MADA: Meta-Adaptive Optimizers Through Hyper-Gradient Descent

Kaan Ozkara [1] **Can Karakus** [2] **Parameswaran Raman** [2] **Mingyi Hong** [2 3] **Shoham Sabach** [4] **Branislav Kveton** [2] **Volkan Cevher** [2 5]

Abstract

Following the introduction of Adam, several novel adaptive optimizers for deep learning have been proposed. These optimizers typically excel in some tasks but may not outperform Adam uniformly across all tasks. In this work, we introduce Meta-Adaptive Optimizers (MADA), a unified optimizer framework that can generalize several known optimizers and dynamically learn the most suitable one during training. The key idea in MADA is to parameterize the space of optimizers and dynamically search through it using hyper-gradient descent during training. We empirically compare MADA to other popular optimizers on vision and language tasks, and find that MADA consistently outperforms Adam and other popular optimizers, and is robust against sub-optimally tuned hyper-parameters. MADA achieves a greater validation performance improvement over Adam compared to other popular optimizers during GPT-2 training and fine-tuning. We also propose AVGrad, a modification of AMSGrad that replaces the maximum operator with averaging, which is more suitable for hyper-gradient optimization. Finally, we provide a convergence analysis to show that parameterized interpolations of optimizers can improve their error bounds (up to constants), hinting at an advantage for meta-optimizers.

[1]Department of Electrical and Computer Engineering, University of California Los Angeles, USA [2]Amazon Web Services [3]Department of Electrical and Computer Engineering, University of Minnesota, USA [4]Faculty of Data and Decision Sciences, Technion – Israel Institute of Technology, Israel [5]LIONS, IEM, STI, Ecole Polytechnique F´ed´erale de Lausanne, Switzerland. SS, MH, and VC hold concurrent appointments as an Amazon Scholar and as a faculty at Technion, University of Minnesota, and EPFL, respectively. This paper describes their work performed at Amazon. Correspondence to: Can Karakus <cakarak@amazon.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

1. Introduction

The choice of an optimization algorithm plays a critical role in determining the downstream performance of a machine learning model. Adaptive moment optimizers are the most preferred class of optimizers employed in most learning tasks such as training Large Language Models (LLMs) (Brown et al., 2020; Touvron et al., 2023) and Diffusion Models (Rombach et al., 2022). In particular, Adam (Kingma and Ba, 2015) is still the "go-to" optimizer in LLM training, despite the emergence of many other optimizers since then.

Recently proposed optimizers (Chen et al., 2023; Xie et al., 2023; Liu et al., 2023; You et al., 2020; Foret et al., 2021) report improved performance compared to Adam in specific tasks. However, it is unclear if their strong performance generalizes across a wide range of tasks as Adam's does (Schmidt et al., 2021). It is also unclear if a single optimizer can be uniformly the best across all learning tasks and training regimes, such as different batch sizes, hyper-parameters, and datasets.

In this work, we introduce the concept of a *parameterized optimizer*, which can be viewed as a unified parameterization of a collection of given optimizers. The parameters of a parameterized optimizer define a convex polytope (e.g. hypercube), whose vertices may correspond to individual base optimizers, while the interior represents new optimizers formed by interpolating between them. To make this concept operational, we propose the meta-adaptive optimizer (MADA), which combines the parameterized optimizer with hyper-gradient descent (Baydin et al., 2018) to learn the optimizer coefficients. MADA dynamically adjusts the parameterized optimizer coefficients *during* training, and effectively adapts the optimizer choice to the learning task. While MADA bears some connections to optimizer search methods, such as (Chen et al., 2023), it does not solely output a final optimizer state. It dynamically adapts the optimizer to the current neighborhood of the loss landscape on-the-fly, removing the need for an offline optimizer search stage, or an outer hyper-parameter selection loop.

Contributions. We make the following contributions:

- We introduce the concept of a parameterized optimizer,

which takes a collection of existing optimizers, and unifies them into a single optimizer, combining the individual update rules through learnable coefficients.

- We propose MADA, a meta-optimization framework that combines parameterized optimizers with hyper-gradient descent to learn a specific optimizer instance during training.
- We find that not all optimizers are suitable to use in a hyper-gradient optimization framework. Specifically, among popular optimizers, using AMSGrad (Reddi et al., 2018) results in poor performance within MADA due to its use of the maximum operator. Motivated by this, we propose a modification of it called AVGrad, which replaces the maximum operator on the second-order moments with time-averaging, and leads to better performance when used as part of MADA. We also analyze its convergence properties (see Appendix B). We show that MADA converges to AVGrad in a simple example where it is known to perform better than Adam, providing evidence for the effectiveness of MADA in adapting the optimizer to the task (see Appendix C).
- To demonstrate that optimizer interpolations can improve convergence bounds in an analytically tractable scenario, we theoretically analyze the convergence behavior for interpolations between two optimizers, namely AVGrad and Adam. Our analysis shows that the interpolated optimizer improves the convergence bounds of the base optimizers up to constant factors.
- We develop a specific parameterized optimizer, which interpolates between Adam (Kingma and Ba, 2015), AVGrad, Yogi (Zaheer et al., 2018), Adan (Xie et al., 2023), and Lion (Chen et al., 2023). On language tasks we compare MADA, which is based on this parameterized optimizer, against Adam, Lion, Adan, and HyperAdam[1], and show that MADA consistently outperforms all baselines. We also illustrate the robustness of MADA to initial hyper-parameters and analyze the evolution of hyper-parameters during training. On vision tasks we compare MADA to Adam, SGD with momentum, and HyperAdam, and observe consistent performance improvement.

Related work. Motivated by the high cost of training large language models, a large number of novel optimizers have been proposed in recent years to speed up the training, increase generalization performance or train more resource-efficient models. As mentioned before, Adam (Kingma and Ba, 2015) is the most commonly employed optimizer, and succeeding methods, in general, try to improve upon it under different scenarios. (Reddi et al., 2018; Zaheer et al., 2018) propose AMSGrad and YOGI, respectively, to fix Adam's potentially non-decreasing effective learning rate. (Xie et al., 2023; Dozat, 2016) introduce Adan and Nadam, respectively, to replace heavy-ball momentum in Adam with Nesterov momentum. (You et al., 2017; 2020) propose LARS and LAMB to improve the performance of Adam in large-batch regime. (Heo et al., 2021) proposes AdamP to avoid premature decay of scale-invariant weights. AdaBound (Luo et al., 2019) and AdaBelief (Zhuang et al., 2020) try to estimate a more stable second-order moment term. (Foret et al., 2021) proposes sharpness-aware minimization (SAM) and (Chen et al., 2020) proposes Padam to increase the generalization performance of the trained models. (Liu et al., 2020) stabilizes the training by reducing gradient variance. The work in (Chen et al., 2023) is similar in spirit to our work, in that it introduces a method to symbolically search a space of optimizers; however unlike our work they consider an offline search method, whereas our method learns the optimizer *during* actual model training, and uses hyper-gradients.

A separate line of work proposed learned optimizers to delegate the optimization task to neural networks (fully connected or LSTMs) (Almeida et al., 2021; Metz et al., 2022; 2020). Instead of directly using first order information as in optimizers, these methods treat gradient information, alongside other features such as training loss, validation loss (Almeida et al., 2021) and so on, as inputs to a neural network which outputs the update to be applied on a particular weight. Learned optimizers require resource-intensive offline training (e.g. thousands of TPU-months in (Metz et al., 2022)) since the updates are handled through a separate neural network that needs to be trained before deployment. Integration of a separate neural network to the optimization process introduces additional complexity, which MADA avoids by requiring only a few parameters to be learned on-the-fly.

Another related line of work is on gradient-based hyper-parameter optimization (Almeida et al., 1999; Maclaurin et al., 2015; Franceschi et al., 2017; Baydin et al., 2018; Chandra et al., 2022); our work applies a similar idea in a new setting, namely optimizer search and adaptation. Finally, we note that our work more broadly relates to a long line of research on AutoML and meta-learning (Andrychowicz et al., 2016; Wichrowska et al., 2017; Hospedales et al., 2021; Real et al., 2020).

2. Parameterized Optimizers

We focus on minimizing a loss function $F : \mathbb{R}^d \to \mathbb{R}$, that is, optimization problems of the following form

$$\min_{x \in \mathbb{R}^d} F(x).$$

[1]Throughout this paper we will refer to the version of Adam that uses hyper-gradients to tune β_1 and β_2 parameters, as in (Chandra et al., 2022), as HyperAdam.

Table 1 A unified framework to express adaptive moment optimizers.

Method	First-Order Moment	Second-Order Moment
Adam (Kingma and Ba, 2015)	$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$	$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
AMSGrad (Reddi et al., 2018)	$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$	$\bar{v}_t = \beta_2 \bar{v}_{t-1} + (1-\beta_2) g_t^2$ $v_t = \max\{v_{t-1}, \bar{v}_t\}$
Adan (Xie et al., 2023)	$\bar{m}_t = \beta_1 \bar{m}_{t-1} + (1-\beta_1) g_t$ $n_t = \beta_3 n_{t-1} + (1-\beta_3)(g_t - g_{t-1})$ $m_t = \bar{m}_t + \beta_3 n_t$	$\hat{g}_t = g_t + \beta_3 (g_t - g_{t-1})$ $v_t = \beta_2 v_{t-1} + (1-\beta_2) \hat{g}_t^2$
Yogi (Zaheer et al., 2018)	$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$	$\hat{g}_t = v_{t-1} + g_t^2 \cdot \text{sign}(g_t^2 - v_{t-1})$ $v_t = \beta_2 v_{t-1} + (1-\beta_2) \hat{g}_t$

Throughout the paper, we denote by $f : \mathbb{R}^d \to \mathbb{R}$ a random function computed on a minibatch sampled from the underlying data distribution. Therefore, for any $x \in \mathbb{R}^d$, we have $\mathbb{E}[f(x)] = F(x)$. Moreover, we assume F is differentiable and that $\mathbb{E}[\nabla f(x)] = \nabla F(x)$ for all $x \in \mathbb{R}^d$. We use f_t to denote the random function evaluated with input mini-batch sampled at t.

Informally, a parameterized optimizer can be described as the convex hull of a set of optimizers, when mapped to a Euclidean space under a certain parameterization. This section is devoted to explaining this notion in more detail, before Section 3 discusses how to perform learning in this Euclidean space.

In order to build a parameterized optimizer, we begin with a collection of existing optimizers. For the sake of concreteness, we illustrate the idea through the following four optimizers: Adam, AMSGrad, Adan and Yogi. A careful inspection reveals that these four optimizers can be described in one update rule that involves three iteratively generated sequences: model parameters, first-order moments, and second-order moments, which will be denoted throughout using the vectors x, m, and v, respectively. Each optimizer only differs in the way it updates these sequences. More precisely, starting with any $x_0 \in \mathbb{R}^d$ and setting $v_0 = m_0 = 0$, the generic update rule is given by

$$x_t = x_{t-1} - \alpha_t \frac{m_t}{\sqrt{v_t} + \epsilon}, \tag{1}$$

where $\alpha_t > 0$ is the learning rate and $\epsilon > 0$ is given. Table 1 shows how each optimizer defines its first- and second-moment iterates within this formulation, where we define $g_t := \nabla f_t(x_{t-1})$ and use $\beta_1, \beta_2, \beta_3$ to denote hyper-parameters controlling moment updates. For simplicity of the exposition, we omit the bias-correction terms for all optimizers.

To parameterize the design space of these four optimizers, we introduce real coefficients restricted between 0 and 1 that interpolate terms arising from different optimizers. Particular choices of these coefficients (typically at extreme values of 0 and 1) recover the underlying optimizers, but they express a new optimizer for their intermediate values.

As an example, observe that the first-order moment update rule of Adan already subsumes those of Adam, AMSGrad, and Yogi (where $\{\bar{m}_t\}$ and $\{n_t\}$ are new sequences introduced by Adan):

$$\begin{aligned} \bar{m}_t &= \beta_1 \bar{m}_{t-1} + (1-\beta_1) g_t \\ n_t &= \beta_3 n_{t-1} + (1-\beta_3)(g_t - g_{t-1}) \\ m_t &= \bar{m}_t + \beta_3 n_t, \end{aligned} \tag{2}$$

where taking $\beta_3 = 0$ recovers the update rules of the other three optimizers since $\bar{m}_t = m_t$. With respect to the second-order moments, Adan similarly covers Adam when $\beta_3 = 0$, since we get that $\hat{g}_t = g_t$. In order to incorporate the second-order moments of AMSGrad and Yogi in the same parameterization, we introduce two new coefficients $c, \rho \in [0, 1]$, and unify the second-moment computation as follows:

$$\begin{aligned} \hat{g}_t &= g_t + \beta_3 (g_t - g_{t-1}) \\ \tilde{g}_t^2 &= c \hat{g}_t^2 + (1-c)(v_{t-1} + \hat{g}_t^2 \cdot \text{sign}(\hat{g}_t^2 - v_{t-1})) \\ \tilde{v}_t &= \beta_2 \tilde{v}_{t-1} + (1-\beta_2) \tilde{g}_t^2 \\ v_t^{(max)} &= \max\{v_{t-1}^{(max)}, \tilde{v}_t\} \\ v_t &= \rho \tilde{v}_t + (1-\rho) v_t^{(max)}. \end{aligned} \tag{3}$$

It is easy to check that, for instance, the second-order moments of Adam can be recovered when $\beta_3 = 0$, and $c = \rho = 1$; those of AMSGrad are recovered when $\beta_3 = \rho = 0$ and $c = 1$; Adan can be recovered when $c = \rho = 1$; and Yogi can be recovered with $\beta_3 = c = 0$ and $\rho = 1$.

To summarize, in this example, for the four optimizers Adam, AMSGrad, Adan and Yogi, our parameterized optimizer is given by the three updating rules: (1), (2) and (3). Following the same line of arguments one can generate parameterized optimizers for other collections of given optimizers as well. However, unless we know a priori how to set the corresponding coefficients, the parameterized optimizer is not readily usable in practice. In the next section, we develop our meta-optimizer which uses a parameterized optimizer and learns the coefficients in an online fashion.

