Analysis of the Effectiveness of Hypergradient Descent on New Optimization Algorithms

Jialiang Sun **Jierui Zhu** **Leyi Wang**
Department of Computer Science, University of Toronto
{jialiang.sun, jierui.zhu, aw.wang}@mail.utoronto.ca

Abstract

In applications, applying gradient-based algorithms require tuning their hyperparameters, which is time-consuming and error-prone. To address this, previous work has shown that hyperparameters can be updated alongside the other parameters via "hypergradients" - the gradient of the loss with respect to the hyperparameter. Extending, a recent paper called "Gradient Descent: The Ultimate Optimizer" [2] develops a way to automatically compute hypergradients in a way that's efficient, modular, and scalable. In this project, We reproduce the results of the paper, combine their methodology with two new promising optimizers AdaBelief [8] and AMSGrad [7], benchmark their performance on UNet [5] on the CIFAR-10 dataset, and offer a qualitative analysis of the results.[1]

1 Introduction

Variants of SGD, such as RMSProp [6], AdaGrad [4], and Adam [3], have established themselves with a proven record of desirable theoretical properties, empirical performance, and computational efficiency. However, the learning rate α is often fixed prior to GD, which might be problematic in complex optimization landscapes. At different stages, one may prefer different learning rates, which inspires updating the learning rate itself every iteration. Further, we are not limited to updating only the learning rate. In newer optimizers involving other hyperparameters, such as β_1 and β_2 that control the exponential decay rates of the gradient averages. We aim to investigate the modular method to automatically update these hyperparameters on each iteration and to highlight its strengths and weaknesses.

2 Related Work

2.1 Notation

Following the conventions of [3], we define $f(\theta) \in \mathbb{R}$ for parameter $\theta \in \mathbb{R}^d$ to be the loss function, $\prod_{\mathcal{F},\mathcal{M}}(y) = \arg\min_{x \in \mathcal{F}} ||M^{\frac{1}{2}}(x-y)||$ to be the projection of y onto a convex & feasible set $\mathcal{F}$; g_t to be the gradient at step t; m_t to be the exponential moving average(EMA) of g_t; v_t to be the EMA of g_t^2; s_t to be the EMA of $(g_t - m_t)^2$; α to be the learning rate; ϵ to be a small number, and β_1, β_2 to be the smoothing parameters.

2.2 Gradient Descent: The Ultimate Optimizer

In this paper [2], Chandra et al. develop the key insight from [1]. At the expense of introducing a new hyperparameter called the hyper-stepsize, one also updates the learning rate via computing

[1] Code is available at https://github.com/JackSun72/CSC413-Final-Project

Preprint. Under review.

$\frac{\partial f(\theta_t)}{\partial \alpha_t} = \frac{\partial f(\theta_t)}{\partial \theta_t} \frac{\partial \theta_t}{\partial \alpha_t}$. However, computing this by hand is tedious and unscalable. The remarkable extension is the applying automatic differentiation for gradients to modularize the process, which can generalize to find better values for hyperparameters not limited to the learning rate.

2.3 AMSGrad

Wang et al. proposed a moving average variant, AMSGrad [7], to guarantee convergence while maintaining the practical benefits of Adam [3]. Mathematically, they identify the undesirable property that the matrix $\Gamma_{t+1} = \sqrt{v_{t+1}} - \sqrt{v_t}$ may be indefinite in Adam, which leads to undesirable convergence behavior. AMSGrad ensures that Γ_t remains positive semi-definite by maintaining the maximum of all v_t values up to the current time step and using this maximum value to normalize the running average of the gradient. This approach results in a non-increasing step size, avoiding the pitfalls of Adam and enhancing overall performance. Comparisons with Adam and AMSGrad on logistic regression, feed-forward neural network, and other architectures highlight AMSGrad's ability to maintain a more stable learning rate, leading to improved performance in practice.

2.4 AdaBelief

Contrary to optimizers in the SGD family, adaptive optimizers like Adam, while converging faster, don't generalize as well on complex tasks because they don't bounce out of local extrema as well. In pursuit of an optimizer that enjoys fast convergence and strong generalization, Zhuang et al. propose AdaBelief [8], which they purport to have those two properties, and also training stability in complex settings. The idea is to modify Adam by the way they scale the exponential moving average (EMA) m_t. Instead of dividing by the root of the EMA of g_t^2, AdaBelief divides by the root of the EMA of $(g_t - m_t)^2$. Intuitively, we think of m_t as a prediction of the gradient. Hence, if the observed g_t deviates too much from m_t, then we have little "belief," hence lowering the step-size as we divide by a larger number, and vice versa. It scales the update direction by the change in gradient, which means it considers curvature information, aiding its performance.

