FAME: FAST ADAPTIVE MOMENT ESTIMATION BASED ON TRIPLE EXPONENTIAL MOVING AVERAGE

Anonymous authors
Paper under double-blind review

ABSTRACT

Network optimization is a key-step in deep learning, which broadly impacts different domains (e.g. natural language, computer vision). Over the years, several optimizers have been developed - some are adaptive and converge quickly, while others are not adaptive but may be more accurate. However, due to the fact that most current optimizers' use simple Exponential Moving Average, gradient trends and their rapid changes may not be accurately identified, resulting in sub-optimal network performance. In this paper, we propose the first deep optimizer based on the Triple Exponential Moving Average (TEMA), a technical indicator originally developed to predict stock market trends. TEMA adds richer multi-level information about data changes and trends compared to the simple Exponential Moving Average. As a result, the gradients moments are better estimated. Furthermore, instead of using TEMA in the same way as the stock domain, here we use it as part of a continuous average during an optimization procedure. We extensively validated our method. Five benchmarks (CIFAR-10, CIFAR-100, PASCAL-VOC, MS-COCO and Cityscapes) were used to test our method, as well as 14 different learning architectures, five different optimizers, and various vision tasks (detection, segmentation, and classification). The results clearly indicate that the robustness and accuracy of our FAME optimizer are superior to those of others.

1 INTRODUCTION

Modern machine learning research relies heavily on optimization algorithms; therefore, advances in optimization have a major and broad impact on this field (Maheswaranathan et al. (2020)). Despite improvements in the design, training, and performance of optimizers, fundamental questions about their behavior remain open. Over the last years, there has been some progress in first-order optimization algorithms that can be broadly categorized into two branches: the stochastic gradient descent (SGD) family (Robbins & Monro (1951)), and the adaptive learning rate methods. SGD methods use a global learning rate for all parameters, while adaptive methods compute an individual learning rate for each parameter ((Kingma & Ba (2017)), (Duchi et al. (2011)), (Tieleman & Hinton (2012)) and (Reddi et al. (2019))). Compared to the SGD family, these adaptive methods typically converge faster but may have worse generalization performance than their non-adaptive counterparts (Wilson et al. (2017)).

2 RELATED WORK

Common-used Optimizers and Gradients Characteristics. Stochastic Gradient Descent (*SGD*) optimizer performs well across many applications. However, SGD scales the gradient uniformly in all directions. Adding *momentum* to SGD introduces gradient accumulation of all past data points, leading to better updates and smoother descent of the loss surface (Qian (1999)). *AdaDelta* suggests looking at a moving window of the past gradient updates to generate more "local" view w.r.t time, adding scaling to the update itself (Zeiler (2012)). *AdaGrad* (Ioffe & Szegedy (2015)) supplies better performance compared with SGD when the gradients are sparse, or in general small. AdaGrad dynamically incorporates knowledge of the data geometry observed in earlier iterations to perform more informative gradient-based learning. It applies larger updates (e.g. high learning rates) for those parameters that are related to infrequent features and smaller updates (i.e. low learning rates) for

the frequent ones. As a result, it is well-suited when dealing with sparse data where each parameter has its own learning rate that improves performance. However, AdaGrad may lead to an extremely small learning rate over time. To resolve AdaGrad's diminishing learning rate problem, ***RMSProp*** optimizer was developed Tieleman & Hinton (2012). The major difference between RMSProp and AdaGrad is that the gradient is calculated by an exponentially decaying average, instead of the sum of its gradients. In a sense, RMSProp basically "slows" down movement near the global minimum, by adjusting and adapting the learning rate accordingly. While SGD with momentum speeds up the optimization process for gradients whose directions are the same, RMSProp adapts the learning rate dynamically to be more cautious as training progresses. *Adam* is another popular optimizer that computes adaptive learning rate for each parameter with a square of the gradients like RMSprop. However, Adam also has an exponentially decaying average of past gradients unlike RMSprop. Reddi et al. proposed a variant of Adam called *AMSGRAD* (Reddi et al. (2019)) that attempts to improve the convergence properties, avoiding large abrupt changes in the learning rate for each input variable. *AdaBound*, which aims to be as fast as Adam and as good as SGD, is another Adam variant that employs dynamic bounds on learning rates to achieve a gradual and smooth transition to SGD (Luo et al. (2019)). Loshchilov and Hutter developed the *AdamW* (Loshchilov & Hutter (2018)) optimizer that generalizes better than Adam and is thus able to compete with SGD while training faster. Recently, a few additional works explored the added-value of combining both SGD and Adam, to achieve fast convergence as Adam and generalization ability as SGD. *SWATS* is a nice example for such a joint framework. It switches from Adam to SGD either with a hard schedule (Keskar & Socher (2017)), or with a smooth transition as AdaBound. However, when SWATS was compared with Adam and SGD, the latter two achieved the best results, not SWATS. Another paper proposed a new optimizer called MAS (Mixing Adam and SGD) that integrates SGD and Adam simultaneously by weighting their contributions (Landro et al. (2020)). Despite the interesting idea, the proposed method uses a straight-forward summation of Adam and SGD through the assignment of *constant* weights for each optimizer, supplying only a limited performance improvement.