3. Meta-Adaptive Optimizers

In the previous section, we described how one can parameterize the design space of a given collection of optimizers. Here, we define MADA as a meta-optimization framework that dynamically learns the coefficients of a parameterized optimizer during training. In this work, we use hyper-gradient descent (Baydin et al., 2018; Chandra et al., 2022) to learn these coefficients, which avoids an expensive hyper-parameter optimization loop that involves multiple training runs. However, we note that in principle other techniques can also be combined with parameterized optimizers to learn the coefficients.

Learning interpolation coefficients. Hyper-gradient descent views the hyper-parameters as trainable parameters of the loss function, and thus differentiates the loss with respect to them and updates them through gradient steps. (Chandra et al., 2022) re-purposes PyTorch `autograd` machinery (Paszke et al., 2017) to automatically compute gradients with respect to optimization hyper-parameters such as learning rate and Adam β_1 and β_2 parameters. Since the update rule of a parameterized optimizer is differentiable with respect to its learnable coefficients, we can also update them using hyper-gradient descent steps.

To precisely describe our meta-optimizer MADA in detail, we need additional notation. We will denote a parameterized optimizer by $\mathcal{O}_q$, where $q \in \mathcal{D}$ denotes the vector of coefficients that defines the optimizer, and $\mathcal{D}$ represents the domain of the vector q. In the case of the example from Section 2, we have that $q = (\beta_1, \beta_2, \beta_3, \rho, c)$ and $\mathcal{O}_q$ is given by (1), (2), and (3). Note that $\mathcal{O}_q$ also encapsulates non-learnable state parameters (such as first-order and second-order moments) and other hyper-parameters such as the learning rate, weight decay, and stability parameter. In the example from Section 2, the domain is the unit hypercube where each element is in the range $[0, 1]$. We denote by $\Pi_{\mathcal{D}}$ the orthogonal projection onto the set $\mathcal{D}$. We provide a pseudo code to illustrate this (see Algorithm 1).

Algorithm 1 Pseudocode for a generic MADA

Input: A parameterized optimizer $\mathcal{O}_q$, where $q \in \mathcal{D}$, a hyper-learning rate α, number of total iterations T.
Init.: x_0 and q_0.

1: **for** t=1 to T **do**
2: Sample f_t.
3: Update model parameters: $x_t = \mathcal{O}_{q_{t-1}}(x_{t-1})$.
4: Update optimizer coefficients: $q_t = \Pi_{\mathcal{D}}\left[q_{t-1} - \alpha \nabla_q f_t(x_{t-1})\right]$.
5: **end for**

Output: Model wights x_T.

Hyper-gradient computation. Before concluding this section, we briefly illustrate hyper-gradient computation[2]. In order to compute the hyper-gradient of the loss with respect to a particular optimizer coefficient, we treat the updated model weights as a function of the coefficient. For instance, considering again the example from Section 2, we will show how to compute the gradient of the function f_t with respect to the coefficient ρ using the parameterized optimizer as given in (3). Using the chain rule, we obtain

$$\begin{aligned}\frac{\partial f_t(x_{t-1})}{\partial \rho} &= \frac{\partial f_t(x_{t-1})}{\partial x_{t-1}} \cdot \frac{\partial x_{t-1}}{\partial v_{t-1}} \cdot \frac{\partial v_{t-1}}{\partial \rho} \\ &= \frac{\partial f_t(x_{t-1})}{\partial x_{t-1}} \cdot \frac{\alpha_{t-1} m_{t-1}}{2\sqrt{v_{t-1}}(\sqrt{v_{t-1}} + \epsilon)^2} \cdot \left(\tilde{v}_{t-1} - v_{t-1}^{(max)}\right),\end{aligned} \tag{4}$$

where the first term $\partial f_t(x_{t-1})/\partial x_{t-1}$ is readily computed using standard back-propagation[3]. Note that the latter two objects of (4) are not explicitly constructed in practice; instead `autograd` simply *continues* back-propagating the already-computed parameter gradients into optimizer coefficients, while handling the necessary book-keeping for hyper-gradient computation.

4. On Convergence of Interpolated Optimizers

Since MADA uses hyper-gradients to update the optimizer, a prerequisite for a base optimizer to work well within MADA is that its update rules must allow efficient flow of hyper-gradients to the parameterization coefficients. In particular, in our experiments with MADA we have found that including AMSGrad among the base optimizers has an adverse effect on the end performance of the trained model. We conjecture that this is caused by the maximum operator in the second-moment term of AMSGrad. Specifically, note that back-propagation of gradients through $\max(a, b)$ corresponds to a simple routing operation to the larger one of a and b, where the smaller one is passed 0 gradients. In AMSGrad, $v_t = \max\{v_{t-1}, \bar{v}_t\}$ which means that for most steps the first term will be greater, causing insufficient hyper-gradient updates on β_2 parameter through $\bar{v}_t$ path[4]. To remedy this, we introduce AVGrad, which replaces the maximum operator with time-averaging of second moments, and results in better hyper-gradient flow and validations loss.

[2] Further details on hyper-gradients can be found in (Baydin et al., 2018; Chandra et al., 2022).

[3] In (4), the first term is a row vector, the second term is a Jacobian matrix, and the third term is a column vector, and thus the $\cdot$ operator refers to a matrix multiplication.

[4] To be accurate, considering the example parameterization (3), $\tilde{v}_t$ term provides another path for hyper-gradients on β_2. However, in practice we observed that when AMSGrad is used, ρ parameter tends to vanish, diminishing the effect of this path as well.

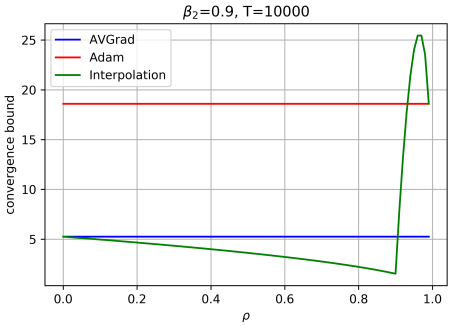

Figure 1: Value of the right-hand side of (6) in Theorem 1 for a representative case where $\beta_2 = 0.9, T = 10,000$.

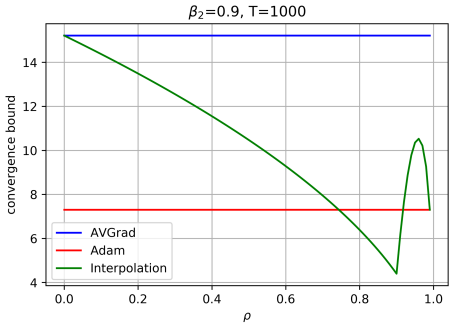

Figure 2: Value of the right-hand side of (6) in Theorem 1 for a representative case where $\beta_2 = 0.9, T = 1,000$.

4.1. AVGrad and its interpolation with Adam

Following the notation in Table 1, we define AVGrad as an optimizer where the first-order moments and second-order moments are defined by

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ \bar{v}_t &= \beta_2 \bar{v}_{t-1} + (1-\beta_2) g_t^2 \\ \tilde{v}_t &= \frac{1}{t}(\bar{v}_t + (t-1)\tilde{v}_{t-1}), \\ v_t &= \tilde{v}_t, \end{aligned} \tag{5}$$

where $\tilde{v}_t$ is the running average of past $\bar{v}_t$'s. We provide the convergence analysis of AVGrad in Appendix B in Theorems 2 and 3

In what follows, we focus on the interpolation between Adam and AVGrad, with the interpolation coefficient ρ_t. Specifically, we replace last line in (5) with $v_t := \rho_t \bar{v}_t + (1-\rho_t)\tilde{v}_t$ where ρ_t interpolates between second-order moments of Adam and AVGrad.

Soundness of AVGrad. We start with a proposition showing that AVGrad is a valid alternative to AMSGrad in mitigating the sub-optimal convergence issue in Adam (Reddi et al., 2018). Specifically, when ρ_t decays with $1/t$ (*i.e.*, converges to AVGrad), the interpolated optimizer fixes the non-decreasing learning rate problem in Adam, similar to AMSGrad.

Proposition 1. *The following inequality is a sufficient condition for non-increasing effective learning rate:*

$$\rho_t \leq \frac{1}{t(1-\beta_2)+1}.$$

In Appendix C, we use MADA to solve the convex problem given in (Reddi et al., 2018), an example where Adam fails. We found that MADA quickly converges to AVGrad, and thus to the optimum, by adapting $\rho_t \to 0$, even when we initialize it from Adam ($\rho_0 = 1$).

4.2. Convergence of the interpolated optimizer

Due to the complexity of our overall parameterization, deriving theoretical guarantees for MADA at its full scope is challenging. Therefore, in this section, we reduce the scope by examining convergence properties of interpolations between two optimizers, namely, AVGrad and Adam. In this setting, we will show that the parameterized optimizer allows the design of novel interpolated optimizers which improve the convergence rate of either optimizer up to constant factors, demonstrating the value of the parameterized optimizer formulation. We defer the proof and other technical details to Appendix A.

We make the following standard assumptions: F is bounded from below, is L-smooth, and with stochastic gradients that are bounded almost surely.

(Défossez et al., 2022) proposed a unified analysis for the convergence analysis of Adagrad and Adam in the non-convex setting. Since AVGrad can be seen as a version of Adagrad, the proof technique of (Défossez et al., 2022) can be adapted to AVGrad with some modifications. The modification includes the introduction of ρ parameter which interpolates between the second-order moment and the moment average, and results in an additional degree of freedom. Subsequently, ρ can be chosen to skew the updates towards the advantageous term under different scenarios.

Here we state the convergence result of the interpolated optimizer without momentum, that is when $\beta_1 = 0$ (see precise statement in Appendix A). The extension to the case with momentum can be done in the same way we extend Theorem 2 to Theorem 3, similar to (Défossez et al., 2022).

Theorem 1 (Convergence of interpolation of AVGrad and Adam without momentum)**.** *Under the assumptions above and $\alpha_t = \frac{\alpha}{\sqrt{t}}$ for some $\alpha > 0$, and for $\rho_t = \frac{\rho}{t}$ for a constant*

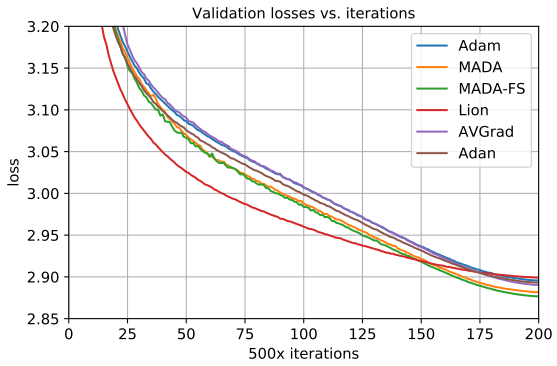

Figure 3: Validation losses of competing methods on OpenWebText for GPT-2 (125M) model using the same random seed.

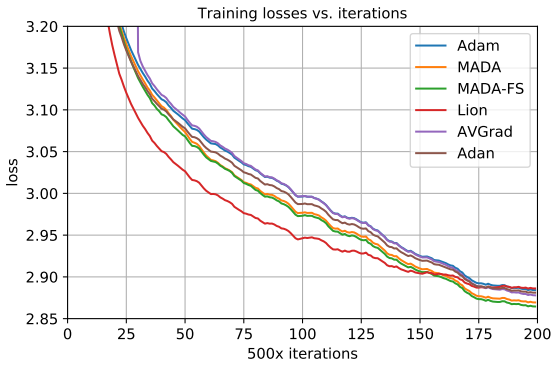

Figure 4: Training losses of competing methods on OpenWebText for GPT-2 (125M) model using same random seed, smoothed for convenience.

ρ:

$$G_T \leq E(T)\left[\frac{C_1}{T} + \frac{C_2 \ln\left(E(T)\right)}{T} + C_3\left[\ln\left(\frac{\rho}{\beta_2}\right)\right]_+\right], \tag{6}$$

where, $G_T = \frac{1}{T}\sum_{t=1}^{T}\|\nabla F(x_t)\|^2$, $E(T) = \frac{\sqrt{\rho+(1-\rho)T}}{\sqrt{1-\beta_2}}$, *and* C_1, C_2, C_3 *are constants independent of* T, ρ, β_2.

Remark. Aligned with the condition in Proposition 1, the contribution from v_t in Theorem 1 is multiplied with $1/t$ in order to avoid divergent terms in the bound. Observe that when $\rho = 0$ or $\rho = 1$ we recover the convergence rates for AVGrad (see Theorem 2 in Appendix B) and Adam (Défossez et al., 2022) respectively. We note that for the right-hand side to remain bounded, one would need either $0 \leq \rho \leq \beta_2$ (all terms vanish as in AVGrad), or $\rho = 1$ (a constant term remains as in Adam). To gain more insights, in Figures 1 and 2, we plot the value of the bound in Theorem 1 with respect to ρ for two representative cases . The first case with larger T represents a more favorable setting for AVGrad as all the terms vanish with T. The second is more favorable for Adam since the vanishing terms vanish faster than AVGrad. In both cases, the interpolated optimizer provides the best convergence bound when $\rho = \beta_2$, which suggests an advantage for the interpolated optimizers.

5. Experiments

In our experiments, we aim to answer how the generalization and downstream performance of MADA compares against fixed optimizers, and how robust MADA is to poor hyper-parameter initializations compared to fixed optimizers. We focus on the pre-training of models from scratch. We also try to gain insights into the behavior of MADA by monitoring the evolution of the optimizer coefficients. Additional details of our experimental setup, and results can be found in Appendix C.

Table 2 Validation loss of MADA on OpenWebText vs other adaptive optimizer baselines.

