CAME: Confidence-guided Adaptive Memory Efficient Optimization

Yang Luo[1*], **Xiaozhe Ren**[2], **Zangwei Zheng**[1], **Zhuo Jiang**[1], **Xin Jiang**[2], **Yang You**[1]

[1]School of Computing, National University of Singapore
[2]Noah's Ark Lab, Huawei
{yangluo,zangwei,jiangz,youy}@comp.nus.edu.sg
{renxiaozhe,jiang.xin}@huawei.com

Abstract

Adaptive gradient methods, such as Adam and LAMB, have demonstrated excellent performance in the training of large language models. Nevertheless, the need for adaptivity requires maintaining second-moment estimates of the per-parameter gradients, which entails a high cost of extra memory overheads. To solve this problem, several memory-efficient optimizers (e.g., Adafactor) have been proposed to obtain a drastic reduction in auxiliary memory usage, but with a performance penalty. In this paper, we first study a confidence-guided strategy to reduce the instability of existing memory efficient optimizers. Based on this strategy, we propose CAME to simultaneously achieve two goals: fast convergence as in traditional adaptive methods, and low memory usage as in memory-efficient methods. Extensive experiments demonstrate the training stability and superior performance of CAME across various NLP tasks such as BERT and GPT-2 training. Notably, for BERT pre-training on the large batch size of 32,768, our proposed optimizer attains faster convergence and higher accuracy compared with the Adam optimizer. The implementation of CAME is publicly available[1].

1 Introduction

Robust training of large language models (LLMs) often relies on adaptive gradient-based optimization methods (Li et al., 2022; Kingma and Ba, 2015; Zhuang et al., 2020). Through the use of cumulative second-order statistics, these methods adapt the per-parameter learning rate and demonstrate superior convergence speed during the training process of LLMs. However, the remarkable performance of adaptive methods incurs an extra cost of memory usage indeed. For example, Adam requires to preserve the first moment estimate and second raw moment estimate of each gradient in order to tune the learning rate for each parameter, which inevitably triples the memory usage concerning the optimizer states. Besides, with the growing size of the model, LLMs are becoming increasingly expensive in terms of memory, and the limitation of memory is gradually emerging as a main bottleneck for training LLMs.

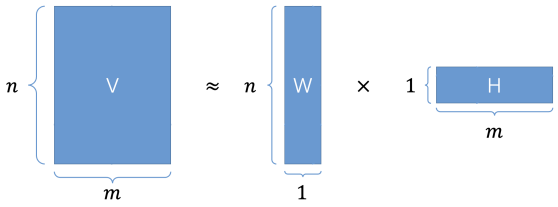


Figure 1: Visualization of Non-negative Matrix Factorization (NMF). Generally, NMF reduces the memory requirements from $O(nm)$ to $O(n+m)$. In this paper, we focus on the special case of rank-1 factors.

Many existing memory-efficient optimizers attempt to store second-order statistics with sublinear memory requirement while retaining the exceptional convergence property of adaptivity (Shazeer and Stern, 2018; Anil et al., 2019). Adafactor optimizer achieves remarkable memory cost reduction by applying the non-negative matrix factorization algorithm (Lee and Seung, 2000) to factorize the accumulator matrix for squared gradients into two rank-1 factors as shown in Figure 1, where the memory requirement for the original matrix V decreases from $O(nm)$ to $O(n+m)$. Whereas, it is observed that Adafactor suffers a performance degradation in the training of large language models universally compared with conventional adaptive gradient-based optimization methods. The reason for this phenomenon is Adafactor inevitably introduces some errors that cause instability in training deep networks due to the operation of non-negative matrix factorization.

[*]Work was done when Yang Luo was an intern at Huawei Noah's Ark Lab.

[1]https://github.com/huawei-noah/Pretrained-Language-Model/tree/master/CAME

In addition, in the case of large-batch training that aims to accelerate the training of deep neural networks, the memory consumption of each machine (GPU/TPU) is much higher than general batch size training, which further imposes a grave constraint on the performance of the trained model. In comparison to standard training tasks, large-batch training presents more challenges for optimizers. Empirically, when the mini-batch size increases after a certain point (e.g. 1024), the test accuracy of the converged solution decreases significantly compared with the baseline (He et al., 2021). To our knowledge, there is currently no work related to memory-efficient optimizers for large-batch training.

Motivated by these challenges, we firstly study a confidence-guided strategy catered to alleviate the instability of Adafactor by calculating the confidence of the generated update at each training step. On the basis of the adaptation strategy, we propose a novel CAME optimizer that saves nearly the same memory footprint as existing memory-efficient optimizers while attaining faster convergence and superior generalization performance. To further assess the scalability of our proposed algorithm, we consider an additional challenging experiment - performing large-batch training on BERT using CAME optimizer.