3 Method

In addition to following the usual update rules (see Algorithms 1 & 2) for the above optimizers, we implement hyperoptimization on them as outlined in [2]: detaching the parameters and their gradients from the computation graph at each iteration, detaching the parents of the hyperparameters, and "clamping" the hyperparameters into a certain range if need be. In the Torch library, we simply call `.detach()` on the variables we need to remove from the graph. This way, backpropagation deposits the gradient with respect to each hyperparameter, giving a fully modular hyperoptimization algorithm.

To evaluate, we first experiment on the MNIST data set using a one-layer fully-connected neural network with the hidden size 128 to compare with the results summarized in [2], where we use the same hyperparameters as them for uniformity. Then, we test our optimizers on CIFAR-10 using UNet [5], comparing the performance and the computation time with and without hyperoptimization.

Algorithm 1 Hyper-Optimized AMSGrad

Initialize: θ_0, $m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $\hat{v}_0 \leftarrow 0$, $t \leftarrow 0$
while θ_t not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla f_t(x_t)$
  $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$
  $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$
  $\hat{v}_t \leftarrow \max(\hat{v}_{t-1}, v_t)$
  Update
    $\theta_{t+1} \leftarrow \prod_{\mathcal{F}, \sqrt{\hat{v}_t}} (\theta_t - \frac{\alpha \cdot m_t}{\sqrt{\hat{v}_t}})$
end while

Algorithm 2 Hyper-Optimized AdaBelief

Initialize: $\theta_0, m_0 \leftarrow 0, s_0 \leftarrow 0, t \leftarrow 0$
while θ_t not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  $s_t \leftarrow \beta_2 s_{t-1} + (1 - \beta_2)(g_t - m_t)^2 + \epsilon$
  Bias Correction
    $\hat{m}_t \leftarrow \frac{m_t}{1 - \beta_1^t}, \hat{s}_t \leftarrow \frac{s_t}{1 - \beta_2^t}$
  Update
    $\theta_t \leftarrow \prod_{\mathcal{F}, \sqrt{\hat{v}_t}} (\theta_{t-1} - \frac{\alpha \hat{m}_t}{\sqrt{\hat{s}_t + \epsilon}})$
end while

4 Results

In this section, we present the results of carrying out our methods section. After reproducing the results of [2] on MNIST, we experiment with different combinations also involving AMSProp [7] and AdaBelief [8] on both the MNIST and CIFAR-10 data sets. We concisely summarize and offer an analysis of our results.

4.1 Reproduction of Results

Table 1 summarizes our attempt at the reproduction of results in [2]. We produced similar results for Adagrad, Adam, and RMSProp, but our hyperoptimized SGD test errors hover around 8% compared to the 5% in [2]. We attribute this to the highly noisy nature of SGD. Nonetheless, we do evidently see that the hyperoptimized versions of each optimizer outperforms its not-hyperoptimized counterpart, as claimed in [2]. In addition, we also recorded the computation time of each combination, observing that on MNIST, the computation time of hyperoptimized optimizers does not meaningfully differ from that of the baseline.

Optimizer / Hyper	Time(s)	Test Error
SGD	41.21	13.45%
SGD / SGD	39.17	7.81%
SGD / Adam	39.71	8.46%
SGD / Adagrad	39.31	8.21%

(a) MNIST with SGD

Optimizer / Hyper	Time(s)	Test Error
Adagrad	39.82	10.46%
Adagrad / SGD	39.92	7.82%
Adagrad / Adadelta	40.29	7.57%

(b) MNIST with AdaGrad

Optimizer / Hyper	Time(s)	Test Error
Adam	43.36	4.66%
Adam / SGD	43.69	3.02%
Adam / Adam	45.04	3.14%
Adam$^{\alpha}$ / SGD	40.05	2.93%
Adam$^{\alpha}$ / Adam	41.05	2.99%

(c) MNIST with Adam

Optimizer / Hyper	Time(s)	Test Error
RMSProp	41.75	6.25%
RMSProp / SGD	41.66	3.10%
RMSProp / RMSProp	42.38	3.23%
RMSProp$^{\alpha}$ / SGD	40.25	3.68%
RMSProp$^{\alpha}$ / RMSProp	40.65	3.18%

(d) MNIST with RMSProp

Table 1: Hyperoptimization Results for Different Optimizer Combinations on MNIST. Note optimizer$^{\alpha}$ means hyperoptimization is done only on α.