Identify trends. (Kolkova (2018)) is the first paper that compares two approaches to identifying trends in the financial domain; 1) the Holt's smoothing, and 2) the technical indicator approach. In computer vision, Maiya et al. introduced the Tom optimizer (Trend over Momentum) that helps to predict the gradients trend (Maiya et al. (2021)). The model uses time series prediction model that is based on Holt's Linear Trend model. The authors introduce a trend component to leverage the rate of change of gradients between two successive time steps for boosting convergence. However, their Tom optimizer assumes that the gradients computed during the optimization process have a consistent increasing, decreasing or constant trend. Unfortunately, this is not always the case. Kolkova et al. found that technical indicators are appropriate to predict trends in data that does not have to contain a seasonality term (the cyclic data changes that are considered in Holt's model), thus deals better with the general case of unclear trends (Kolkova (2018)).

Exponential Moving Averages. The well known Exponential Moving Average (EMA, as is used by many of the current optimizers) is such a technical indicator. It can be formulated as following,

$$EMA_t = EMA_{t-1} \times \beta + (1 - \beta)x_t \tag{1}$$

where t is a specific time step and x_t is the data point at time step t. β is a hyper-parameter to be tuned. EMA has two extensions called DEMA (Double Exponential Moving Average) and TEMA (Triple Exponential Moving Average) that were first introduced in the financial domain as indicators to evaluate trends of stocks (Mulloy (1994)). DEMA is defined as follows,

$$DEMA = 2 \times EMA_1 - EMA_2 \tag{2}$$

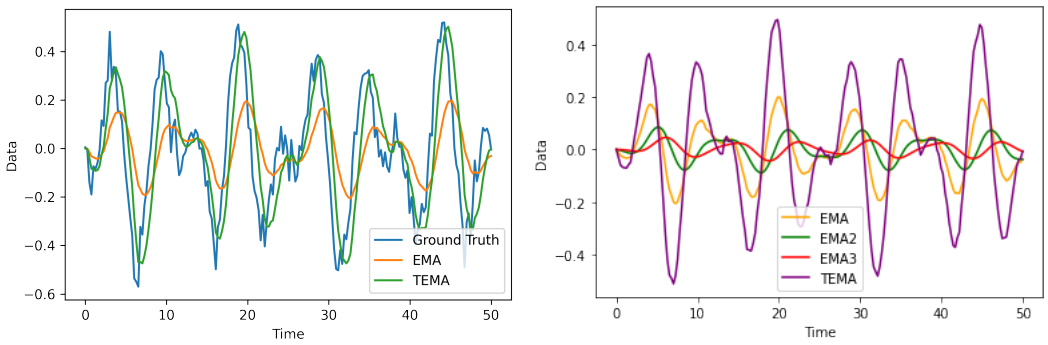
where EMA_1 and EMA_2 are the simple exponential moving average (EMA), and $EMA(EMA_1)$, respectively. Similarly, we define TEMA as,

$$TEMA = 3 \times EMA_1 - 3 \times EMA_2 + EMA_3 \tag{3}$$

where EMA_3 is the $EMA(EMA_2)$.

EMA, DEMA and TEMA can be intuitively considered as multi-level trend estimators. Means, while EMA represents the original data, DEMA includes an additional term (i.e. EMA_2) that considers the changes in the moving average of the original data (i.e. EMA_1). TEMA extends DEMA even further and incorporates EMA_2 changes, thus can be intuitively considered as the 2^{nd} derivative of

the original data itself. As a result, TEMA is a technical indicator that identify trends and their rapid changes more accurately than the EMA or DEMA because it adds important and richer information about the changes in data (by its derivatives). Furthermore, TEMA was originally developed in an attempt to reduce or avoid the inevitable issue of lagging (time difference between predicted output and the ground-truth) that takes place when using the simple Exponential Moving Averages (EMA). Equation 3 shows that TEMA triples the EMA but then cancels out the lagging by subtracting smoothed versions of EMA (EMA_2 and EMA_3 data points already represent different types of averages over the original data). Figure 2 shows the advantage of TEMA over the EMA in terms of identifying trends and responding more quickly to rapid changes in data. It also demonstrates the smoothed versions of EMA - which are the TEMA's components. Considering 1) the response time to changes in the ground truth, and 2) the ability to follow the amplitude of the changes, the TEMA provides much better results than the EMA.



