Adaptive Momentum Coefficient for Neural Network Optimization

Zana Rashidi[1](✉), Kasra Ahmadi K. A.[1], Aijun An[1], and Xiaogang Wang[2]

[1] Department of Electrical Engineering and Computer Science
{zrashidi,kasraah,aan}@eecs.yorku.ca
[2] Department of Mathematics and Statistics
stevenw@mathstat.yorku.ca
York University, Toronto ON M3J 1P3, CA

Abstract. We propose a novel and efficient momentum-based first-order algorithm for optimizing neural networks which uses an adaptive coefficient for the momentum term. Our algorithm, called *Adaptive Momentum Coefficient* (AMoC), utilizes the inner product of the gradient and the previous update to the parameters, to effectively control the amount of weight put on the momentum term based on the change of direction in the optimization path. The algorithm is easy to implement and its computational overhead over momentum methods is negligible. Extensive empirical results on both convex and neural network objectives show that AMoC performs well in practise and compares favourably with other first and second-order optimization algorithms. We also provide a convergence analysis and a convergence rate for AMoC, showing theoretical guarantees similar to those provided by other efficient first-order methods.

Keywords: Adaptive Momentum · Neural Networks · Optimization · Accelerated Gradient Descent · Convex Optimization

1 Introduction

First order optimization methods such as Stochastic Gradient Descent (SGD) with Momentum [21] and their variants are the methods of choice for optimizing neural networks. While there has been extensive work on developing second-order methods such as Hessian-Free optimization [11] and Natural Gradients [1, 12], such methods have not been successful in replacing the first-order ones due to their large per-iteration costs in time and memory.

Although Nesterov's accelerated gradient and its modifications have been very effective in deep neural network optimization [21], it has been shown that Nesterov's method might perform suboptimal for strongly convex functions [2] without looking at local geometry of the objective function. Further, in order to get the best of both worlds, search for optimization methods which combine the efficiency of first-order methods and the effectiveness of second-order updates is still underway.

In this work, we introduce an adaptive coefficient for the momentum term in the Heavy Ball and Nesterov methods as an effort to combine first-order and second-order methods. We call our algorithm *Adaptive Momentum Coefficient* (AMoC). The adaptive coefficient effectively weights the momentum term based on the change in direction on the loss surface in the optimization process. The change in direction can contribute as implicit local curvature information without resorting to the expensive second-order information such as the Hessian or the Fisher Information Matrix.

Our experiments show the effectiveness of the adaptive coefficient on both strongly-convex functions with Lipschitz gradients and neural network objectives. AMoC can speed up the convergence process significantly in convex problems and performs well in the neural network experiments. AMoC has similar time-efficiency as first-order methods (e.g. Heavy Ball, Nesterov) while reaching lower errors.

The structure of the paper is as follows. In section 2, we give a brief background on neural network optimization methods. In section 3, we introduce our adaptive momentum coefficient algorithm and discuss its merits. In section 4, we analyse the convergence of the algorithm and provide a convergence rate. In section 4, we discuss some of the related work in the literature. Section 5 illustrates the algorithm's performance on convex and non-convex benchmarks. Proofs and details regarding the algorithm and the experiments can be found in the Appendix.

2 Background

We consider a neural network with a differentiable loss function $f : \mathbb{R}^D \rightarrow \mathbb{R}$ with the set of parameters θ. The objective is to minimize the loss function f with a set of iterative updates to the parameters θ. Gradient descent methods use the following update:

$$\theta_{t+1} = \theta_t - \epsilon \nabla f(\theta_t) \tag{1}$$

where ϵ is the learning rate. However, this update can be very slow and determining the learning rate ϵ can be hard. A large learning rate can cause oscillations and overshooting. A small learning rate can slow down the convergence drastically.

Heavy Ball method. In order to speed up the convergence of gradient descent, one can add a momentum term [18].

$$d_{t+1} = \mu d_t - \epsilon \nabla f(\theta_t); \quad \theta_{t+1} = \theta_t + d_{t+1} \tag{2}$$

where d is the velocity and μ is the momentum parameter.

Nestrov's method. Nesterov's accelerated gradient [16] can be rewritten as a momentum method [21]:

$$d_{t+1} = \mu d_t - \epsilon \nabla f(\theta_t + \mu d_t); \quad \theta_{t+1} = \theta_t + d_{t+1} \tag{3}$$

Algorithm 1: AMoC (Adaptive Momentum Coefficient)

```
1 Initialize $\theta_1$
2 Set $d_1 = \mathbf{0}$
3 for $t = 1$ to $T$ do
4     Calculate the gradient $g_t = \nabla f(\theta_t)$
5     Calculate adaptive coefficient $\gamma_t = \mu\big(1 - \beta(\bar{g}_t \cdot \bar{d}_t)\big)$
6     Calculate update $d_{t+1} = \gamma_t d_t - \epsilon g_t$
7     Update parameters $\theta_{t+1} = \theta_t + d_{t+1}$
```

Nesterov's momentum is different from the heavy ball method only in where we take the gradient. Note that in both methods d_t is the previous update $\theta_t - \theta_{t-1}$.

