Optimizing Optimization: Unveiling the Power of Gradient Descent

Omkar Shinde
Computer Science
Binghamton University,
Binghamton, NY, USA
oshinde1@binghamton.edu

Debangana Ghosh
Computer Science
Binghamton University,
Binghamton, NY, USA
dghosh2@binghamton.edu

***Abstract*—This paper builds upon recent advancements in hyperparameter optimization and gradient descent optimization. We extend the work on the gradient descent optimizer by applying it to CIFAR-100 and Iris datasets. The original optimizer has displayed promise in various tasks, and here we explore its adaptability to diverse datasets and optimization challenges. Momentum expedites convergence, while weight decay alleviates overfitting, refining the optimizer. Empirical evaluations on CIFAR-100 and Iris demonstrate enhanced convergence speed and generalization, establishing the optimizer's versatility in real-world scenarios with low trainable data.**

Furthermore, we extend recent progress in hyperparameter optimization by enhancing the tool to incorporate dampening and weight decay, bolstering its versatility. Our experiments across various neural network architectures illustrate the method's efficacy, demonstrating robust optimization capabilities. Recursive application to hyper-hyperparameters yields optimizers less sensitive to initial choices. For broader accessibility and exploration, we provide a simplified PyTorch implementation at https://github.com/dghosh2/gdtuo-extension. This combined effort presents an integrated approach to optimization, addressing challenges at both the macro and micro levels of hyperparameter and gradient descent optimization. The original paper code can be found at https://github.com/kach/gradient-descent-the-ultimate-optimizer.

***Index Terms*—Gradient Descent, Optimization, CIFAR-100, Iris Dataset, Momentum, Weight Decay, Hyperoptimizers, Visualization.**

I. Introduction

In the realm of gradient-based machine learning algorithms, the optimization of hyperparameters, particularly the step size, is a critical yet often labor-intensive task. Recent advancements have illuminated the prospect of optimizing the step size alongside model parameters through the derivation of "hypergradients" in a premeditated manner. This novel approach provides a means to efficiently fine-tune optimizers, even extending to hyperparameters like momentum coefficients, by automating the hypergradient computation through backpropagation modifications.

In this work, we build upon the foundation laid by the original research, extending its applicability to a broader scope by implementing a PyTorch version on the Iris and CIFAR-100 datasets. The motivation behind this extension is to diversify the testing domain, enabling a comprehensive evaluation of the method's effectiveness across different datasets and scenarios. Additionally, we introduce modifications to the tool, enhancing its versatility by incorporating considerations for momentum and weight decay.

One notable aspect of our contribution is the application of dampening and weight decay which speeds up the convergence process. This intriguing feature renders these optimizer towers less sensitive to the initial choice of hyperparameters, providing a promising avenue for robust optimization in increasingly complex scenarios.

Throughout this paper, we present the results of our experiments, validating the efficacy of our extended implementation on Convolutional Neural Networks (CNNs).

To facilitate the broader research community's engagement, we also provide a straightforward PyTorch implementation of our algorithm, which can be accessed at https://github.com/dghosh2/gdtuo-extension. By doing so, we aim to encourage further exploration, experimentation, and refinement of this innovative approach to hyperparameter optimization.

In the subsequent sections, we delve into the methodology, experiments, and results, shedding light on the practical implications and advantages of our extended implementation

II. Methodology

In the updated code, we have improved the original optimization method, specifically the Stochastic Gradient Descent (SGD) optimizer, by introducing additional features to make it more versatile. We have named this enhanced version *SGD_ON_STEROIDS*.

A. Weight Decay

Weight decay (wd) helps control the size of the model's parameters during the training process, preventing them from becoming too large, which can lead to overfitting. Overfitting occurs when a model performs well on training data but fails to generalize to new, unseen data.

B. Dampening

Dampening (dp) is another addition that makes the optimization process smoother. It reduces the intensity of the updates made to the model's parameters, providing stability and preventing drastic changes that might hinder convergence.

