / SGD, and RMSProp / SGD. For each optimizer stack, the same neural network architecture was used. The neural networks with Adam / SGD and RMSProp / SGD stacks were trained for 5 epochs, and the networks with SGD / SGD and AdaGrad / SGD stacks were trained for 30 epochs and no more than that due to the possibility of overfitting (Chandra et al., 2022).

Tables 1 to 4 depict the result of the use of the gradient clipping methodology for the SGD / SGD, Adam / SGD, AdaGrad / SGD, and RMSProp / SGD stacks, respectively, for this task. For each optimizer stack, various values of the learning rate α were passed into the SGD secondary optimizer. No momentum was used for the SGD secondary optimizer, and default hyperparameter values were used for the primary optimizers. For the SGD primary optimizer, the following default hyperparameter values were used: $\alpha = 0.01$ and $\mu = 0$. For the Adam primary optimizer, the following default hyperparameter values were used: $\alpha = 0.001$, $\beta_1 = 0.9$, and $\beta_2 = 0.999$. For the

AdaGrad primary optimizer, the following default hyperparameter value was used: $\alpha = 0.01$. For the RMSProp primary optimizer, the following default hyperparameter values were used: $\alpha = 0.01$ and $\gamma = 0.99$.

SGD / SGD(α)		
α	average test error, no gradient clipping	average test error, gradient clipping
0.01	4.90%	5.06%
0.05	6.18%	6.04%
0.1	6.97%	6.68%
0.5	7.01%	7.27%
1	6.78%	6.91%
5	5.31%	5.97%
10	4.72%	5.20%
50	**7.81%**	**3.91%**
75	**38.05%**	**15.92%**
100	**25.26%**	**13.01%**
250	22.75%	40.09%
500	70.28%	72.26%
1000	**80.43%**	**58.04%**

Table 1: Effect of gradient clipping with the SGD / SGD(α) stack on average test error for the MNIST task. Baseline with SGD alone resulted in 8.96% average test error. The bolded rows are where gradient clipping reduced average test error by at least 1%.

Adam / SGD(α)		
α	average test error, no gradient clipping	average test error, gradient clipping
10^{-5}	3.14%	3.23%
$5 * 10^{-5}$	3.09%	3.21%
10^{-4}	3.05%	3.06%
$5 * 10^{-4}$	3.59%	3.20%
$\mathbf{10^{-3}}$	**4.71%**	**3.35%**
$\mathbf{5 * 10^{-3}}$	**35.89%**	**16.6%**
$\mathbf{10^{-2}}$	**79.57%**	**23.2%**
$\mathbf{5 * 10^{-2}}$	**86.8%**	**84.27%**
10^{-1}	76.17%	86.44%
$5 * 10^{-1}$	83.16%	82.36%

Table 2: Effect of gradient clipping with the Adam / SGD(α) stack on average test error for the MNIST task. Baseline with Adam alone resulted in 4.63% average test error. The bolded rows are where gradient clipping reduced average test error by at least 1%.

AdaGrad / SGD(α)		
α	average test error, no gradient clipping	average test error, gradient clipping
0.01	6.92%	6.71%
0.05	7.02%	7.16%
0.1	6.75%	7.04%
0.5	5.70%	5.99%
1	**7.86%**	**5.19%**
2.5	**7.34%**	**4.11%**
5	**57.91%**	**3.49%**
7.5	**36.62%**	**30.33%**
10	27.08%	49.9%
50	**64.51%**	**20.79%**
100	**90.19%**	**49.04%**

Table 3: Effect of gradient clipping with the AdaGrad / SGD(α) stack on average test error for the MNIST task. Baseline with AdaGrad alone resulted in 7.38% average test error. The bolded rows are where gradient clipping reduced average test error by at least 1%.

RMSProp / SGD(α)		
α	average test error, no gradient clipping	average test error, gradient clipping
10^{-5}	4.43%	4.26%
$5 * 10^{-5}$	3.47%	3.84%
10^{-4}	3.36%	3.55%
$5 * 10^{-4}$	4.26%	3.75%
10^{-3}	5.16%	4.86%
$5 * 10^{-3}$	6.16%	6.12%
10^{-2}	6.36%	6.52%
$\mathbf{2.5 * 10^{-2}}$	**88.80%**	**6.14%**
$\mathbf{5 * 10^{-2}}$	**90.66%**	**9.72%**
$7.5 * 10^{-2}$	90.03%	90.03%
$\mathbf{10^{-1}}$	**89.17%**	**64.56%**
$5 * 10^{-1}$	90.2%	90.02%

Table 4: Effect of gradient clipping with the RMSProp / SGD(α) stack on average test error for the MNIST task. Baseline with RMSProp alone resulted in 3.91% average test error. The bolded rows are where gradient clipping reduced average test error by at least 1%.