3 Adaptive Momentum Coefficient

We propose Adaptive Momentum Coefficient (AMoC) as an alternative to the fixed momentum parameter in the Heavy Ball and Nesterov's algorithms. The adaptive coefficient utilizes the angle between the direction of the current gradient vector and the previous update to the parameters. This angle is characterized by the inner product between the normalized vectors of these two values, namely

$$\bar{g}_t \cdot \bar{d}_t = \cos(\pi - \phi_t) \quad \text{where} \quad \bar{g}_t = \frac{g_t}{\|g_t\|}; \quad \bar{d}_t = \frac{d_t}{\|d_t\|} \tag{4}$$

where ϕ_t is the angle between the negative gradient $-g_t$ and the previous update d_t. Our goal with the adaptive coefficient is to automatically reinforce the momentum term when these two directions align and gradually decrease this effect when they don't. Thus we propose the following coefficient for the momentum term:

$$\gamma_t = \mu\big(1 - \beta(\bar{g}_t \cdot \bar{d}_t)\big) \tag{5}$$

where $\mu \geq 0$ is the regular momentum parameter and $\beta \geq 0$ is a parameter controlling the amount of weight put on the inner product. With a large β, the algorithm will behave more aggressively, meaning, moving rapidly when directions align and bouncing back when they don't, which is suitable for convex functions. A small β however, leads to a more conservative behaviour, which is expected when optimizing non-convex objectives, e.g., neural networks (to avoid being trapped in local optima, for example). Since ϕ can range from 0 to π,

$$\mu(1 - \beta) \leq \gamma_t \leq \mu(1 + \beta) \tag{6}$$

Note that if we set β to zero, γ_t reduces to the regular momentum parameter for Heavy Ball, which can be seen as a special case of our algorithm. The adaptive coefficient embeds a notion of change of direction of the optimization path. This notion can be interpreted as implicit second-order information where it tells us how much the current gradient's direction is different from the previous gradients which is similar to what second-order information (e.g. the Hessian) provides intuitively:

$$H(\theta_t) = \nabla^2 f(\theta_t) \tag{7}$$

The Hessian provides the rate of change in the gradient of a function at a point while the gradient tells us the rate of change of the function itself. Since the previous update contains a running estimate of the past gradients, the dot product is basically comparing the current gradient against all aggregated past gradients.

The algorithm (AMoC) is shown in Algorithm 1. We also incorporate Nesterov's lookahead gradient into AMoC by simply taking the gradient at a further point, $g_t = \nabla f(\theta_t + \mu d_t)$, thus only changing line 4 of Algorithm 1. We call this version of the algorithm AMoC-N, with "N" standing for Nesterov.

4 Convergence Analysis

To analyse AMoC's convergence, consider a differentiable convex function f with Lipschitz gradients. On this class of functions, similar to the Heavy Ball algorithm, AMoC benefits from a convergence rate of order $1/T$ with a slightly different factor. To prove this proposition, we need to define the following sequence of coefficients:

Definition. *The sequence $\{\lambda_k\}$ is defined as:*

$$\lambda_{k+1} = \frac{\lambda_k}{\gamma_k} - 1, \qquad \lambda_1 = \frac{\mu(1-\beta)}{1-\mu(1-\beta)} \tag{8}$$

Lemma. *For arbitrarily large integer T, there exists $\beta > 0$ such that the first $T+1$ elements of the sequence $\{\lambda_k\}$ are positive.*

Proof. See appendix A.

This lemma provides us with the proper β for the following theorem. This theorem guarantees the convergence of the algorithm, and provides an upper-bound for the convergence rate of a specific weighted average of all iterates.

Theorem. *For any differentiable convex function f with L-Lipschitz gradients, the sequence generated by AMoC with sufficiently small β, $\mu \in [0, \frac{1}{1+\beta})$, and $\epsilon \in (0, \frac{\mu(1+\beta)}{L\lambda_1})$ satisfies the following:*

$$f(\tilde{\theta}_T) - f(\theta^*) \leq \frac{\|\theta_1 - \theta^*\|^2}{2T(1+\lambda_{T+1})}\Big(\frac{1}{\epsilon} + \frac{\lambda_1^2 L}{\mu}\Big) \tag{9}$$

where θ^* is the optimal point and $\tilde{\theta}_T = (\sum_{k=1}^{T} \frac{\lambda_k}{\gamma_k}\theta_k)/(\sum_{k=1}^{T} \frac{\lambda_k}{\gamma_k})$.

Proof. See appendix A.

It can be easily verified that by setting β to 0, all the elements of the sequence $\{\lambda_k\}$ will be equal to $\frac{\mu}{1-\mu}$ and (9) becomes the same bound as the one for the Heavy Ball algorithm presented in [6].

5 Related Work

There has been extensive work on large-scale optimization techniques for neural networks in recent years. A good overview can be found in [3]. Here, we discuss some of the work more related to ours in three parts.

5.1 Gradient Descent Variants

Adagrad [5] is an optimization technique that extends gradient descent and adapts the learning rate according to the parameters. Adadelta [24] and RMSprop [22] improve upon Adagrad by reducing its aggressive deduction of the learning rate. Adam [9] improves upon the previous methods by keeping an additional average of the past gradients which is similar to what momentum does. Adaptive Restart [17] proposes to reset the momentum whenever rippling behaviour is observed in accelerated gradient schemes. AggMo [10] keeps several velocity vectors with distinct parameters in order to damp oscillations. AMSGrad [19] on the other hand, keeps a longer memory of the past gradients to overcome the suboptimality of the Adam on simple convex problems.

5.2 Accelerated Methods

Several recent works have been focusing on acceleration for gradient descent methods. In [13], the authors propose an adaptive method to accelerate Nesterov's algorithm in order to close a small gap in its convergence rate for strongly convex functions with Lipschitz gradients adding a possibility of more than one gradient call per iteration. In [20], the authors propose a differential equation for modeling Nesterov inspired by the continuous version of gradient descent, a.k.a. gradient flow. The authors in [23] take this further and suggest that all accelerated methods have a continuous time equivalent defined by a Lagrangian functional, which they call the Bregman Lagrangian. Recently, in [4] the authors propose a differential geometric interpretation of Nesterov's method for strongly-convex functions with links to continuous time differential equations mentioned earlier and their Euler discretization.

