
Review: Momentum-Based Variance Reduction in Non-Convex SGD

Alan Nawoj
ECE 273 Final Project
Spring 2026
University of California San Diego
anawoj@gmail.com

Abstract

This report analyzes the landmark conference paper “Momentum-Based Variance Reduction in Non-Convex SGD” by Cutkosky and Orabona [1] and the innovative STORM algorithm that it unveils. After discussing the motivation of this work, the fundamentals of STORM, its core underlying algorithmic design principles, and a summary of the theoretical analysis on its convergence bounds, this report also provides empirical evidence to demonstrate a faster convergence rate on a proposed enhancement to the original STORM algorithm using a widely-adopted non-convex objective function benchmark.

1 Introduction and Motivation

Stochastic Gradient Descent (SGD) has become a cornerstone algorithm in solving modern optimization problems, especially in the field of deep learning where objective functions are typically non-convex, datasets are massive in size, and algorithmic performance is often a critical bottleneck. Instead of computing a full gradient $\nabla F(\mathbf{x}_t)$ at every iteration, like standard gradient descent, SGD simply picks one random sample (or a mini batch of samples) and computes the gradient of this subset of all inputs per iteration (called a stochastic gradient, $\nabla f(\mathbf{x}_t, \xi_t)$) in order to update the optimization variable. Since the gradient of a single sample (or mini batch) is an unbiased estimator of the true gradient for randomly selected samples, $\mathbb{E}[\nabla f(\mathbf{x}_t, \xi_t)] = \nabla F(\mathbf{x}_t)$, SGD ultimately makes progress toward a critical point where $\nabla F(\mathbf{x}_*) = 0$. However, there are trade-offs with this approach in terms of performance and accuracy which have led to numerous subsequent SGD-based algorithms that aim to improve upon its shortcomings, including the novel STOchastic Recursive Momemtum (STORM) algorithm discussed in this review.

1.1 Limitations of Standard SGD

While there are several benefits of using SGD over standard gradient descent, SGD is unfortunately plagued with a persistent variance in its stochastic gradients that makes convergence to a stationary point challenging unless the step size is progressively decreased, which further degrades performance. This is due to the fact that a stochastic gradient is “noisy” and not necessarily aligned with the true gradient, thus often causing the algorithm to endlessly bounce around in the vicinity of a stationary point within a region that is bounded by the variance of all possible stochastic gradients and the step size. As a result, the optimal convergence rate of SGD for convex functions over T iterations is $\mathcal{O}(1/T^{1/2})$ and $\mathcal{O}(1/T)$ for strongly convex functions, where the optimality gap, $F(\mathbf{x}_t) - \inf_x F(\mathbf{x})$, is used as the target metric. Meanwhile, the optimal convergence rate of SGD for non-convex functions is $\mathcal{O}(1/T^{1/4})$, where the expected value of the gradient norm $\mathbb{E}[\|\nabla F(\mathbf{x}_t)\|]$ is used as the target metric.

1.2 Variance Reduction Techniques

Given that stochastic gradient variance is an inherent limitation on the performance of standard SGD, several algorithms, such as SVRG [2], SARAH [3], and SPIDER [4] have been developed with the specific goal of reducing this variance. Each of these stochastic variance-reduced gradient algorithms has improved the convergence rate for non-convex objective functions to critical points down to an optimal $\mathcal{O}(1/T^{1/3})$ bound. However, despite their theoretical appeal, these traditional variance-reduction methods generally underperform on practical deep learning problems [5] largely due to the fact that they must periodically compute excessively large, static, low-noise "checkpoint" (stage) gradients. This architectural design, in turn, introduces new tightly coupled hyperparameters that must be carefully tuned, including the batch size for the checkpoint gradients, checkpoint frequency, and the non-adaptive learning rate.

1.3 Goals and Benefits of STORM

The STOchastic Recursive Momemtum (STORM) algorithm mitigates many of the shortcomings of these traditional variance reduction algorithms while also achieving the same optimal performance of $\mathcal{O}(1/T^{1/3})$. Specifically, STORM claims a convergence rate of $\mathbb{E}[\|\nabla F(x_t)\|] = \mathcal{O}(1/T^{1/2} + \sigma^{1/3}/T^{1/3})$ over T iterations and σ^2 variance in the gradients. The authors of STORM avoid the need for computationally expensive periodic checkpoint gradients and instead achieve variance reduction through recursively correcting gradient estimates using a momentum-like recursion with a learning rate that adapts at each iteration. At a high level, momentum is a technique rooted in physics for accelerating progress in gradient-based optimization algorithms, but in the case of STORM, momentum is uniquely being used as a variance control mechanism and the algorithm also avoids the need for extensive hyperparameter tuning.

1.4 Key Assumptions

In addition to the assumption that the objective function $f(\cdot)$ is non-convex, this paper also assumes that $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is a differentiable function that is both L -smooth and G -Lipschitz and that there is some upper bound σ^2 on the noise of its stochastic gradients. The condition for L -smoothness implies L -Lipschitz gradients where $\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|$ for all $\mathbf{x}$ and $\mathbf{y}$. Likewise, a function is G -Lipschitz if it satisfies the condition $\|\nabla f(\mathbf{x})\| \leq G$ for all $\mathbf{x}$, meaning that G is an upper bound on the magnitude of its gradients. For context, the constants L and G both appear in the equations of the STORM algorithm constants k , w , and c discussed in the next section.

