IO-ADAM: RETHINKING MEMORY-EFFICIENT ADAPTIVE OPTIMIZERS FROM GRADIENT COMPUTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Adaptive Moment Estimation (Adam) is one of the most popular stochastic optimizers for deep neural network training and has become the default optimizer in many scenarios, especially on language tasks. With the first and second moment estimation, Adam provides adaptive learning rates for each parameter, significantly outperforming Stochastic Gradient Descent (SGD). However, as the deep neural networks become larger, the estimation of the first and second moments takes up substantial memory, motivating methods to reduce the memory usage for adaptive optimizers. In this paper, we propose to rethink the first and second moment estimation from a gradient computation perspective. The gradient of the weight matrix is the multiplication of the input and the gradient of the output. Instead of trying to find a low-rank approximation for the first and second moment estimation as in previous works, we propose to track the input and the output gradient for efficient moment estimation. We provide analyses on the connection and difference between our proposed method, the widely used Adam optimizer, and previous memory-efficient optimizers proposed to reduce the memory usage. We conduct experiments to verify the effectiveness of our method, where our method reduces the memory usage by up to 30% while preserving similar performance or even improving the performance of Adam.

1 INTRODUCTION

Deep learning has achieved remarkable success across a wide range of domains, driven in large part by the design of effective optimization algorithms for deep neural network training. From Stochastic Gradient Descent (SGD) [28] to adaptive learning rate methods such as AdaGrad [7] and RMSprop [10], efforts have been made to accelerate convergence and improve neural network performance. Specifically, Adaptive Moment Estimation (Adam) [13] proposes to estimate the first and second moments of the gradient and calculates an adaptive learning rate for each parameter, combining the advantages of AdaGrad and RMSprop. Due to its fast convergence and minimal need for hyperparameter tuning, Adam has become one of the most popular optimizers. Despite the satisfactory performance of Adam, maintaining the estimation of the first and second moments of the gradient requires twice the memory for the parameters. As neural networks become larger, the substantial memory usage of Adam motivates the development of memory-efficient adaptive optimizers. A typical approach to reduce memory usage is to find a low-rank approximation for the estimation of the first or second moment of the gradient. Adafactor [29] proposes to remove the first moment estimation and estimate the second moment with only the per-row and per-column sums of the square gradients. There are various approaches [8; 11; 23; 2] analyzing and utilizing the low-rank approximation of weight gradients. More recently, Galore [34] proposes to project gradients to a low-dimensional subspace based on singular value decomposition (SVD) and apply adaptive moment estimation on the projected gradients.

From a gradient computation perspective, this paper proposes to investigate and rethink previous memory-efficient approaches using gradient low-rank approximation. Without loss of generality, we consider the fundamental component of most neural networks, a linear transformation with a weight matrix. Through back propagation, the gradient of the weight matrix is calculated by multiplying the input and the gradient of the output. It reveals a key structural property: the weight gradient can be naturally decomposed into the product of the activation from the previous layer (input) and the error signal from the next layer (output gradient), where the number of samples determines the rank

of the gradient. This gradient decomposition perspective in each sample's contribution to the weight gradient provides a principled and fine-grained perspective for rethinking how optimizer states can be represented and updated more efficiently. Unlike previous approaches trying to decompose the gradients, we propose to track the input and output gradients before the weight gradient computation and utilize these two components to estimate the first and second moments of the weight gradient. By separately storing the input and output gradients instead of the moment estimation in full-matrix form, we reduce the memory usage while preserving an accurate moment estimation.

We analyze the connection and difference between our method and the Adam optimizer. We show that the second moment estimation from our method is the upper bound of that in Adam, which also provides the same regret bound on the convergence rate as that for Adam. We further analyze the difference brought by the larger second-moment estimation from our method. A larger second-moment estimation reduces the element-wise learning rate, which means our method requires a larger learning rate to compensate for this discrepancy. We introduce two complementary adjustments to narrow the gap further in a more principled manner. First, we incorporate a lightweight buffering mechanism that accumulates input and output gradients across multiple batches, which would produce more accurate second-moment estimation and more closely approximate the moment estimation in Adam. Second, inspired by Hölder's inequality, we adjust the exponents applied to the input and output gradients, allowing for a tighter upper bound on the second moment estimate. To verify the effectiveness of our method, we conduct experiments on various language and vision datasets, where our method achieves similar performance as the Adam optimizer with nearly half of the memory usage. **We summarize the contribution of this paper in the following.**

- We introduce a novel memory-efficient adaptive optimizer. Unlike prior gradient-factorization approaches, we propose to track the input and the output gradient for the first and second moment estimation, reflecting the intrinsic structure of the gradient.
- We provide analyses of the difference and connection between the proposed optimizer and the widely used Adam optimizer. We show that our estimated second-order moment is an upper bound of that in the Adam optimizer and provide a similar regret bound on the convergence rate.
- We conduct extensive experiments to verify the effectiveness of our proposed memory-efficient optimizer. Our optimizer achieves substantial memory savings, reducing memory usage by up to 40%, while preserving or improving upon Adam's optimization performance.

2 Related Works

Adaptive Moment Estimation (Adam). Adam [13] is a widely adopted adaptive learning rate optimizer, which combines the advantages from AdaGrad [7] and RMSprop [10]. By maintaining exponential moving averages of both the weight gradient itself and the squared weight gradient, Adam dynamically adjusts individual learning rates for each parameter. We provide the Adam algorithm in Alg. 1. Thanks to its fast convergence and minimal requirement for hyperparameter tuning, the Adam optimizer has become the default optimizer in many scenarios. Following the wide adoption of Adam, many works have sought to deepen the theoretical understanding of its behavior [33; 15; 4] and to address its limitations through refinement on the algorithm [27; 20; 33]. Specifically, the first and second moment estimation of the Adam optimizer takes up twice the memory as the parameters, and has motivated approaches to reduce the memory usage.

Memory Efficient Approaches for Adaptive Optimizers. Various approaches have been introduced to reduce the memory usage. A pioneering approach is Adafactor [29], which introduces a factorized second moment estimation using the per-row and per-column sums of the square gradients. [5; 16; 17] propose to quantize the optimizer states to reduce the memory usage, while [21] proposes to fuse the back propagation and the optimizer update such that the gradient is released right after its computation. Notably, a branch of work focuses on finding a low-rank approximation of the gradient, where the low-rank property of the gradient has been studied and used to reduce the training cost [29; 34]. Most recently, Galore [34] proposes projecting the gradient to low-rank subspaces and estimating moments for the projected gradients. AdamSNSM [24] also proposes the subset norm and subspace momentum, estimating the moment in the subspace. Unlike previous approaches, this paper proposes rethinking gradient decomposition from a gradient computation perspective, where one of the most natural decompositions of the weight gradient is to decompose it into the input and output gradients. From this perspective, we propose a novel memory-efficient

Algorithm 1 Adam

Input: Loss function $\mathcal{L}$, trainable parameters $\theta = \{W_1, W_2, \cdots, W_n\}$, step size α, hyperparameters β_1, β_2, and ϵ

Initialize $M^0_{W_i} = 0$, $V^0_{W_i} = 0$;
for $t = 1$ to T **do**
  for $W_i \in \theta$ **do**
    $G^t_{W_i} \leftarrow \nabla_{W_i}\mathcal{L}$;
    $M^t_{W_i} \leftarrow \beta_1 M^{t-1}_{W_i} + (1-\beta_1)\cdot G^t_{W_i}$;
    $V^t_{W_i} \leftarrow \beta_2 V^{t-1}_{W_i} + (1-\beta_2)\cdot (G^t_{W_i})^2$;
    $\hat{M}^t_{W_i} \leftarrow M^t_{W_i}/(1-\beta^t_1)$;
    $\hat{V}^t_{W_i} \leftarrow V^t_{W_i}/(1-\beta^t_2)$;
    $W_i \leftarrow W_i - \alpha\hat{M}^t_{W_i}/(\sqrt{\hat{V}^t_{W_i}} + \epsilon)$;
  end for
end for

Table 1: Comparison between optimizers. We provide a description and the complexity of second-moment estimation.