(a) (b)

Figure 1: (a) Simulated demonstration of Trend Estimation and Lagging. TEMA, EMA and the ground-truth are shown. (b) Demonstration of the different smoothed components of TEMA

Having presented the strengths and capabilities of TEMA, **our paper introduces the following contributions:**

- The proposed FAME optimizer is the first to take advantage of the Triple Exponential Moving Average, which is a technical indicator derived from a completely different domain - the stock market. Through the use of TEMA, which provides richer information about gradients and their trend, we improve the fundamental block of EMA that is used in nearly all current optimizers, thus contributing to the entire deep learning field.
- As opposed to using TEMA in the same way as the stock market domain (by comparing its specific value at a certain time point to the market's change), here we use its value as part of a whole continuous average during a deep optimization procedure.
- We examine the relationship between gradients variance and the optimizers performance.
- Our method was extensively evaluated on 1) five different datasets (CIFAR-10, CIFAR-100, PASCAL-VOC, MS-COCO, and Cityscapes), 2) 14 different architectures, and 3) different computer vision tasks - detection, and classification. It was also thoroughly compared with 4) five different optimizers (SGD+momentum, Adam, AdamW, AdaBound and AdaGrad).

3 Our proposed FAME Optimizer

The purpose of this section is to present our proposed FAME optimizer, which incorporates TEMA as its key component. Consequently, FAME can estimate gradient trends more accurately while being more responsive to rapid gradient changes and reducing lagging (and for these reasons it is considered as **Fast** Adaptive Moment Estimation).
Let $f_t(\theta)$ be a noisy objective function: a stochastic scalar function that is differentiable w.r.t. parameters θ. We are interested in minimizing the expected value of this function, $E[f(\theta)]$ w.r.t. its parameters θ. $f_1(\theta), ..., , f_T(\theta)$ represent the realisations of the stochastic function at subsequent time

steps 1,...,T. The stochasticity might come from the evaluation at random sub-samples (mini-batches) of data points instead of using the whole data as a piece, or arise from inherent function noise. We denote the gradient, i.e. the vector of partial derivatives of f_t, w.r.t θ evaluated at time step t:

$$g_t = \nabla_\theta f_t(\theta) \tag{4}$$

First and second moments of the gradients, (m_t, v_t), are represented by the Exponential Moving Average (eq. 1) in the following way -

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \end{aligned} \tag{5}$$

where the hyper-parameters $\beta_1, \beta_2 \in [0, 1)$ control the exponential decay rates of these moving averages. Using the 1^{st} moment (m_t) and the 2^{nd} moment (v_t) equations, we can define EMA_2 (dm_t, dv_t) as:

$$\begin{aligned} dm_t &= \beta_3 dm_{t-1} + (1-\beta_3) m_t \\ dv_t &= \beta_5 dv_{t-1} + (1-\beta_5) v_t \end{aligned} \tag{6}$$

and EMA_3 (tm_t, tv_t) as:

$$\begin{aligned} tm_t &= \beta_4 tm_{t-1} + (1-\beta_4) dm_t \\ tv_t &= \beta_5 tv_{t-1} + (1-\beta_5) dv_t \end{aligned} \tag{7}$$

tm_t, dm_t, tv_t, dv_t are all initialized to 0. Incorporating Eqs. (5-7) into the Triple Exponential Moving Average equation (3) results in the following:

$$\begin{aligned} m_{FAME_t} &= 3m_t - 3dm_t + tm_t \\ &= 3(\beta_1 m_{t-1} + (1-\beta_1) g_t) - 3(\beta_3 dm_{t-1} + (1-\beta_3) m_t) \\ &+ \beta_4 tm_{t-1} + (1-\beta_4) dm_t \\ v_{FAME_t} &= 3v_t - 3dv_t + tv_t \\ &= 3(\beta_2 v_{t-1} + (1-\beta_2) g_t^2) - 3(\beta_5 dv_{t-1} + (1-\beta_5) v_t) \\ &+ (\beta_5 tv_{t-1} + (1-\beta_5) dv_t) \end{aligned} \tag{8}$$

Our final parameter update equation will be calculated as follows:

$$\theta_t = \theta_{t-1} + \alpha \times \frac{m_{FAME_t}}{v_{FAME_t} + \epsilon} \tag{9}$$

In our experiments, we assigned the following values for the hyper-parameters - $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$. These values are commonly used by Adam, SGD+Momentum and others. We also selected $\beta_3 = 0.3$, $\beta_4 = 0.5$, $\beta_5 = 0.8$ and $\epsilon = 0.01$. These values were chosen empirically as they supplied the best results.