Method	Validation Loss
Adam	2.8956
Adan	2.8896
HyperAdam	2.8950
Lion	2.8892
AVGrad	2.8895
MADA	**2.8806**
MADA-FS	**2.8766**
Poor initialization	
MADA$^-$	2.8921
Adam$^-$	2.9157

5.1. A concrete parameterized optimizer

We use a concrete parameterization that is similar to the example presented in Section 2, with two changes. First, we replace AMSGrad with AVGrad, and second, we include Lion among the optimizers that we interpolate. Hence, second-order moment in (3) becomes $v_t = \rho\bar{v}_t + (1-\rho)\tilde{v}_t$ as in the interpolation in Section 4. Moreover, the update term becomes $\gamma\frac{m_t}{\sqrt{v_t}+\epsilon} + (1-\gamma)\text{sign}(u_t)$ where u_t is the moment term from Lion (Chen et al., 2023). We refer the reader to Appendix C for a complete description of the parameterization. Lion was omitted in the example in Section 2 for the sake of simplicity, since it integrates less naturally with the rest of the optimizers.

5.2. Experimental setting

Data and models. In recent years, auto-regressive models have been widely used for benchmarking and algorithm evaluation (Radford et al., 2019b; Liu et al., 2023). Motivated by this, we evaluate MADA on the causal language modeling task with GPT-2, over two datasets: Shakespeare (Karpathy,

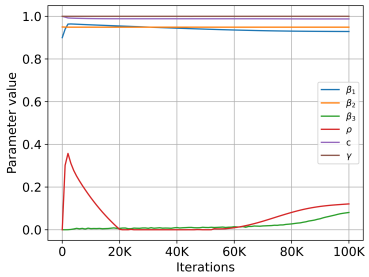

Figure 5: Parameter evolution for $\beta_{1,0} = 0.9, \beta_{2,0} = 0.95, \beta_{3,0} = 0$.

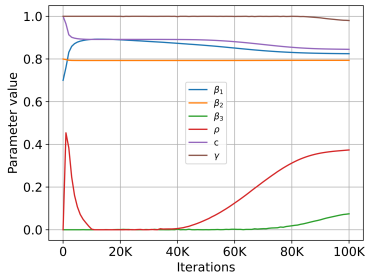

Figure 6: Parameter evolution for $\beta_{1,0} = 0.7, \beta_{2,0} = 0.8, \beta_{3,0} = 0$.

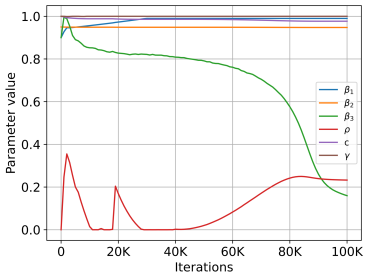

Figure 7: Parameter evolution for $\beta_{1,0} = 0.9, \beta_{2,0} = 0.95, \beta_{3,0} = 0.9$.

2015), and OpenWebText (Gokaslan and Cohen, 2019). On the Shakespeare dataset, we train a 6 layer transformer with context size of 256 (11M parameters) and fine-tune GPT-2 (XL) with 48 layers (1.5B parameters). On OpenWebText, we train GPT-2 (small) with 12 layers, 1024 context size (125M parameters) and GPT-2 (medium) with 24 layers, 1024 context size (335M parameters). Our code is available at `https://github.com/amazon-science/mada_optimizer_search`.[5]

Baselines and learned optimizers. On OpenWebText, we compare MADA to Adam (Kingma and Ba, 2015), HyperAdam (Chandra et al., 2022) (Adam with β_1 and β_2 parameters learned through hyper-gradients), Adan (Xie et al., 2023), Lion (Chen et al., 2023), and AVGrad. For all the methods we used decoupled weight decay (Loshchilov and Hutter, 2019), i.e. the AdamW variant of Adam is used. We keep the learning hyper-parameters such as learning rate schedule and weight decay identical, except for Lion where we choose a lower initial learning rate as suggested in (Liu et al., 2023). For a fair comparison, for all optimizers, we use the same codebase based on (Chandra et al., 2022). For OpenWebText experiments, we use established parameters for Adam ($\beta_1 = 0.9, \beta_2 = 0.95, \epsilon = 10^{-6}$) and also use these values as the initial parameters for MADA, HyperAdam, and AVGrad; for other methods we use the parameters suggested in respective papers; and measure the validation loss. For Shakespeare experiments, we compare MADA to Adam and HyperAdam; the relatively small model size in this experiment allows us to sweep the grid of initial β parameters and compare the final training loss across many different hyper-parameter choices. In some experiments, we also evaluate the final state of MADA statically, *i.e.*, we take the final optimizer learned by MADA and use it as the fixed optimizer from the beginning of training, which we refer to as MADA-FS.

Tuning hyper-learning rates. While hyper-learning rates are additional hyper-parameters to be tuned; the tasks are less sensitive to hyper-learning rates compared to other hyper-parameters such as learning rate, most likely because each hyper-learning rate controls the update of a single parameter. As a result, hyper-learning rates can be fine-tuned easily and can be transferred across similar tasks. In general, for the hyper-learning rates of β_1, β_2, since the gradients are relatively large, we search in a set of smaller values: $\{10^{-3}, 5 \times 10^{-4}, 10^{-4}\}$. For the other parameters we search in the set $\{10^{-1}, 5 \times 10^{-2}, 10^{-2}\}$.

[5]We use nanoGPT (`https://github.com/karpathy/nanoGPT`) code base for the implementation.

5.3. Results

In this section, we first present empirical results of training GPT-2 models on OpenWebText, and provide a discussion of generalization performance and robustness of MADA. We also show how the interpolation coefficients evolve during training. Next, we provide a visualization of the optimal training loss given various initializations of β_1, β_2 values using the smaller Shakespeare dataset. We present additional results in Appendix C including a toy problem to show that MADA discovers optimal solutions that Adam cannot find, and comparing MADA and Adam for GPT-2 (medium).

GPT-2 on OpenWebText. When we initialize MADA from AVGrad and Adan with $\beta_{1,0} = 0.9, \beta_{2,0} = 0.95, \beta_{3,0} = 0.9, \rho_0 = 0$ (where subscript 0 denotes the time index) from Table 2 we observe that MADA consistently outperforms Adam and other recently proposed optimizers, including Lion, Adan, and HyperAdam. In particular, MADA-FS outperforms Adam with established parameters by 0.019 in validation loss which is a significant improvement for this task. In Figure 3 and Figure 4 we see that MADA is able to converge to a lower loss than baseline methods (both in terms of validation and training loss). Note that MADA is able to converge faster than AVGrad and Adam with the same learning rate and schedule, which may be attributed to theoretical result Theorem 1 on interpolated optimizers.

Perplexity on benchmark datasets. To measure the generalization of MADA to other datasets, we compute the vali-

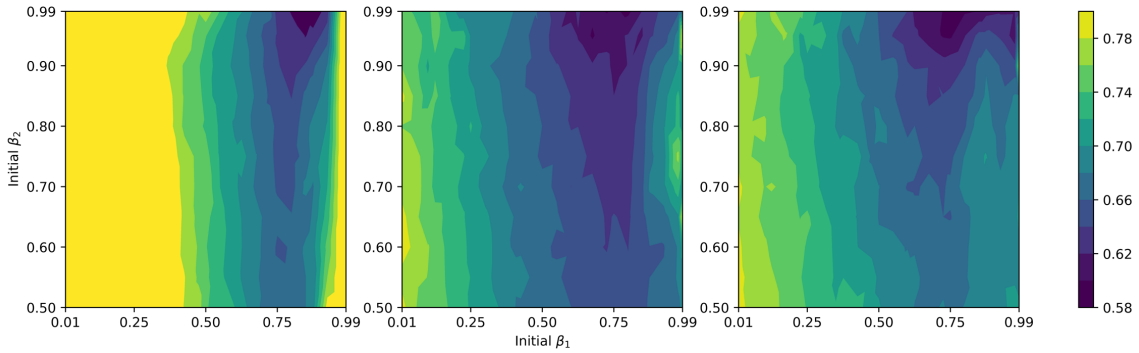

Figure 8: Final training loss of Adam, MADA, and HyperAdam vs. initial β values on Shakespeare dataset. MADA yields better performance for wider choice of initial β values, illustrating its robustness.

dation perplexity of the trained models on three datasets as shown in Table 3. We see that on OpenWebText MADA outperforms Adam and HyperAdam by 0.34 and 0.33 points); while on Wikitext (Merity et al., 2016), MADA outperforms these two baselines by 4.45 and 2.37 respectively. On Lambada (Radford et al., 2019a) it is the second-best-performing method with a small gap behind HyperAdam (0.88).

MADA-FS versus MADA. We find that MADA-FS performs slightly better than MADA for OpenWebText (0.004 in validation loss and 0.07, 1.85, 0.68 points on validation perplexity on OpenWebText, Wikitext and Lambada). More details are in Tables 2 and 3. On the other hand, on Shakespeare we observe that, the dynamically-evolving optimizer performs generally better than using the fixed final optimizer. We conjecture that the difference in the number of iterations, the model size or dataset may explain this phenomenon.

Learning interpolation coefficients is crucial. We also see that HyperAdam performance is very close to Adam (2.8950 vs. 2.8956), which suggests that the main performance improvement of MADA does not simply originate from the tuning of the β_1, β_2 parameters, but rather from the adaptation of the interpolation coefficients in the optimizer space.

Robustness to initial hyper-parameters. Additionally, we compare MADA to Adam when we initialize from the suboptimal hyperparameters $\beta_{1,0} = 0.7, \beta_{2,0} = 0.8$ (we call these optimizers MADA$^-$ and Adam$^-$ respectively). Notably, we observe that even suboptimally initialized MADA$^-$ is able to outperform Adam with the established initial parameters, as well as HyperAdam. This particular example indicates the robustness of MADA. We provide more examples with poor initial hyper-parameter choices on Shakespeare dataset in the next subsections.

Evolving hyper-parameters. We monitor the evolution of the interpolation coefficients during training of GPT-2 (125M) model on OpenWebText for various choices of initialization of β_1, β_2 (Figures 5, 6, and 7).

Table 3 Validation perplexities of competing methods on OpenWebText, Wikitext and Lambada datasets.

Method	OpenWebText	Wikitext	Lambada
Adam	18.0940	63.8544	77.3314
Adan	17.9863	63.5518	74.6970
HyperAdam	18.0843	61.7717	**72.6803**
Lion	17.9792	61.8661	75.3158
AVGrad	17.9840	64.2620	75.1317
MADA	**17.8249**	61.2513	74.2480
MADA-FS	**17.7544**	59.4086	**73.5623**
Poor initialization			
MADA$^-$	18.0317	**57.1613**	75.3550
Adam$^-$	18.4624	72.9017	79.1217

First, we observe that sub-optimal initialization of hyper-parameters (as in Figure 6) results in a more significant update of the coefficients. This suggests that default optimizer state with $\beta_{1,0} = 0.9, \beta_{2,0} = 0.95$ is close to a local minimum. A closer inspection yields insight into how MADA accelerates training. First, in Figures 5, 6 we see that β_1 increases at the beginning which facilitates taking larger steps, while towards the end it decreases to give more emphasis on individual gradients. Second, we observe that the interpolation factor of AVGrad, ρ, follows a three-phase pattern. This suggests that MADA puts more importance on individual normalization term in the beginning and towards the end, and it puts more emphasis on the less noisy averaging term during the intermediate stages.

Another interesting behavior we observe is when we initialize β_3 (which governs the weight of Adan) from a higher value such as 0.9 (Figure 7), we see β_1 reaches 0.99 and stays constant, in contrast with Figures 5, 6. Remarkably, MADA initialized with $\beta_3 = 0.9$ automatically trains β_1 to be

0.99; i.e. MADA, when initialized from combination of Adan and AVGrad, automatically finds the β_1 that is suggested by the authors of (Xie et al., 2023). Note that $\beta_1, \beta_2, \beta_3$ in this work correspond to $1-\beta_1, 1-\beta_3, 1-\beta_2$ in (Xie et al., 2023).

Shakespeare dataset. On Shakespeare dataset we compare MADA to Adam and HyperAdam. We show that it results in significant increase in the performance across many initial β_1, β_2 values when training 11M parameter model. Particularly, in Figure 8, we see that MADA results in better performance for the vast majority of initial β_1 and β_2 parameters, illustrating robustness.

Fine-tuning GPT-2 XL on Shakespeare dataset. We also fine-tune the pre-trained GPT-2 XL (1.5B parameters) model on Shakespeare, for approximately 2 epochs, using MADA, Adam, and HyperAdam methods. Table 4 shows the resulting training loss values, where MADA results in a large improvement. Notably, HyperAdam achieves no gain over Adam.

Table 4 Training losses after 2 epochs of fine-tuning on Shakespeare dataset.

Method	Training loss
MADA	**0.255**
Adam	0.276
HyperAdam	0.278

Vision tasks. We also validated the performance of MADA on two computer vision models: 5-layer CNN and ResNet-9 on CIFAR-10 dataset. We observe that MADA shows consistent improvement over our baselines (see test accuracy results below in Table 5).

Table 5 Test accuracy of competing methods for CNN and ResNet models.

Method	5-layer CNN	ResNet-9
MADA	$\mathbf{66.12 \pm 0.14}$	$\mathbf{93.79 \pm 0.11}$
Adam	65.84 ± 0.11	93.73 ± 0.10
HyperAdam	65.80 ± 0.21	93.69 ± 0.06
Momentum SGD	65.97 ± 0.94	92.60 ± 0.10

6. Conclusion

In this paper we propose an approach to unify a set of optimizers into a parameterized formulation. We further show how to dynamically learn the most suitable optimizer for a particular task during training using hyper-gradient descent. We employ MADA to train language models and observe that it consistently outperforms Adam and other fixed optimizers both in terms of best validation performance, and in terms of robustness to initial hyper-parameters.