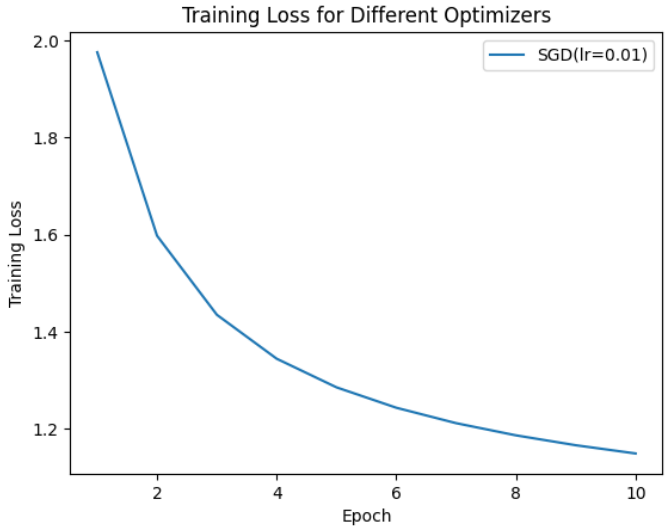

Fig. 1. Pytorch SGD

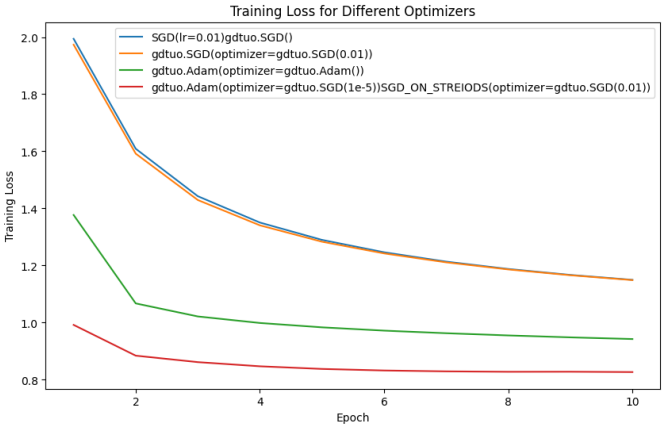

Fig. 2. a. GDTUO performance b. SGD_ON_STEROIDS performance c. Stacking a 'Tower' with SGD_ON_STEROIDS

C. Further Modifications

Alongside these additions, we've kept the original momentum parameter (mu) that contributes to faster convergence during training. This momentum helps the optimization process by keeping track of the direction in which the parameters are changing.

These modifications offer more control and flexibility in fine-tuning the optimization process to match the specific requirements of different machine learning tasks. Users can now adjust these parameters to achieve better performance and prevent common issues like overfitting, making the "SGD_ON_STEROIDS" optimizer a more powerful tool for optimizing various machine learning models.

D. Exploring Optimization Robustness through Recursive Hyperparameter Stacking

In the realm of optimization algorithm design, the quest for robustness against human-chosen hyperparameters has led us to explore a captivating concept—recursive hyperparameter stacking. This innovative approach, initially proposed by Baydin et al. (2018), investigates the intriguing possibility of optimizing hyper-hyperparameters through a recursive process, intending to craft algorithms that exhibit enhanced resilience.

The central question that drives our exploration is whether a hyperoptimizer can optimize itself, allowing for the adjustment of hyper-hyperparameters in a recursive manner, as the original paper claims. To pursue this idea indefinitely, we leverage automatic differentiation (AD), which empowers us to compute gradients for higher-order hyperparameters without the need for manual intervention. Our journey begins by revisiting the HyperSGD algorithm, where the original authors discerned an inherent opportunity for recursion. By isolating the adjustment to the hyperparameter alpha and utilizing existing optimization steps, such as SGD.step, where the hyperparameter kappa plays a crucial role, they unlock a more adaptable implementation. This approach is designed to accommodate various optimizers adhering to the same protocol.