3.1 BIAS CORRECTION

Simple EMAs tend to be biased towards initial values, especially during the initial time steps, and especially when decay rates are small (β close to 1). Due to Adam's initialization as a vector of 0's, the EMAs are biased toward 0. For the reasons listed below, we can disregard a bias correction step in our framework:

- Most current optimizers are based on EMA alone, while our FAME is based on $TEMA$, which also includes additional EMA_2, EMA_1. As a result of considering these additional terms, our FAME is less dependent on the original data and has an additional understanding of the data changes. Consequently, gradient changes are handled faster and initial values are less likely to affect.

- Larger β implies a slower decay rate, means that EMA considers previous gradients for a relatively long time window. The moments' decay rates that are used by well-known optimizers are usually within the range of [0.9, 0.99]. These values result relatively slow decay rate. In our proposed method, smaller values are assigned to β_3, β_4, and β_5, leading to a faster decay rate, which ignores gradients from previous time steps.
  It is worth mentioning - as was already explored in published papers, choosing β_1, β_2 that are smaller than 0.9 for Adam results much worse performance, so it is not a reasonable solution. On contrary, TEMA supplies a type of multi-level information for trend estimation that is completely different than using smaller β_1, β_2 for Adam, (that just minimizes the size of the look-back window).

4 EXPERIMENTS

Our experimental phase includes extensive method validation on a variety of:

- **Benchmarks** - CIFAR-10, CIFAR-100, PASCAL-VOC, MS-COCO and Cityscapes.
- **Architectures** - total of 14 different models/ architectures with different complexities.
- **Vision tasks** - we explored the FAME abilities in both detection and classification.
- **Optimizers** - comparison with five optimizers (SGD-momentum, Adam, AdamW, AdaBound and AdaGrad). Running time for all optimizers and for FAME was comparable.

To ensure a fair comparison and evaluation of different optimizers and learning models, we applied:

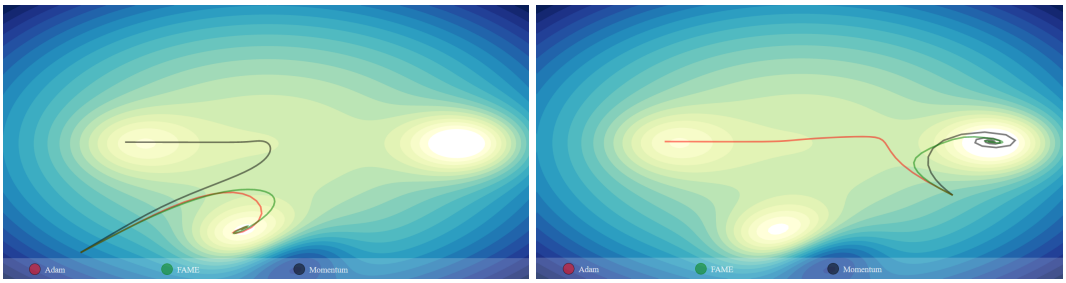
- **Weight Initialization** - identical initial weights that were generated prior to training, for all architectures and optimizers.
- **Optimal Hyper-parameters** - every optimizer was used with its best hyper-parameters. That way we ensured that the performance difference is only due to optimizer choice.
- **Training from Scratch** - all models were trained from scratch to evaluate the pure end-to-end effect of a specific optimizer.

All experiments were performed on Google Cloud Virtual Machines range between k80 and V100, depends on the computational requirements for the specific analyzed data. The machines had up to 52GB RAM, 16 CPUs cores and 2 GPUs.

5 RESULTS

5.1 SIMULATED DATA

To evaluate the abilities of our proposed optimizer, we initially tested it on a simulated data. To do that, we generated a 2D space with several local minima inside (based on Dupont). Each local minimum has its own "depth"/spatial dominance. SGD with Momentum, Adam, and our FAME were all initialized at the same location on the space. We explored the convergence of each optimizer towards the correct minimum point ("correct" - considering both aspects; 1) the local minimum's dominance and 2) its distance from the initial chosen point). Figure 2(a) shows that our proposed FAME converged towards the correct local minimum (the lower one, compared with the left upper one), same as Adam and better than the wrong SGD performance. Figure 2(b) shows that our FAME correctly converged towards the right local minimum which is also the global minimum of the whole 2D space, same as SGD and much better than Adam. These observations clearly show that while different initializations greatly affect the performance of the already-known optimizers (i.e. SGD, Adam), our proposed FAME is more robust and less sensitive to the initial points.