For relatively low learning rates for the respective optimizer stacks for the MLP for this task, $\alpha \leq \approx 10$ for SGD / SGD(α), $\alpha \leq \approx 5 * 10^{-4}$ for Adam / SGD(α), $\alpha \leq \approx 0.5$ for AdaGrad / SGD(α), and $\alpha \leq \approx 10^{-2}$ for RMSProp / SGD(α), gradient clipping had negligible impact on the performance of the optimizer stack. However, for learning rates that are considerably large for the secondary optimizer of the stacks, $\alpha > \approx 10$ for SGD / SGD(α), $\alpha > \approx 5 * 10^{-4}$ for Adam / SGD(α), $\alpha > \approx 0.5$ for AdaGrad / SGD(α), and $\alpha > \approx 10^{-2}$ for RMSProp / SGD(α), gradient clipping mitigated deterioration of the model's performance until $\alpha > \approx 100$ (SGD / SGD(α)), $\alpha > \approx 5 * 10^{-2}$ (Adam / SGD(α)), $\alpha > \approx 7.5$ (AdaGrad / SGD(α)), and $\alpha > \approx 5 * 10^{-2}$ (RMSProp / SGD(α)), in which gradient clipping may or may not decrease test error. For values of α in which gradient clipping reduced test error by at least 1%, the reduction ranged from anywhere between a few percentage points to around 80%.

4.2 Optimizer Stacks and Gradient Clipping with CIFAR-10

For the CIFAR-10 task, we used a CNN with two 3 by 3 kernel 16 filter convolutional layers (stride of 1, padding of 0) and relu activations, followed by the same fully-connected layers and tanh function as with the neural network for MNIST in our experimentation. We used a standard 256 batch size (Chandra et al., 2022). We conducted standard data normalization: we normalized each channel by the respective mean and standard deviation of the channels from the train dataset. We tested the effect of the proposed gradient clipping methodology on test error for the Adam / SGD

Adam / SGD(α)		
α	average test error, no gradient clipping	average test error, gradient clipping
10^{-9}	39.26%	39.87%
10^{-8}	38.76%	39.12%
10^{-7}	40.02%	39.26%
$\mathbf{10^{-6}}$	$\mathbf{42.17\%}$	$\mathbf{40.01\%}$
$\mathbf{10^{-5}}$	$\mathbf{45.95\%}$	$\mathbf{43.52\%}$
$\mathbf{10^{-4}}$	$\mathbf{46.96\%}$	$\mathbf{43.99\%}$
$2.5 * 10^{-3}$	89.82%	90.49%
$5 * 10^{-3}$	90.00%	90.00%
$\mathbf{10^{-3}}$	$\mathbf{90.00\%}$	$\mathbf{73.88\%}$
10^{-2}	90.00%	90.00%

Table 5: Effect of gradient clipping with the Adam / SGD(α) stack on average test error for the CIFAR-10 task. Baseline with Adam alone resulted in 39.65% average test error. The bolded rows are where gradient clipping reduced average test error by at least 1%.

optimizer stack as the optimizer of the neural network. As with MNIST, the Adam / SGD optimizer network was trained for 5 epochs.

Table 5 depicts the result of the use of the gradient clipping methodology for the Adam / SGD stack for this task. For the optimizer stack, various values of the learning rate α were passed into the SGD secondary optimizer. The same default values as with the MNIST task for the Adam primary optimizer was used: $\alpha = 0.001$, $\beta_1 = 0.9$, and $\beta_2 = 0.999$.

A similar pattern was seen for Adam / SGD(α) with the CNN for the CIFAR-10 task as with the optimizer stacks with the MLP for the MNIST task. For relatively low learning rates for the Adam / SGD(α) optimizer stack for the neural network for this task, $\alpha \leq \approx 10^{-7}$, gradient clipping had negligible impact on the performance. However, for learning rates that are considerably large for Adam / SGD, $\alpha > \approx 10^{-7}$, gradient clipping mitigated deterioration of the model's performance until $\alpha > \approx 10^{-4}$, in which gradient clipping may or may not decrease test error. For values of α in which gradient clipping reduced test error by at least 1%, the reduction was around 2% to 3%.