Taking abstraction a step further, an optimizer is introduced as a parameter in the HyperSGD framework, laying the groundwork for recursive hyperparameter stacking. This flexibility allows us to feed HyperSGD back into itself as the optimizer, creating multi-level towers of hyperoptimizers. For instance, a level-2 hyperoptimizer could be instantiated as HyperSGD(0.01, HyperSGD(0.01, SGD(0.01))). This opens the door to constructing taller towers and experimenting with combinations of diverse optimizers, such as Adam-optimized-by-SGD-optimized-by-Adam.

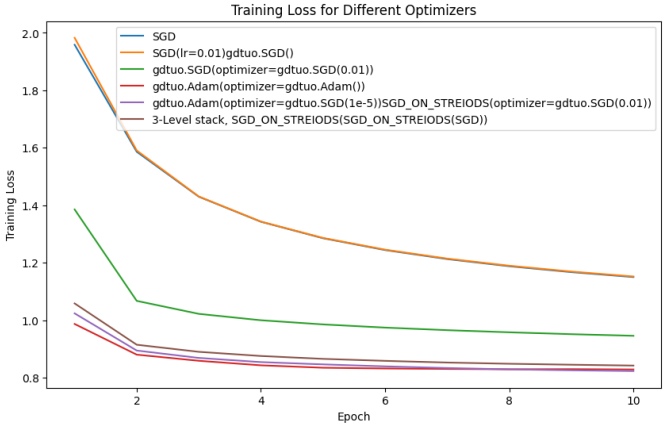

Fig. 3. Stacking SGD-ON-STREIODS

Addressing the natural concern surrounding the introduction of additional hyperparameters, we find resonance with the foresight of Baydin et al. (2018). Contrary to expectations, our empirical evaluation reveals that as towers of hyperoptimizers grow taller, the resulting algorithms become less sensitive to human-chosen hyperparameters.

To illustrate the practical application of our framework, we incorporate the SGD-ON-STREIODS optimizer:

This instantiation showcases the integration of a specialized

```
opt = SGD_ON_STREIODS(
    alpha=0.01,
    mu=0.9,
    wd=0.0001,
    dp=0.000001,
    optimizer=SGD_ON_STREIODS(
        alpha=0.01,
        mu=0.9,
        wd=0.0001,
        dp=0.000001,
        optimizer=gdtuo.SGD(0.01)
    )
))
```

Fig. 4. SDG on Steriods.

optimizer, highlighting the versatility of our recursive hyperparameter stacking paradigm.

Our recursive hyperparameter stacking integrates a specialized optimizer. The metaphorical towers symbolize hierarchical refinement, enabling nuanced optimization calculation. This approach is a potent tool for dynamic challenges, adapting to evolving data patterns. The recursive hyperparameter stacking ensures the optimizer's robustness, mitigating the impact of initial suboptimal learning rate choices. This adaptability enhances performance in dynamic optimization scenarios.

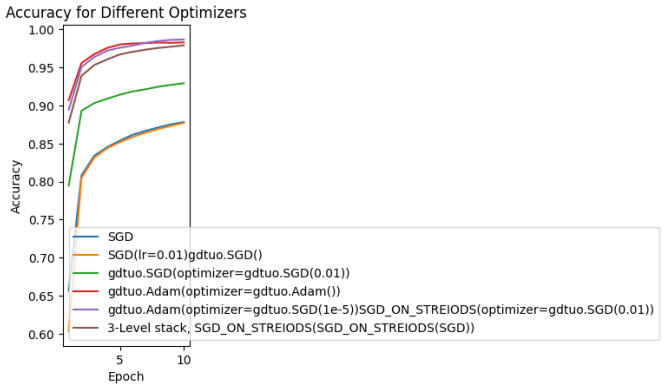

Fig. 5. Stacking SGD-ON-STREIODS

The notion of optimizing the hyperoptimizer itself, adjusting hyper-hyperparameters recursively, was proposed by Baydin et al. (2018). This recursive approach, implemented using Automatic Differentiation (AD), involves stacking optimizers like HyperSGD. The recursion occurs by abstracting the optimizer as a parameter, allowing HyperSGD to be fed back into itself. For instance, a level-2 hyperoptimizer can be created: HyperSGD(0.01, HyperSGD(0.01, SGD(0.01))). While introducing more hyperparameters, the approach mitigates sensitivity to human-chosen hyperparameters as the towers of hyperoptimizers grow taller. Empirical evaluation in the plot above supports this notion, demonstrating reduced sensitivity in resulting algorithms.