(a) (b)

Figure 2: Simulated data of a 2D space with several local minima. Our Fame, Adam and SGD optimizers were tested. (a-b) Examples for two different initial points.

5.2 Public Benchmarks

5.2.1 Better Generalization and Higher Accuracy of the proposed FAME

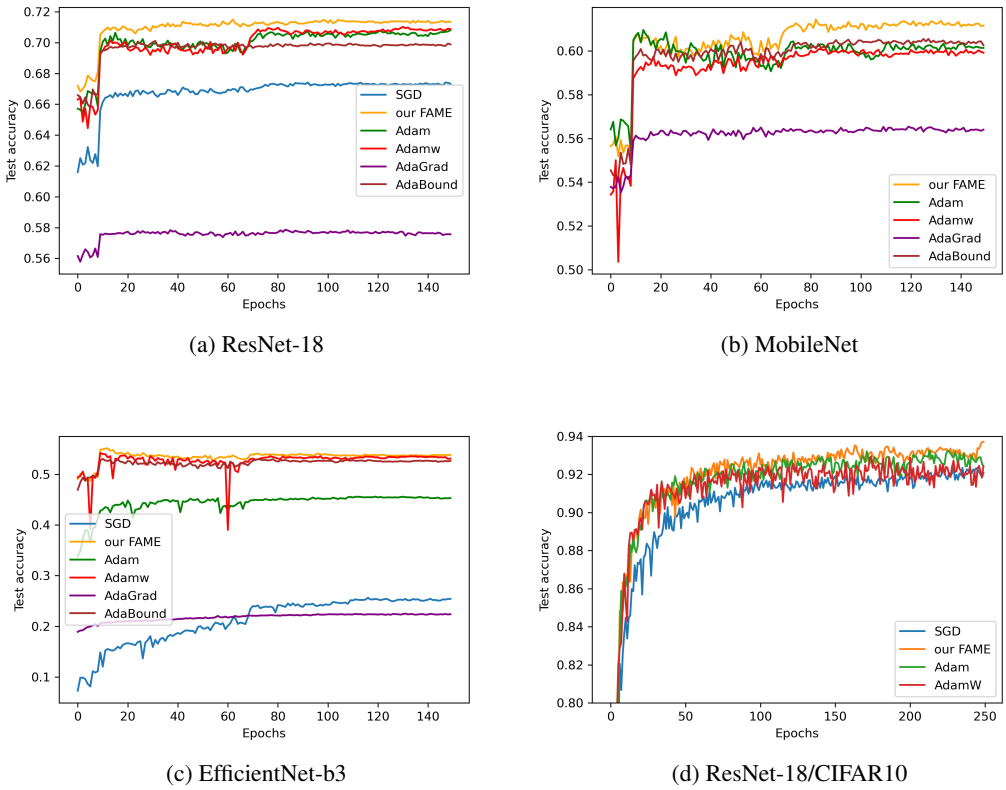
CIFAR-10 and CIFAR-100 benchmarks. The following hyper-parameters were chosen for CIFAR-10/CIFAR-100 datasets (Krizhevsky et al. (2009)) - learning rate=0.01, momentum=0.937, weight decay=0.005, 3 warmup epochs, batch size of 128, and 200 epochs.
Table 1 presents the summary of the results for CIFAR-10 and CIFAR-100 benchmarks. For *top5* classification accuracy, in 9 out of 11 tested architectures (81.8%) our FAME optimizer outperformed both Adam and SGD. For *top1* accuracy, in 9 out of the 12 (75%) learning architectures that were tested (11 for CIFAR-100 and additional one for CIFAR-10), our proposed FAME outperformed both Adam and SGD. For the three remaining architectures, our FAME was at the 2^{nd} place. It has not been assigned to the 3^{rd} place in any of these 11 architectures (not for top5 nor top1).

Dataset	Architecture	Our FAME	Adam	SGD+Momentum
CIFAR-10 ***(top1)***	Resnet18	**93.72**	92.41	92.08
CIFAR-100 (***top5/top1***)	Efficentnet-b0	**75.5 / 49.2**	74.3 / 45.9	52.6 / 35.2
	Efficentnet-b3	**78.5 / 53.8**	74.2 / 45.3	53.4 / 25.4
	MobileNet	85 / 61.2	83.2 / 60.1	**87.4 / 64.4**
	DenseNet-121	**91.8 / 73.8**	88.1 / 66.4	91.7 / 72.7
	DenseNet-201	**92.1** / 73.4	92 / **73.9**	92 / 73.3
	SqueezeNet	**89.4 / 66.2**	88.6 / 65.1	88.6 / 64.7
	Resnet-18	**90.8 / 71.3**	90.7 / 70.8	88.9 / 67.3
	Resnet-34	**91.4 / 72.6**	90.7 / 71.1	89.4 / 68.8
	SEResnet-18	**91.1 / 72.5**	90.5 / 70.4	88.9 / 67.2
	Inception-v3	92.7 / 74.5	**93 / 74.9**	90.1 / 70.2
	WideResnet 40-4	**91.7 / 71.4**	91.4 / 70.6	91.5 / 70.8