5.3 Second-order Methods

Second-order methods are desirable because of their fine convergence properties due to dealing with bad-conditioned curvature by using local second-order information. Hessian-Free optimization [11] is based on the truncated-Newton approach where the conjugate gradient algorithm is used to optimize the quadratic approximation of the objective function. The natural gradient method [1] reformulates the gradient descent in the space of the prediction functions instead of the parameters. This space is then studied using concepts in differential geometry. K-FAC [12] approximates the Fisher information matrix which is based on the natural gradient method. Our method is different since we are not using explicit second-order information but rather implicitly deriving curvature information using the change in direction.

Table 1: Number of iterations to convergence for the strongly-convex functions with Lipschitz gradients experiment. Note that in our experiments, Adam did not converge for the Smooth-BPDN and Ridge Regression problem.

	AMoC	**AMoC-N**	**Heavy Ball**	**Nesterov**	**Adam**
Anisotropic Bowl	78	**50**	1901	1749	888
Smooth-BPDN	**59**	77	110	115	–
Ridge Regression	**1160**	1817	> 3000	> 3000	–

6 Experiments

We evaluated AMoC on strongly convex functions with Lipschitz gradients and neural network objectives including Autoencoders, Residual Networks and LSTMs. We compare our algorithm with Heavy-Ball, Nesterov, and Adam in addition with K-FAC in the Autoencoder experiment. Our neural network experiment setups closely follow [10], were tuned for best performance on the validation set and implemented in PyTorch (except K-FAC, where we used its official Tensorflow code[3] with the optimal parameters). See Appendix B for a detailed discussion of the inner product values during these experiments.

6.1 Strongly Convex and Lipschitz functions

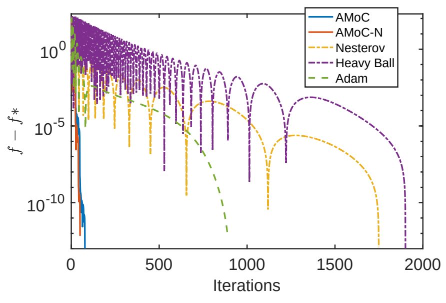
We borrow these three minimization problems from [13] where the authors try to accelerate Nesterov's method by using adaptive step sizes. The problems are Anisotropic Bowl, Ridge Regression and Smooth-BPDN. We fix the momentum parameter μ for all methods which is set to 0.99 for the Anisotropic Bowl and 0.9 for the other two problems. The learning rate ϵ for all methods is tuned for best performance. The parameter β in our algorithms is set to 1 for the Anisotropic Bowl for both AMoC and AMoC-N, 0.1 for Ridge Regression for both methods and 1 for AMoC in Smooth-BPDN and 0.1 for AMoC-N in the same problem. Results are shown in Figure 1 and Table 1.

Anisotropic Bowl. The Anisotropic Bowl is a bowl-shaped function with a constraint to get Lipschitz continuous gradients:

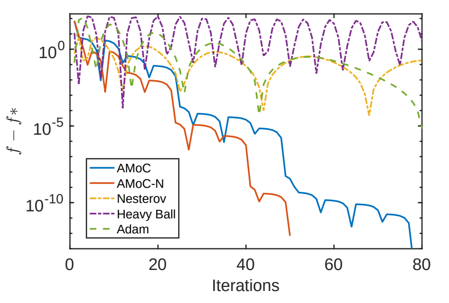
$$f(\theta) = \sum_{i=1}^{n} i \cdot \theta_{(i)}^4 + \frac{1}{2}\|\theta\|_2^2, \qquad \text{subject to } \|\theta\|_2 \leq \tau \tag{10}$$

As in [13], we set $n = 500$, $\tau = 4$ and $\theta_0 = \frac{\tau}{\sqrt{n}}\mathbf{1}$. Figure 1a and 1b (magnified) show the convergence results for our algorithms and the baselines. The algorithms terminate when $f(\theta) - f(\theta^*) < 10^{-12}$. AMoC-N and AMoC take only 50 and 78 iterations to converge, while the closest result is that of Adam which takes 888 iteration to converge.

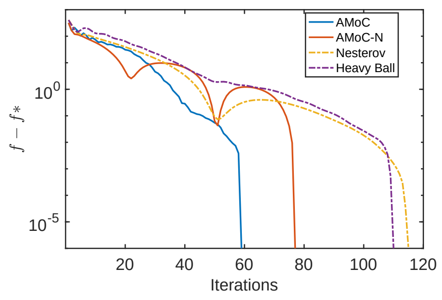
[3] https://github.com/tensorflow/kfac



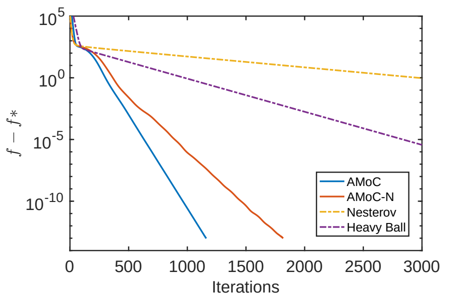
(a) Anisotropic Bowl



(b) Anisotropic Bowl (magnified)



(c) Smooth-BPDN



(d) Ridge Regression

Fig. 1: Results from experiments on strongly convex functions with Lipschitz gradients. All methods start from the same point. The vertical axes show the distance to the optimal value ($f - f^*$, where $f^* = f(\theta^*)$) and the horizontal axes show the number of iterations.

Ridge Regression. The Ridge Regression problem is a linear least squares function with Tikhonov regularization:

$$f(\theta) = \frac{1}{2}\|A\theta - b\|_2^2 + \frac{\lambda}{2}\|\theta\|_2^2 \tag{11}$$

where $A \in \mathbb{R}^{m \times n}$ is a measurement matrix, $b \in \mathbb{R}^m$ is the response vector and $\lambda > 0$ is the ridge parameter. The function $f(\theta)$ is a positive definite quadratic function with the unique solution of $\theta^* = (A^TA + \lambda I)^{-1}A^Tb$.