5 Related Work

Hyperparameter tuning plays a pivotal role in training of machine learning models across a multitude of applications. When utilizing machine learning models, it is necessary to optimize different parameters and hyperparameters that can be critical to the performance of the models. Among the various hyperparameters, the learning rate stands out as one of the most crucial to optimize, particularly for gradient-descent based algorithms (Bengio, 2012)

To enhance the performance of machine learning models, numerous strategies including learning rate scheduling (Smith, 2017) and Bayesian optimization methods (Snoek et al., 2012) have been employed to tune the learning rates. Additionally, heuristics have been employed in certain instances for the fine-tuning of the learning rate. Notably, all these methods involve adjusting the learning rate after some number of iterations, or in the case of heuristics, after several rounds of training based on the same data set.

A seminal contribution to this domain was made by Almeida et al. (1999), who, to the best of our knowledge, were the first to propose a method for updating the learning rate at each iteration of the optimization process. Their proposed methodology demonstrated success when applied to state-of-the-art gradient-based algorithms of that era, such as Stochastic Gradient Descent (SGD) and gradient descent. Subsequently, Baydin et al. (2017) inspired by their work, extended this methodology to modern machine optimization techniques, including the widely used Adam optimizer. In their methods, Baydin et al. (2017), manually computed the hypergradients, derivatives of the optimization problem with respect to hyperparameters, successfully bringing suboptimal step sizes to optimal values.

Chandra et al. (2022), further built upon this work by leveraging reverse mode automatic differentiation for hypergradient computation, making the optimization of additional parameters more accessible. Despite the success of the hypergradient based optimization, they acknowledged the presence of instability and divergence of the solution when handling large initial hyperparameters.This issue is fairly common throughout machine learning and is known as the exploding gradient problem, where a step size might be too large for the optimization of further layers in a deep network. Exploding gradients are further defined by Philipp et al. (2018) in which they are qualified with a gradient scale coefficient which when well approximated by an exponential function indicates exploding gradients. A solution is suggested by Pascanu et al. (2013) where the norm of a gradient is rescaled every time it exceeds a certain threshold.

6 Future Work

The results presented in this study primarily focused on the MNIST dataset and fully connected neural networks. The next phase of the work will focus on the empirical evaluation of the method on other combinations of optimizers to ascertain the robustness of the methodology. Evaluating the robustness of the normed-based averaged gradient clipping technique across different optimization stacks can provide valuable insights into its generalizability and effectiveness for gradient parameter search.

7 Conclusion

In this paper, we have looked at the effect of gradient clipping on large hyperparameters in gradient descent. We have investigated the results of various optimizer stacks with stochastic gradient descent as the secondary optimizer, SGD / SGD, Adam / SGD, AdaGrad / SGD, and RMSProp / SGD. With an MLP on the MNIST dataset and a CNN on the CIFAR-10 dataset, we have shown that while there is very little impact when using small learning rates, there is a noticeable improvement of test error as it increases. The incorporation of the norm-based gradient clipping into the methodologies proposed by Chandra et al. (2022), can significantly improve the optimality of hyperparameters, overcoming the challenges posed by initially large values.

References

Luís B Almeida, Thibault Langlois, José D Amaral, and Alexander Plakhov. Parameter adaptation in stochastic optimization. In *On-line learning in neural networks*, pp. 111–134. 1999.

Atilim Gunes Baydin, Robert Cornish, David Martinez Rubio, Mark Schmidt, and Frank Wood. Online learning rate adaptation with hypergradient descent. *arXiv preprint arXiv:1703.04782*, 2017.

Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade: Second Edition*, pp. 437–478. Springer, 2012.

Kartik Chandra, Audrey Xie, Jonathan Ragan-Kelley, and Erik Meijer. Gradient descent: The ultimate optimizer. *NeurIPS*, 2022. URL `https://arxiv.org/abs/1909.13371`.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pp. 1310–1318. Pmlr, 2013.

George Philipp, Dawn Song, and Jaime Carbonell. The exploding gradient problem demystified - definition, revalence, impact, origin, tradeoffs, and solutions. In *International Conference on Learning Representations*, 2018. URL `https://arxiv.org/pdf/1712.05577.pdf`.

Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pp. 464–472. IEEE, 2017.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.