Limitations. The main limitation of our framework is the additional computational requirement and memory usage of the parameterized optimizer due to the maintained optimizer states. The additional computation can be broken down into two components: additional per-parameter computational steps during optimizer update, and the computation of hyper-gradients. Note that the first component is negligible compared to the overall FLOPs requirement of a single training step in a language model. This is because for a model with N parameters trained over a batch of T tokens, the model parameter update in the parameterized optimizer involves cN FLOPs (with c being on the range of 10-20 depending on the parameterization), while the forward-backward passes require approximately $6TN$ FLOPs, with T being on the order of thousands. To analyze the second component, consider the hyper-gradient example in (4), where the computation of hyper-gradient with respect to ρ involves several vector-level element-wise operations (note that the multiplication of the first term with second does not require a matrix-vector multiplication, since it can be implemented by a dot product with the numerator followed by element-wise division with the denominator), where each vector is of size N. As a result, the FLOPs requirement is still $O(N)$ (does not scale with T), and much smaller than the $6TN$ required for the forward-backward passes. The derivative with respect to the other coefficients can similarly be shown to have $O(N)$ complexity.

The additional memory usage arises from the fact that we need to maintain a larger optimizer state per parameter. To mitigate, one can employ sharded data parallelism techniques (Rajbhandari et al., 2020), which shards the optimizer state across a large number of data-parallel devices. Another approach is to analyze the contributions of the constituent optimizers to the learning performance, and prune the ones that have little contribution.

In Appendix D, we provide profiling results showing the actual increase in iteration time and memory usage.

Future work. Theoretically, we have shown the convergence of interpolation of AVGrad and Adam; as a future work we would like to construct a generic theoretical framework that can characterize the convergence of optimizer interpolations more generally. Empirically, we aim to gain a deeper understanding of the dynamics of the optimizer learning process in practice.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning and neural network optimization. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Diogo Almeida, Clemens Winter, Jie Tang, and Wojciech Zaremba. A generalizable approach to learning optimizers. *arXiv preprint arXiv:2106.00958*, 2021.

Luís B. Almeida, Thibault Langlois, Jose D. Amaral, and Alexander Plakhov. *Parameter Adaptation in Stochastic Optimization*, page 111–134. Publications of the Newton Institute. Cambridge University Press, 1999. doi: 10.1017/CBO9780511569920.007.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29, 2016.

Atilim Gunes Baydin, Robert Cornish, David Martinez Rubio, Mark Schmidt, and Frank Wood. Online learning rate adaptation with hypergradient descent. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=BkrsAzWAb.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Kartik Chandra, Audrey Xie, Jonathan Ragan-Kelley, and Erik Meijer. Gradient descent: The ultimate optimizer. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=-Qp-3L-5ZdI.

Jinghui Chen, Dongruo Zhou, Yiqi Tang, Ziyan Yang, Yuan Cao, and Quanquan Gu. Closing the generalization gap of adaptive gradient methods in training deep neural networks, 2020.

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Yao Liu, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, and Quoc V. Le. Symbolic discovery of optimization algorithms, 2023.

Alexandre Défossez, Leon Bottou, Francis Bach, and Nicolas Usunier. A simple convergence proof of adam and adagrad. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL https://openreview.net/forum?id=ZPQhzTSWA7.

Timothy Dozat. Incorporating nesterov momentum into adam. 2016.

Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=6Tm1mposlrM.

Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. Forward and reverse gradient-based hyperparameter optimization. In *International Conference on Machine Learning*, pages 1165–1173. PMLR, 2017.

Aaron Gokaslan and Vanya Cohen. Openwebtext corpus. http://Skylion007.github.io/OpenWebTextCorpus, 2019.

Byeongho Heo, Sanghyuk Chun, Seong Joon Oh, Dongyoon Han, Sangdoo Yun, Gyuwan Kim, Youngjung Uh, and Jung-Woo Ha. Adamp: Slowing down the slowdown for momentum optimizers on scale-invariant weights, 2021.

Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):5149–5169, 2021.

Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. http://karpathy.github.io/2015/05/21/rnn-effectiveness/, 2015.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Hong Liu, Zhiyuan Li, David Hall, Percy Liang, and Tengyu Ma. Sophia: A scalable stochastic second-order optimizer for language model pre-training, 2023.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkgz2aEKDr.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Liangchen Luo, Yuanhao Xiong, and Yan Liu. Adaptive gradient methods with dynamic bound of learning rate. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg3g2R9FX.

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International conference on machine learning*, pages 2113–2122. PMLR, 2015.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

Luke Metz, Niru Maheswaranathan, C Daniel Freeman, Ben Poole, and Jascha Sohl-Dickstein. Tasks, stability, architecture, and compute: Training more effective learned optimizers, and using them to train themselves. *arXiv preprint arXiv:2009.11243*, 2020.

Luke Metz, James Harrison, C Daniel Freeman, Amil Merchant, Lucas Beyer, James Bradbury, Naman Agrawal, Ben Poole, Igor Mordatch, Adam Roberts, et al. Velo: Training versatile learned optimizers by scaling up. *arXiv preprint arXiv:2211.09760*, 2022.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019a.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019b.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

Esteban Real, Chen Liang, David So, and Quoc Le. AutoML-zero: Evolving machine learning algorithms from scratch. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 8007–8019. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/real20a.html.

Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=ryQu7f-RZ.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

Robin M Schmidt, Frank Schneider, and Philipp Hennig. Descending through a crowded valley - benchmarking deep learning optimizers. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 9367–9376. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/schmidt21a.html.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *International conference on machine learning*, pages 3751–3760. PMLR, 2017.

Xingyu Xie, Pan Zhou, Huan Li, Zhouchen Lin, and Shuicheng Yan. Adan: Adaptive nesterov momentum algorithm for faster optimizing deep models, 2023.

Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks, 2017.

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Syx4wnEtvH.

Manzil Zaheer, Sashank Reddi, Devendra Sachan, Satyen Kale, and Sanjiv Kumar. Adaptive methods for nonconvex optimization. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/90365351ccc7437a1309dc64e4db32a3-Paper.pdf.

Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar C Tatikonda, Nicha Dvornek, Xenophon Papademetris, and James Duncan. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 18795–18806. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/d9d4f495e875a2e075a1a4a6e1b9770f-Paper.pdf.

A. Setup, Details, and Proof for Theorem 1

Assumptions. (i) F is bounded below by F^*, (ii) stochastic gradients, that is $\forall x \in \mathbb{R}^d,\ \|\nabla f(x)\|_\infty \leq R$ a.s. , (iii) the objective function is smooth, that is $\forall x, y \in \mathbb{R}^d, \quad \|\nabla F(x) - \nabla F(y)\|_2 \leq L\|x - y\|_2$.

Setup. Our setup follows (Défossez et al., 2022). Let $d \in \mathbb{N}$ be the dimensionality of the model. Given a function $h : \mathbb{R}^d \to \mathbb{R}$ we define $\nabla_i h$ as the i'th component of the gradient. For the stochastic loss, we assume we have access to an oracle providing i.i.d samples $(f_t)_{t\in\mathbb{N}}$. $\mathbb{E}_{t-1}$ is defined as the expectation conditioned upon observing the past stochastic function values: $f_1, \ldots, f_{t-1}$. Unlike (Défossez et al., 2022), for simplicity of calculations, we define $\epsilon_t = \epsilon/t$ where ϵ is a small constant for stability. Next we define adaptive moment updates in a generic way, and then specialize them to AVGrad. These updates are different from those in Section 2, but they are equivalent, and facilitates easier analysis.

Adaptive updates. Similar to (Défossez et al., 2022) we define the following vectors iteratively.

$$m_{t,i} = \beta_1 m_{t-1,i} + \nabla_i f_t(x_{t-1})$$
$$v_{t,i} = \beta_2 v_{t-1,i} + \nabla_i f_t(x_{t-1})^2$$

Note that $\hat{m}_{t,i} = (1-\beta_1)m_{t,i}$ and $\hat{v}_{t,i} = (1-\beta_2)v_{t,i}$ give the updates in Adam. (Défossez et al., 2022) uses the learning rate to absorb $(1-\beta)$ terms. In particular, if one has an update $x_{t,i} = x_{t-1,i} - \alpha_t \frac{m_{t,i}}{\sqrt{\epsilon + v_{t,i}}}$; Adam is recovered with $\alpha_t = \alpha\frac{1-\beta_1}{\sqrt{1-\beta_2}}$.

AVGrad updates. Let $\alpha_t = \frac{\alpha}{\sqrt{t}}$ be the learning rate. AVGrad is defined via the following iterative steps

$$v_{t,i}^{(sum)} = v_{t-1,i}^{(sum)} + v_{t,i} = \sum_{j=1}^{t} v_{j,i} \text{ and } v_{t,i}^{(avg)} = \frac{v_{t,i}^{(sum)}}{t}$$
$$x_{t,i} = x_{t-1,i} - \alpha_t \frac{m_{t,i}}{\sqrt{\epsilon_t + v_{t,i}^{(avg)}}} = x_{t-1,i} - \alpha \frac{m_{t,i}}{\sqrt{\epsilon + v_{t,i}^{(sum)}}}$$

Note that to obtain the AVGrad updates we also need to scale the learning rate with $\frac{1-\beta_1}{\sqrt{1-\beta_2}}$; but, for now, we assume it is already absorbed in the α for the sake of simplicity. We also drop the bias correction for simplicity as justified in (Défossez et al., 2022).

A.1. Proof of Theorem 1

We multiply the normalization term of the update with t from the decaying learning rate; hence, there is a multiplying factor of t difference in definition of ψ in Theorem 1 and $\bar{\psi}$ in this proof. Let us start by defining $\bar{\psi}_{t,i}$

$$\bar{\psi}_{t,i} = \rho v_{t,i} + (1-\rho)v_{t,i}^{(sum)} \tag{7}$$

We also define $\tilde{\psi}_{t,i}$:

$$\begin{aligned}\tilde{\psi}_{t,i} = \rho\tilde{v}_{t,i} + (1-\rho)\tilde{v}_{t,i}^{(sum)} &= \rho(\beta_2 v_{t-1,i} + \mathbb{E}_{t-1}\nabla_i f_t(x_{t-1})^2) + (1-\rho)(v_{t-1,i}^{(sum)} + \beta_2 v_{t-1,i} + \mathbb{E}_{t-1}\nabla_i f_t(x_{t-1})^2) \\ &= (1-\rho)v_{t-1,i}^{(sum)} + \beta_2 v_{t-1,i} + \mathbb{E}_{t-1}\nabla_i f_t(x_{t-1})^2, \end{aligned} \tag{8}$$

where the contribution of the last gradient is replaced by its expected value conditioned on the past iteration. Let us also define

$$u_{t,i} = \frac{\nabla_i f_t(x_{t-1})}{\sqrt{\epsilon + \bar{\psi}_{t,i}}}.$$

From the smoothness of the objective function F, we can use the Descent Lemma to obtain that

$$F(x_t) \leq F(x_{t-1}) - \alpha\nabla F(x_{t-1})^\top u_t + \frac{\alpha^2 L}{2}\|u_t\|^2.$$

Taking the expectation of both sides conditioned on $f_1, \ldots, f_{t-1}$, we have

$$\mathbb{E}_{t-1}F(x_t) \leq F(x_{t-1}) - \alpha \sum_{i\in[d]} \mathbb{E}_{t-1}\left[\nabla_i F(x_{t-1})\frac{\nabla_i f_t(x_{t-1})}{\sqrt{\epsilon + \overline{\psi}_{t,i}}}\right] + \frac{\alpha^2 L}{2} + \mathbb{E}_{t-1}[\|u_t\|^2]. \tag{9}$$

To upper bound the second term on the right hand side, we use the following lemma which generalizes Lemma 5.1 in (Défossez et al., 2022) to be used with interpolated optimizer and replaces v_t with $\psi_{t,i}$ as the normalization term.

Lemma 1 (Updates approximately follow a descent direction)**.**

$$\mathbb{E}_{t-1}\left[\nabla_i F(x_{t-1})\frac{\nabla_i f_t(x_{t-1})}{\sqrt{\epsilon + \overline{\psi}_{t,i}}}\right] \geq \frac{\nabla_i F(x_{t-1})^2}{2\sqrt{\epsilon + \tilde{\psi}_{t,i}}} - 2R\mathbb{E}_{t-1}\left[\frac{\nabla_i f_t(x_{t-1})^2}{\epsilon + \psi_{t,i}}\right]. \tag{10}$$

Proof. For the sake of simplicity, we define $G = \nabla_i F(x_{t-1}),\ g = \nabla_i f_t(x_{t-1}), \psi = \bar{\psi}_{t,i}, \tilde{\psi} = \tilde{\psi}_{t,i}$. First obviously, we have

$$\frac{Gg}{\sqrt{\epsilon+\psi}} = \frac{Gg}{\sqrt{\epsilon+\tilde{\psi}}} + \underbrace{\frac{Gg}{\sqrt{\epsilon+\psi}} - \frac{Gg}{\sqrt{\epsilon+\tilde{\psi}}}}_{A}. \tag{11}$$

For the first term, we use the fact that $\mathbb{E}_{t-1}[g] = G$ to obtain that

$$\mathbb{E}_{t-1}\left[\frac{Gg}{\sqrt{\epsilon+\tilde{\psi}}}\right] = \frac{G^2}{\sqrt{\epsilon+\tilde{\psi}}}. \tag{12}$$

In order to bound A, we begin with

$$A = Gg\frac{\tilde{\psi}-\psi}{\sqrt{\epsilon+\psi}\sqrt{\epsilon+\tilde{\psi}}(\sqrt{\epsilon+\psi}+\sqrt{\epsilon+\tilde{\psi}})} = Gg\frac{\mathbb{E}_{t-1}g^2 - g^2}{\sqrt{\epsilon+\psi}\sqrt{\epsilon+\tilde{\psi}}(\sqrt{\epsilon+\psi}+\sqrt{\epsilon+\tilde{\psi}})},$$

where the last equality follows from the fact that $\tilde{\psi} - \psi = \mathbb{E}_{t-1}[g^2] - g^2$ (see (8) and the definition of $\psi_{t,i}$). Hence, from the triangle inequality we have that

$$|A| \leq \underbrace{|Gg|\frac{\mathbb{E}_{t-1}g^2}{\sqrt{\epsilon+\psi}(\epsilon+\tilde{\psi})}}_{A_1} + \underbrace{|Gg|\frac{g^2}{\sqrt{\epsilon+\tilde{\psi}}(\epsilon+\psi)}}_{A_2},$$

where in the inequality follows from the fact that $\sqrt{\epsilon+\psi} + \sqrt{\epsilon+\tilde{\psi}} \geq \max(\sqrt{\epsilon+\psi}, \sqrt{\epsilon+\tilde{\psi}})$. A useful fact that will be used later is the following

$$\forall \lambda > 0,\ x, y \in \mathbb{R} \quad xy \leq \frac{\lambda x^2}{2} + \frac{y^2}{2\lambda}. \tag{13}$$