Contributions of our paper can be summarized in the following:

- Inspired by training instability of Adafactor, we explore a confidence-guided strategy centered on the existing error in the raw updates of Adafactor for parameters of large language models.
- In light of the dedicated strategy, we propose a novel optimization algorithm, CAME, for achieving faster convergence and less performance degradation catered at memory-efficient optimization. We further investigate the effect of the proposed memory-efficient optimization algorithm in large-batch training settings.
- We demonstrate the powerful performance of CAME with extensive NLP experiments: CAME shows faster convergence and better generalization capability than Adam in BERT pre-training task with two different batch sizes (32k and 8k); in the training of GPT-2 model and T5 model, CAME achieves fast convergence speed as Adam without degrading of performance. Notably, in the large-batch training of the BERT model, CAME obtains comparable validation accuracy with LAMB using around 15% less memory usage.

2 Related Work

Memory Efficient Adaptive Optimization Memory efficient optimizers maintain the benefits of standard per-parameter adaptivity while significantly reducing memory footprint. Adafactor (Shazeer and Stern, 2018) proposes to reconstruct a low-rank approximation of the exponentially smoothed accumulator at each training step that is optimal with respect to the generalized Kullback-Leibler divergence. SM3 (Anil et al., 2019) divides the elements in the second-order gradient matrix into sets by the observed similarity of the elements, and each item in the generated approximation matrix is the minimum of the maximum value of each set in which it is located. The methods mentioned above behave poorly in the training of large language models and converge slowly, which raises a significant challenge for memory-efficient optimization methods.

Large Batch Training A large-batch training scheme is preferred in distributed machine learning because of its ability to increase parallelism by enhancing large-scale cluster utilization. It has seen growing interest in recent years in large-batch training (Liu et al., 2022; Li et al., 2021; Huo et al., 2021). In particular, a layer-wise adaptive learning rate algorithm LARS (You et al., 2017a) is proposed to scale the batch size to 32k for ResNet-50. Based on LARS, LAMB optimizer (You et al., 2019) can finish the BERT training in 76 minutes through TPU v3 Pod. Despite the success of these approaches for BERT models, the much larger batch size highly boosts the GPU usage which is prohibitively expensive and inaccessible to most researchers.

Moreover, training with a large batch size incurs additional challenges (Hoffer et al., 2017; Keskar et al., 2016). Large-batch training is prone to converge to sharp local minima, since the number of interactions will decrease when the batch size is increased if the number of epochs is fixed, which causes a wide gap in generalization of the model(Keskar et al., 2016). Traditional methods seek to narrow the generalization gap by carefully

tuning hyperparameters, such as learning rate, momentum, and label smoothing, to narrow the generalization gap (Goyal et al., 2017a; Shallue et al., 2018; You et al., 2017b). Yet there have been few attempts to reduce memory usage in large-batch training, and the underlying challenge remains unclear.

3 Method

In this section, we firstly provide a brief description of the Adafactor optimizer and discuss the errors contained in the update of Adafactor (erroneous update). We further study a confidence-guided strategy and introduce the proposed CAME in detail in light of the strategy.

3.1 An overview of Adafactor

The $\mathcal{L}(\theta) \in \mathbb{R}$ represents the loss function that we plan to minimize, where $\theta \in \mathbb{R}^{n \times m}$ is the parameter of the model. g_t is the gradient at step t, η is the learning rate, r_t and c_t are the exponential moving average of two low-rank factors for the second moments of the gradient. ϵ_1 is a small regularization constants and u_t is the current approximate update.

In the training of large language models, Adafactor is required to apply momentum to ensure the convergence (Chowdhery et al., 2022), and the corresponding pseudocode is illustrated in Algorithm 1. The problem setting is as follows. Assume that we aim to minimize the expected value of an objective function $f(\theta)$. At each training step, we receive the loss derived from a mini-batch of data, and calculate the gradient g_t of the function based on the previous parameters. Subsequently, we update the exponential running averages of two factors for second moments of the gradient r_t and c_t, compute approximations for the second moments of the gradient v_t, and adjust the generated update (u_t) when $RMS(u_t)$ surpasses a specific threshold value d as in:

$$\hat{u}_t = \frac{u_t}{\max(1, RMS(u_t)/d)} \tag{1}$$

where $RMS(u_t)$ refers to the root-mean-square calculation of the components of u_t. Finally, the first moment of the adjusted update m_t is utilized to update the parameter, resulting in a new iteration θ_t. The optimization continues until the parameters converge and returns the final iteration θ_T as our approximate solution.