4.2 Experiments

Applying different combinations of AMSGrad and AdaBelief to MNIST, we find that they take similar training time as above. AdaBelief strongly benefits from the SGD hyperoptimizer, achieving a test error second to only to the Adam$^{\alpha}$ optimizer with SGD hyperoptimization. However, it benefits less so from Adam. Interestingly, the baseline AMSGrad optimizer is already strong, and hyperoptimization with SGD actually hurts performance. Hence, hyperoptimization need not always improve performance.

Optimizer / Hyper	Time(s)	Test Error
AdaBelief	45.39	6.61%
AdaBelief / SGD	44.96	2.98%
AdaBelief / Adam	45.68	4.43%

(a) MNIST with AdaBelief

Optimizer / Hyper	Time(s)	Test Error
AMSGrad	42.72	3.70%
AMSGrad / SGD	42.78	3.93%
AMSGrad / Adam	44.45	3.31%

(b) MNIST with AMSGrad

Table 2: Hyperoptimization Results for AdaBelief & AMSGrad on MNIST.

As a richer dataset than MNIST, experiments on CIFAR-10 demonstrate more diverse results, as summarized in Table 3. In the UNet architecture, the combination of Adam and its variants performed better than SGD-based combinations. The lowest validation error was achieved by AdaBelief/AdaBelief at 38.8%, closely followed by Adam/AMSGrad at 38.1%.

Optimizer / Hyper	Time(s)	Val. Error
SGD	32.04	58.8%
SGD / SGD	32.22	58.5%
SGD / Adam	34.81	57.0%
SGD / AMSgrad	33.80	55.1%

(a) UNet on CIFAR-10 with SGD

Optimizer / Hyper	Time(s)	Val. Error
Adam	79.64	39.3%
Adam / Adam	82.53	38.4%
Adam / AMSGrad	83.88	38.1%

(b) UNet on CIFAR-10 with Adam

Optimizer / Hyper	Time(s)	Val. Error
AdaBelief	96.39	39.0%
AdaBelief / SGD	84.28	39.1%
AdaBelief / Adam	123.63	39.3%
AdaBelief / AdaBelief	98.82	38.8%

(c) UNet on CIFAR-10 with AdaBelief

Optimizer / Hyper	Time(s)	Val. Error
AMSGrad	66.09	40.4%
AMSGrad / SGD	66.74	40.2%
AMSGrad / Adam	68.80	39.7%
AMSGrad / AMSGrad	74.15	39.3%

(d) UNet on CIFAR-10 with AMSGrad

Table 3: Hyperoptimization Results for Different Optimizer Combinations for UNet on CIFAR-10.

4.3 Analysis

We make several key observations. First, the optimizer combined with its hyperoptimizer generally achieved the best validation accuracy, demonstrating an improvement of approximately 1% compared to the original optimizer. It indicates the potential benefits of using hyperoptimization for fine-tuning the performance of optimizers. Second, the choice of hyperoptimizer can drastically impact the performance. In some cases, the best hyperoptimizer is SGD, while it is AMSGrad in others. The choice of hyperoptimizer adds complexity to empirical use since finding the optimal one is problem-dependent and requires experimentation, so it is effectively another hyperparameter. Lastly, the use of the `.detach()` function is proved beneficial in controlling the growth of the computational graph, thereby ensuring that the hyperoptimizer maintains the same asymptotic time complexity. The training times in the tables empirically support this idea.

5 Limitations and Future Works

In this study, several limitations exist that warrant further investigation. The limited number of experimental runs may affect the generalizability of our findings, necessitating increased iterations in future research. Moreover, the use of high initial hyperparameter values could lead to suboptimal performance in complex problems or large-scale datasets. Alternative initialization strategies should be explored to ensure a more suitable starting point in the search space. Additionally, future work should examine a broader range of hyperparameters and their combinations, which could provide a more comprehensive understanding of their impact on the performance of various optimization algorithms.

6 Conclusion

We reproduced and extended the hyperoptimization methodology to AdaBelief and AMSGrad optimizers, achieving improved performance on MNIST and CIFAR-10 datasets. In general, we find that hyperoptimized versions of each optimizer often perform better, the choice of hyperoptimizer matters, and the training computation time remain similar.