To bound A_1, we use (13) with $\lambda = \frac{\sqrt{\epsilon+\tilde{\psi}}}{2}, x = \frac{|G|}{\sqrt{\epsilon+\tilde{\psi}}}, y = |g|\frac{\mathbb{E}_{t-1}g^2}{2\sqrt{\epsilon+\tilde{\psi}}\sqrt{\epsilon+\psi}}$, which yields that

$$A_1 \leq \frac{G^2}{4\sqrt{\epsilon+\tilde{\psi}}} + \frac{g^2\mathbb{E}_{t-1}[g^2]^2}{(\epsilon+\tilde{\psi})^{3/2}(\epsilon+\psi)}.$$

Taking the expectation and noting $\epsilon + \tilde{\psi} \geq \mathbb{E}_{t-1}[g^2]$ ensures that

$$\mathbb{E}_{t-1}[A_1] \leq \frac{G^2}{4\sqrt{\epsilon+\tilde{\psi}}} + \frac{\mathbb{E}_{t-1}[g^2]}{\sqrt{\epsilon+\tilde{\psi}}}\mathbb{E}_{t-1}\left[\frac{g^2}{\epsilon+\psi}\right] \leq \frac{G^2}{4\sqrt{\epsilon+\tilde{\psi}}} + R\mathbb{E}_{t-1}\left[\frac{g^2}{\epsilon+\psi}\right]$$

where the last inequality uses our boundedness assumption $\sqrt{\mathbb{E}_{t-1}[g^2]} \leq R$ and the fact that $\sqrt{\epsilon+\tilde{\psi}} \geq \sqrt{\mathbb{E}_{t-1}[g^2]}$. Similarly, for A_2, we use (13) with $\lambda = \frac{\sqrt{\epsilon+\tilde{\psi}}}{2\mathbb{E}_{t-1}g^2}, x = \frac{|Gg|}{\sqrt{\epsilon+\tilde{\psi}}}, y = \frac{g^2}{\epsilon+\psi}$, which gives us (we used similar bounding arguments)

$$\mathbb{E}_{t-1}[A_2] \leq \frac{G^2}{4\sqrt{\epsilon+\tilde{\psi}}} + R\frac{\mathbb{E}_{t-1}[g^2]}{\epsilon+\psi}.$$

Combining both upper bounds we have,

$$\mathbb{E}_{t-1}[|A|] \leq \frac{G^2}{2\sqrt{\epsilon+\tilde{\psi}}} + 2R\frac{\mathbb{E}_{t-1}[g^2]}{\epsilon+\psi},$$

which can be equivalently written as follows

$$\mathbb{E}_{t-1}[-|A|] \geq -\left(\frac{G^2}{2\sqrt{\epsilon+\tilde{\psi}}} + 2R\frac{\mathbb{E}_{t-1}[g^2]}{\epsilon+\psi}\right),$$

Finally, combining further with (12) and substituting into (11) gives us,

$$\mathbb{E}_{t-1}\left[\frac{Gg}{\sqrt{\epsilon+\psi}}\right] \geq \frac{G^2}{\sqrt{\epsilon+\tilde{\psi}}} - \left(\frac{G^2}{2\sqrt{\epsilon+\tilde{\psi}}} + 2R\frac{\mathbb{E}_{t-1}[g^2]}{\epsilon+\psi}\right) = \frac{G^2}{2\sqrt{\epsilon+\tilde{\psi}}} - 2R\frac{\mathbb{E}_{t-1}[g^2]}{\epsilon+\psi},$$

which proves the desired result. □

Using Lemma 1 in (9) yields that

$$\mathbb{E}_{t-1}[F(x_t)] \leq F(x_{t-1}) - \left(\frac{\alpha\sqrt{1-\beta_2}}{2R\sqrt{\rho+(1-\rho)t}}\|\nabla F(x_{t-1})\|^2 - 2\alpha R\mathbb{E}_{t-1}[\|u_t\|^2]\right) + \frac{\alpha^2 L}{2}\mathbb{E}_{t-1}[\|u_t\|^2].$$

Summing both sides for $t = 1, \ldots, T$, and noting $\sqrt{t} \leq \sqrt{T}$ implies that

$$\mathbb{E}[F(x_T)] \leq F(x_0) - \frac{\alpha\sqrt{1-\beta_2}}{2R\sqrt{\rho+(1-\rho)T}}\sum_{t=0}^{T-1}\|\nabla F(x_t)\|^2 + \left(2\alpha R + \frac{\alpha^2 L}{2}\right)\sum_{t=0}^{T-1}\mathbb{E}[\|u_t\|^2].$$

Equivalently, we can write,

$$\frac{\alpha\sqrt{1-\beta_2}}{2R\sqrt{\rho+(1-\rho)T}}\sum_{t=0}^{T-1}\|\nabla F(x_t)\|^2 \leq F(x_0) - \mathbb{E}[F(x_T)] + \left(2\alpha R + \frac{\alpha^2 L}{2}\right)\sum_{t=0}^{T-1}\mathbb{E}[\|u_t\|^2]. \tag{14}$$

Now, we would like to bound the last term on the right hand side. For this, we introduce the following lemma that generalizes Lemma 5.2 in (Défossez et al., 2022).

Lemma 2. *Define* $b_t = \rho\sum_{j=1}^{t}\beta_2^{t-j}a_j + (1-\rho)\sum_{\tau=1}^{t}\sum_{j=1}^{\tau}\beta_2^{\tau-j}a_j$ *for* $t > 0$ *and* $b_0 = 0$*, we have*

$$\sum_{t=1}^{T}\frac{a_t}{\epsilon+b_t} \leq \ln\left(1 + \frac{R^2(\rho+(1-\rho)T)}{\epsilon(1-\beta_2)}\right) + T\left[\ln\left(\frac{\rho}{\beta_2}\right)\right]_+.$$

Proof. Let $l_t = \sum_{j=1}^{t}\beta_2^{t-j}a_j$, $r_t = \sum_{\tau=1}^{t}\sum_{j=1}^{\tau}\beta_2^{\tau-j}a_j$, we have $b_t = \rho l_t + (1-\rho)r_t$. Since $b_t > a_t \geq 0$ we have that $1 - z \leq \exp^{-z}$ with $z = \frac{a_t}{\epsilon+b_t} < 1$. Hence,

$$\frac{a_t}{\epsilon+b_t} \leq \ln(\epsilon+b_t) - \ln(\epsilon+b_t-a_t)$$

$$
\begin{aligned}
&= \ln(\epsilon + b_t) - \ln(\epsilon + \rho(l_t - a_t) + (1-\rho)(r_t - a_t)) \\
&= \ln(\epsilon + b_t) - \ln\Big(\epsilon + \rho\beta_2 l_{t-1} + (1-\rho)r_{t-1} + (1-\rho)(\sum_{j=1}^{t}\beta_2^{t-j}a_j - a_t)\Big) \\
&= \ln(\epsilon + b_t) - \ln\Big(\epsilon + \rho\beta_2 l_{t-1} + (1-\rho)r_{t-1} + (1-\rho)\beta_2 \underbrace{\sum_{j=1}^{t-1}\beta_2^{t-1-j}a_j}_{l_{t-1}}\Big) \\
&= \ln(\epsilon + b_t) - \ln\left(\epsilon + \beta_2 l_{t-1} + (1-\rho)r_{t-1}\right) \\
&= \ln\left(\frac{\epsilon + b_t}{\epsilon + b_{t-1}}\right) + \ln\left(\frac{\epsilon + \rho l_{t-1} + (1-\rho)r_{t-1}}{\epsilon + \beta_2 l_{t-1} + (1-\rho)r_{t-1}}\right) \\
&\le \ln\left(\frac{\epsilon + b_t}{\epsilon + b_{t-1}}\right) + \ln\left(\max\{1, \frac{\rho}{\beta_2}\}\right) \\
&= \ln\left(\frac{\epsilon + b_t}{\epsilon + b_{t-1}}\right) + \left[\ln\left(\frac{\rho}{\beta_2}\right)\right]_+,
\end{aligned}
$$

where the first equality is due to definition of b_t. Summing the inequality above for $t = 1, 2, \ldots, T$, yields that (recall that $b_0 = 0$)

$$
\sum_{t=1}^{T}\frac{a_t}{\epsilon + b_t} \le \ln\left(1 + \frac{b_T}{\epsilon}\right) + T\left[\ln\left(\frac{\rho}{\beta_2}\right)\right]_+
$$

Also note that $b_T \le \frac{R^2(\rho + (1-\rho)T)}{1-\beta_2}$. □

Using Lemma 2 in (14), dividing both sides by T, and after some algebra we have

$$
\begin{aligned}
\frac{1}{T}\sum_{t=1}^{T}\|\nabla F(x_t)\|^2 \le{}& \frac{2R\sqrt{\rho + (1-\rho)T}(F_0 - F^*)}{\alpha T\sqrt{1-\beta_2}} \\
&+ \frac{2R\sqrt{\rho + (1-\rho)T}d}{\sqrt{1-\beta_2}T}(2R + \alpha L)\left(\ln\left(1 + \frac{R^2(\rho + (1-\rho)T)}{\epsilon(1-\beta_2)}\right) + T\left[\ln\left(\frac{\rho}{\beta_2}\right)\right]_+\right)
\end{aligned}
$$

which concludes the proof.

B. Convergence of AVGrad

Using the notation and assumptions in Section 3, we give the following convergence results for AVGrad.

Theorem 2 (Convergence of AVGrad without momentum)**.** *Under the above assumptions and $\alpha_t = \frac{\alpha}{\sqrt{t}}$ for some $\alpha > 0$, we have:*

$$
\frac{1}{T}\sum_{t=1}^{T}\|\nabla F(x_t)\|^2 \le \frac{2R(F_0 - F^*)}{\alpha\sqrt{T}\sqrt{1-\beta_2}} + \frac{2Rd}{\sqrt{T}\sqrt{1-\beta_2}}(2R + \alpha L)\ln\left(1 + \frac{R^2T}{(1-\beta_2)\epsilon}\right).
$$

Theorem 3 (Convergence of AVGrad with momentum)**.** *Let $\tau_T \in \{0, \ldots, T-1\}$ denote a random index such that $\forall j \in \mathbb{N}, j < T, \mathbb{P}[\tau = j] \propto 1 - \beta_1^{T-j}$. Under the above assumptions and $\alpha_t = \frac{\alpha}{\sqrt{t}}$ for some $\alpha > 0$, we have:*

$$
\mathbb{E}\|\nabla F(x_\tau)\|^2 \le \frac{2(1-\beta_1)R\sqrt{T}}{\alpha\sqrt{1-\beta_2}\tilde{T}}(F(x_0) - F^*) + C\frac{\sqrt{T}d}{\tilde{T}}\ln\left(1 + \frac{R^2T}{(1-\beta_2)\epsilon}\right),
$$

where $C = \frac{\alpha RL}{\sqrt{1-\beta_2}(1-\beta_1)} + \frac{2\beta_1\alpha^2L^2}{(1-\beta_2)(1-\beta_1)^3} + \frac{12R^2}{\sqrt{1-\beta_1}}$ and $\tilde{T} = T - \frac{\beta_1}{1-\beta_1}$.

Remark. Comparing our Theorems 2 and 3 to Theorems 3 and 4 in (Défossez et al., 2022), we observe that AVGrad does not have the non-vanishing, constant term that Adam has (see (Défossez et al., 2022), Theorem 2). Moreover, unlike the result for Adam, we do not require $\beta_2 > \beta_1$. Analogous to how momentum slows down the convergence bound of algorithms (by multiplicative factors) (Défossez et al., 2022) AVGrad slows down the convergence of Adagrad by multiplicative factors of $(1 - \beta_2)$.

B.1. Proof of Theorem 2

Proof of convergence of AVGrad can be obtained by replacing the interpolated second order moment term ψ_t, by $v_t^{(sum)}$ in the previous section. In other words, setting $\rho = 0$ in the Proof of Theorem 1, going through the analysis will give the desired result

$$\frac{1}{T}\sum_{t=1}^{T}\|\nabla F(x_t)\|^2 \leq \frac{2R(F_0 - F^*)}{\alpha\sqrt{T}\sqrt{1-\beta_2}} + \frac{2Rd}{\sqrt{1-\beta_2}\sqrt{T}}\left(2R + \alpha L\right)\ln\left(1 + \frac{TR^2}{(1-\beta_2)\epsilon}\right).$$

B.2. Proof of Theorem 3

The idea in this proof is to essentially change gradient terms in the Descent Lemma with first order moments, as the difference in consequent model weights will now depend on the moment term rather than a gradient at some time point. The necessary changes in the lemmas closely follow the proof for Theorem 3,4 in (Défossez et al., 2022). We start by redefining some iterative vectors.

$$\begin{aligned}
m_{t,i} &= \beta_1 m_{t-1,i} + \nabla_i f_t(x_{t-1}) \\
v_{t,i} &= \beta_2 v_{t-1,i} + \nabla_i f_t(x_{t-1})^2 \\
x_{t,i} &= x_{t-1,i} - \alpha_t \frac{m_{t,i}}{\sqrt{\epsilon_t + v_{t,i}^{(avg)}}} = x_{t-1,i} - \alpha \frac{m_{t,i}}{\sqrt{\epsilon + v_{t,i}^{(sum)}}}
\end{aligned}$$

Let us further define $G_t = \nabla F(x_{t-1}), g_t = \nabla f_t(x_{t-1}), u_{t,i} = \frac{m_{t,i}}{\sqrt{\epsilon + v_{t,i}^{(sum)}}}$ and $U_{t,i} = \frac{g_{t,i}}{\sqrt{\epsilon + v_{t,i}^{(sum)}}}$. And also define:

$$\tilde{v}_{t,k,i}^{(sum)} = v_{t-k,i}^{(sum)} + \mathbb{E}_{t-k-1}[\sum_{\tau=t-k+1}^{t}\sum_{j=1}^{\tau}\beta_2^{\tau-j} g_{j,i}^2]$$

note that for $j \leq t-k$ we have $\mathbb{E}_{t-k-1}[g_j^2] = g_j^2$, so $\tilde{v}_{t,k,i}^{(sum)}$ essentially replaces the contribution of last k gradients with their expected values. From the smoothness of the objective function F, we have

$$F(x_t) \leq F(x_{t-1}) - \alpha G_t^\top u_t + \frac{\alpha^2 L}{2}\|u_t\|_2^2.$$

Taking the expectations of both sides,

$$\mathbb{E}[F(x_t)] \leq \mathbb{E}[F(x_{t-1})] - \alpha\sum_{i\in[d]}\mathbb{E}[G_{t,i}^\top u_{t,i}] + \frac{\alpha^2 L}{2}\mathbb{E}[\|u_t\|_2^2]. \tag{15}$$

To bound the second term on the right hand side, we introduce the following approximate descent lemma, whose proof is provided at the end of section.