Following [13], $m = 1200$, $n = 2000$ and $\lambda = 1$. A is generated from $U\Sigma V^T$ where $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times m}$ are random orthonormal matrices and $\Sigma \in \mathbb{R}^{m \times m}$ is diagonal with entries linearly distanced in $[100, 1]$ while $b = \text{randn}(m, 1)$ is drawn i.i.d. from the standard normal distribution. Figure 1d shows the results where AMoC and AMoC-N converge in 1160 and 1817 iterations respectively. The Nesterov version of the algorithm (AMoC-N) is not able to outperform the regular version as is the case with the original Heavy Ball and Nesterov algorithms. The tolerance is set to $f(\theta) - f(\theta^*) < 10^{-13}$. We have not included Adam since it was not able to perform well in this problem (also see [19]).

Smooth-BPDN. Smooth-BPDN is a smooth and strongly convex version of the BPDN (basis pursuit denoising) problem:

$$f(\theta) = \frac{1}{2}\|A\theta - b\|_2^2 + \lambda\|\theta\|_{\ell_1,\tau} + \frac{\rho}{2}\|\theta\|_2^2 \tag{12}$$

where:

$$\|\theta\|_{\ell_1,\tau} = \begin{cases} |\theta| - \frac{\tau}{2} & \text{if } |\theta| \geq \tau \\ \frac{1}{2\tau}\theta^2 & \text{if } |\theta| < \tau \end{cases}$$

and $\|\cdot\|_{\ell_1,\tau}$ is a smoothed version of the ℓ_1 norm also known as Huber penalty function with half-width of τ.

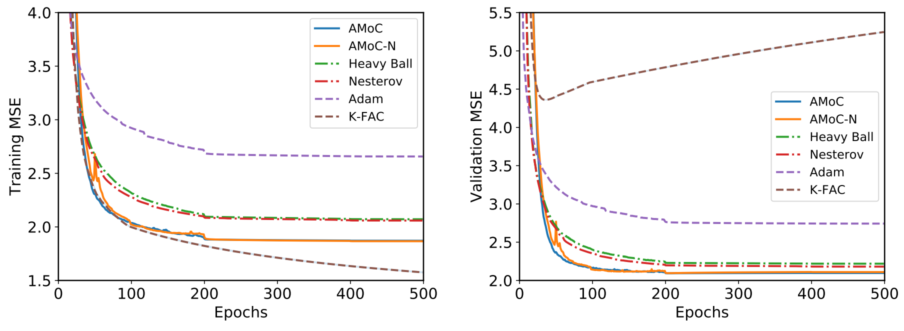
As in [13], we set $A = \frac{1}{\sqrt{n}} \cdot \text{randn}(m, 1)$ where $m = 800$ and $n = 2000$, $\lambda = 0.05$, $\tau = 0.0001$. The real signal is a random vector with 40 non-zero values and $b = A\theta^* + e$ where $e = 0.01\frac{\|b\|_2}{\sqrt{m}} \cdot \text{randn}(m, 1)$ is Gaussian noise. Since we cannot find the solution analytically, Nesterov's method is used as an approximation to the solution $(f(\theta_N^*))$ and the tolerance is set to $f(\theta) - f(\theta_N^*) < 10^{-12}$. Figure 1c shows the results for the algorithms. AMoC-N and AMoC converge in 77 and 59 iterations respectively, outperforming all other methods. We observe the weakness of the lookahead gradient in this problem as well. Similarly, Adam was not able to perform well and hence not included it in the graph (also see [19]).

6.2 Deep Autoencoders

To evaluate the performance of AMoC, we apply it to the benchmark deep autoencoder problem first introduced in [8] on the MNIST dataset. We use the same network architectures as in [8] which is [1000 500 250 30 250 500 1000] except we use ReLU activation throughout the model. Our baselines are the Heavy Ball algorithm [18], SGD with Nesterov's Momentum [21] and K-FAC [12], a second-order method utilizing natural gradients using an approximation of the Fisher information matrix.

We use 90% of the training data for training and 10% for validation. All methods use the same parameter initialization scheme. All methods except K-FAC (which use a special learning rate and momentum schedule along with an increasing batch size schedule), use a fixed momentum parameter and we decay the learning rate by 0.1 at epochs 200 and 400. The parameter β is set to 0.1. For the momentum parameter μ, we did a search in {0.9,0.99,0.999} and the learning rate in {0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001}. The minibatch size was set to 200. We set β_2 in Adam to 0.999 and searched over {0.9, 0.99, 0.999} for β_1.

The training and validation results for 500 epochs of training are shown in Figure 2 and the test set results are shown in Table 2. AMoC and AMoC-N outperform the baselines in terms of validation error and perform similarly to each



(a) Training Error (b) Validation Error

Fig. 2: Training and Validation Error during training for the MNIST Autoencoder experiment. The vertical axes show the Mean Squared Error (reconstruction error) and the horizontal axes show the number of epochs.

Table 2: Test errors from the MNIST Autoencoder experiment.

AMoC	**AMoC-N**	**Heavy Ball**	**Nesterov**	**Adam**	**K-FAC**
2.08	**2.08**	2.20	2.16	2.72	6.07

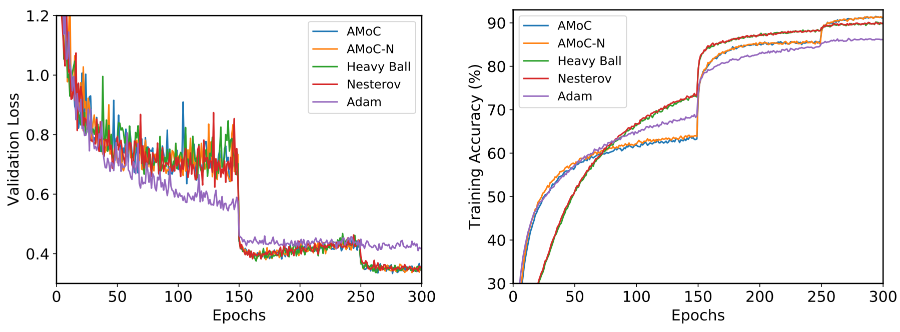
other. K-FAC outperforms all methods in terms of training error but overfits to the dataset with the default parameters. Use of Nesterov's lookahead gradient does not affect the performance significantly in both Heavy Ball and AMoC. Adam performs poorly relative to other methods in this experiment. Further, AMoC and AMoC-N reach the lowest testing error among the algorithms followed by Nesterov and Heavy Ball.