Contributions

Jialiang implemented the code for result reproduction and running the experiments. Jierui formed the idea. Leyi and Jierui equally contributed to the report.

References

[1] A. G. Baydin, R. Cornish, D. M. Rubio, M. Schmidt, and F. Wood. Online learning rate adaptation with hypergradient descent. *arXiv preprint arXiv:1703.04782*, 2017.

[2] K. Chandra, E. Meijer, S. Andow, E. Arroyo-Fang, I. Dea, J. George, et al. Gradient descent: The ultimate optimizer. *arXiv preprint arXiv:1909.13371*, 2019.

[3] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[4] A. Lydia and S. Francis. Adagrad—an optimizer for stochastic gradient descent. *Int. J. Inf. Comput. Sci*, 6(5):566–568, 2019.

[5] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *arXiv preprint arXiv:1505.04597*, 2015. conditionally accepted at MICCAI 2015.

[6] T. Tieleman and G. Hinton. Lecture 6.5 - rmsprop, coursera: Neural networks for machine learning. Technical report, Technical report, 2012.

[7] Y. Wang, P. Zhou, and W. Zhong. An optimization strategy based on hybrid algorithm of adam and sgd. In *MATEC Web of Conferences*, volume 232, page 03007. EDP Sciences, 2018.

[8] J. Zhuang, T. Tang, Y. Ding, S. C. Tatikonda, N. Dvornek, X. Papademetris, and J. Duncan. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. *Advances in neural information processing systems*, 33:18795–18806, 2020.

7 Appendix

Optimizer / Hyper	Initial α
SGD	0.01
SGD / SGD	0.01 / 0.01
SGD / Adam	0.01 / 0.1
SGD / Adagrad	0.01 / 0.01

(a) MNIST with SGD

Optimizer / Hyper	Initial α
Adagrad	0.01
Adagrad / SGD	0.01 / 0.01
Adagrad / Adadelta	0.01 / 0.01

(b) MNIST with AdaGrad

Optimizer / Hyper	Initial α
Adam	0.001
Adam / SGD	0.01 / 1e-5
Adam / Adam	0.001 / 0.001
Adam$^{\alpha}$ / SGD	0.01 / 1e-5
Adam$^{\alpha}$ / Adam	0.001 / 0.001

(c) MNIST with Adam

Optimizer / Hyper	Initial α
RMSProp	0.01
RMSProp / SGD	0.01 / 1e-4
RMSProp / RMSProp	0.01 / 1e-4
RMSProp$^{\alpha}$ / SGD	0.01 / 1e-4
RMSProp$^{\alpha}$ / RMSProp	0.01 / 1e-4

(d) MNIST with RMSProp

Table 4: Choice of Hyperparameters for Table 1 experiments.

Optimizer / Hyper	Initial α
AdaBelief	0.01
AdaBelief / SGD	0.01 / 1e-4
AdaBelief / Adam	0.01 / 1e-4

(a) MNIST with AdaBelief

Optimizer / Hyper	Initial α
AMSGrad	0.01
AMSGrad / SGD	0.01 / 1e-4
AMSGrad / Adam	0.01 / 1e-4

(b) MNIST with AMSGrad

Table 5: Choice of Hyperparameters for Table 2 experiments.

Optimizer / Hyper	Initial α
SGD	0.01
SGD / SGD	0.01 / 0.01
SGD / Adam	0.01 / 1e-5
SGD / AMSgrad	0.01 / 1e-5

(a) UNet on CIFAR-10 with SGD

Optimizer / Hyper	Initial α
Adam	0.01
Adam / Adam	0.01 / 1e-5
Adam / AMSGrad	0.01 / 1e-5

(b) UNet on CIFAR-10 with Adam

Optimizer / Hyper	Initial α
AdaBelief	0.01
AdaBelief / SGD	0.01 / 1e-6
AdaBelief / Adam	0.01 / 1e-6
AdaBelief / AdaBelief	0.01 / 1e-6

(c) UNet on CIFAR-10 with AdaBelief

Optimizer / Hyper	Initial α
AMSGrad	0.02
AMSGrad / SGD	0.02 / 1e-5
AMSGrad / Adam	0.02 / 1e-4
AMSGrad / AMSGrad	0.02 / 1e-5

(d) UNet on CIFAR-10 with AMSGrad

Table 6: Choice of Hyperparameters for Table 3 experiments.