Method	Description	Complexity
Adam [13]	Estimate the first moment and second moment of weight gradient by exponential moving average of the weight gradient and the squared weight gradient.	mn
Adafactor [29]	Remove first moment estimate, use the per-column sum and per-row sum of the squared weight gradient for the second moment estimate.	$m+n$
Galore [34]	Project gradients into subspaces and estimate the first and second moments in the subspace.	$mr+nr$, r is the rank.
IO-Adam (Ours)	We track the input and output gradient before gradient computation and estimate moments with the input and output gradient.	$mc+nc$, c is the buffer size.

adaptive optimizer, namely adaptive moment estimation by input and output gradient (IO-Adam). We provide a comparison between our method and previous works in Table 1.

3 Reducing Memory by Tracking Input and Output Gradient

3.1 Preliminaries

Without loss of generality, we consider the fundamental component of neural networks, a linear transformation $\mathbf{y} = W\mathbf{x}$, where $\mathbf{y} \in \mathbb{R}^n, \mathbf{x} \in \mathbb{R}^m$ is the input and output and $W \in \mathbb{R}^{n\times m}$ is the weight matrix. For a loss $\mathcal{L}$, the gradient of the weight W is calculated through back propagation:

$$\nabla_W \mathcal{L} = \nabla_{\mathbf{y}}\mathcal{L} \otimes \mathbf{x} = (\nabla_{\mathbf{y}}\mathcal{L})\mathbf{x}^\top. \tag{1}$$

Since the input $\mathbf{x}$ and output gradient $\nabla_{\mathbf{y}}\mathcal{L}$ are vectors, the gradient is of rank 1. Similarly, when the inputs are in batches with $Y = WX$ where $Y = (\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_{bs}) \in \mathbb{R}^{n\times bs}, X = (\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_{bs}) \in \mathbb{R}^{m\times bs}$ and $bs \in \mathbb{R}$ is the batch size, we have:

$$\nabla_W \mathcal{L} = (\nabla_Y \mathcal{L})X^\top = \sum_i^{bs} \nabla_{\mathbf{y}_i}\mathcal{L} \otimes \mathbf{x}_i, \tag{2}$$

where the rank of the weight gradient is equal to the batch size bs. From the gradient computation perspective, a natural decomposition of the weight gradient is to decompose the weight gradient into the input and the output gradient. Motivated by this perspective, we propose to separately track the input and the output gradient for a memory-efficient adaptive optimizer.

3.2 The First Moment Estimation

Fused Momentum Update. The first moment estimate, or the momentum, is introduced to accelerate the convergence of stochastic optimizers. In Adam, the first moment estimation is stored and updated as an exponential moving average of the gradient.

$$M^t_W = \beta_1 M^{t-1}_W + (1-\beta_1)\cdot \nabla_W \mathcal{L}, \tag{3}$$

where M^t_W is the momentum at the t-th step and β_1 is the coefficient. Since we propose to track the input and the output gradient before weight gradient calculation, the momentum update could be fused with the gradient calculation, similar to the fused backpropagation as in [21]. Instead of conducting backpropagation and then updating the momentum, the weight gradient does not need to be stored in memory, where the momentum could be directly updated by

$$M^t_W = \beta_1 M^{t-1}_W + (1-\beta_1)\cdot (\nabla_Y \mathcal{L})X^\top. \tag{4}$$

Algorithm 2 IO-Adam (ours)

Input: Loss function $\mathcal{L}$, trainable parameters $\{W_1, W_2, \cdots, W_n\}$ and corresponding input and output at every step $\{X_W^t, Y_W^t\}$, step size α, buffer size b, hyperparameters β_1, β_2, and ϵ.

Initialize $M_{W_i}^0 = 0$, $C_{W_i}^0 = 0$, $R_{W_i}^0 = 0$;
for $t = 1$ to T **do**
  for $W_i \in \theta$ **do**
    $M_{W_i}^t \leftarrow \beta_1 M_{W_i}^{t-1} + (1-\beta_1) \cdot (\nabla_{Y_{W_i}^t} \mathcal{L}) X_{W_i}^{t\top}$;
    ▷ Fused moment update.
    $C_{W_i}^t \leftarrow \beta_2 C_{W_i}^{t-1} + (1-\beta_2) X_{W_i}^{t2} (1_{bs} e_{(t \bmod b)}^\top)$;
    $R_{W_i}^t \leftarrow \beta_2 R_{W_i}^{t-1} + (1-\beta_2) (\nabla_{Y_{W_i}^t} \mathcal{L})^2 (1_{bs} e_{(t \bmod b)}^\top)$;
    ▷ Separately track input and output gradient.
    $V_{W_i}^t = C_{W_i}^t R_{W_i}^{t\top}$;
    $\hat{M}_{W_i}^t \leftarrow M_{W_i}^t / (1-\beta_1^t)$;
    $\hat{V}_{W_i}^t \leftarrow V_{W_i}^t / (1-\beta_2^t)^2$;
    ▷ Modified bias correction.
    $W_i \leftarrow W_i - \alpha \hat{M}_{W_i}^t / (\sqrt{\hat{V}_{W_i}^t} + \epsilon)$;
  end for
end for

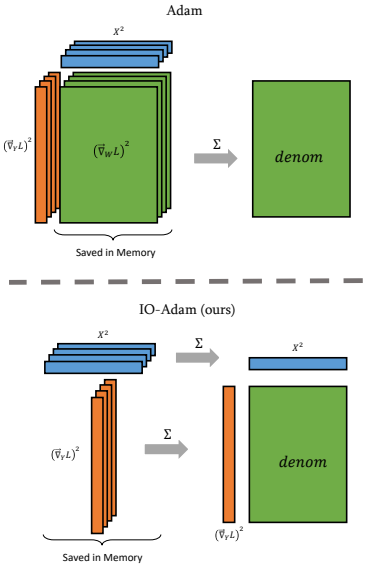


Figure 1: Illustration of the difference between our method and Adam.

Buffering Mechanism. For scenarios where the batch size is rather small, we could keep a buffer to record the input and the output gradient, such that

$$\begin{aligned} B_{\mathbf{x}}^t &= \left((1-\beta_1)\mathbf{x}^t, \beta_1(1-\beta_1)\mathbf{x}^{t-1}, c \cdots, \beta_1^c(1-\beta_1)\mathbf{x}^{t-c}\right), \\ B_{\nabla_y \mathcal{L}}^t &= \left((1-\beta_1)\nabla_{y^t}\mathcal{L}, \beta_1(1-\beta_1)\nabla_{y^{t-1}}\mathcal{L}, \cdots, \beta_1^c(1-\beta_1)\nabla_{\mathbf{y}^{t-c}}\mathcal{L}\right). \end{aligned} \tag{5}$$

$B_{\mathbf{x}}^t$ and $B_{\nabla_y \mathcal{L}}^t$ represent the buffer and $\mathbf{x}^t$ and $\nabla_{y^t}\mathcal{L}$ is the input and output gradient at the t-th step. By tracking the most recent c batches, the moment estimation is provided by

$$M_W^t = B_{\nabla_y \mathcal{L}}^t B_{\mathbf{x}}^{t\top}. \tag{6}$$

As long as $c \cdot bs \cdot (m+n) < m \cdot n$, keeping the buffer would reduce the memory usage. Note that, when c is set to 1, it is equivalent to an optimizer without momentum, such as the Adafactor [29]. Given the large scale of modern neural networks, separately storing the input and output gradients could still generally reduce the memory usage compared to storing the whole weight gradient.