Adafactor derives an effective solution for non-negative matrix factorization in the special case

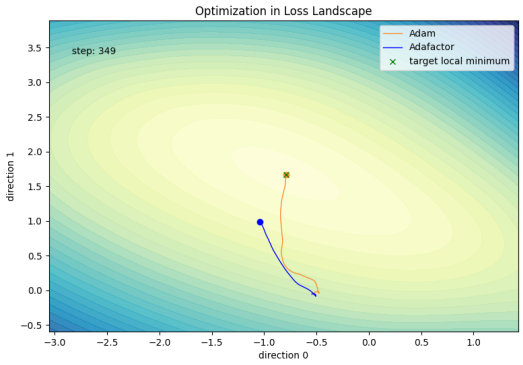


Figure 2: Loss landscape visualization for erroneous update of Adafactor in 1-layer multilayer perceptron (MLP) (Haykin, 1994) with same training steps. Adafactor deviates from the training curve of Adam.

of rank-1 factors, which obtains the minimal Kullback–Leibler divergence (Lee and Seung) between the matrix V and the approximated matrix WH. The formulation of the solution is as follows, in which $1_m = (1, ..., 1) \in \mathbb{R}^m$ represents a column vector of m ones:

$$W = V1_m, \quad H = \frac{1_n^T V}{1_n^T V 1_m}. \tag{2}$$

It should be noted that Adafactor stores only the moving averages of these factors rather than the entire matrix V, yielding considerable memory savings and requiring memory usage proportional to $O(n + m)$ instead of $O(nm)$.

3.2 Erroneous Update

The non-negative matrix factorization operation in Adafactor will inevitably incur erroneous update in the training of deep neural networks. As shown in Figure 2, Adafactor always converge slower than Adam due to the existing error in calculated updates, which further limits the application scenarios of memory-efficient optimizers.

As shown in Figure 3, two scenarios demonstrate how two types of erroneous updates are supposed to be handled in the ideal case. In Figure 3(a), the difference between the momentum of updates m_t and the current update u_t is large, illustrating that the historical experience for the update of original Adafactor contains high level of errors that will inevitably influence the stability of the training process. If we utilize the raw m_t to take an optimization step, the direction of optimization will deviate increasingly from the desired direction, which is reflected by the slow convergence and performance

Algorithm 1: Adafactor Optimizer

Input: Initial parameters θ_0, learning rate η, momentum of update m_0, r_0, c_0, step t, regularization constant ϵ_1, exponential moving average parameters β_1, β_2, clipping threshold d

while θ_t *not converge* **do**

Compute $g_t = \nabla f(\theta_{t-1})$

$r_t = \beta_2 r_{t-1} + (1-\beta_2)(g_t^2 + \epsilon_1 1_n 1_m^T) 1_m$

$c_t = \beta_2 c_{t-1} + (1-\beta_2) 1_n^T (g_t^2 + \epsilon_1 1_n 1_m^T)$

$v_t = r_t c_t / 1_n^T r_t$

$u_t = g_t / \sqrt{v_t}$

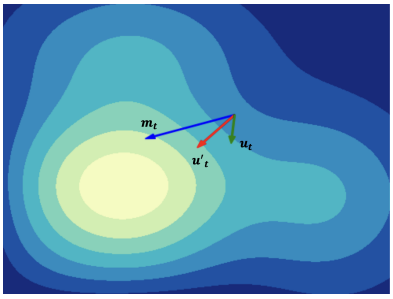
$\hat{u}_t = u_t / \max(1, RMS(u_t)/d)$

$m_t = \beta_1 m_{t-1} + (1-\beta_1)\hat{u}_t$

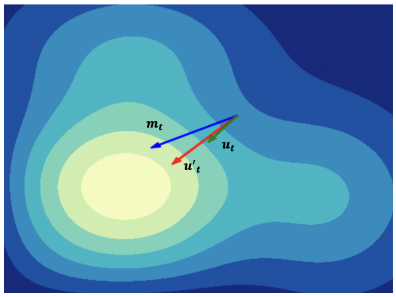
$\theta_t = \theta_{t-1} - \eta m_t$

end

degradation of existing memory-efficient optimizers. By contrast, when the difference between m_t and u_t is small as shown in Figure 3(b), the momentum m_t is stable with limited errors and high confidence therefore a large optimization step is required with the updating direction close to m_t.



(a) scenario 1



(b) scenario 2

Figure 3: Visualization of two scenarios where Adafactor updates have different stability.

Inspired by the erroneous update that is universal in existing memory-efficient optimizers, we firstly consider an efficient approach to decrease the side effect caused by insecure updating. Given m_t and u_t, we take the residual between them as the instability in the preserved momentum and set generated instability as the denominator of original m_t to more adaptively take an update step. Following is the formulation of the adjusted update u', where ϵ is the regularization constant:

$$u'_t = \frac{m_t}{\sqrt{(m_t - u_t)^2 + \epsilon}} \tag{3}$$

Extending further on the plain method, we propose a confidence-guided strategy that enables self-adjusted updates by taking the confidence of the raw update of Adafactor into consideration. The intuition behind the proposed strategy is to calculate the residual between the exponential moving average (EMA) of the update and the current update, which represents the deviation of the approximated update. The larger the deviation of the EMA value from the current generated update, the wider the error EMA of update contains, resulting in a lower level of confidence in the EMA of update. Obviously, we expect the optimizer to take a small update when it incorporates huge error (a large residual from the present update), while updating parameters more when the optimization process is stable (involved error of EMA is limited).