Table 1: Comparison of classification accuracy supplied by different optimizers across architectures on CIFAR-10 and CIFAR-100 datasets. For CIFAR-100 - results represent top5 (left) and top1 (right) accuracies (in %). For CIFAR-10 - results are top1. Architectures were trained from *scratch*. **Best results for each architecture are bolded**.

Figure 3 presents the comparison of our FAME with additional optimizers - AdaBound, AdamW and AdaGrad, for a subgroup of five architectures. We chose a random representative for each architectures family that appears in Table 1 - ResNet, DenseNet, MobileNet and EfficientNet. The results for the other optimizers that we compared with are fully consistent with the literature. Figure 3 clearly shows that our FAME constantly supplies the most accurate performance.

Pascal-VOC benchmark. The following hyper-parameters were chosen for the Pascal Visual Object Classes (VOC) benchmark (Everingham et al. (2015)) - learning rate=0.0334, momentum=0.74832, weight decay=0.00025, 3 warmup epochs, batch size of 48, and 200 epochs.
We incorporated our proposed FAME in both Yolov5-m and Yolov5-s architectures and compared its performance with Adam, AdamW and SGD. For example, for Yolo5v-m our FAME obtained



(a) ResNet-18

(b) MobileNet

(c) EfficientNet-b3

(d) ResNet-18/CIFAR10

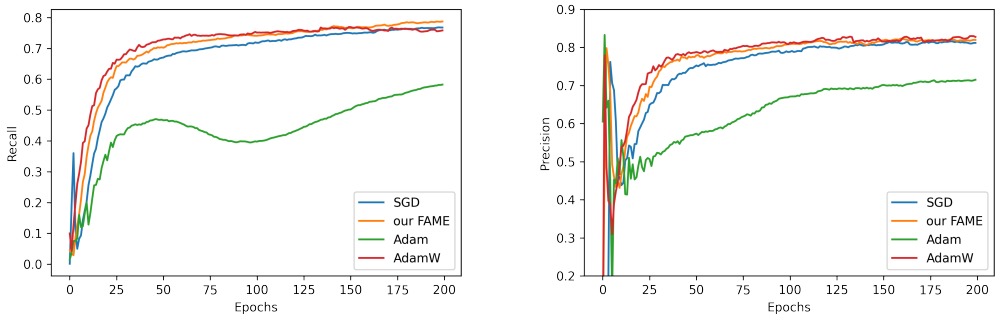
Figure 3: Optimizers Comparison. (a) ResNet-18/CIFAR-100, (b) MobileNet/CIFAR-100, (c) Efficient-B3/CIFAR-100, (d) ResNet-18/CIFAR-10. *For clear visualization - the CIFAR-100 accuracy results (a-c) are shown from epoch 50 and later.*

mAP@0.5 (Mean Average Precision over IoU threshold of 0.5) of 0.85 while SGD, Adam and AdamW supplied mAP@0.5 of 0.83, 0.66 and 0.83, respectively (Table 2). These results show a significant improvement of the learning architecture when using our FAME. Moreover, figure 4 shows that our FAME optimizer was better than SGD, Adam and AdamW optimizers in term of Precision as well. For Recall, our FAME was better than SGD and ADAM and comparable to AdamW.

Architecture	Our FAME	SGD	Adam	AdamW
Yolo5v-s	**81.2**	78.7	65.1	80.1
Yolo5v-m	**85**	82.8	66.2	83

Table 2: Comparison of classification Mean Average Precision (mAP@0.5) supplied by different optimizers across architectures on PASCAL-VOC dataset. **Best results for each architecture are bolded**.

MS-COCO benchmark. We validated our FAME on MS-COCO as well (Lin et al. (2014)), by incorporating it in the Yolov5-n architecture.
The following hyper-parameters were chosen - learning rate=0.01, momentum=0.937, weight decay=0.0005, 3 warmup epochs, batch size of 128, and 250 epochs.



(a) Recall (b) Precision

Figure 4: Recall and Precision. Comparison of SGD, Adam, AdamW and FAME on the PASCAL-VOC benchmark.

Our FAME results were compared to SGD, Adam, and AdamW (Table 3). For all statistical parameters of mAP@0.5, Precision, Recall, and F1-score, FAME was found to be superior to others.