3.3 The Second Moment Estimation

The second moment estimate is used to adjust the learning rate for each parameter adaptively. In Adam, the second moment estimation is achieved with the exponential moving average of the squared weight gradient:

$$V_W^t = \beta_2 V_W^{t-1} + (1-\beta_2) \cdot (\nabla_W \mathcal{L})^2, \tag{7}$$

where V_W^t is the second moment estimation at the t-th step and β_2 is the coefficient. The essence of the second moment estimate is that the learning rate would be adaptively changed for each parameter, where the parameters with larger gradient norm would have a smaller learning rate. From the gradient calculation perspective, as in Eq. 1, parameters that generally receive larger inputs or error signals (output gradient) would have a smaller learning rate. Therefore, we propose to track the squared input and the squared output gradient for the estimation of the second moment.

$$\begin{aligned} \mathbf{c}_W^t &= \beta_2 \mathbf{c}_W^{t-1} + (1-\beta_2) \cdot X^2 1_{bs}, \\ \mathbf{r}_W^t &= \beta_2 \mathbf{r}_W^{t-1} + (1-\beta_2) \cdot (\nabla_Y \mathcal{L})^2 1_{bs}, \\ V_W^t &= \mathbf{r}_W^t \mathbf{c}_W^{t\top}. \end{aligned} \tag{8}$$

$1_{bs} \in \mathbb{R}^{bs}$ represents an all-one vector with bs elements. $\mathbf{c}_W^t \in \mathbb{R}^m$ and $\mathbf{r}_W^t \in \mathbb{R}^n$ are the exponential moving averages of the sum of the squared input and the squared output gradient over a batch.

We emphasize the connection and difference between our method and the Adafactor [29]. In Adafactor, the second moment estimate comprises the squared gradient's per-row sum and per-column sum. The per-row sum contains the information from the squared output gradient, while the per-column sum contains the information from the squared input. Specifically, when batch size is 1, the per-row sum and per-column sum in Adafactor are equivalent to $\|\mathbf{x}\|^2 \cdot \nabla_{\mathbf{y}}\mathcal{L}^2$ and $\|\nabla_{\mathbf{y}}\mathcal{L}\|^2 \cdot \mathbf{x}^2$. Instead of operating on the weight gradient, we separately track the input and output gradients, avoiding the variance brought by mixing the squared values from the input and output gradients.

Generally, the second moment estimation in our method would be larger than that in Adam, indicating our method requires a larger learning rate for a similar performance. The reasons are twofold. Firstly, the second moment estimate in Adam tracks the square of the sum across a batch of samples, while our method tracks the sum of squared values. By Cauchy-Buniakowsky-Schwarz Inequality, our second moment estimate is the upper bound of that in Adam. We provide a more detailed analysis in Sec. 3.4. Secondly, the C_W^t and R_W^t in Eq. 8 accumulate values across different batches, multiplying them to introduce the product of input and output gradient from different batches. To better approximate the second moment estimate in Adam, we propose to maintain two matrices instead of vectors with $C_W^t \in \mathbb{R}^{m\times b}$ and $R_W^t \in \mathbb{R}^{n\times b}$, where the columns in the matrices are updated alternately. This means that each column is updated for every b batch, reducing the product of input and output gradients from different batches. Here, we rewrite Eq. 8 as:

$$\begin{aligned} C_W^t &= \beta_2 C_W^{t-1} + (1-\beta_2)\cdot X^2(1_{bs}e_{(t \bmod b)}^\top), \\ R_W^t &= \beta_2 R_W^{t-1} + (1-\beta_2)\cdot (\nabla_Y \mathcal{L})^2(1_{bs}e_{(t \bmod b)}^\top), \\ V_W^t &= C_W^t R_W^{t\top}. \end{aligned} \tag{9}$$

Let $e_{(t \bmod b)} \in \mathbb{R}^b$ denote a standard basis vector with its $(t \bmod b)$-th component equal to 1 and all other components are 0, which controls which column of C and R to be updated. Since C and R are separately updated, the bias correction is also modified. We present our method in Algorithm 2.

3.4 The Connection and Difference between Our Method and Adam

The primary difference between our method and Adam lies in the estimation of the second moment, where Adam utilizes the squared gradient, whereas our method employs the squared input and output gradients. In this section, we analyze the connection between our method and Adam and further investigate the theoretical convergence properties.

3.4.1 The Upper Bound of the Second Moment Estimate

In this section, we show that our second moment estimate is the upper bound of the second moment estimate in Adam. Let $\frac{\partial \mathcal{L}}{\partial \mathbf{y}_{ij}}$ denotes the j-th element in $\nabla_{\mathbf{y}_i}\mathcal{L}$ and $\mathbf{x}_{ij}$ denotes the j-th element in $\mathbf{x}$. For the weight gradient, with Eq. 2 we have:

$$\nabla_W \mathcal{L} = \begin{pmatrix} \sum_i^{bs} \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{i1}}\cdot \mathbf{x}_{i1} & \sum_i^{bs} \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{i1}}\cdot \mathbf{x}_{i2} & \cdots & \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{i1}}\cdot \mathbf{x}_{im} \\ \sum_i^{bs} \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{i2}}\cdot \mathbf{x}_{i1} & \sum_i^{bs} \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{i2}}\cdot \mathbf{x}_{i2} & \cdots & \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{i2}}\cdot \mathbf{x}_{im} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_i^{bs} \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{in}}\cdot \mathbf{x}_{i1} & \sum_i^{bs} \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{in}}\cdot \mathbf{x}_{i2} & \cdots & \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{in}}\cdot \mathbf{x}_{im} \end{pmatrix}. \tag{10}$$

For the element $(\nabla_W \mathcal{L})_{[j,k]}$ at the j-th column and the k-th row of the matrix, we have:

$$(\nabla_W \mathcal{L})_{[j,k]} = \sum_i^{bs} \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{ij}}\cdot \mathbf{x}_{ik}. \tag{11}$$

Similarly, for the squared weight gradient, we have,

$$(\nabla_W \mathcal{L})^2_{[j,k]} = \left(\sum_i^{bs} \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{ij}}\cdot \mathbf{x}_{ik}\right)^2. \tag{12}$$

In our method, we take the sum of the squared input and output gradients, where

$$X^2 1_{bs} = \left(\sum_i^{bs} \mathbf{x}_{i1}^2, \quad \sum_i^{bs} \mathbf{x}_{i2}^2, \quad \cdots, \sum_i^{bs} \mathbf{x}_{im}^2 \right)^\top,$$
$$(\nabla_Y \mathcal{L})^2 1_{bs} = \left(\sum_i^{bs} (\frac{\partial \mathcal{L}}{\partial y_{i1}})^2, \quad \sum_i^{bs} (\frac{\partial \mathcal{L}}{\partial y_{i2}})^2, \quad \cdots, \sum_i^{bs} (\frac{\partial \mathcal{L}}{\partial y_{in}})^2 \right)^\top. \tag{13}$$

Our second moment estimate is based on the outer product between the two vectors in Eq. 13. The element at the j-th column and the k-th row of the outer product is given by

$$\left((\nabla_Y \mathcal{L})^2 1_{bs} \otimes X^2 1_{bs} \right)_{[j,k]} = \left(\sum_i^{bs} (\frac{\partial \mathcal{L}}{\partial \mathbf{y}_{ij}})^2 \right) \cdot \left(\sum_i^{bs} \mathbf{x}_{ik}^2 \right). \tag{14}$$

Comparing Eq. 12 to Eq. 14, by the Cauchy-Buniakowsky-Schwarz Inequality, we have

$$\forall j, k : (\nabla_W \mathcal{L})^2_{[j,k]} \leq \left((\nabla_Y \mathcal{L})^2 1_{bs} \otimes X^2 1_{bs} \right)_{[j,k]}. \tag{15}$$

Eq. 15 indicates that the outer product of squared input and squared output gradient is the upper bound of the squared weight gradient. Therefore, the second moment estimate in our method is larger than the second moment estimate in the Adam optimizer.