Specifically, the EMA of update m_t is directly used to take an update step in Adafactor, while in our proposed strategy, m_t is divided by $\sqrt{U_t}$, where U_t is the calculated instability matrix. Therefore, $\frac{1}{\sqrt{U_t}}$ is the confidence in the observation: viewing m_t as the prediction of the update, if m_t deviates greatly from u_t (U_t is large), which indicates a weak confidence in m_t, the optimizer performs a small optimization step; if u_t closely matches m_t, we have solid confidence in m_t, and correspondingly take a large optimization step.

3.3 CAME Algorithm

Based on the proposed confidence-guided strategy, we develop a brand-new variant of memory-

Algorithm 2: CAME Optimizer

Input: Initial parameters θ_0, learning rate η, momentum of update $m_0 = 0$, $r_0 = 0, c_0 = 0$, step t = 0, regularization constants ϵ_1, ϵ_2, exponential moving average parameters $\beta_1, \beta_2, \beta_3$, clipping threshold d

while θ_t *not converge* **do**

Compute $g_t = \nabla f(\theta_{t-1})$

$r_t = \beta_2 r_{t-1} + (1 - \beta_2)(g_t^2 + \epsilon_1 1_n 1_m^T)1_m$

$c_t = \beta_2 c_{t-1} + (1 - \beta_2)1_n^T(g_t^2 + \epsilon_1 1_n 1_m^T)$

$v_t = r_t c_t / 1_n^T r_t$

$u_t = g_t / \sqrt{v_t}$

$\hat{u}_t = u_t / \max(1, RMS(u_t)/d)$

$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\hat{u}_t$

$U_t = (\hat{u}_t - m_t)^2$

$R_t = \beta_3 R_{t-1} + (1 - \beta_3)(U_t + \epsilon_2 1_n 1_m^T)1_m$

$C_t = \beta_3 C_{t-1} + (1 - \beta_3)1_n^T(U_t + \epsilon_2 1_n 1_m^T)$

$S_t = R_t C_t / 1_n^T R_t$

$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{S_t}} m_t$

end

efficient optimization methods with faster convergence. Our proposed CAME optimization method successfully obtains the same rate of convergence as prevailing first-order optimization algorithms (e.g., Adam) and with almost equal memory cost to available memory-efficient optimizers (e.g., Adafactor). The pseudocode of CAME algorithm is specified in Algorithm 2.

By calculating U_t at each training step, we employ non-negative matrix factorization on the instability matrix U_t following (Shazeer and Stern, 2018) where the generalized Kullback-Leibler divergence between V and WH is minimal. With U_t factorized into R_t and C_t, it is sufficient to store only the moving averages of these factors rather than the full matrix U_t, thus saving considerable memory footprint.

We simply validate intuitions and the corresponding example is shown in Figure 4, in which the proposed CAME reaches the optimal point much faster than Adafactor. Learning rate is 10^{-3} for all optimizers. In the example, we set the parameters of CAME to be the same as the default in Adafactor, $\beta_1 = 0.9, \beta_2 = 0.999$ and set extra $\beta_3 = 0.9999$ for CAME.

4 Experiments

In this section, we present extensive comparisons with existing optimizers on training tasks of three important large language models: BERT (Devlin et al., 2019), GPT-2 (Radford et al., 2018a) and T5 (Raffel et al., 2022).

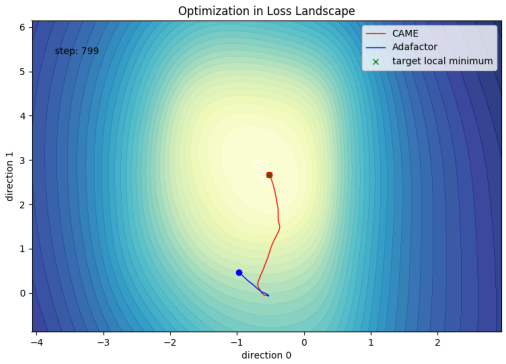


Figure 4: Loss trajectories of Adafactor and CAME. CAME reaches the target local minimum (marked as green cross in 2D plots) much faster than Adafactor.

4.1 Setup

Dataset We perform experiments on the BookCorpus (Radford et al., 2018a) and English Wikipedia with 800M and 2.5B words respectively. Furthermore, we focus on the GLUE benchmark (Peters et al., 2018), SQuAD v1.1 dataset (Rajpurkar et al., 2016) and SQuAD v2.0 dataset (Rajpurkar et al., 2018) to demonstrate the performance of pre-trained BERT models with CAME optimizer.