6.3 Residual Networks

The classification experiments were done using 34-layer residual networks [7] on two datasets of CIFAR10 and CIFAR100. Results are shown in Figure 3 and Table 3. We trained each model for 300 epochs using 80% of the training data while holding out a random 20% for validation.

We used a batch size of 128 and searched for the learning rate in $\{0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001\}$ while decaying by 0.1 at epochs 150 and 250. β_1, β_2 and μ were set similar to the Autoencoder experiment. β was set to 0.2 for CIFAR10 and 0.1 for CIFAR100. Batch normalization and a weight decay of 0.0005 were used in the models. Data augmentation was also limited to random resized cropping and horizontal flips.

In the CIFAR10 experiment, the algorithms AMoC, AMoC-N, Heavy Ball and Nesterov outperform Adam and perform similarly among themselves. However, AMoC achieves the highest accuracy on the test set. For CIFAR100, AMoC



(a) Validation Loss for CIFAR10. (b) Training Accuracy for CIFAR100.

Fig. 3: Results from the ResNet-34 experiments on CIFAR10 and CIFAR100. The vertical axis for the figure on the left shows the loss and of the figure on the right shows the accuracy. The horizontal axes show the number of epochs.

Table 3: Test accuracy (%) from the ResNet-34 experiments.

	AMoC	**AMoC-N**	**Heavy Ball**	**Nesterov**	**Adam**
CIFAR10	**89.37**	88.23	88.8	88.75	86.48
CIFAR100	68.05	**68.26**	64.04	64.04	63.58

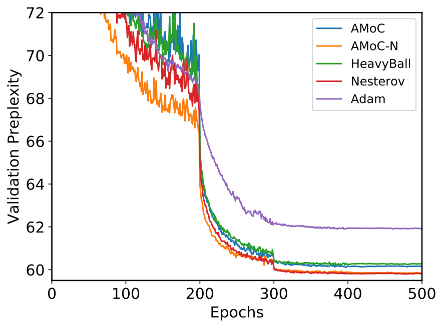
and AMoC-N outperform other methods by a noticeable margin, while the baselines perform similarly. Our algorithms also reach the highest accuracy on the test set with AMoC-N slightly higher.

6.4 LSTMs

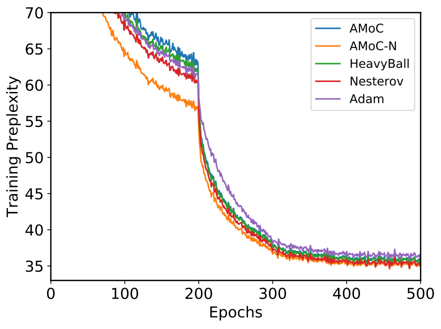
We also experimented with LSTM word-level language models on the Penn Treebank dataset following the experimental setting of [14]. The LSTM model used has 3 layers each with 1150 hidden nodes with an embedding size of 400. Dropout is used with the probability of 0.1 on the embedding layer, 0.65 on the input embedding layer and 0.3 in the hidden layers. A weight decay of 1.2e-6 is used along with weight drop with a probability of 0.5 and temporal activation regularization with scaling of 1. L2 regularization is also used on the activations with a scale of 2.

We used a batch size of 80 and searched over {20, 15, 10, 5, 1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001} for the learning rate. We decay the learning rate by 0.1 at epochs 200, 300 and 400 while training for 500 epochs. β_1, β_2 and μ were set similar to the previous experiments. β in our algorithms was set to 0.2.

The results are shown in Figure 4 and Table 4. All methods reach similar training perplexity with AMoC-N and Nesterov reaching lower validation



(a) Validation Perplexity



(b) Training Perplexity

Fig. 4: Training and Validation perplexity from the LSTM experiments on the Penn Treebank dataset. The vertical axes show the perplexity of the language model and the horizontal axes show the number of epochs.

Table 4: Test perplexity from the LSTM Penn Treebank experiment.

AMoC	**AMoC-N**	**Heavy Ball**	**Nesterov**	**Adam**
58.16	**57.61**	57.91	57.80	59.47

perplexity followed by AMoC and Heavy Ball. AMoC-N achieves the lowest perplexity on the test set.

7 Conclusions

We proposed a novel and efficient momentum-based algorithm, AMoC, by introducing an adaptive coefficient for the momentum term. We compared AMoC against SGD with Nesterov's momentum, regular momentum (Heavy Ball), Adam and a recently proposed second-order method, K-FAC, on both convex functions and non-convex neural network objectives including Autoencoders, CNNs and RNNs. We saw that AMoC is able to perform well in various settings compared to popular first order and second-order methods. AMoC's implementation is straightforward and is computationally efficient. We also analyzed AMoC's convergence properties and proposed a convergence rate similar to that of the Heavy Ball algorithm. We believe that AMoC offers a new and promising direction in convex and non-convex optimization research and in particular, neural network optimization.