3.4.2 Convergence Analysis

Following Adam [13] and the adjustment pointed out in AMSgrad [27], we provide a similar theoretical analysis on the convergence of our method. Based on the online learning framework [35], the optimizer aims to predict the parameter θ_t with an arbitrary, unknown sequence of the convex cost functions $f_1(\theta), f_2(\theta), \cdots, f_T(\theta)$ and evaluate the parameter on a previously unknown cost function f_t. The algorithm is evaluated with the regret, the sum of all previous differences between the online prediction $f_t(\theta_t)$ and the best fixed point parameter $f_t(\theta^*)$ for all the previous steps. The regret is defined as:

$$R(T) = \sum_{t=1}^{T} [f_t(\theta_t) - f_t(\theta^*)], \tag{16}$$

where $\theta^* = \arg\min_\theta \sum_{t=1}^{T} f_t(\theta)$. The theoretical analysis in [27] shows that Adam has $O(\sqrt{T})$ regret bound when the cost function f_t has bounded gradients, such that $\lim_{T \to \infty} \frac{R(T)}{T} = 0$ when the second moment estimation is monotonically increasing (the variant of Adam, AMSgrad [27]). We show that our method has a similar regret bound. The proof is in the Appendix A.

Proposition 3.1. ***[Regret Bound]*** *Assume that the cost function f_t has bounded gradients, $\forall \theta : \|\nabla f_t(\theta)\|_2 \leq G, \|\nabla f_t(\theta)\|_\infty \leq G_\infty$ and distance between θ_t generated by our method is bounded, $\forall m, n \in \{1, 2, \cdots, T\} : \|\theta_m - \theta_n\|_2 \leq D, \|\theta_m - \theta_n\|_\infty \leq D_\infty$. Our method achieves the following guarantee:*

$$\forall T \geq 1 : \frac{R(T)}{T} = O(\frac{1}{\sqrt{T}}).$$

Proposition 3.1 states that our method also has $O(\sqrt{T})$ regret bound such that $\lim_{T \to \infty} \frac{R(T)}{T} = 0$. The proof of Proposition 3.1 is similar to the analysis provided in the Adam paper [13]. Generally, our second moment estimate is larger than the second moment estimate in Adam while also being bounded when the gradient is bounded, which provides us with a similar regret bound as Adam.

3.4.3 A Possible Modification on Our Method Inspired by Hölder's inequality

While Sec. 3.4.1 shows that the outer product between the squared input and the squared output gradient is the upper bound of the squared weight gradient, in this section, we discuss a possible approach to tighten the bound. By Hölder's inequality [9], for all $p, q \in [1, \infty]$, $\frac{1}{p} + \frac{1}{q} = 1$ we have,

$$\sum_i^n |a_i b_i| \leq \left(\sum_i^n a_i^p \right)^{\frac{1}{p}} \cdot \left(\sum_i^n b_i^q \right)^{\frac{1}{q}}. \tag{17}$$

Table 2: Results of fine-tuning RoBERTa-Base [18] on GLUE [31]. We use same hyperparameters across all the tasks. The peak memory allocation data is collected on CoLA with a batch size of 16.

Mehods	CoLA	STSB	MRPC	RTE	SST2	MNLI	QNLI	QQP	Mem (MB)
AdamW	**62.82**	90.67	**92.17**	74.01	92.09	86.59	91.51	91.46	2409.89
Adafactor	58.08	90.25	91.03	74.45	93.15	87.04	**92.84**	89.93	1133.41
Galore	58.54	90.66	91.81	73.65	92.66	85.80	90.83	90.74	1768.14
IO-Adam (Ours)	61.59	**90.68**	91.30	**75.09**	**94.04**	**87.12**	92.40	**91.68**	1599.61

The Hölder's inequality is an extension of the Cauchy-Buniakowsky-Schwarz Inequality. When $p = 2, q = 2$, Eq. 17 becomes the Cauchy Inequality. By adjusting the p and q, one may tighten the bound. For neural networks, the inputs are generally more sparse, *e.g.*, negative elements become zero after ReLU activation. On the other hand, the output gradients could drastically change, where previous works [1; 3] show that the loss landscape's sharpness may even increase as the model becomes closer to the minimum. Therefore, to adjust p and q for the input and output gradient, we could use a smaller $p \in [1, 2]$ for the input and leave the $q \in [2, +\infty]$ for the output gradient. By doing so, the second moment estimation in our method, Eq. 8, could be adjusted to:

$$\begin{aligned} \mathbf{c}_W^t &= \beta_2 \mathbf{c}_W^{t-1} + (1-\beta_2) \cdot |X|^p 1_{bs}, \\ \mathbf{r}_W^t &= \beta_2 \mathbf{r}_W^{t-1} + (1-\beta_2) \cdot |\nabla_Y \mathcal{L}|^q 1_{bs}, \\ \sqrt{V_W^t} &= (\mathbf{r}_W^t)^{\frac{1}{p}} (\mathbf{c}_W^{t\top})^{\frac{1}{q}}. \end{aligned} \tag{18}$$

This modification introduces a potential hyperparameter p, which could be adjusted for better performance when the input or the output gradients are sparse. In Sec. 4.4, we provide experimental results, shedding light on how the p could influence performance. While this is an additional modification, not the main purpose, unless otherwise specified, the p is set to 2 in our experiments.

4 Experiments

4.1 The Glue Benchmark

We follow the setting in Galore [34] to conduct experiments on GLUE [31] with pretrained RoBERTa-Base [18]. GLUE is a benchmark for evaluating language models on various tasks, including sentiment analysis, question answering, and textual entailment. With experiments on GLUE, we compare our method with AdamW[1] [19], Adafactor [29], and Galore [34]. The hyperparameters are set the same for all the tasks. The model is fine-tuned for 30 epochs with the learning rate set at $3e-5$ and a linear learning rate scheduler. The other settings follow the default settings in PyTorch and Galore for all the optimizers. Our method uses fused first moment estimation and a buffering mechanism for the second moment estimation as introduced in Sec. 3.2 and Sec. 3.3. The buffer size for our method and the rank for Galore are both set to 16. We will provide more details about the hyperparameter setting in the Appendix.

As shown in Table 2, we present the results of fine-tuning RoBERTa-Base on GLUE. Generally, our method achieves similar performance to Adam and outperforms Adafactor and Galore on most tasks. Regarding memory usage, Adafactor achieves the lowest memory usage with no first moment estimation and two vectors for the second moment estimation. Our method generally achieves similar memory reduction as Galore with fused first moment estimation and a buffer for second moment estimation. Note that we track the first moment estimate with a full matrix, which could be combined with Galore, projecting the moment estimate to subspaces for further memory reduction.

4.2 Pretraining LLaMA on C4 and MiniPile

To further test our optimizer, we pretrain small LLaMA [30] on the C4 dataset [26] and MiniPile [12] following Galore [34] and AdamSNSM [24]. The Colossal Cleaned Common Crawl (C4) dataset is a large-scale multilingual corpus of text extracted from the web, and MiniPile is a filtered subset of the Pile corpus. Following the setting in Galore [34], we use models of size 60M, 130M and 1B. We

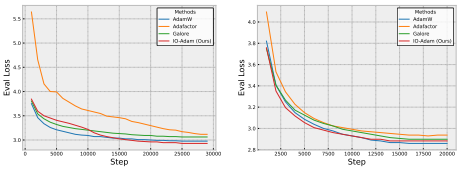
[1]The Adam with its weight decay disentangled from moment estimation

Table 4: Results of training ViT-small on CIFAR10 with different optimizers. We test our method both with and without a buffer for the second moment estimation as mentioned in Sec. 3.3.