	Our FAME	SGD	Adam	AdamW
mAP@0.5	**0.529**	0.446	0.191	0.441
Precision	**0.649**	0.59	0.331	0.542
Recall	**0.486**	0.412	0.234	0.417
F1-Score	**0.556**	0.485	0.274	0.471

Table 3: Comparison of classification Mean Average Precision (mAP@0.5) supplied by different optimizers and by using YOLOv5-n architecture on MS-COCO dataset. Results of other optimizers we compared with fully agree with the literature. **Best results for each tested parameter are bolded**.

Cityscapes benchmark. We used a ResNet-50 backbone for Deeplabv3+ architecture proposed by (Chen et al. (2018)) to analyze Cityscapes benchmark (Cordts et al. (2016)). This is the 14^{th} architecture that was tested in this paper. The following hyper-parameters were chosen - momentum=0.9, learning rate=0.001, weight decay=0.0001, batch size of 16, with 163 epochs.
Table 4 presents the mean IoU results, indicating that our FAME outperformed the others.

	Our FAME	SGD	Adam	AdamW
Mean IoU	**0.76**	0.63	0.74	0.74

Table 4: Comparison of classification mean IoU supplied by different optimizers for Cityscapes benchmark. **Best result for mean IoU is bolded**.

5.2.2 Sensitivity to Parameters

Having a grid search for hyper-parameters is not a common procedure in deep learning due to its time consuming and its significant computational load. However, in this paper we applied a grid search to show the high robustness of the proposed FAME and its low sensitivity to hyper-parameters choice. For computational efficiency, the hyper-parameters for CIFAR-10 were searched on an extended grid. We tested a wide range of values for each hyper-parameter - $\beta_{3,4,5}$ - [0,1] with discrete steps of 0.1 for each β, and ϵ - [0.001, 0.02] with increments of 0.005. Applying all these options, the largest difference between the classification accuracies while using different sets of parameters was only 5.54%. These results support the high robustness of our FAME over different values of hyper-parameters. For CIFAR-100, Cityscapes, MS-COCO and Pascal-VOC, we tested the following range of values for each hyper-parameter - $\beta_{3,4,5}$ - [0, 0.8] with discrete steps of 0.1 for each β and ϵ - [0.01, 0.1] with increments of 0.01. Applying all these options, the largest difference between the classification accuracies while using these different sets of parameters was only 3.72%. The results support the high robustness of our FAME over different values of hyper-parameters.

5.2.3 FAME PERFORMANCE AND ITS EFFECT ON THE GRADIENTS VARIANCE

Faghri et al. (2020) and Defazio & Bottou (2018) found that gradients variance depends on the distance to local minima and can be affected by challenging data or by noisy data estimated from random data samples (i.e. mini-batches) rather than that from the entire data. If the noisy gradients have a large variance, the stochastic gradient algorithm might spend much time bouncing around, leading to slower convergence and worse performance (Wang et al. (2013)). To demonstrate our FAME's capabilities, we also examined the gradients' variance and its change over time. Is the gradient distribution/variance affected differently by various optimizers during the learning process? Figure 5 demonstrates the gradients variance that was calculated within the last layer (i.e. Fully-connected layer) during the last epoch compared to the equivalent gradient variance during the first epoch. Meaning, as long as the last/first epochs ratio is smaller than 1 - it indicates that the learning model converged. The results that appear in figure 5 are in agreement with table 1 - for each architecture where our FAME optimizer supplied the best classification performance, the gradient variance ratio was lower than the equivalent values provided by SGD and Adam. On contrary, when our FAME did not provide the best performance (e.g. MobileNet), its gradient variance ratio was higher than the equivalent ratio provided by the winner optimizers.

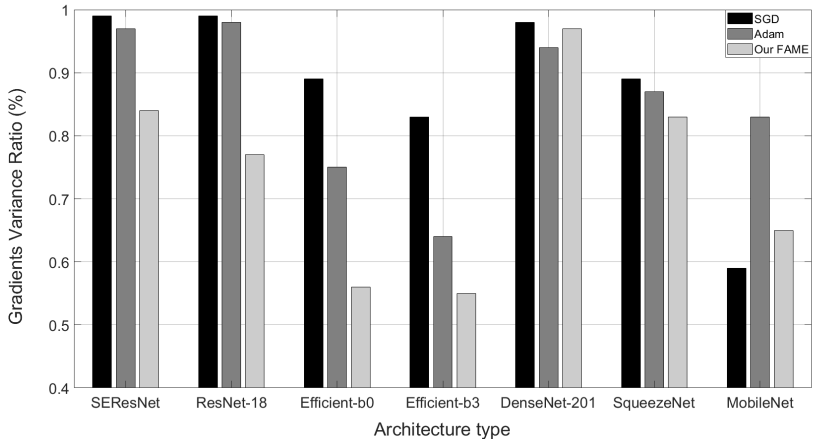


Figure 5: Gradients Variance Ratio (last epoch/1^{st} epoch). Lower ratio is better.