A Proofs

Lemma. *For arbitrarily large integer T, there exists $\beta > 0$ such that the first $T+1$ elements of the sequence $\{\lambda_k\}$ are positive.*

Proof. Consider λ_1 to λ_k are positive and λ_{k+1} is negative. We have:

$$\begin{aligned}\lambda_{j+1} - \frac{\mu(1+\beta)}{1-\mu(1+\beta)} &= \frac{\lambda_j}{\gamma_j} - \frac{1}{1-\mu(1+\beta)} \\ &\geq \frac{\lambda_j}{\mu(1+\beta)} - \frac{1}{1-\mu(1+\beta)} \\ &= \frac{\lambda_j - \frac{\mu(1+\beta)}{1-\mu(1+\beta)}}{\mu(1+\beta)}\end{aligned}$$

Combining all the inequalities for $j = 1, ..., k$ results:

$$\begin{aligned}-\frac{\mu(1+\beta)}{1-\mu(1+\beta)} &\geq \lambda_{k+1} - \frac{\mu(1+\beta)}{1-\mu(1+\beta)} \\ &\geq \frac{\lambda_1 - \frac{\mu(1+\beta)}{1-\mu(1+\beta)}}{(\mu(1+\beta))^k} \\ &= \frac{-2\mu\beta}{(1-\mu)^2 - \mu^2\beta^2} / (\mu(1+\beta))^k\end{aligned}$$

therefore:

$$k \geq ln\Big(\frac{\mu(1+\beta)(1-\mu(1-\beta))}{2\mu\beta}\Big) / ln\big(\mu(1+\beta)\big)$$

The right-hand side of this inequality approaches $+\infty$ as β approaches 0. Thus, by choosing a small enough β, we achieve arbitrarily large number of positive terms.

Theorem. *For any differentiable convex function f with L-Lipschitz gradients, the sequence generated by AMoC with sufficiently small β, $\mu \in [0, \frac{1}{1+\beta})$, and $\epsilon \in (0, \frac{\mu(1+\beta)}{L\lambda_1})$ satisfies the following:*

$$f(\tilde{\theta}_T) - f(\theta^*) \leq \frac{\|\theta_1 - \theta^*\|^2}{2T(1+\lambda_{T+1})}\Big(\frac{1}{\epsilon} + \frac{\lambda_1^2 L}{\mu}\Big) \tag{13}$$

where θ^* is the optimal point and $\tilde{\theta}_T = (\sum_{k=1}^T \frac{\lambda_k}{\gamma_k}\theta_k)/(\sum_{k=1}^T \frac{\lambda_k}{\gamma_k})$.

Proof. To prove this theorem, we follow a similar approach as [6]. By definition we have:

$$\begin{aligned}d_k &= \theta_k - \theta_{k-1} \\ g_k &= \nabla f(\theta_k) \\ \gamma_k &= \mu\left(1 - \beta\bar{g}_k \cdot \bar{d}_k\right) \\ \theta_{k+1} &= \theta_k - \epsilon g_k + \gamma_k d_k\end{aligned}$$

therefore:

$$\begin{aligned}\theta_{k+1} + \lambda_{k+1} d_{k+1} &= (\lambda_{k+1} + 1)\,\theta_{k+1} - \lambda_{k+1}\theta_k \\ &= \theta_k - \frac{\epsilon\lambda_k}{\gamma_k} g_k + \lambda_k d_k\end{aligned}$$

By subtracting θ^* and seting $\delta_k = \theta_k - \theta^*$, we get:

$$\begin{aligned}\|\delta_{k+1} + \lambda_{k+1} d_{k+1}\|^2 &= \|\delta_k + \lambda_k d_k\|^2 + \left(\frac{\epsilon\lambda_k}{\gamma_k}\right)^2 \|g_k\|^2 \\ &\quad - \frac{2\alpha\lambda_k}{\gamma_k}\delta_k \cdot g_k - \frac{2\epsilon\lambda_k^2}{\gamma_k} g_k \cdot d_k\end{aligned} \tag{14}$$

According to [15, Theorem 2.1.5], $\delta_k \cdot g_k \geq f(\theta_k) - f(\theta^*) + \frac{1}{2L}\|g_k\|^2$ which combined with (14) results:

$$\begin{aligned}\|\delta_{k+1} + \lambda_{k+1} d_{k+1}\|^2 &\leq \|\delta_k + \lambda_k d_k\|^2 + \left(\frac{\epsilon\lambda_k}{\gamma_k}\right)^2 \|g_k\|^2 \\ &\quad - \frac{2\epsilon\lambda_k}{\gamma_k}\left(f(\theta_k) - f(\theta^*) + \frac{1}{2L}\|g_k\|^2\right) \\ &\quad - \frac{2\epsilon\lambda_k^2}{\gamma_k} g_k \cdot d_k \\ &\leq \|\delta_k + \lambda_k d_k\|^2 - \frac{2\epsilon\lambda_k}{\gamma_k}(f(\theta_k) - f(\theta^*)) \\ &\quad - \frac{2\epsilon\lambda_k^2}{\gamma_k} g_k \cdot d_k\end{aligned}$$

Summing up all the inequalities for $k = 1, ..., T$ yields:

$$\begin{aligned}0 \leq \|\delta_{T+1} + \lambda_{T+1} d_{T+1}\|^2 &\leq \|\delta_1\|^2 - 2\epsilon \sum_{k=1}^{T} \frac{\lambda_k}{\gamma_k}\left(f(\theta_k) - f(\theta^*)\right) \\ &\quad - 2\epsilon \sum_{k=1}^{T} \frac{\lambda_k^2}{\gamma_k} g_k \cdot d_k \\ &\leq \|\delta_1\|^2 - 2\epsilon(\sum_{k=1}^{T} \frac{\lambda_k}{\gamma_k})\left(f(\tilde{\theta}_k) - f(\theta^*)\right) \\ &\quad - \frac{2\epsilon}{\mu}\sum_{k=1}^{T} \lambda_k^2 \|g_k\|\,\|d_k\| \frac{\bar{g}_k \cdot \bar{d}_k}{\left(1 - \beta\bar{g}_k \cdot \bar{d}_k\right)}\end{aligned}$$