Methods	eval-acc(%)	eval-loss	mem-alloc(MB)
AdamW	76.06	0.76402	112.02
Adafactor	76.32	0.75529	**64.52**
IO-Adam(Ours) w/buffer size $b = 16$	**77.63**	**0.69870**	68.40
IO-Adam(Ours) w/buffer size $b = 8$	76.87	0.71221	66.84
IO-Adam(Ours) w/buffer size $b = 4$	77.01	0.70820	66.06
IO-Adam(Ours) w/buffer size $b = 2$	77.13	0.72194	65.66
IO-Adam(Ours) w/buffer size $b = 1$	75.46	0.73762	65.47

compare our method with AdamW [19], Adafactor [29], Galore [34], and AdamSNSM [24]. The rank for Galore and the buffer size in our method are set to the same, 128 for LLaMA-60M, 256 for LLaMA-130M, and 1024 for LLaMA-1B. For each optimizer, the learning rate is grid searched in the set $\{0.025, 0.02, 0.01, 0.002, 0.001\}$.

As shown in Figure 2, we report the evaluation loss of the LLaMA-60M and LLaMA-130M through the training progress on MiniPile. Generally, our method achieves a performance similar to or better than that of AdamW. As analyzed in Sec. 3.4, our method yields a larger second moment estimation compared to Adam, which means our method could use a larger learning rate than AdamW. In this pretraining scenario, the optimal learning rate for Adam is 0.002, while the optimal learning rate for our method is 0.025, differing by an order of magnitude. It indicates that in some scenarios, the learning rate for our method should be substantially higher than that for Adam or AdamW.



(a) LLaMA-60M (b) LLaMA-130M

Figure 2: The evaluation loss of LLaMA-60M and 130M on MiniPile. Our method outperforms Adafactor and Galore and achieves a similar performance to AdamW.

As shown in Table 3, we report the final evaluation perplexity of LLaMA models trained on the C4 dataset. We follow the setting in AdamSNSM [24] to conduct our experiments. The learning rate of our method for the 60M and 130M models is searched in the set $\{0.025, 0.02, 0.01, 0.002, 0.001\}$. The learning rate for the 1B model is set to 0.002 due to the high computational cost. The results in Table 3 demonstrate the effectiveness of our method, where our method achieves a similar performance to AdamW [19]. Details are in Appendix B. We also provide a time complexity analysis in Appendix C.

Table 3: The final evaluation perplexity of the LLaMA model [30] of different sizes trained on the C4 dataset, following AdamSNSM [24]. (* denotes the results referenced from AdamSNSM [24].)

Perplexity (↓)	LLaMA-60M	LLaMA-130M	LLaMA-1B
Adam [13]	30.46*	24.60*	16.00*
AdamW [19]	29.62	22.61	14.51
Galore [34]	34.09	25.31	16.76*
AdamSNSM [24]	29.84	22.71	14.05*
IO-Adam	29.75	22.40	14.36

4.3 Vision Transformer on CIFAR10

To evaluate our optimizer on vision tasks, we train vision-transformers (ViT) [6] on CIFAR10 [14]. We use the code from vit-pytorch to conduct our experiments and compare our method to AdamW [19] and Adafactor [29]. For each optimizer, we train a ViT-small with default hyperparameters except the learning rate, where the learning rate is set at $5e-4$ for AdamW and $1e-3$ for our method, selected with a grid search. Note that Adafactor is designed not to require a learning rate; in our experiment, we use the default learning rate $1e-2$ in PyTorch. We also conduct experiments to verify the impact of buffer size b for second moment estimation on our method, where we select the buffer size from $\{1, 2, 4, 8, 16\}$. Details about hyperparameter settings are in Appendix B.

The results are shown in Table 4. While the Adafactor achieves the lowest memory usage, our method with a buffer of size 16 achieves the best performance compared to AdamW and Adafactor with a relatively low memory usage. Comparing the performance of our method with different buffer sizes, we show that a buffer for second moment estimation is important for better perfor-

mance, where a buffer reduces the cross-batch terms when we multiply the moving average of the squared input and the squared output gradient, leading to a more accurate and smaller second moment estimation. Generally, a larger buffer size means our second moment estimation contains fewer cross-batch terms corresponding to the product of inputs and output gradients from different batches.

4.4 Experiments on WikiText

We conduct experiments on WikiText-2 [22] with GPT-2 [25]. WikiText-2 is a language-modeling benchmark that comprises clean text from English Wikipedia articles. We follow the setting in the official example from the Transformers library [32] to fine-tune a pretrained GPT-2 on WikiText-2. While GPT-2 employs 1-dimensional convolution, which is basically equivalent to a linear layer, we replace each 1D convolution with a linear layer. Our method, IO-Adam, is compared to AdamW and Adafactor. The learning rate is set at $5e-4$ for IO-Adam and AdamW and $5e-5$ for Adafactor. The buffer size of IO-Adam is 1. The results are shown in Table 5, where our method achieves a lower perplexity compared to Adafactor and AdamW.

Table 5: Results of GPT-2 fine-tuned with different optimizers on WikiText-2.

Methods	Perplexity(↓)
AdamW	22.63
Adafactor	22.72
IO-Adam (Ours)	**22.36**

Experiments on the p modification as introduced in Sec. 3.4.3. We further conduct experiments on the modification of p inspired by Hölder's inequality as introduced in Sec. 3.4.3. By changing p, we want to achieve a more accurate second moment estimation. While the output gradient could be more drastically changed, the p should be smaller for the input, leaving a larger $q = \frac{p}{p-1}$ for the output gradient. To test how this modification would affect our method, we fine-tune GPT-2 on WikiText-2 by IO-Adam with different p. As shown in Fig. 3, as the p increases from 1.2 to 2.5, the perplexity first decreases and then increases. $p = 1.8$ achieves the lowest perplexity, which means the most proper p is around 1.8 for fine-tuning GPT-2 on WikiText-2 in this scenario. The results verify our analysis, such that lowering the p could further improve the performance. While p is a potential hyperparameter, we want to emphasize that, according to our other experimental results, setting p to 2 generally works well. With the hyperparameter p introduced, we want to offer an option to treat the input and the output gradient differently for adaptive learning adjustment, where the input is generally stable when the learning rate is small, and the output gradient could drastically change when the loss landscape is sharp. While previous works treat the weight gradient as a whole, it is a unique feature for our method to be able to adjust for the difference between the input and the output gradient.

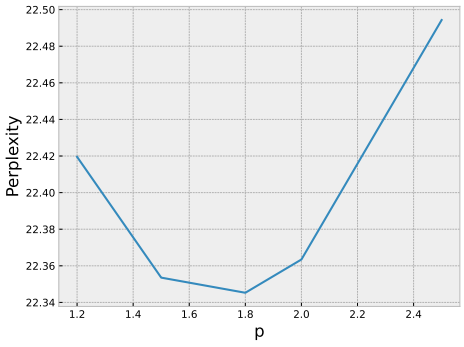


Figure 3: The perplexity of models finetuned by IO-Adam with different p as discussed in Sec.3.4.3. According to the result, p around 1.8 achieves the best performance.

5 Conclusion

In this paper, we propose rethinking previous memory-efficient approaches for adaptive optimizers from the perspective of gradient computation. We show that the input and the output gradient are a natural decomposition of the weight gradient. From this perspective, we propose a memory-efficient adaptive optimizer that separately tracks the input and output gradients for adaptive moment estimation, namely IO-Adam. We analyze the connection and difference between IO-Adam and vanilla Adam, showing that our method has the same regret bound for convergence rate as Adam. We conduct experiments on language modeling and vision tasks, where our method substantially reduces memory usage while maintaining a similar or superior performance to Adam. The limitation of our method is primarily due to structural limitations, as our method focuses on the fundamental component—the linear layer—in neural networks. Further modifications to our method may be required to apply it to other modules, such as convolution. We hope our work will inspire future works to enhance the effectiveness and efficiency of adaptive optimizers.