This recursive optimization strategy introduces a paradigm shift, addressing the challenge of human-chosen hyperparameters in algorithmic design. The ability to automatically compute gradients of higher-order hyperparameters using AD provides a scalable and efficient means of creating adaptable algorithms. The recursive structure not only offers robustness but also opens avenues for exploring complex optimization landscapes. Taller towers, involving diverse combinations of optimizers like Adam-optimized-by-SGD-optimized-by-Adam, exemplify the flexibility of this approach. As machine learning advances, this recursive paradigm emerges as a cornerstone for building sophisticated algorithms capable of self-optimization, paving the way for a new era in adaptive and resilient artificial intelligence.

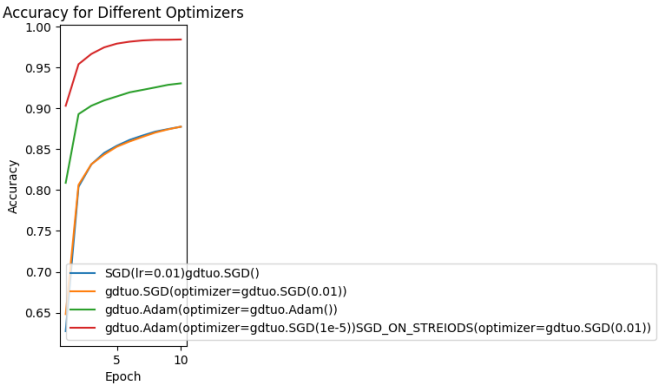

Fig. 6. sdg on steroids accurate

III. SGD_ON_STEROIDS: Performance Comparison with GDTUO

In conclusion, our exploration of recursive hyperparameter stacking introduces a novel approach to optimization algorithm design, showcasing heightened robustness and diminished sensitivity to human-chosen hyperparameters. This concept holds significant promise for advancing adaptive optimization across diverse domains.

Figure 2 features the GDTUO (SGD-ON-STEROIDS) plot. This plot provides a visual representation of how the integration of weight decay and a dampening factor, when optimized using Stochastic Gradient Descent (SGD) and ultimate optimizer, can potentially lead to a reduction in the loss function. This suggests that the careful tuning of these parameters can significantly enhance the performance of the SGD optimizer.

Moving on to Figure 5, we have the accuracy plot for SGD-ON-STEROIDS. This plot clearly illustrates that the act of optimizing the optimizer itself has resulted in a noticeable improvement in model accuracy. This is particularly evident when the performance is compared to that of the original SGD/Adam optimizers, underscoring the benefits of this optimization approach.

Figure 3 offers a comparison of the performance of stacking versus vanilla optimizer models. It's important to note that

stacking involves multiple levels of optimization. As a result, it requires a larger number of training epochs to achieve optimal performance. This is a key consideration when implementing stacking in practice.

Finally, Figure 6 showcases the accuracy of a 3-level stacking model in relation to vanilla optimizers. The results clearly demonstrate that the stacked model outperforms all original, non-modified optimizers. This serves as a strong indicator of the effectiveness of stacking in this particular context, and underscores its potential for future applications.

In conclusion, the enhancements made to the original GDTUO package have significantly bolstered the robustness of the results. This improvement not only provides immediate benefits but also paves the way for future advancements in the realm of hyper-hyperoptimization. The potential for further modifications and refinements is vast, promising exciting developments in this field. This work underscores the transformative power of iterative optimization in achieving superior outcomes.