Model We evaluate the efficiency of our proposed CAME on three trending large language models: BERT, GPT-2 and T5. We further test the performance of CAME for large-batch training

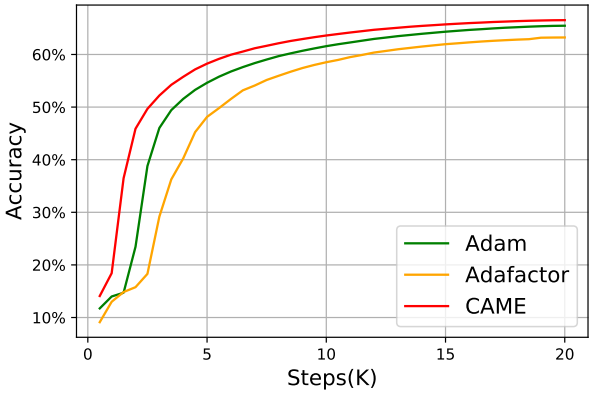


Figure 5: Masked LM test accuracy of BERT-Large model trained on Wikipedia dataset with 8k batch size.

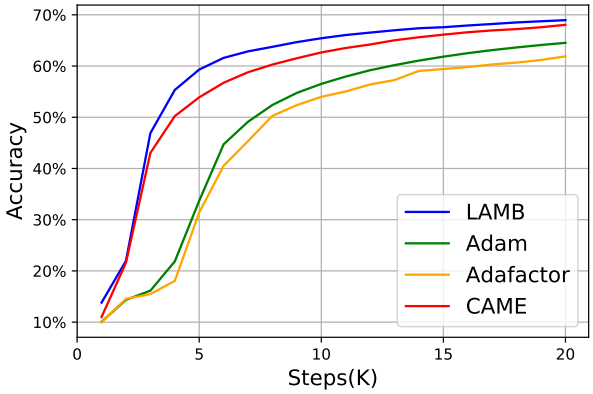


Figure 6: Masked LM test accuracy of BERT-Large model trained on Wikipedia dataset with 32k batch size. CAME achieves comparable accuracy with Adafactor using around only half of required training steps (10k).

with BERT-Large.

Compared methods The main baselines comprise two widely-used optimizers: classic optimizer Adam and memory-efficient optimizer Adafactor. With regard to large-batch training, LAMB optimizer is additionally considered when setting baselines.

Implementation Detail We implement our optimization algorithm in Pytorch (Paszke et al., 2019). The parameters β_1 and β_2 in Algorithm 2 are set as 0.9 and 0.999 respectively, and we search for optimal β_3 among $\{0.9, 0.99, 0.999, 0.9999, 0.99999\}$. We use 8 Tesla V-100 GPUs and set ϵ_1, ϵ_2 as 10^{-30}, 10^{-16} in all experiments with gradient accumulation and model parallelism. Besids, we set η as 2×10^{-4}, 6×10^{-4}, 3×10^{-4} for BERT-Large (32K), GPT-2, T5 training and apply learning rate warmup scheduling (Goyal et al., 2017b) to avoid divergence due to the large learning rate, by starting with a smaller learning rate η and gradually increasing to the large learning rate η. To make sure we are comparing with solid baselines, we use grid search to tune the hyperparameters for Adafactor, Adam and LAMB. We further improve the performance of large-batch training by applying Mixup (Zhang et al., 2017) to scale the batch size up to 32,768.

4.2 BERT Training

We firstly present empirical results in the training task of BERT model to evaluate the performance of our proposed CAME optimizer, focusing on its larger variant, BERT-Large, which has 340M parameters in all. Following the default setting, we pre-train the BERT-Large model ($L = 24, H = 1024$) with a sequence length of 128 on 8 Tesla V-100 GPUs. The experiments were implemented with the code from NVIDIA [2] and mainly include two types of batch sizes: 8k and 32k, one of which represents the widely used setting for pre-training BERT and the other denotes the training scenario under large-batch training. The empirical results are presented in Figure 5 and Figure 6. As illustrated in Figure 5, CAME achieves a significant improvement compared with Adam and Adafactor. To be specific, CAME (66.5%) increases validation accuracy at with an increment 3.4% in comparison to Adafactor (63.1%) using same number of training steps (20k). Apart from Adafactor, our proposed CAME achieves better performance than Adam in the pre-training of BERT-Large model with a huge reduction of memory cost.