Lemma 3. *[Updates approximately follow a descent direction]*

$$\mathbb{E}\left[G_{t,i}\frac{m_{t,i}}{\sqrt{\epsilon + v_{t,i}^{(sum)}}}\right] \geq \frac{1}{2}\left(\sum_{i\in[d]}\sum_{k=0}^{t-1}\beta_1^k\mathbb{E}\left[\frac{G_{t-k,i}^2}{\sqrt{\epsilon + \tilde{v}_{t,k+1,i}^{(sum)}}}\right]\right) - \frac{3R\sqrt{1-\beta_2}}{\sqrt{1-\beta_1}}\sum_{k=0}^{t-1}\sqrt{k+1}\beta_1^k\mathbb{E}[\|U_{t-k}\|^2]$$

$$- \frac{\alpha^2 L^2}{4R\sqrt{1-\beta_2}}\sqrt{1-\beta_1}\sum_{l=1}^{t-1}\|u_{t-l}\|^2\sum_{k=l}^{t-1}\beta_1^k\sqrt{k}.$$

Using Lemma 3 in (15) for the second term on right hand side, yields that

$$\mathbb{E}[F(x_t)] \leq \mathbb{E}[F(x_{t-1})] - \frac{\alpha}{2}\left(\sum_{i\in[d]}\sum_{k=0}^{t-1}\beta_1^k\mathbb{E}\left[\frac{G_{t-k,i}^2}{\sqrt{\epsilon+\tilde{v}_{t,k+1,i}^{(sum)}}}\right]\right) + \frac{3\alpha R\sqrt{1-\beta_2}}{\sqrt{1-\beta_1}}\sum_{k=0}^{t-1}\sqrt{k+1}\beta_1^k\mathbb{E}[\|U_{t-k}\|_2^2]$$
$$+ \frac{\alpha^3 L^2}{4R\sqrt{1-\beta_2}}\sqrt{1-\beta_1}\sum_{l=1}^{t-1}\|u_{t-l}\|_2^2\sum_{k=l}^{t-1}\beta_1^k\sqrt{k} + \frac{\alpha^2 L}{2}\mathbb{E}[\|u_t\|_2^2]. \tag{16}$$

Let us define $\Omega_t := \sqrt{\sum_{\tau=1}^{t}\sum_{j=1}^{\tau}\beta_2^{\tau-j}}$, hence, from our boundedness assumption, $\sqrt{\epsilon+\tilde{v}_{t,k+1,i}^{(sum)}} \leq R\sqrt{\sum_{\tau=1}^{t}\sum_{j=1}^{\tau}\beta_2^{\tau-j}} = R\Omega_t$, inserting in (16) implies that

$$\mathbb{E}[F(x_t)] \leq \mathbb{E}[F(x_{t-1})] - \frac{\alpha}{2R\Omega_t}\left(\sum_{k=0}^{t-1}\beta_1^k\mathbb{E}\left[G_{t-k,i}^2\right]\right) + \frac{\alpha^2 L}{2}\mathbb{E}\|u_t\|_2^2 + \frac{3\alpha R\sqrt{1-\beta_2}}{\sqrt{1-\beta_1}}\sum_{k=0}^{t-1}\sqrt{k+1}\beta_1^k\mathbb{E}\|U_{t-k}\|_2^2$$
$$+ \frac{\alpha^3 L^2}{4R\sqrt{1-\beta_2}}\sqrt{1-\beta_1}\sum_{l=1}^{t-1}\|u_{t-l}\|_2^2\sum_{k=l}^{t-1}\beta_1^k\sqrt{k}$$

Summing for $t = 1, \ldots, T$ results in

$$\underbrace{\frac{\alpha}{2R}\sum_{t=1}^{T}\frac{1}{\Omega_t}\sum_{k=0}^{t-1}\beta_1^k\mathbb{E}\|G_{t-k}^2\|_2^2}_{A} \leq F(x_0) - F^* + \underbrace{\frac{\alpha^2 L}{2}\sum_{t=1}^{T}\mathbb{E}[\|u_t\|^2]}_{B} + \underbrace{\frac{\alpha^3 L^2}{4R\sqrt{1-\beta_2}}\sqrt{1-\beta_1}\sum_{t=1}^{T}\sum_{l=1}^{t-1}\mathbb{E}[\|u_{t-l}\|^2]\sum_{k=l}^{t-1}\beta_1^k\sqrt{k}}_{C}$$
$$+ \underbrace{\frac{3\alpha R\sqrt{1-\beta_2}}{\sqrt{1-\beta_1}}\sum_{t=1}^{T}\sum_{k=0}^{t-1}\sqrt{k+1}\beta_1^k\mathbb{E}[\|U_{t-k}\|^2]}_{D}. \tag{17}$$

We will examine each term separately, for B, we state the following lemma (whose proof is given at the end of section) to bound the sum of squared norm term

Lemma 4. *Assume, $0 \leq \beta_1 < 1$, $0 \leq \beta_2 < 1$ and we have sequence of real numbers $(a_t)_{t\in[T]}$. Let $c_t = \sum_{\tau=1}^{t}\beta_1^{t-\tau}a_t$, $b_t = \sum_{\tau=1}^{t}\sum_{j=1}^{\tau}\beta_2^{\tau-j}a_t^2$. Then,*

$$\sum_{t=1}^{T}\frac{c_t^2}{\epsilon+b_t} \leq \frac{1}{(1-\beta_1)^2}\ln\left(1+\frac{b_T}{\epsilon}\right).$$

Lemma 4 implies that

$$\frac{\alpha^2 L}{2}\sum_{t=1}^{T}\mathbb{E}[\|u_t\|^2] \leq \frac{\alpha^2 L}{2(1-\beta_1)^2}\sum_{i\in[d]}\ln\left(1+\frac{v_{T,i}^{(sum)}}{\epsilon}\right) \leq \frac{\alpha^2 L}{2(1-\beta_1)^2}\sum_{i\in[d]}\ln\left(1+\frac{R^2T}{(1-\beta_2)\epsilon}\right). \tag{18}$$

Before moving on with other terms, we state some useful facts that are proven in (Défossez et al., 2022).

Fact 1. *Given $0 < a < 1$ and $Q \in \mathbb{N}$ we have,*

$$\sum_{q=0}^{Q-1}a^q q \leq \frac{a}{(1-a)^2}.$$

Fact 2. *Given* $0 < a < 1$ *and* $Q \in \mathbb{N}$ *we have,*

$$\sum_{q=0}^{Q-1} a^q \sqrt{q} \leq \frac{2}{(1-a)^{3/2}}.$$

Fact 3. *Given* $0 < a < 1$ *and* $Q \in \mathbb{N}$ *we have,*

$$\sum_{q=0}^{Q-1} a^q \sqrt{q}(q+1) \leq \frac{4a}{(1-a)^{5/2}}.$$

Now we move onto examining other terms in (17). For C, we make the following change in the index $j = t - l$ which yields the following steps

$$\begin{aligned}
\frac{\alpha^3 L^2}{4R\sqrt{1-\beta_2}}\sqrt{1-\beta_1}\sum_{t=1}^{T}\sum_{l=1}^{t-1}\mathbb{E}[\|u_{t-l}\|^2]\sum_{k=l}^{t-1}\beta_1^k\sqrt{k} &= \frac{\alpha^3 L^2}{4R\sqrt{1-\beta_2}}\sqrt{1-\beta_1}\sum_{t=1}^{T}\sum_{j=1}^{t}\mathbb{E}[\|u_j\|^2]\sum_{k=t-j}^{t-1}\beta_1^k\sqrt{k} \\
&= \frac{\alpha^3 L^2}{4R\sqrt{1-\beta_2}}\sqrt{1-\beta_1}\sum_{j=1}^{T}\mathbb{E}[\|u_j\|^2]\sum_{t=j}^{T}\sum_{k=t-j}^{t-1}\beta_1^k\sqrt{k} \\
&= \frac{\alpha^3 L^2}{4R\sqrt{1-\beta_2}}\sqrt{1-\beta_1}\sum_{j=1}^{T}\mathbb{E}[\|u_j\|^2]\sum_{k=0}^{T-1}\beta_1^k\sqrt{k}\sum_{t=j}^{j+k}1 \\
&= \frac{\alpha^3 L^2}{4R\sqrt{1-\beta_2}}\sqrt{1-\beta_1}\sum_{j=1}^{T}\mathbb{E}[\|u_j\|^2]\sum_{k=0}^{T-1}\beta_1^k\sqrt{k}(k+1) \\
&\overset{(a)}{\leq} \frac{\alpha^3 L^2}{R\sqrt{1-\beta_2}}\sum_{j=1}^{T}\mathbb{E}[\|u_j\|^2]\frac{\beta_1}{(1-\beta_1)^2} \\
&\overset{(b)}{\leq} \frac{\alpha^3 L^2}{R\sqrt{1-\beta_2}}\frac{\beta_1}{(1-\beta_1)^4}\sum_{i\in[d]}\ln\left(1+\frac{v_{T,i}^{(sum)}}{\epsilon}\right) \qquad (19) \\
&\overset{(c)}{\leq} \frac{\alpha^3 L^2}{R\sqrt{1-\beta_2}}\frac{\beta_1}{(1-\beta_1)^4}\sum_{i\in[d]}\ln\left(1+\frac{R^2 T}{(1-\beta_2)\epsilon}\right), \qquad (20)
\end{aligned}$$

where in (a) we use Fact 3, in (b) we use Lemma 4, and in (c) the definition of $v_{T,i}^{(sum)}$. For D, we make the following change in the index $j = t - k$, and obtain that

$$\begin{aligned}
\frac{3\alpha R\sqrt{1-\beta_2}}{\sqrt{1-\beta_1}}\sum_{t=1}^{T}\sum_{k=0}^{t-1}\sqrt{k+1}\beta_1^k\mathbb{E}[\|U_{t-k}\|^2] &= \frac{3\alpha R\sqrt{1-\beta_2}}{\sqrt{1-\beta_1}}\sum_{t=1}^{T}\sum_{j=1}^{t}\sqrt{1+t-j}\beta_1^{t-j}\mathbb{E}[\|U_j\|^2] \\
&= \frac{3\alpha R\sqrt{1-\beta_2}}{\sqrt{1-\beta_1}}\sum_{j=1}^{T}\mathbb{E}[\|U_j\|^2]\sum_{t=j}^{T}\sqrt{1+t-j}\beta_1^{t-j} \\
&\overset{(a)}{\leq} \frac{3\alpha R\sqrt{1-\beta_2}}{\sqrt{1-\beta_1}}\sum_{j=1}^{T}\mathbb{E}[\|U_j\|^2]\frac{2}{(1-\beta_1)^{3/2}} \\
&\overset{(b)}{\leq} \frac{6\alpha R\sqrt{1-\beta_2}}{(1-\beta_1)^2}\sum_{i\in[d]}\ln\left(1+\frac{v_{T,i}^{(sum)}}{\epsilon}\right) \qquad (21) \\
&\overset{(b)}{\leq} \frac{6\alpha R\sqrt{1-\beta_2}}{(1-\beta_1)^2}\sum_{i\in[d]}\ln\left(1+\frac{R^2}{(1-\beta_2)\epsilon}\right), \qquad (22)
\end{aligned}$$

where in (a) we use Fact 2 and in (b) we use Lemma 2. For A, first note that,

$$\Omega_t = \sqrt{\sum_{\tau=1}^{t}\sum_{j=1}^{\tau}\beta_2^{\tau-j}} \leq \sqrt{\frac{t}{(1-\beta_2)}} \leq \sqrt{\frac{T}{(1-\beta_2)}}. \tag{23}$$

Let us change index, $j = t-k$, and use (23):

$$\begin{aligned}
\frac{\alpha}{2R}\sum_{t=1}^{T}\frac{1}{\Omega_t}\sum_{k=0}^{t-1}\beta_1^k\mathbb{E}[\|G_{t-k}^2\|^2] &\geq \frac{\alpha\sqrt{1-\beta_2}}{2R\sqrt{T}}\sum_{t=1}^{T}\sum_{j=1}^{t}\beta_1^{t-j}\mathbb{E}[\|G_j\|^2] \\
&= \frac{\alpha\sqrt{1-\beta_2}}{2R\sqrt{T}}\sum_{j=1}^{T}\mathbb{E}[\|G_j\|^2]\sum_{t=j}^{T}\beta_1^{t-j} \\
&= \frac{\alpha\sqrt{1-\beta_2}}{2(1-\beta_1)R\sqrt{T}}\sum_{j=1}^{T}(1-\beta_1^{T-j+1})\mathbb{E}[\|G_j\|^2] \\
&= \frac{\alpha\sqrt{1-\beta_2}}{2(1-\beta_1)R\sqrt{T}}\sum_{j=0}^{T-1}(1-\beta_1^{T-j})\mathbb{E}[\|\nabla F(x_j)\|^2].
\end{aligned}$$