ETHICS STATEMENT

All the authors of this paper have read the ICLR Code of Ethics and will adhere to it during the submission process. This paper focuses on memory-efficient adaptive optimizers applied to deep neural networks, such as large language models and convolutional neural networks. This paper does not involve human subjects and poses no potentially harmful insights, methodologies, or applications.

REPRODUCIBILITY STATEMENT

To ensure reproducibility, we provide details on the experimental setting of our experiments in the main text and the appendix. We provide detailed descriptions of how the experiments are conducted. We will provide the code upon acceptance of this paper.

DECLARATION OF AI USE

We used Grammarly to check the grammar and fix typos in our writing. All ideas, analyses, and conclusions remain our own.

REFERENCES

[1] Jeremy M Cohen, Simran Kaur, Yuanzhi Li, J Zico Kolter, and Ameet Talwalkar. Gradient descent on neural networks typically occurs at the edge of stability. *arXiv preprint arXiv:2103.00065*, 2021.

[2] Romain Cosson, Ali Jadbabaie, Anuran Makur, Amirhossein Reisizadeh, and Devavrat Shah. Low-rank gradient descent. *IEEE Open Journal of Control Systems*, 2:380–395, 2023.

[3] Alex Damian, Eshaan Nichani, and Jason D Lee. Self-stabilization: The implicit bias of gradient descent at the edge of stability. *arXiv preprint arXiv:2209.15594*, 2022.

[4] Steffen Dereich and Arnulf Jentzen. Convergence rates for the adam optimizer. *arXiv preprint arXiv:2407.21078*, 2024.

[5] Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 8-bit optimizers via block-wise quantization. *arXiv preprint arXiv:2110.02861*, 2021.

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[7] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[8] Mary Gooneratne, Khe Chai Sim, Petr Zadrazil, Andreas Kabel, Françoise Beaufays, and Giovanni Motta. Low-rank gradient approximation for memory-efficient on-device training of deep neural network. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3017–3021. IEEE, 2020.

[9] GH Hardy and JE Littlewood. G. polya (1934) inequalities.

[10] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8):2, 2012.

[11] Siyuan Huang, Brian D Hoskins, Matthew W Daniels, Mark D Stiles, and Gina C Adam. Low-rank gradient descent for memory-efficient training of deep in-memory arrays. *ACM Journal on Emerging Technologies in Computing Systems*, 19(2):1–24, 2023.

[12] Jean Kaddour. The minipile challenge for data-efficient language models. *arXiv preprint arXiv:2304.08442*, 2023.

[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[14] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[15] Frederik Kunstner, Jacques Chen, Jonathan Wilder Lavington, and Mark Schmidt. Noise is not the main factor behind the gap between sgd and adam on transformers, but sign descent might be. *arXiv preprint arXiv:2304.13960*, 2023.

[16] Bingrui Li, Jianfei Chen, and Jun Zhu. Memory efficient optimizers with 4-bit states. *Advances in Neural Information Processing Systems*, 36:15136–15171, 2023.

[17] Yixiao Li, Yifan Yu, Chen Liang, Pengcheng He, Nikos Karampatziakis, Weizhu Chen, and Tuo Zhao. Loftq: Lora-fine-tuning-aware quantization for large language models. *arXiv preprint arXiv:2310.08659*, 2023.

[18] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[19] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[20] Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. Adaptive gradient methods with dynamic bound of learning rate. *arXiv preprint arXiv:1902.09843*, 2019.

[21] Kai Lv, Hang Yan, Qipeng Guo, Haijun Lv, and Xipeng Qiu. Adalomo: Low-memory optimization with adaptive learning rate. *arXiv preprint arXiv:2310.10195*, 2023.

[22] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[23] Ionut-Vlad Modoranu, Aleksei Kalinov, Eldar Kurtic, Elias Frantar, and Dan Alistarh. Error feedback can accurately compress preconditioners. *arXiv preprint arXiv:2306.06098*, 2023.

[24] Thien Hang Nguyen and Huy Le Nguyen. Lean and mean adaptive optimization via subset-norm and subspace-momentum with convergence guarantees. *arXiv preprint arXiv:2411.07120*, 2024.

[25] 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, 2019.

[26] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[27] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

[28] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

[29] Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pp. 4596–4604. PMLR, 2018.

[30] 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.

[31] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[32] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[33] Zeke Xie, Xinrui Wang, Huishuai Zhang, Issei Sato, and Masashi Sugiyama. Adaptive inertia: Disentangling the effects of adaptive learning rate and momentum. In *International conference on machine learning*, pp. 24430–24459. PMLR, 2022.

[34] Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient llm training by gradient low-rank projection. *arXiv preprint arXiv:2403.03507*, 2024.

[35] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th international conference on machine learning (icml-03)*, pp. 928–936, 2003.

A PROOF FOR PROPOSITION 3.1

Generally, we follow the analysis in [13] for the proof of Proposition 3.1. The main difference between our method and Adam is that we separately track squared input and squared output gradient for the second moment estimation, which will result in a larger second moment estimate. We start with the definition of convex functions.

Definition A.1. *A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if for all $x, y \in \mathbb{R}^d$, for all $\lambda \in [0, 1]$,*

$$\lambda f(x) + (1 - \lambda) f(y) \geq f(\lambda x + (1 - \lambda) y). \tag{19}$$

Lemma A.2. *If a function $f : \mathbb{R}^d \to \mathbb{R}$ is convex, for all $x, y \in \mathbb{R}^d$,*

$$f(y) \geq f(x) + \nabla f(x)^\top (y - x) \tag{20}$$

Similar to that in [13], we use the above lemma to upper bound the regret. Since all the parameters are reshaped to a vector, to align the notation with [13], we use g_t to denote $f_t(\theta_t)$ and use m_t to denote the momentum at the t-th step updated by $m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla f_t(\theta_t)$ and $\hat{m}_t$ is $\frac{m_t}{(1-\beta_1^t)}$. For the second moment estimate, we use v_t to represent our second moment estimate. Specifically $\theta_{t,i}$, $g_{t,i}$, $m_{t,i}$, and $v_{t,i}$ are used to denote the i-th element of the corresponding vector.

Lemma A.3. *Let $\gamma \triangleq \frac{\beta_1}{\beta_2}$. For $\beta_1, \beta_2 \in [0, 1]$ that satisfy $\gamma < 1$, and bounded g_t with $\|g_t\| \leq G$, $\|g_t\|_\infty \leq G_\infty$, the following inequality holds,*

$$\sum_{t=1}^{T} \frac{\hat{m}_{t,i}^2}{\sqrt{t\hat{v}_{t,i}}} < \frac{2(1-\beta_2^t)(1-\beta_1)^2}{(1-\beta_2)^2(1-\beta_1^t)^2} G_\infty \sqrt{T}. \tag{21}$$

Proof. According to Eq. 15, our second moment estimation is generally larger than that of the Adam,

$$v_{t,i} \geq \sum_{j=1}^{t} (1-\beta_2)^2 \beta_2^{2(t-j)} g_{j,i}^2. \tag{22}$$

For the first moment estimate, we have

$$m_{t,i}^2 = \left[\sum_{j=1}^{t} (1-\beta_1)\beta_1^{t-j} g_{j,i} \right]^2. \tag{23}$$