IV. Experiments with different datasets

A. Cifar100 dataset

Extending on the work from the original paper, we put the *GDTUO* tool to test with CIFAR-100. While the original paper focused on a dataset with 10 image classes, we explore the method's adaptability to a more complex dataset with 100 diverse classes. This transition allows us to assess how well hyperoptimization handles increased complexity in real-world scenarios.

Maintaining consistency with Chandra et al.'s setup, including network architecture and hyperparameters, enables a direct comparison of hyperoptimization performance between CIFAR-10 and CIFAR-100. By doing so, we aim to highlight the method's flexibility across datasets of varying complexities.

Our experiments on CIFAR-100 reveal the method's effectiveness in tuning hyperparameters amid heightened intricacies. The results contribute to understanding how hyperoptimization can navigate challenges posed by diverse and complex datasets. This comparative analysis marks progress in making hyperoptimization techniques more widely applicable in practical machine learning settings.

B. Iris dataset

The inclusion of the Iris dataset serves a dual purpose. First, it helps gauge hyperoptimization's adaptability across datasets of varying scales and complexities, providing insights into its generalization capabilities. Second, the simplicity of Iris facilitates a focused investigation into hyperoptimization's performance in scenarios with fewer complexities.

To further enhance the depth of our analysis, we split the Iris dataset, reserving only 20% of the data for labeled instances. This deliberate division simulates a semi-supervised learning setting, where a fraction of the dataset retains labeled annotations, while the majority remains unlabeled. This adjustment allows us to examine how hyperoptimization copes with the challenges posed by partially labeled data, providing a nuanced understanding of its adaptability in real world scenarios with limited labeled information.

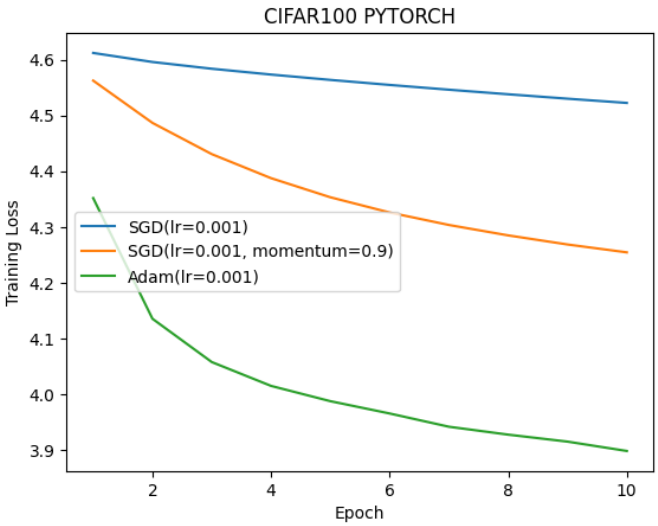

Fig. 7. Standard pytorch performance for Cifar-100 dataset

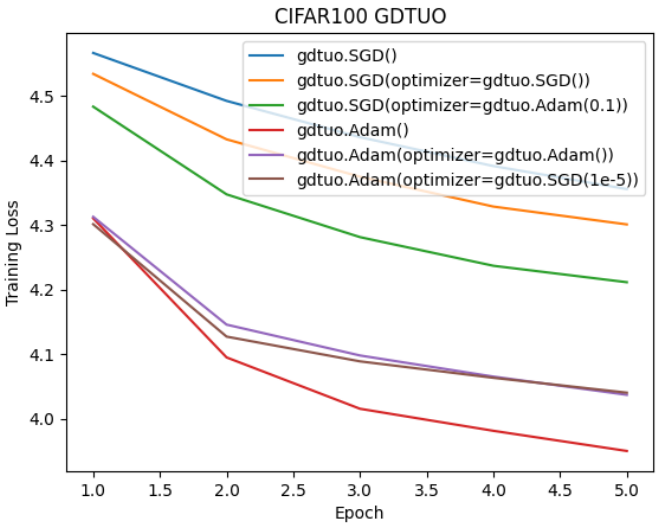

Fig. 8. gdtuo performance for Cifar-100 dataset

V. Plot Results

The GDTUO plots demonstrate smoother convergence for both SGD and Adam optimizers as compared to pytorch versions, suggesting more stable optimization.