Note, $\sum_{j=0}^{T-1}(1-\beta_1^{T-j}) = T - \beta_1\frac{1-\beta_1^T}{1-\beta_1} \geq T - \frac{\beta_1}{1-\beta_1} = \tilde{T}$, and let $\tau \in \{0,\dots,T-1\}$ and $\mathbb{P}[\tau = j] \propto 1-\beta_1^{T-j}$, then we have:

$$A \geq \frac{\alpha\sqrt{1-\beta_2}}{2(1-\beta_1)R\sqrt{T}}\tilde{T}\mathbb{E}[\|\nabla F(x_\tau)\|^2]. \tag{24}$$

Inserting (18), (19), (21), (24) in (17) yields that

$$\begin{aligned}
\underbrace{\frac{\alpha\sqrt{1-\beta_2}}{2(1-\beta_1)R\sqrt{T}}\tilde{T}\mathbb{E}[\|\nabla F(x_\tau)\|^2]}_{A} \leq F(x_0) - F^* + \underbrace{\frac{\alpha^2 L}{2}\frac{1}{(1-\beta_1)^2}\sum_{i\in[d]}\ln\left(1+\frac{R^2T}{(1-\beta_2)\epsilon}\right)}_{B} \\
+ \underbrace{\frac{\alpha^3L^2}{R\sqrt{1-\beta_2}}\frac{\beta_1}{(1-\beta_1)^4}\sum_{i\in[d]}\ln\left(1+\frac{R^2T}{(1-\beta_2)\epsilon}\right)}_{C} \\
+ \underbrace{\frac{6\alpha R\sqrt{1-\beta_2}}{(1-\beta_1)^2}\sum_{i\in[d]}\ln\left(1+\frac{R^2}{(1-\beta_2)\epsilon}\right)}_{D}.
\end{aligned}$$

and after some algebra we have,

$$\begin{aligned}
\mathbb{E}[\|\nabla F(x_\tau)\|^2] \leq \frac{2(1-\beta_1)R\sqrt{T}}{\alpha\sqrt{1-\beta_2}\tilde{T}}(F(x_0)-F^*) + \frac{\alpha RL\sqrt{T}d}{\sqrt{1-\beta_2}(1-\beta_1)\tilde{T}}\ln\left(1+\frac{R^2T}{(1-\beta_2)^2\epsilon}\right) \\
+ \frac{2\beta_1\alpha^2L^2\sqrt{T}d}{(1-\beta_2)(1-\beta_1)^3\tilde{T}}\ln\left(1+\frac{R^2T}{(1-\beta_2)\epsilon}\right) + \frac{12R^2\sqrt{T}d}{\sqrt{1-\beta_1}\tilde{T}}\ln\left(1+\frac{R^2T}{(1-\beta_2)\epsilon}\right),
\end{aligned}$$

Equivalently,

$$\mathbb{E}\|\nabla F(x_\tau)\|^2 \leq \frac{2(1-\beta_1)R\sqrt{T}}{\alpha\sqrt{1-\beta_2}\tilde{T}}(F(x_0)-F^*) + C\frac{\sqrt{T}d}{\tilde{T}}\ln\left(1+\frac{R^2T}{(1-\beta_2)\epsilon}\right),$$

where $C = \frac{\alpha RL}{\sqrt{1-\beta_2}(1-\beta_1)} + \frac{2\beta_1\alpha^2L^2}{(1-\beta_2)(1-\beta_1)^3} + \frac{12R^2}{\sqrt{1-\beta_1}}$ which concludes the proof.

Proof of Lemma 3. We start by separating the main term into two:

$$G_{t,i}\frac{m_{t,i}}{\sqrt{\epsilon+v_{t,i}^{(sum)}}} = \sum_{k=0}^{t-1} G_{t,i}\beta_1^{t-k}\frac{g_{t-k,i}}{\sqrt{\epsilon+v_{t,i}^{(sum)}}} \tag{25}$$

$$= \underbrace{\sum_{k=0}^{t-1} G_{t-k,i}\beta_1^{t-k}\frac{g_{t-k,i}}{\sqrt{\epsilon+v_{t,i}^{(sum)}}}}_{A} + \underbrace{\sum_{k=0}^{t-1}(G_{t,i}-G_{t-k,i})\beta_1^{t-k}\frac{g_{t-k,i}}{\sqrt{\epsilon+v_{t,i}^{(sum)}}}}_{B}. \tag{26}$$

For B, we again utilize the fact (13) that $\forall\lambda > 0,\ x,y \in \mathbb{R}\quad xy \le \frac{\lambda x^2}{2} + \frac{y^2}{2\lambda}$, for each dimension with $\lambda = \frac{\sqrt{1-\beta_1}}{2R\sqrt{1-\beta_2}\sqrt{k+1}}, x = |G_{t,i}-G_{t-k,i}|, y = \frac{|g_{t-k,i}|}{\sqrt{\epsilon+v_{t,i}^{(sum)}}}$. Then,

$$|B| \le \sum_{i\in[d]}\sum_{k=0}^{t-1}\beta_1^t\left(\frac{\sqrt{1-\beta_1}}{4R\sqrt{1-\beta_2}\sqrt{k+1}}(G_{t,i}-G_{t-k,i})^2 + \frac{2R\sqrt{1-\beta_2}\sqrt{k+1}g_{t-k,i}^2}{\sqrt{1-\beta_1}(\epsilon+v_{t-k,i}^{(sum)})}\right).$$

Note that $\epsilon + v_{t,i}^{(sum)} \ge \epsilon + v_{t-k,i}^{(sum)}$, hence,

$$\frac{g_{t-k,i}^2}{\epsilon+v_{t,i}^{(sum)}} \le \frac{g_{t-k,i}^2}{\epsilon+v_{t-k,i}^{(sum)}} = U_{t-k,i}^2.$$

Moreover, smoothness of objective function implies

$$\|G_t - G_{t-k}\|^2 \le L^2\|x_{t-1}-x_{t-k-1}\|^2 = L^2\|\sum_{l=1}^{k}\alpha u_{t-l}\|^2 \le \alpha^2L^2k\sum_{l=1}^{k}\|u_{t-l}\|^2$$

As a result,

$$|B| \le \sum_{k=0}^{t-1}\frac{\alpha^2L^2\beta_1^k\sqrt{1-\beta_1}\sqrt{k}}{4R\sqrt{1-\beta_2}}\sum_{l=1}^{k}\|u_{t-l}\|^2 + \sum_{k=0}^{t-1}\frac{R\sqrt{1-\beta_2}\sqrt{k+1}}{\sqrt{1-\beta_1}}\beta_1^k\|U_{t-k}\|^2$$

$$= \frac{\alpha^2L^2}{4R\sqrt{1-\beta_2}}\sqrt{1-\beta_1}\sum_{l=1}^{t-1}\|u_{t-l}\|^2\sum_{k=l}^{t-1}\beta_1^k\sqrt{k} + \frac{R\sqrt{1-\beta_2}}{\sqrt{1-\beta_1}}\sum_{k=0}^{t-1}\sqrt{k+1}\beta_1^k\|U_{t-k}\|^2. \tag{27}$$

For term A, we focus on the main term of the summation $\mathbb{E}\left[G_{t-k}\frac{g_{t-k,i}}{\sqrt{\epsilon+v_{t,i}^{(sum)}}}\right]$. Let $G = G_{t-k,i}, g = g_{t-k,i}, \tilde{v} = \tilde{v}_{t,k+1,i}^{(sum)}, v = v_{t,i}^{(sum)}$. Note that,

$$\tilde{v} - v = v_{t-k,i}^{(sum)} + \mathbb{E}_{t-k-1}\Big[\sum_{\tau=t-k}^{t}\sum_{j=1}^{\tau}\beta_2^{\tau-j}g_j^2\Big] - v_{t,i}^{(sum)}$$

$$= \mathbb{E}_{t-k-1}[\sum_{\tau=t-k}^{t}\sum_{j=1}^{\tau}\beta_2^{\tau-j}g_j^2] - \sum_{\tau=t-k}^{t}v_\tau$$

$$= \underbrace{\mathbb{E}_{t-k-1}[\sum_{\tau=t-k}^{t}\sum_{j=1}^{\tau}\beta_2^{\tau-j}g_j^2]}_{A_1^2} - \underbrace{\sum_{\tau=t-k}^{t}\sum_{j=1}^{\tau}\beta_2^{\tau-j}g_j^2}_{A_2^2}.$$

We continue similar to Lemma 1.

$$\frac{Gg}{\sqrt{\epsilon+v}} = \frac{G^2}{\sqrt{\epsilon+\tilde{v}}} + \underbrace{Gg\frac{A_1^2-A_2^2}{\sqrt{\epsilon+v}\sqrt{\epsilon+\tilde{v}}(\sqrt{\epsilon+v}+\sqrt{\epsilon+\tilde{v}})}}_{C}. \tag{28}$$

We have

$$|C| \leq \underbrace{\frac{|Gg|A_1^2}{\sqrt{\epsilon+v}(\epsilon+\tilde{v})}}_{C_1} + \underbrace{\frac{|Gg|A_2^2}{(\epsilon+v)\sqrt{\epsilon+\tilde{v}}}}_{C_2}.$$

We will utilize (13), for C_1, let $\lambda = \frac{\sqrt{(1-\beta_1)}\sqrt{\epsilon+\tilde{v}}}{2}$, $x = \frac{|G|}{\sqrt{\epsilon+\tilde{v}}}$, $y = \frac{|g|A_1^2}{\sqrt{\epsilon+\tilde{v}}\sqrt{\epsilon+v}}$. Then,

$$C_1 \leq \frac{G^2}{4\sqrt{\epsilon+\tilde{v}}} + \frac{1}{\sqrt{1-\beta_1}} \frac{g^2 A_1^4}{(\epsilon+\tilde{v})^{\frac{3}{2}}(\epsilon+v)}. \tag{29}$$

Given $\epsilon + \tilde{v} \geq A_1^2$ and taking the expectation:

$$\mathbb{E}_{t-k-1}[C_1] \leq \frac{G^2}{4\sqrt{\epsilon+\tilde{v}}} + \frac{1}{\sqrt{1-\beta_1}} \frac{A_1^2}{\sqrt{\epsilon+\tilde{v}}} \mathbb{E}_{t-k-1}\left[\frac{g^2}{(\epsilon+v)}\right].$$

Similarly, for C_2, we let $\lambda = \frac{\sqrt{1-\beta_1}\sqrt{\epsilon+\tilde{v}}}{2A_1^2}$, $x = \frac{|GA_2|}{\sqrt{\epsilon+\tilde{v}}}$, $y = \frac{|A_2 g|}{\epsilon+v}$. Then,

$$C_2 \leq \frac{G^2}{4\sqrt{\epsilon+\tilde{v}}} \frac{A_2^2}{A_1^2} + \frac{1}{\sqrt{1-\beta_1}} \frac{A_1^2}{\sqrt{\epsilon+\tilde{v}}} \frac{g^2 A_2^2}{(\epsilon+v)^2}.$$

Using $\epsilon + v \geq A_2^2$ and $\mathbb{E}_{t-k-1}[\frac{A_2^2}{A_1^2}] = 1$,

$$\mathbb{E}_{t-k-1}[C_2] \leq \frac{G^2}{4\sqrt{\epsilon+\tilde{v}}} + \frac{1}{\sqrt{1-\beta_1}} \frac{A_1^2}{\sqrt{\epsilon+\tilde{v}}} \mathbb{E}_{t-k-1}\left[\frac{g^2}{(\epsilon+v)}\right].$$

Hence,

$$\mathbb{E}_{t-k-1}[|C|] \leq \frac{G^2}{2\sqrt{\epsilon+\tilde{v}}} + \frac{1}{\sqrt{1-\beta_1}} \frac{2A_1^2}{\sqrt{\epsilon+\tilde{v}}} \mathbb{E}_{t-k-1}\left[\frac{g^2}{(\epsilon+v)}\right].$$

Using $A_1 \leq \sqrt{\epsilon+\tilde{v}}$, thus, $A_1 \leq R\sqrt{k+1}\sqrt{1-\beta_2}$:

$$\mathbb{E}_{t-k-1}[|C|] \leq \frac{G^2}{2\sqrt{\epsilon+\tilde{v}}} + \frac{2R\sqrt{k+1}\sqrt{1-\beta_2}}{\sqrt{1-\beta_1}} \mathbb{E}_{t-k-1}\left[\frac{g^2}{(\epsilon+v)}\right].$$

Taking complete expectation, using $\epsilon + v_{t,i}^{(sum)} \geq \epsilon + v_{t-k,i}^{(sum)}$ and reintroducing the indices:

$$\mathbb{E}[|C|] \leq \frac{1}{2}\mathbb{E}\left[\frac{G_{t-k,i}^2}{\sqrt{\epsilon+\tilde{v}_{t,k+1,i}^{(sum)}}}\right] + \frac{2R\sqrt{k+1}\sqrt{1-\beta_2}}{\sqrt{1-\beta_1}} \mathbb{E}_{t-k-1}\left[\frac{g_{t-k,i}^2}{(\epsilon+v_{t-k,i}^{(sum)})}\right].$$

Equivalently,

$$\mathbb{E}[-|C|] \geq -\frac{1}{2}\mathbb{E}\left[\frac{G_{t-k,i}^2}{\sqrt{\epsilon+\tilde{v}_{t,k+1,i}^{(sum)}}}\right] - \frac{2R\sqrt{k+1}\sqrt{1-\beta_2}}{\sqrt{1-\beta_1}} \mathbb{E}_{t-k-1}\left[\frac{g_{t-k,i}^2}{(\epsilon+v_{t-k,i}^{(sum)})}\right] \tag{30}$$

Note,

$$\mathbb{E}[|A|] \geq \sum_{i\in[d]} \sum_{k=0}^{t-1} \beta_1^k \left(\mathbb{E}\left[\frac{G_{t-k,i}^2}{\sqrt{\epsilon+\tilde{v}_{t,k+1,i}^{(sum)}}}\right] + \mathbb{E}[-|C|]\right)$$