For $\beta < 1$, function $\frac{x}{1-\beta x}$ is convex for $x \in [-1, 1]$, thus:

$$\begin{aligned}0 \leq \|\delta_1\|^2 &- 2\epsilon(\sum_{k=1}^{T} \frac{\lambda_k}{\gamma_k})\left(f(\tilde{\theta}_k) - f(\theta^*)\right) \\ &- \frac{2\epsilon}{\mu}\frac{\sum_{k=1}^{T} \lambda_k^2 g_k \cdot d_k}{1 - \beta\frac{\sum_{k=1}^{T}\lambda_k^2 g_k \cdot d_k}{\sum_{k=1}^{T}\lambda_k^2\|g_k\|\|d_k\|}}\end{aligned}$$

Function $\frac{x}{1-\beta x}$ is also increasing, and $g_k \cdot d_k \geq f(\theta_k) - f(\theta_{k-1})$ [15, Theorem 2.1.5], therefore:

$$2\epsilon(\sum_{k=1}^{T} 1 + \lambda_{k+1}) \left(f(\tilde{\theta}_k) - f(\theta^*)\right) \leq \|\delta_1\|^2 - \frac{2\epsilon}{\mu} \frac{\sum_{k=2}^{T} \lambda_k^2 \left(f(\theta_k) - f(\theta_{k-1})\right)}{1 - \beta \frac{\sum_{k=2}^{T} \lambda_k^2 (f(\theta_k) - f(\theta_{k-1}))}{\sum_{k=1}^{T} \lambda_k^2 \|g_k\| \|d_k\|}}$$

Furthermore, easily one can show that sequence $\{\lambda_k\}$ is decreasing, and

$$\sum_{k=2}^{T} \lambda_k^2 \left(f(\theta_k) - f(\theta_{k-1})\right) \geq -\lambda_1^2 \left(f(\theta_1) - f(\theta^*)\right)$$

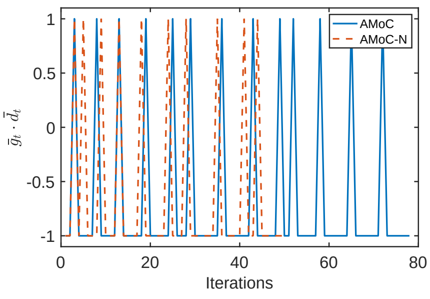
Therefore:

$$\begin{aligned} 2\epsilon T(1 + \lambda_{k+1}) \left(f(\tilde{\theta}_k) - f(\theta^*)\right) &\leq \|\delta_1\|^2 + \frac{2\epsilon}{\mu} \frac{\lambda_1^2 \left(f(\theta_1) - f(\theta^*)\right)}{1 + \beta \frac{\lambda_1^2 (f(\theta_1) - f(\theta^*))}{\sum_{k=1}^{T} \lambda_k^2 \|g_k\| \|d_k\|}} \\ &\leq \|\delta_1\|^2 + \frac{2\epsilon}{\mu} \lambda_1^2 \left(f(\theta_1) - f(\theta^*)\right) \\ &\leq \|\delta_1\|^2 (1 + \frac{\epsilon}{\mu} \lambda_1^2 L) \end{aligned}$$

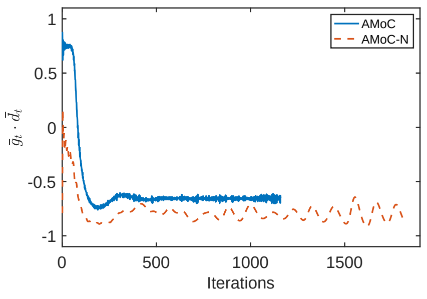
where the final inequality follows from $f(\theta_1) - f(\theta^*) \leq \frac{L}{2} \|\delta_1\|^2$ [15, Theorem 2.1.5], and concludes the proof.

B Inner Product Analysis

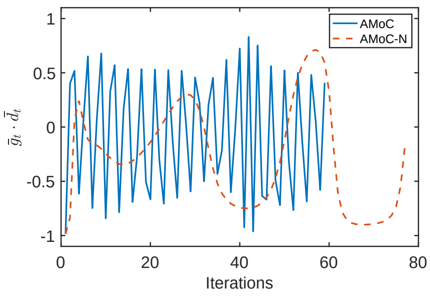
In this section we report the value of the inner product $(\bar{g}_t \cdot \bar{d}_t)$ for AMoC and AMoC-N per iteration/epoch for each of the experiments in Figures 5 and 6. The results from the Anisotropic Bowl and the Smooth-BPDN experiments are particularly interesting. In the first case (Figure 5a), with the inner product oscillating between positive and negative values, we can infer that the algorithm is crossing the optimum multiple times (without overshooting) but is able to bounce back and reach the optimum point eventually and in less iterations than the baselines. The second case (Figure 5c) behaves in a similar way, except the algorithm seems to be moving close to the optimum but going up again and bouncing back several times (most likely in an oval-shaped trajectory) until it gets close enough that it terminates. In the Ridge Regression problem (Figure 5b), the inner product drops from values between 0.5 and 1 to values between -0.5 to -1, indicating that the algorithm is moving towards the optimum steadily, with the updates always keeping a low angle with the gradient. In the neural network experiments (Figures 6a to 6d), the inner product gets closer to 0 by the end of training. We link this behaviour to the algorithm moving in a circular fashion around the optima with the direction of the negative gradient almost perpendicular ($\phi = \pi/2$) to the previous update.