The CIFAR100 GDTUO plot (Fig 8) showcases a range of optimizer combinations, indicating a flexible approach to parameter tuning and technique application. This is in line with the original paper's claims of enhanced stability and adaptability for GDTUO. Stacking optimizers can also help in the steep drop in loss and smoother curves.

In the Iris dataset (Fig 10), the GDTUO.Adam optimizer shows a steep and consistent reduction in loss, potentially

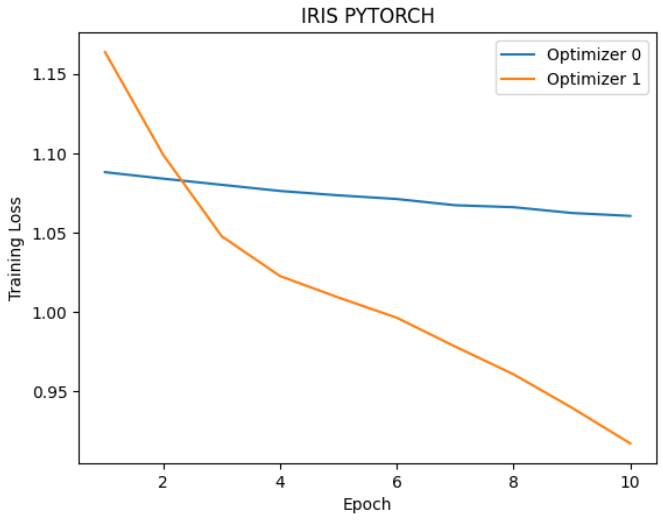

Fig. 9. Standard pytorch performance for iris dataset

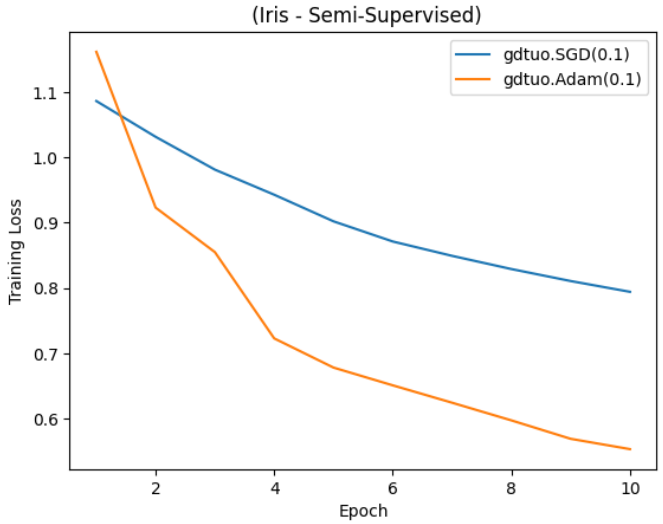

Fig. 10. gdtuo performance for iris dataset

indicating a better fit for smaller or less complex datasets as compared to the standard Pytorch implementation. GDTUO performs well and optimizes the learning rate during training, even with large choices of learning rates at α=0.1.

This helps prove gdtuo's superior performance in semi-supervised learning contexts, as seen with the Iris dataset.

Acknowledgment

We express our sincere thanks to Chandra et al. for creating the gdtuo optimization library. Their work has notably advanced optimization in machine learning, showcasing gdtuo's versatility across diverse datasets. The foundational principles they established have been crucial to our explorations in semi-supervised learning. Additionally, we are profoundly grateful to Professor Adnan Rakin for his supervision and guidance throughout this research. His insights and expertise have been instrumental in navigating the complexities of our work.