5.2.4 MEMORY COST AND COMPUTATIONAL TIME

The proposed optimizer requires a slightly higher memory cost and computational time, compared with other optimizers. For example, compared with Adam, it requires twice the memory and a 5% increase in computational time. Compared with SGD, the equivalent values were 2.5 times in memory cost and 7% in computational time. It is important to clarify, that this increase in both aspects is not critical and can be easily handled by commonly- used hardware.

6 DISCUSSION AND FUTURE WORK

In this paper, we introduce the FAME optimizer, which is the first to utilize the Triple Exponential Moving Average, a technical indicator that comes from a completely different domain - the stock market. TEMA, which provides richer information about gradients and their trends, improves the fundamental block of EMA used in nearly all deep learning optimizers, therefore providing a significant contribution and a broad effect across the entire field. Using a significant number of benchmarks, architectures, and computer vision tasks, we validated our FAME against other commonly used optimizers. The results demonstrate that our FAME is superior to others, supplying the most accurate and robust performance.

Future work can include an implementation of our FAME in learning architectures that require heavier computational resources. Because the training of such frameworks from scratch requires extremely strong computational resources, we were limited in our ability to do that at this point. In addition, testing the FAME optimizer in the natural language domain can be interesting as well.

REFERENCES

Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. *CoRR*, abs/1802.02611, 2018. URL http://arxiv.org/abs/1802.02611.

Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3213–3223, 2016.

Aaron Defazio and Léon Bottou. On the ineffectiveness of variance reduced optimization for deep learning. *CoRR*, abs/1812.04529, 2018. URL http://arxiv.org/abs/1812.04529.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 07 2011.

Emilien Dupont. Optimization algorithms visualization. URL https://bl.ocks.org/EmilienDupont/aaf429be5705b219aaaf8d691e27ca87.

M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, January 2015.

Fartash Faghri, David Duvenaud, David J. Fleet, and Jimmy Ba. A study of gradient variance in deep learning. *CoRR*, abs/2007.04532, 2020. URL https://arxiv.org/abs/2007.04532.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL http://arxiv.org/abs/1502.03167.

Nitish Shirish Keskar and Richard Socher. Improving generalization performance by switching from adam to sgd. *arXiv preprint arXiv:1712.07628*, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Andrea Kolkova. Indicators of technical analysis on the basis of moving averages as prognostic methods in the food industry. *Journal of Competitiveness*, 10(4):102–119, 2018.

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

Nicola Landro, Ignazio Gallo, and Riccardo La Grassa. Mixing ADAM and SGD: a combined optimization method. *CoRR*, abs/2011.08042, 2020. URL https://arxiv.org/abs/2011.08042.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pp. 740–755. Springer, 2014.

Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam, 2018. URL https://openreview.net/forum?id=rk6qdGgCZ.

Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. Adaptive gradient methods with dynamic bound of learning rate. *CoRR*, abs/1902.09843, 2019. URL http://arxiv.org/abs/1902.09843.

Niru Maheswaranathan, David Sussillo, Luke Metz, Ruoxi Sun, and Jascha Sohl-Dickstein. Reverse engineering learned optimizers reveals known and novel mechanisms. *CoRR*, abs/2011.02159, 2020. URL https://arxiv.org/abs/2011.02159.

Anirudh Maiya, Inumella Sricharan, Anshuman Pandey, and Srinivas K. S. Tom: Leveraging trend of the observed gradients for faster convergence. *CoRR*, abs/2109.03820, 2021. URL https://arxiv.org/abs/2109.03820.

Patrick G. Mulloy. Smoothing data with faster moving averages. *Technical Analysis of Stocks Commodities*, 1994. URL http://technical.traders.com/archive/volume-2014.asp?yr=1994#Jan.

N. Qian. On the momentum term in gradient descent learning algorithms. *Neural networks : the official journal of the International Neural Network Society*, 12 1:145–151, 1999.

Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond, 2019. URL https://arxiv.org/abs/1904.09237.

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

T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

Chong Wang, Xi Chen, Alexander J Smola, and Eric P Xing. Variance reduction for stochastic gradient optimization. *Advances in neural information processing systems*, 26, 2013.

Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning, 2017. URL https://arxiv.org/abs/1705.08292.

Matthew D. Zeiler. Adadelta: An adaptive learning rate method, 2012.