To evaluate the performance of our proposed CAME for large-batch training, we scale the batch size for BERT-Large training to 32,768 on Wikipedia dataset. As illustrated in Figure 6, CAME consistently reaches a more remarkable improvement compared with Adafactor. We notice that the accuracy of CAME on BERT-Large pre-training is 68.0%, which is highly over the original Adafactor (61.9%) with same number of training steps. In addition, CAME reaches comparable accuracy with only half the training steps required for Adafactor. With batch size getting larger from 8k to 32k, CAME brings more enhancements to the training of BERT-Large in comparison with Adam and Adafactor. Compared with LAMB in large-batch training, CAME saves a high-level of

[2] https://github.com/NVIDIA/DeepLearningExamples

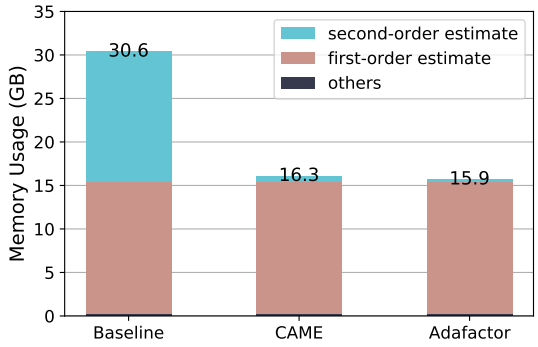


Figure 7: The memory reduction about optimizer states of CAME when training BERT-4B using PyTorch.

memory footprint with slight training performance degradation.

Memory Usage Comparison We set batch size to 1 to measure the memory usage of each optimizer more efficiently. As shown in Table 1, the two optimizers (Adam and LAMB) frequently employed for training large language models consume the highest amount of memory usage. Meanwhile, our proposed CAME optimizer exhibits a reduced memory footprint over the existing SM3 memory-efficient optimizer. As a consequence of our confidence-guided strategy in CAME, there is no doubt that CAME will introduce an increased memory footprint in comparison with Adafactor. However, the extra memory footprint incurred of CAME is almost negligible (1%) with a substantial performance improvement.

For further demonstration of the memory saving effect of CAME, we expand BERT model to BERT-4B with 4 billion weights using the scaling method of GPT-3 (Brown et al., 2020). We set the mini-batch size to 64 and the accumulation steps to 16 in this experiment. In Figure 7, we train BERT-4B with three different optimizers using PyTorch framework. As a result, CAME can save 47% memory footprint about optimizer states compared with Baseline (Adam) when the weights number of a model get to 4 billion.

4.3 Downstream Tasks

We select a representative set of downstream tasks to further demonstrate the performance of BERT models pre-trained by our proposed CAME. In this part we adopt BERT-Base model for the fine-tuning task and follow the originally published BERT-Base results in (Devlin et al., 2019) and (Liu et al., 2019) as the main baseline. The learning rate is tuned on the dev set for each setting and each task is fine-tuned for three epochs.

Table 1: Quantitative memory usage per GPU (GB) comparison in the pre-training of BERT-Large model.

Optimizer	Memory Cost (GB)
Adam	8.24
LAMB	8.23
Adafactor	7.00
SM3	7.44
CAME	7.07

We compare the end-task performance of BERT-Base with the baseline on typical downstream tasks and the empirical results are presented in Table 2. The experimental results demonstrate the efficiency of our proposed CAME optimizer by showing that BERT-Base model trained with CAME on two batch sizes both achieve comparable performance to the baseline with less memory cost. In particular, we observe that BERT-Base model trained with large batch (32k) presents no performance degradation and even attains higher evaluation metrics scores on some downstream tasks. Specifically, the BERT-Base model trained on CAME improves on average by 0.5 across five metrics compared to the baseline, proving the feasibility of CAME for the large-batch training task.

4.4 GPT-2 Training

In addition to BERT pre-training task, we perform CAME-based training task on another typical large language model, GPT-2. Using the original structure of GPT-2 (Radford et al., 2018b), we specifically adopt GPT-medium ($L = 24, H = 1024$) with 345M parameters in our experiment. This implementation is based on the code provided by Megatron[3]. Identically, we take English Wikipedia as the training dataset for this section. Unlike the pre-training of BERT in Section 4.2, we only concentrate on standard training batch size (128) for GPT-2 pre-training.

The empirical results of validation loss are shown in Figure 8. We are able to find that CAME achieves similar convergence and final accuracy compared to Adam, which reveals an impressive improvement over the performance of Adafactor

[3] https://github.com/NVIDIA/Megatron-LM

Table 2: Results of fine-tuning performance on MNLI-m, SST-2, MRPC and two SQuAD datasets. The F1 and EM for SQuAD v1.1 dataset are firstly averaged, and the average of all results across five datasets is further calculated.