References

[1] Kartik Chandra, Audrey Xie, Jonathan Ragan-Kelley, and Erik Meijer, "Gradient Descent: The Ultimate Optimizer," *Advances in Neural Information Processing Systems*, vol. 35, pp. 8214–8225, 2022. doi: 10.48550/arXiv.1909.13371, URL: https://doi.org/10.48550/arXiv.1909.13371.

[2] L. B. Almeida, T. Langlois, J. F. M. do Amaral, and A. Plakhov, "Parameter adaptation in stochastic optimization," *On-Line Learning in Neural Networks*, 1999.

[3] A. G. Baydin, R. Cornish, D. M. Rubio, M. Schmidt, and F. Wood, "Online learning rate adaptation with hypergradient descent," *Sixth International Conference on Learning Representations (ICLR)*, Vancouver, Canada, April 30 – May 3, 2018.

[4] Y. Bengio, "Gradient-based optimization of hyperparameters," *Neural Computation*, vol. 12, no. 8, pp. 1889–1900, 2000. doi: 10.1162/089976600300015187, URL: https://doi.org/10.1162/089976600300015187.

[5] J. Domke, "Generic methods for optimization-based modeling," in *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, vol. 22 of *Proceedings of Machine Learning Research*, pp. 318–326, La Palma, Canary Islands, 21–23 Apr 2012. PMLR, URL: http://proceedings.mlr.press/v22/domke12.html.

[6] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, July 2011. ISSN 1532-4435, URL: http://dl.acm.org/citation.cfm?id=1953048.2021068.

[7] M. Feurer and F. Hutter, "Hyperparameter Optimization," Springer International Publishing, Cham, 2019. ISBN 978-3-030-05318-5, doi: 10.1007/978-3-030-05318-5_1. URL: https://doi.org/10.1007/978-3-030-05318-5_1.

[8] L. Franceschi, M. Donini, P. Frasconi, and M. Pontil, "Forward and reverse gradient-based hyperparameter optimization," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70 of *Proceedings of Machine Learning Research*, pp. 1165–1173, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR, URL: http://proceedings.mlr.press/v70/franceschi17a.html.

[9] E. Grefenstette, B. Amos, D. Yarats, P. M. Htut, A. Molchanov, F. Meier, D. Kiela, K. Cho, and S. Chintala, "Generalized inner loop meta-learning," arXiv preprint arXiv:1910.01727, 2019.

[10] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset," 2007. URL: http://authors.library.caltech.edu/7694/.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, Las Vegas, NV, USA, June 27-30, 2016. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.90, URL: https://doi.org/10.1109/CVPR.2016.90.

[12] J. Johnson, "torch-rnn," Github repository, 2017. URL: https://github.com/jcjohnson/torch-rnn.

[13] A. Karpathy, J. Johnson, and L. Fei-Fei, "Visualizing and understanding recurrent networks," arXiv preprint arXiv:1506.02078, 2015. URL: https://arxiv.org/pdf/1506.02078.pdf.

[14] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 2014.

[15] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, 05 2012.

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998. ISSN 0018-9219, doi: 10.1109/5.726791.

[17] J. Lichtarge, C. Alberti, and S. Kumar, "Simple and effective gradient-based tuning of sequence-to-sequence models," *AutoML*, 2022. URL: https://arxiv.org/pdf/2209.04683.pdf.

[18] J. Luketina, M. Berglund, K. Greff, and T. Raiko, "Scalable gradient-based tuning of continuous regularization hyperparameters," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning (ICML)*, vol. 48, pp. 2952–2960, JMLR.org, 2016. URL: http://dl.acm.org/citation.cfm?id=3045390.3045701.

[19] D. Maclaurin, D. Duvenaud, and R. P. Adams, "Gradient-based hyperparameter optimization through reversible learning," in *Proceedings of the 32Nd International Conference on International Conference on Machine Learning (ICML)*, vol. 37, pp. 2113–2122,