Applying the Cauchy–Schwarz inequality, we have

$$m_{t,i}^2 \leq \Big(\sum_{j=1}^{t} \left[\frac{(1-\beta_1)\beta_1^{t-j}}{(1-\beta_2)\beta_2^{t-j}} \right]^2 \Big)\Big(\sum_{j=1}^{t} (1-\beta_2)^2 \beta_2^{2(t-j)} g_{j,i}^2\Big) \tag{24}$$

Combining Eq. 22 and Eq. 24, we further have

$$\begin{aligned} \frac{\hat{m}_{t,i}^2}{\sqrt{\hat{v}_{t,i}}} &= \frac{(1-\beta_2^t)}{(1-\beta_1^t)^2} \frac{m_{t,i}^2}{\sqrt{v_{t,i}}} \\ &\leq \frac{(1-\beta_2^t)}{(1-\beta_1^t)^2} \Big(\sum_{j=1}^{t} \left[\frac{(1-\beta_1)\beta_1^{t-j}}{(1-\beta_2)\beta_2^{t-j}} \right]^2 \Big) \frac{\sum_{j=1}^{t} (1-\beta_2)^2 \beta_2^{2(t-j)} g_{j,i}^2}{\sqrt{\sum_{j=1}^{t} (1-\beta_2)^2 \beta_2^{2(t-j)} g_{j,i}^2}} \\ &\leq \frac{(1-\beta_2^t)(1-\beta_1)^2}{(1-\beta_2)^2(1-\beta_1^t)^2} \sum_{j=1}^{t} \gamma^{2(t-j)} G_\infty. \end{aligned} \tag{25}$$

Since $\gamma < 1$, for all $t \in \{1, 2, \cdots, T\}$ we have

$$\frac{\hat{m}_{t,i}^2}{\sqrt{\hat{v}_{t,i}}} \leq \frac{(1-\beta_2^t)(1-\beta_1)^2}{(1-\beta_2)^2(1-\beta_1^t)^2} G_\infty. \tag{26}$$

Therefore,

$$\begin{aligned}\sum_{t=1}^{T}\frac{\hat{m}_{t,i}^2}{\sqrt{t\hat{v}_{t,i}}} &\leq \frac{(1-\beta_2^t)(1-\beta_1)^2}{(1-\beta_2)^2(1-\beta_1^t)^2}G_\infty\sum_{t=1}^{T}\frac{1}{\sqrt{t}}\\ &< \frac{2(1-\beta_2^t)(1-\beta_1)^2}{(1-\beta_2)^2(1-\beta_1^t)^2}G_\infty\sqrt{T}.\end{aligned} \tag{27}$$

□

Theorem A.4. *Assume the convex function f_t has bounded gradients, $\|\nabla f_t(\theta)\|_2 \leq G$, $\|\nabla f_t(\theta)\|_\infty \leq G_\infty$ for all $\theta \in \mathbb{R}^d$ and the θ generated by IO-Adam is bounded by $\|\theta_n - \theta_m\|_2 \leq D$, $\|\theta_n - \theta_m\|_\infty \leq D_\infty$ for any $n, m \in \{1, 2, \cdots, T\}$, and $\beta_1, \beta_2 \in [0, 1)$ satisfy $\frac{\beta_1}{\beta_2} < 1$. Let $\alpha_t = \frac{\alpha}{\sqrt{t}}$ and $\beta_{1,t} = \beta_1\lambda^{t-1}$, $\lambda \in (0, 1)$. IO-Adam achieves the following guarantee, for all $T > 1$,*

$$R(T) \leq \frac{dD^2}{\alpha(1-\beta_1)}G_\infty\sqrt{T} + \frac{\alpha d(1-\beta_2^t)(1-\beta_1^2)}{(1-\beta_2)^2(1-\beta_1^t)^2}G_\infty\sqrt{T} + \frac{dD_\infty^2 G_\infty}{\alpha(1-\beta_1)(1-\lambda)^2}. \tag{28}$$

Proof. Using Lemma A.2, we have

$$f_t(\theta_t) - f_t(\theta^*) \leq \nabla f_t(\theta_t)^\top(\theta_t - \theta^*). \tag{29}$$

According to the update rule in Algorithm 2,

$$\begin{aligned}\theta_{t+1} &= \theta_t - \alpha_t\hat{m}_t/\sqrt{\hat{v}_t}\\ &= \theta_t - \frac{\alpha_t}{(1-\beta_1^t)\sqrt{\hat{v}_t}}[\beta_{1,t}m_{t-1} + (1-\beta_{1,t})\nabla f_t(\theta_t)].\end{aligned} \tag{30}$$

With the update rule, we have

$$\begin{aligned}(\theta_{t+1} - \theta^*)^2 =&(\theta_t - \theta^* - \alpha_t\hat{m}_t/\sqrt{\hat{v}_t})^2\\ =&(\theta_t - \theta^*)^2 - \frac{2\alpha_t\hat{m}_t}{\sqrt{\hat{v}_t}} \odot (\theta_t - \theta^*) + \frac{\alpha_t^2\hat{m}_t^2}{\hat{v}_t}\end{aligned} \tag{31}$$

Substitute $\hat{m}_t$ with $\frac{\beta_{1,t}m_{t-1}+(1-\beta_{1,t})\nabla f_t(\theta_t)}{(1-\beta_1^t)}$ in Eq. 31, we have

$$\begin{aligned}\nabla f_t(\theta_t) \odot (\theta_t - \theta^*) =&\frac{(1-\beta_1^t)\sqrt{\hat{v}_t}}{2\alpha_t(1-\beta_{1,t})}[(\theta_t - \theta^*)^2 - (\theta_{t+1} - \theta^*)^2] - \frac{\beta_{1,t}}{1-\beta_{1,t}}m_{t-1} \odot (\theta_t - \theta^*)\\ &+ \frac{\alpha_t(1-\beta_1^t)\hat{m}_t^2}{2\sqrt{\hat{v}_t}(1-\beta_{1,t})}\end{aligned} \tag{32}$$

Similar to [13], we use Young's inequality, $ab \leq a^2/2 + b^2/2$ and rearrange Eq. 32

$$\begin{aligned}\nabla f_t(\theta_t) \odot (\theta_t - \theta^*) \leq&\frac{(1-\beta_1^t)\sqrt{\hat{v}_t}}{2\alpha_t(1-\beta_1)}[(\theta_t - \theta^*)^2 - (\theta_{t+1} - \theta^*)^2] + \frac{\beta_{1,t}\sqrt{\hat{v}_{t-1}}}{2\alpha_{t-1}(1-\beta_{1,t})}(\theta^* - \theta_t)^2\\ &+ \frac{\alpha_{t-1}\beta_1}{2(1-\beta_1)\sqrt{\hat{v}_{t-1}}}m_{t-1}^2 + \frac{\alpha_t(1-\beta_1^t)\hat{m}_t^2}{2\sqrt{\hat{v}_t}(1-\beta_1)}\\ \leq&\frac{\sqrt{\hat{v}_t}}{2\alpha_t(1-\beta_1)}[(\theta_t - \theta^*)^2 - (\theta_{t+1} - \theta^*)^2] + \frac{\beta_{1,t}\sqrt{\hat{v}_{t-1}}}{2\alpha_{t-1}(1-\beta_{1,t})}(\theta^* - \theta_t)^2\\ &+ \frac{\alpha_{t-1}\beta_1}{2(1-\beta_1)\sqrt{\hat{v}_{t-1}}}\hat{m}_{t-1}^2 + \frac{\alpha_t\hat{m}_t^2}{2(1-\beta_1)\sqrt{\hat{v}_t}}\end{aligned} \tag{33}$$