Model	**MNLI-m** (Acc)	**SST-2** (Acc)	**MRPC** (Acc)	**SQuAD v1.1** (F1/EM)	**SQuAD v2.0** (F1)	**Average** -
Baseline	84.3	92.8	88.9	88.5/80.8	76.3	85.4
CAME (batch size = 8k)	84.8	92.8	89.9	88.8/81.8	77.9	86.1 (**+0.7**)
CAME (batch size = 32k)	84.5	92.9	89.8	88.5/81.2	77.4	85.9 (**+0.5**)

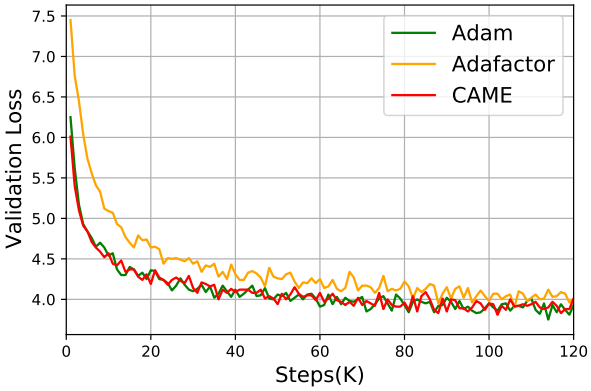


Figure 8: Validation loss of GPT-2 language model. CAME demonstrates similar optimization performance to Adam.

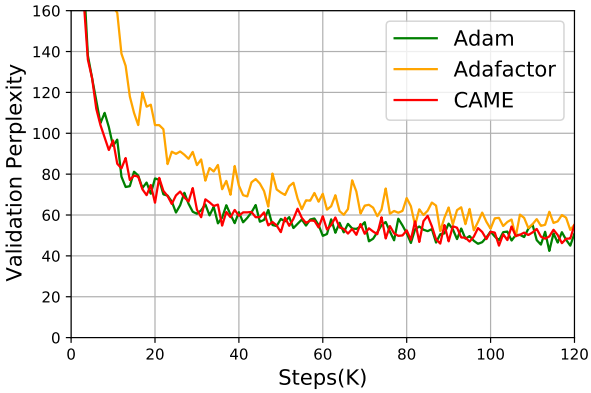


Figure 9: Validation perplexity of GPT-2 language model. CAME demonstrates comparable convergence speed with Adam.

with comparable training steps. Moreover, as indicated in Figure 9, the validation perplexity of CAME presents the same convergence performance as Adam but faster convergence speed than Adafactor, which clearly supports the validity of CAME that has fast convergence as in traditional adaptive methods and low memory usage as in existing memory-efficient methods. For instance, the converged validation perplexity of CAME and Adafactor is 50.1 and 56.9 respectively, which yields a considerable improvement of 12.0%.

4.5 T5 Training

Finally, we report empirical results from a different large language model training task: Text-to-Text Transfer Transformer, T5. Concretely, we follow the architecture of T5 (Raffel et al., 2022) and choose T5-Base ($L = 24, H = 1024$) with 220M parameters for the experiment. All of our implementations are also based on the code provided by Megatron. Similarly, we consider Wikipedia with 2.5B words as the training dataset in this part. As with the training of GPT-2 in Section 4.4, we only concentrate on standard training batch size (128) for T5.

The comparison of CAME with Adafactor and Adam is conducted in the same manner as Section 4.4, and corresponding results of validation loss and validation perplexity are illustrated in Figure 10 and Figure 11 seperately. Note that CAME consistently obtains comparable convergence performance for validation loss and validation perplexity on par with Adam, while reducing similar memory usage as Adafactor.

5 Conclusion

In this paper we propose a novel memory-efficient optimizer called CAME, which supports adaptive confidence-based updating guided by the residual between predicted update and generated update. CAME achieves a considerable improvement compared to existing memory-efficient optimizers in the training of large language models, with an ignorable extra memory footprint. Moreover, CAME shows comparable convergence to Adam and LAMB with huge memory reduction. In particular, CAME has proven effective for large-batch training, which serves as an advantageous extension to memory-efficient optimizers. We hope our work will provide insight into memory reduction

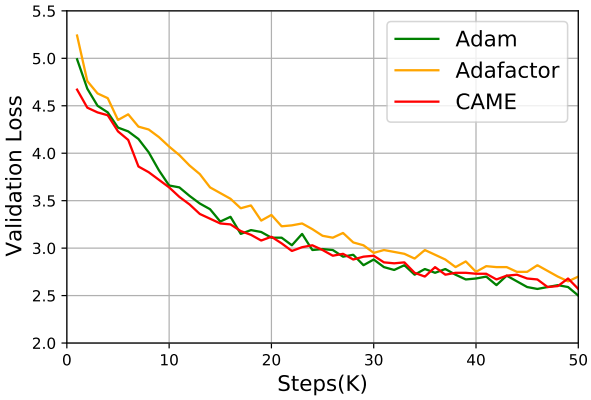


Figure 10: Validation loss of T5 language model. CAME exhibits similar convergence rates to Adam.