Then, inserting (30), we have:

$$\mathbb{E}[|A|] \geq \sum_{i\in[d]}\sum_{k=0}^{t-1}\beta_1^k\left(\mathbb{E}\left[\frac{G_{t-k,i}^2}{\sqrt{\epsilon+\tilde{v}_{t,k+1,i}^{(sum)}}}\right]-\left(\frac{1}{2}\mathbb{E}\left[\frac{G_{t-k,i}^2}{\sqrt{\epsilon+\tilde{v}_{t,k+1,i}^{(sum)}}}\right]+\frac{2R\sqrt{k+1}\sqrt{1-\beta_2}}{\sqrt{1-\beta_1}}\mathbb{E}_{t-k-1}\left[\frac{g^2}{(\epsilon+v_{t-k,i}^{(sum)})}\right]\right)\right)$$
$$=\frac{1}{2}\left(\sum_{i\in[d]}\sum_{k=0}^{t-1}\beta_1^k\mathbb{E}\left[\frac{G_{t-k,i}^2}{\sqrt{\epsilon+\tilde{v}_{t,k+1,i}^{(sum)}}}\right]\right)-\frac{2R\sqrt{1-\beta_2}}{\sqrt{1-\beta_1}}\sum_{k=0}^{t-1}\beta_1^k\sqrt{k+1}\mathbb{E}[\|U_{t-k}\|^2]. \tag{31}$$

Note, in (25) we have,

$$G_{t,i}\frac{m_{t,i}}{\sqrt{\epsilon+v_{t,i}^{(sum)}}}=A+B\geq A-|B|,$$

taking the expectation of both sides yields

$$\mathbb{E}\left[G_{t,i}\frac{m_{t,i}}{\sqrt{\epsilon+v_{t,i}^{(sum)}}}\right]=\mathbb{E}[A]+\mathbb{E}[B]\geq\mathbb{E}[A]+\mathbb{E}[-|B|], \tag{32}$$

Inserting (31) and negated (27) into (32) results in

$$\mathbb{E}\left[G_{t,i}\frac{m_{t,i}}{\sqrt{\epsilon+v_{t,i}^{(sum)}}}\right]\geq\frac{1}{2}\left(\sum_{i\in[d]}\sum_{k=0}^{t-1}\beta_1^k\mathbb{E}\left[\frac{G_{t-k,i}^2}{\sqrt{\epsilon+\tilde{v}_{t,k+1,i}^{(sum)}}}\right]\right)-\frac{3R\sqrt{1-\beta_2}}{\sqrt{1-\beta_1}}\sum_{k=0}^{t-1}\sqrt{k+1}\beta_1^k\mathbb{E}\|U_{t-k}\|^2$$
$$-\frac{\alpha^2L^2}{4R\sqrt{1-\beta_2}}\sqrt{1-\beta_1}\sum_{l=1}^{t-1}\|u_{t-l}\|^2\sum_{k=l}^{t-1}\beta_1^k\sqrt{k},$$

which completes the proof. □

Proof of Lemma 4. For some t,

$$\frac{c_t^2}{\epsilon+b_t}\leq\frac{1}{1-\beta_1}\sum_{\tau=1}^{t}\beta_1^{t-\tau}\frac{a_\tau^2}{\epsilon+b_t}.$$

We have $\epsilon+b_t\geq\epsilon+b_\tau$ for any $\tau\leq t$, then,

$$\begin{aligned}\sum_{t=1}^{T}\frac{c_t^2}{\epsilon+b_t}&\leq\frac{1}{1-\beta_1}\sum_{t=1}^{T}\sum_{\tau=1}^{t}\beta_1^{t-\tau}\frac{a_\tau^2}{\epsilon+b_\tau}\\&=\frac{1}{1-\beta_1}\sum_{\tau=1}^{T}\frac{a_\tau^2}{\epsilon+b_\tau}\sum_{\tau=1}^{t}\beta_1^{t-\tau}\\&\leq\frac{1}{(1-\beta_1)^2}\sum_{\tau=1}^{T}\frac{a_\tau^2}{\epsilon+b_\tau}\\&\leq\frac{1}{(1-\beta_1)^2}\ln\left(1+\frac{b_T}{\epsilon}\right),\end{aligned}$$

where in the last inequality we apply Lemma 2 with $\rho=0$. Note, from the definition of b_T we also have that $b_T\leq\frac{R^2T}{1-\beta_2}$. □

C. Additional Experiments and Details

C.1. Overall Parameterization in the Experiments

For the *first order moment term* we have,

$$m_t=\beta_1m_{t-1}+(1-\beta_1)g_t$$

$$n_t = \beta_3 n_{t-1} + (1-\beta_3)(g_t - g_{t-1})$$
$$u_t = Lion(g_t, m_t^{lion}),$$

where β_1 controls the strength of heavy-ball momentum and β_3 extends the equation to Nesterov momentum. The first order moment equations essentially capture the updates for Adan and Adam. For the *second order moment term* we have,

$$\hat{g}_t = g_t + \beta_3(g_t - g_{t-1})$$
$$\tilde{g}_t^2 = c_{t-1}\hat{g}_t^2 + (1-c_{t-1})(\bar{v}_{t-1} + \hat{g}_t^2 \text{sign}(\hat{g}_t^2 - \bar{v}_{t-1}))$$
$$\bar{v}_t = \beta_2 \bar{v}_{t-1} + (1-\beta_2)\tilde{g}_t^2$$
$$\tilde{v}_t = (\bar{v}_t + (t-1)\tilde{v}_{t-1})/t$$
$$v_t = \rho \bar{v}_t + (1-\rho)\tilde{v}_t$$

where c controls whether the difference of consequent $\bar{v}_t$s grows with $g_t^2 \text{sign}(g_t^2 - \bar{v}_{t-1})$, as in YOGI, or $g_t^2(g_t^2 - \bar{v}_{t-1})$ as in Adam; the former, prevents rapid increases in effective learning rate and provides more controlled updates. For the second order moment term, ρ_t determines whether to use, less noisy, average of past v_ts (we call this method AVGrad and defer its formal introduction to the next section) or the current v_t, which may be more noisy but up to date. Lastly, the *update term* together with the decoupled weight decay is as follows,

$$x_{t-1} = x_{t-1} - \lambda \alpha_t x_{t-1}$$
$$x_t = x_{t-1} - \alpha_t \Big(\gamma \frac{m_t + \beta_3 n_t}{\sqrt{v_t} + \epsilon} + (1-\gamma)\text{sign}(u_t) \Big),$$

where ϵ is the stability parameter, $\text{sign}(u_t)$ is the update term of Lion optimizer (see Table 1), γ controls whether to do a Lion type update or not.

C.2. Training setup for the experiments

On OpenWebText, we use a global batch size of 480 sequences, cosine learning rate schedule with the peak learning rate of 6×10^{-4} (1.5×10^{-4} for Lion) and the final learning rate of 1.5×10^{-5} (as chosen for Sophia algorithm in (Liu et al., 2023)). For Lion we use the tuned learning rate from (Liu, 2023) for the same setting. For Adam we found that employing Sophia's learning rate schedule results in better loss compared to employing Adam's learning rate schedule in (Liu et al., 2023). For our methods: AVGrad, `MADA`, `MADA-FS`, we directly employ Adam's learning rate and learning schedule without any tuning since our goal is to demonstrate that `MADA` can be plugged in place of Adam without any changes in learning rate schedule.

We run the experiment for 100,000 iterations (first 2000 are warmup iterations) which corresponds to training on ~ 492B tokens. We use a weight decay parameter of 0.1. On Shakespeare, we use a batch size of 64, cosine learning rate schedule with the peak learning rate of 10^{-3} and the final learning rate of 10^{-4}. We run the experiment for 5,000 iterations (first 100 are warmup iterations). We run our experiments on AWS `p5.48xlarge` instances equipped with 8 NVIDIA H100 GPUs. To be able utilize multiple GPU's we allreduced hyper-gradients across GPU's.

We initialize `MADA` with $\beta_3 = 0.9, \rho = 0$ on top of Adam's established β_1, β_2 parameters for OpenWebText experiments as it resulted in a better validation loss. For Shakespeare experiments, we compare `MADA` against Adam over a grid of (β_1, β_2) values, where in the case of `MADA` (β_1, β_2) represents the initial values. For the OpenWebText experiments, we do not update hyper-parameters for the first 50 iterations for the sake of stability. For both experiments we use SGD as the hyper-parameter optimizer. On Shakespeare, for 10M model we use $2.5e{-}3$ learning rate and 0.5 momentum (we do not use momentum for γ) for learning the hyper-parameters. On OpenWebText (and for 1.5B model on Shakespeare) we use a learning rate of $5e{-}4$ for training β_1, β_2 and $1e{-}1$ for other hyper-parameters.

Vision Tasks. For 5-layer model experiments, we use a CNN whose first two layers are convolutional layers with 6 and 16 output channels and 5 kernel size; and last 3 layers are fully connected layers with 120, 84, and 10 output dimensionality. We use ReLU activation in all layers except the last one and maxpool on the outputs convolutional layers. We use a constant learning rate of $1e{-}3$ for MADA, Adam, HyperAdam; and $1e{-}2$ learning rate and 0.9 momentum coefficient for SGD with momentum. We initialize `MADA` from Adam state, we set hyper learning rate for β_1, β_2 to $1e{-}4$ and for the other variables to $1e{-}2$. We use batch size of 256 and train for 50 epochs. For ResNet-9 experiments, we use the model implementation

from https://github.com/Moddy2024/ResNet-9. We use one cycle learning rate scheduler with $1e{-}2$ peak learning rate for MADA, Adam, HyperAdam and $1e{-}1$ peak learning rate for SGD with momentum. Again we initialize MADA from Adam, we set hyper learning rate for β_1 to $1e{-}4$, for β_2 to $1e{-}4$ and for the other variables to $1e{-}2$. We use batch size of 400 and train for 50 epochs.

C.3. Additional Experiments

Method	OpenWebText (validation loss)	OpenWebText	Wikitext	Lambada
MADA ($\rho_0 = 0$)	2.8853	17.9084	61.4689	**73.1291**
MADA-FS ($\rho_0 = 0$)	2.8838	17.8822	63.9886	75.6158

Table 6: Validation loss on OpenWebText and validation perplexities on OpenWebText, Wikitext and Lambada datasets of GPT-2 (125M) models trained on OpenWebText with MADA. The initial optimzier state is AVGrad.

Method	OpenWebText (validation loss)	OpenWebText	Wikitext	Lambada
Adam	2.6527	14.1928	50.1410	53.7468
MADA	2.6422	14.0441	43.8067	53.7575

Table 7: Validation loss on OpenWebText and validation perplexities on OpenWebText, Wikitext and Lambada datasets of GPT-2 (355M) models trained on OpenWebText with Adam and MADA. The initial state of MADA is AVGrad + Adan ($\rho = 0, \beta_3 = 0.9$).

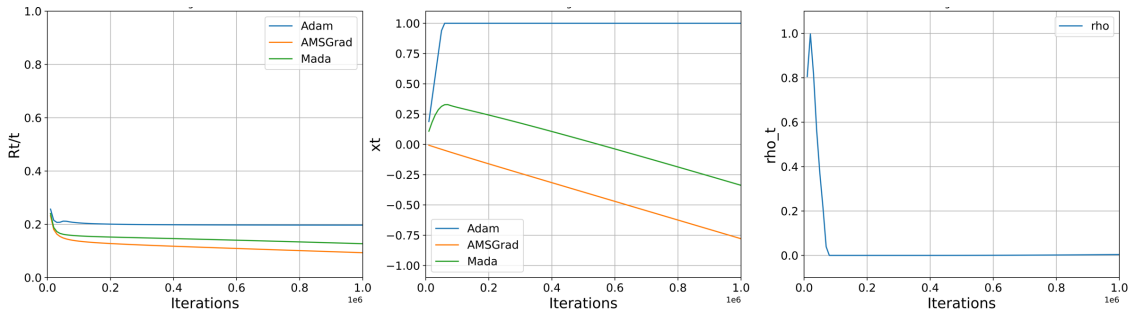

Figure 9: Average regret, x_t, ρ_t with respect to iterations for MADA on simple convex function.

Synthetic convex experiment. The online learning experiment given in (Reddi et al., 2018) to motivate AMSGrad is as follows:

$$g_t(x) := \begin{cases} 1010x & \text{for } t \bmod 101 = 1 \\ -10x & \text{otherwise} \end{cases} \tag{33}$$

with constraint set $x \in [-1, 1]$ and t denotes the time index in the online learning setting. This is an example where Adam notably fails to converge to the optimal solution, $x = -1$, whereas AMSGrad and AVGrad reach the optimum. In Figure 9, we plot the average regret which is defined by $(g_t(x) - g_t(-1))/t$, as well as the evolution of x and ρ with respect to iterations. Here, we reduce the effect of Lion by assigning a small hyper-learning rate to γ, since signSGD based methods neutralize the effect of large gradients which allow faster progress towards the optimum. We observe that even when we initialize MADA from Adam ($\rho_0 = 1$), it quickly recovers AVGrad ($\rho_t \to 0$), which is the right optimizer to use for this example. This experiment shows that MADA can learn to behave like AVGrad even when initialized from Adam.

D. Profiling Results

We profile the memory footprints of the methods and find that the peak memory usage is 15.5 GB for Adam, and 24.9 GB for MADA; time per iteration is 0.65s for Adam and 0.85s for MADA. If we exclude LION (which contributes the least in this setting) MADA results in 22.2GB of memory usage and 0.72s time per iteration. We also investigated a large-batch setting. Comparing Adam and MADA respectively, training GPT-2 (124M) with a batch size of 3840 (~ 4 million tokens) requires memory usage of 51.5GB and 61.5GB while the time per iteration is 4.72s and 5.20s, we would like to note that our method is able to outperform Adam with the same wall-clock time as well. The main reason for the memory increase is the optimizer states; due to the increased number of base optimizers with such a large model size. In contrast, for vision tasks, we observe virtually no additional memory used due to the smaller size of the models and minimal overhead of storing optimizer states.