(a) Anisotropic Bowl



(b) Ridge Regression



(c) Smooth-BPDN

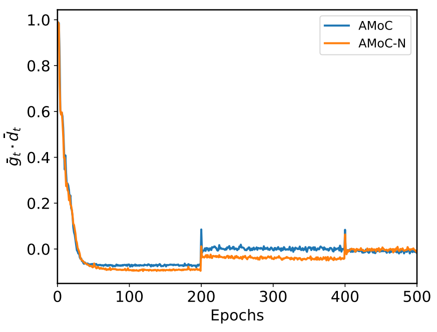
Fig. 5: Inner product, $\bar{g}_t \cdot \bar{d}_t = \cos(\pi - \phi_t)$, during optimization for both AMoC and AMoC-N for convex experiments in the paper.

Acknowledgements

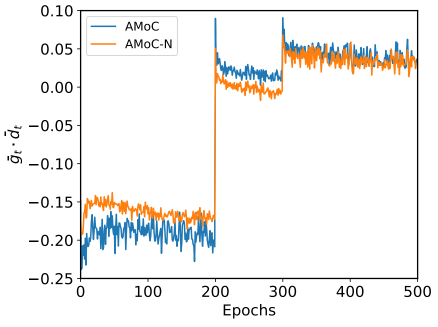
We would like to thank Ruth Urner for valuable discussions and insights. We would also like to thank the anonymous reviewers for their useful suggestions and constructive feedback.

References

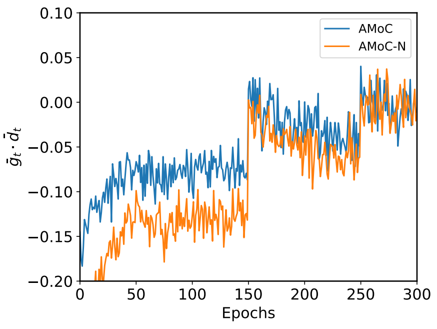
1. Amari, S.I.: Natural gradient works efficiently in learning. Neural computation **10**(2), 251–276 (1998)
2. Aujol, J.F., Rondepierre, A., Aujol, J., Dossal, C., et al.: Optimal convergence rates for nesterov acceleration. arXiv preprint arXiv:1805.05719 (2018)
3. Bottou, L., Curtis, F.E., Nocedal, J.: Optimization methods for large-scale machine learning. Siam Review **60**(2), 223–311 (2018)
4. Defazio, A.: On the curved geometry of accelerated optimization. arXiv preprint arXiv:1812.04634 (2018)



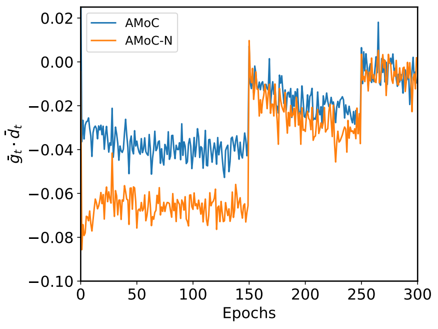
(a) MNIST Autoencoder



(b) LSTM Penn Treebank



(c) ResNet-34 CIFAR 10



(d) ResNet-34 CIFAR 100

Fig. 6: Inner product, $\bar{g}_t \cdot \bar{d}_t = \cos(\pi - \phi_t)$, during training for both AMoC and AMoC-N for NN experiments in the paper.

5. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research **12**(Jul), 2121–2159 (2011)
6. Ghadimi, E., Feyzmahdavian, H.R., Johansson, M.: Global convergence of the heavy-ball method for convex optimization. In: 2015 European Control Conference (ECC). pp. 310–315. IEEE (2015)
7. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
8. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. science **313**(5786), 504–507 (2006)
9. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
10. Lucas, J., Sun, S., Zemel, R., Grosse, R.: Aggregated momentum: Stability through passive damping. arXiv preprint arXiv:1804.00325 (2018)
11. Martens, J.: Deep learning via hessian-free optimization. In: ICML. vol. 27, pp. 735–742 (2010)

12. Martens, J., Grosse, R.: Optimizing neural networks with kronecker-factored approximate curvature. In: International conference on machine learning. pp. 2408–2417 (2015)
13. Meng, X., Chen, H.: Accelerating nesterov's method for strongly convex functions with lipschitz gradient. arXiv preprint arXiv:1109.6058 (2011)
14. Merity, S., Keskar, N.S., Socher, R.: Regularizing and optimizing lstm language models. arXiv preprint arXiv:1708.02182 (2017)
15. Nesterov, Y.: Introductory lectures on convex optimization: A basic course, vol. 87. Springer Science & Business Media (2013)
16. Nesterov, Y.E.: A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In: Dokl. akad. nauk Sssr. vol. 269, pp. 543–547 (1983)
17. O'donoghue, B., Candes, E.: Adaptive restart for accelerated gradient schemes. Foundations of computational mathematics **15**(3), 715–732 (2015)
18. Polyak, B.T.: Some methods of speeding up the convergence of iteration methods. USSR Computational Mathematics and Mathematical Physics **4**(5), 1–17 (1964)
19. Reddi, S.J., Kale, S., Kumar, S.: On the convergence of adam and beyond (2018)
20. Su, W., Boyd, S., Candes, E.: A differential equation for modeling nesterov's accelerated gradient method: Theory and insights. In: Advances in Neural Information Processing Systems. pp. 2510–2518 (2014)
21. Sutskever, I., Martens, J., Dahl, G.E., Hinton, G.E.: On the importance of initialization and momentum in deep learning. ICML (3) **28**(1139-1147), 5 (2013)
22. Tieleman, T., Hinton, G.: Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning **4**(2), 26–31 (2012)
23. Wibisono, A., Wilson, A.C., Jordan, M.I.: A variational perspective on accelerated methods in optimization. proceedings of the National Academy of Sciences **113**(47), E7351–E7358 (2016)
24. Zeiler, M.D.: Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701 (2012)