The regret is the sum of all the elements in $\nabla f_t(\theta_t) \odot (\theta_t - \theta^*)$ over all the iteration, where we have,

$$R(T) \leq \sum_{i=1}^{d} \frac{\sqrt{\hat{v}_{1,i}}}{2\alpha_1(1-\beta_1)}(\theta_{1,i} - \theta^*_{,i})^2 + \sum_{i=1}^{d}\sum_{t=2}^{T} \frac{1}{2(1-\beta_1)}(\theta_{t,i} - \theta^*_{,i})^2 \left(\frac{\sqrt{\hat{v}_{t,i}}}{\alpha_t} - \frac{\sqrt{\hat{v}_{t-1,i}}}{\alpha_{t-1}}\right) + \sum_{i=1}^{d}\sum_{t=1}^{T-1} \frac{\alpha_{t-1}\beta_1}{2(1-\beta_1)\sqrt{\hat{v}_{t,i}}}\hat{m}^2_{t,i} + \sum_{i=1}^{d}\sum_{t=1}^{T} \frac{\alpha_t \hat{m}^2_{t,i}}{2(1-\beta_1)\sqrt{\hat{v}_{t,i}}} + \sum_{i=1}^{d}\sum_{t=1}^{T} \frac{\beta_{1,t}\sqrt{\hat{v}_{t-1}}}{2\alpha_{t-1}(1-\beta_{1,t})}(\theta^*_{,i} - \theta_{t,i})^2. \tag{34}$$

Apply Lemma A.3, we have

$$R(T) \leq \sum_{i=1}^{d} \frac{\sqrt{\hat{v}_{1,i}}}{2\alpha_1(1-\beta_1)}(\theta_{1,i} - \theta^*_{,i})^2 + \sum_{i=1}^{d}\sum_{t=2}^{T} \frac{1}{2(1-\beta_1)}(\theta_{t,i} - \theta^*_{,i})^2 \left(\frac{\sqrt{\hat{v}_{t,i}}}{\alpha_t} - \frac{\sqrt{\hat{v}_{t-1,i}}}{\alpha_{t-1}}\right) + \sum_{i=1}^{d} \frac{\alpha(1-\beta_2^t)(1-\beta_1^2)}{(1-\beta_2)^2(1-\beta_1^t)^2} G_\infty \sqrt{T} + \sum_{i=1}^{d}\sum_{t=1}^{T} \frac{\beta_{1,t}\sqrt{\hat{v}_{t-1}}}{2\alpha_{t-1}(1-\beta_{1,t})}(\theta^*_{,i} - \theta_{t,i})^2. \tag{35}$$

Note that, as pointed out by AMSgrad [27], it requires the second moment estimate to increase monotonically, and they provide a modification to ensure that. From this perspective, our proposed method is similar and could also be modified for better convergence. From the assumption that $\|\theta_n - \theta_m\|_2 \leq D$, $\|\theta_n - \theta_m\|_\infty \leq D_\infty$ for any $n, m \in \{1, 2, \cdots, T\}$, we have

$$R(T) \leq \frac{D^2}{2\alpha(1-\beta_1)} \sum_{i=1}^{d} \sqrt{T\hat{v}_{T,i}} + \sum_{i=1}^{d} \frac{\alpha(1-\beta_2^t)(1-\beta_1^2)}{(1-\beta_2)^2(1-\beta_1^t)^2} G_\infty \sqrt{T} + \frac{D_\infty^2}{2\alpha} \sum_{i=1}^{d}\sum_{t=1}^{T} \frac{\beta_{1,t}\sqrt{t\hat{v}_{t-1,i}}}{(1-\beta_{1,t})}. \tag{36}$$

Since the gradient is bounded, for IO-Adam, we have $\sqrt{\hat{v}_{t,i}} < 2G_\infty$, therefore, we have

$$R(T) \leq \frac{dD^2 G_\infty}{\alpha(1-\beta_1)}\sqrt{T} + \frac{\alpha d(1-\beta_2^t)(1-\beta_1^2)}{(1-\beta_2)^2(1-\beta_1^t)^2} G_\infty \sqrt{T} + \frac{dD_\infty^2 G_\infty}{\alpha} \sum_{t=1}^{T} \frac{\beta_{1,t}\sqrt{t}}{(1-\beta_{1,t})}. \tag{37}$$

For the last term, similar to the proof in [13], we use arithmetic geometric series upper bound $\sum_t t\lambda^t \leq \frac{1}{(1-\lambda)^2}$ for the last term:

$$\begin{aligned} \sum_{t=1}^{T} \frac{\beta_{1,t}\sqrt{t}}{(1-\beta_{1,t})} &\leq \sum_{t=1}^{T} \frac{1}{(1-\beta_1)} \lambda^{t-1}\sqrt{t} \\ &\leq \sum_{t=1}^{T} \frac{1}{(1-\beta_1)} \lambda^{t-1} t \\ &\leq \frac{1}{(1-\beta_1)(1-\lambda)^2} \end{aligned} \tag{38}$$

Therefore, we have the following regret bound,

$$R(T) \leq \frac{dD^2}{\alpha(1-\beta_1)} G_\infty \sqrt{T} + \frac{\alpha d(1-\beta_2^t)(1-\beta_1^2)}{(1-\beta_2)^2(1-\beta_1^t)^2} G_\infty \sqrt{T} + \frac{dD_\infty^2 G_\infty}{\alpha(1-\beta_1)(1-\lambda)^2}. \tag{39}$$

□

Based on Theorem A.4, it is easy to see that $R(T) = O(\sqrt{T})$, such that $\lim_{T\to\infty} \frac{R(T)}{T} = 0$, providing the same regret bound for our method IO-Adam as the Adam.

B Hyperparameter Details

B.1 Experiments on GLUE

We follow the default setting in [34] to conduct experiments. The learning rate is set at $3e-5$ with batch size 16 and a linear learning rate scheduler. The model is fine-tuned on each task for 30 epochs. The rank for Galore and the buffer size for our method are set to 16.

Table 6: The average time of each step when training LLaMA-60M on the C4 dataset. The batch size is set to 64, and the gradient is accumulated for eight batches. The experiments are conducted on a L20Z GPU.

Method	Average Time per step (s)
AdamW	0.2817
Adafactor	0.2892
Galore	0.2949
IO-Adam (Ours)	0.3010

B.2 Experiments on MiniPile and C4

We use the code in [34] to pretrain LLaMA-60M and LLaMA-130M on MiniPile. The learning rate is grid searched in the set $\{0.025, 0.02, 0.01, 0.002, 0.001\}$ for each optimizer while the batch size is set to 300. β_1 is set to 0.9 and β_2 is set to 0.999 as default for Adam, Galore and our method. Weight decay is set to 0.1. We set the number of warm-up steps to 600 for LLaMA-60M and 1000 for LLaMA-130M.

B.3 Experiments on CIFAR10

We use the code in vit-pytorch to conduct our experiment. We set the patch size to 4, dimension to 128, depth to 6, number of heads to 16, and MLP dimension to 2048. All hyperparameters except the learning rate are set to the same. For learning rate, we use 5×10^{-4} for AdamW, default 10^{-2} for Adafactor, and 10^{-3} for ours to let all optimizers perform well. The learning rate will be multiplied by 0.3 if the validation result becomes poor for 2 consecutive epochs. We run 50 epochs in total.

B.4 Experiments on WikiText-2

We use the official example code from the Transformers library [32] to conduct our experiments. We use default hyperparameters with a learning rate set at $5e-4$. Since GPT-2 uses 1-dimensional convolution equivalent to a linear layer, we convert the code to use linear modules.

C Additional Results

C.1 Time Complexity Analysis

Generally, our method requires tracking the input and output gradient, which introduces a little overhead compared to Adam. The runtime is close to those memory-efficient optimizers such as Galore and Adafactor. In Table 6, we report the average time for each step when training LLaMA-60M on the C4 dataset. While our implementation currently uses hooks in Pytorch to track the intermediate hidden states and the gradient, our method is slightly slower than other methods. Better implementation of monitoring the input and output gradients could further improve effectiveness.