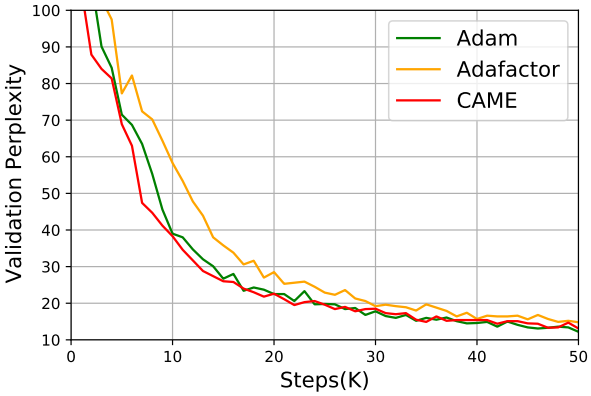


Figure 11: Validation perplexity of T5 language model. CAME demonstrates similar convergence speed to Adam.

of optimizers in future exploration.

6 Limitations

Despite the success of our CAME optimizer in training large language models with memory efficiency, there are still some limitations that need to be addressed in the future.

Our proposed memory-efficient optimizer introduces additional computation costs for the non-negative matrix factorization of the instability matrix in comparison with Adafactor. We observe, however, that the training time of CAME increases only slightly in our experiments. Beyond that, CAME exhibits minor performance degradation in large-batch training of the BERT-Large model versus LAMB, which allows for further improvement in the future. Meanwhile, it is possible to conduct further experiments on other models in other fields, such as Computer Vision and Reinforcement Learning, thereby exploring the effectiveness of CAME training under more application scenarios. As a final point, it would be much more helpful to provide an in-depth theoretical analysis of CAME to improve comprehensiveness of the paper.

Acknowledgements

Yang You's research group is being sponsored by NUS startup grant (Presidential Young Professorship), Singapore MOE Tier-1 grant, ByteDance grant, ARCTIC grant, SMI grant and Alibaba grant. We also thank Huawei Noah's Ark Lab for providing the necessary computing resources and support for datasets.

References

Rohan Anil, Vineet Gupta, Tomer Koren, and Yoram Singer. 2019. *Memory-Efficient Adaptive Optimization*. Curran Associates Inc., Red Hook, NY, USA.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, et al. 2020. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA. Curran Associates Inc.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, et al. 2022. Palm: Scaling language modeling with pathways.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017a. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017b. Accurate, large minibatch sgd: Training imagenet in 1 hour.

Simon Haykin. 1994. *Neural networks: a comprehensive foundation*. Prentice Hall PTR.

Xiaoxin He, Fuzhao Xue, Xiaozhe Ren, and Yang You. 2021. Large-scale deep learning optimizations: A comprehensive survey.

Elad Hoffer, Itay Hubara, and Daniel Soudry. 2017. Train longer, generalize better: closing the generalization gap in large batch training of neural networks.

Zhouyuan Huo, Bin Gu, and Heng Huang. 2021. Large batch optimization for deep learning using new complete layer-wise adaptive rate scaling. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(9):7883–7890.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2016. On large-batch training for deep learning: Generalization gap and sharp minima.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Daniel Lee and H. Sebastian Seung. 2000. Algorithms for non-negative matrix factorization. In *Advances in Neural Information Processing Systems*, volume 13. MIT Press.

Daniel D. Lee and H. Sebastian Seung. 1999. Learning the parts of objects by nonnegative matrix factorization. *Nature*, 401:788–791.

Conglong Li, Ammar Ahmad Awan, Hanlin Tang, Samyam Rajbhandari, and Yuxiong He. 2021. 1-bit lamb: Communication efficient large-scale large-batch training with lamb's convergence speed.

Zhiyuan Li, Srinadh Bhojanapalli, Manzil Zaheer, Sashank Reddi, and Sanjiv Kumar. 2022. Robust training of neural networks using scale invariant architectures. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 12656–12684. PMLR.

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

Yong Liu, Siqi Mai, Xiangning Chen, Cho-Jui Hsieh, and Yang You. 2022. Towards efficient and scalable sharpness-aware minimization.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. Cite arxiv:1802.05365Comment: NAACL 2018. Originally posted to openreview 27 Oct 2017. v2 updated for NAACL camera ready.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2018a. Language models are unsupervised multitask learners.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2018b. Language models are unsupervised multitask learners.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2022. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1).

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Christopher J. Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E. Dahl. 2018. Measuring the effects of data parallelism on neural network training.

Noam M. Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. *ArXiv*, abs/1804.04235.

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

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2019. Large batch optimization for deep learning: Training bert in 76 minutes.

Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. 2017b. Imagenet training in minutes.

Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. 2017. mixup: Beyond empirical risk minimization.

Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar Tatikonda, Nicha Dvornek, Xenophon Papademetris, and James Duncan. 2020. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. *Conference on Neural Information Processing Systems*.