2 Description of Algorithm

In order to appreciate the uniqueness of STORM, it is helpful to first outline the architecture of SGD with momentum since STORM builds directly upon this concept. When we talk about SGD with momentum, the typical implementation involves updating two variables per iteration: the *optimization vector* (e.g. $\mathbf{x}_t$) and the *momentum vector* (e.g. $\mathbf{d}_t$). The momentum vector is typically formed as an exponential moving average (EMA) of past stochastic gradients such that the current value of $\mathbf{d}_t$ is equal to a convex combination of the previous value of the momentum vector and the current stochastic gradient. The two variables are updated using the following recursive formulas, where a is a fixed constant such that $a \in (0, 1)$ and η is the fixed learning rate:

$$\mathbf{d}_t = (1 - a)\mathbf{d}_{t-1} + a\nabla f(\mathbf{x}_t, \xi_t) \quad (1)$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta\mathbf{d}_t \quad (2)$$

While this gradual dampening of historical gradients via an EMA may look appealing, the authors note that this strategy does *not* actually reduce variance since the noise from each stochastic gradient accumulates in the momentum vector over time.

2.1 Core STORM Update Rule

The central innovation in STORM is the addition of a noise correction term, $(1 - a_t)(\nabla f(\mathbf{x}_t, \xi_t) - \nabla f(\mathbf{x}_{t-1}, \xi_t))$, to the momentum update as well as the inclusion of an adaptive learning rate, η_t , and

an adaptive momentum coefficient, a_t , which get updated on each iteration t . The revised update formulas for $\mathbf{x}_t$ and $\mathbf{d}_t$ are as follows:

$$\mathbf{d}_t = (1 - a_t)\mathbf{d}_{t-1} + a_t \nabla f(\mathbf{x}_t, \xi_t) + (1 - a_t)(\nabla f(\mathbf{x}_t, \xi_t) - \nabla f(\mathbf{x}_{t-1}, \xi_t)) \quad (3)$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta_t \mathbf{d}_t \quad (4)$$

STORM also introduces three input parameter constants (k , w , and c) to the algorithm which are included in the adaptive learning rate η_t and momentum coefficient a_t update formulas as follows.

$$\eta_t = \frac{k}{\left(w + \sum_{i=1}^t \|\nabla f(\mathbf{x}_i, \xi_i)\|^2\right)^{1/3}} \quad (5)$$

$$a_{t+1} = c\eta_t^2 \quad (6)$$

2.2 Key Steps of STORM Algorithm

The following steps outline the operational flow of the STORM algorithm and explain what is happening at each step in basic terms.

1. **Initialization:** Choose an initial starting point, $\mathbf{x}_1$, as well as values for the STORM-specific constants (k , w , and c).
 - **Constant k (step-size scaling factor):** This constant appears in the adaptive step size formula to establish the baseline step size of the algorithm. The optimal value of k as stated by the authors is $bG^{2/3}/L$ (for any $b > 0$), but computing the exact values of L and G is often nearly impossible for a deep learning model's objective function, for instance, so k is typically chosen in the range of 0.1 to 1.0 in practice.
 - **Constant w (step-size regularizer):** Without w , the denominator of the step-size would be zero on the initial iteration which would result in a divide by zero error, so this constant acts as a regularizer to smooth out the step sizes for initial iterations of STORM. While the paper presents a value of w based on L and G , a practical choice for this constant is 1.0.
 - **Constant c (momentum scaling factor):** Since the momentum coefficient a_t varies proportional to the square of the step size η_t and the step size will get progressively smaller as stochastic gradients accumulate in its denominator, the constant c helps to prevent the momentum coefficient from collapsing to zero too quickly. While the optimal value of c is based on L and G , a practical choice for this constant is in the range of 1.0 to 10.0.
2. **Iterative Update Steps:** For each time step t from 1 to T :
 1. **Adaptive Step Size:** The step size parameter η_t is inversely proportional to the sum of all historical stochastic gradient magnitudes seen thus far and is computed at the start of this loop as $\eta_t = k/(w + \sum_{i=1}^t \|\nabla f(\mathbf{x}_i, \xi_i)\|^2)^{1/3}$ for the current step t . This naturally leads to smaller step sizes as the algorithm progresses.
 2. **Optimization Variable:** After computing η_t , we can now update the optimization variable using the formula $\mathbf{x}_{t+1} = \mathbf{x}_t - \eta_t \mathbf{d}_t$ where the $\mathbf{d}_t$ momentum vector is the recursively updated proxy to the true gradient.
 3. **Momentum Parameter:** With $\mathbf{x}$ updated, we now prepare the intermediate variables needed for the *subsequent* update of $\mathbf{x}$ which includes the momentum parameter a_t . The update formula is $a_{t+1} = c\eta_t^2$, so it uses the constant c and the squared value of the current learning rate η_t . While the value of η_t is unrestricted, the momentum parameter must fall within the range $(0, 1)$ and so c must be tuned so that all a_t stay within this valid range.
 4. **Next Sample:** Choose the next random sample ξ_{t+1} from the complete set of inputs.
 5. **Norm of Stochastic Gradient:** Using the new sample ξ_{t+1} , compute the norm of the gradient of $f(\cdot)$ on this sample as $\|\nabla f(\mathbf{x}_{t+1}, \xi_{t+1})\|$ using the current weights $\mathbf{x}_{t+1}$ and store this value since it will be used in the denominator of η_t on the next iteration.
 6. **Momentum Vector:** Finally, we update the momentum vector $\mathbf{d}_t$ using the unique recursive gradient noise correction term introduced by STORM. Through algebraic manipulation, Equation (3) above can be simplified to the following: $\mathbf{d}_{t+1} = \nabla f(\mathbf{x}_{t+1}, \xi_{t+1}) + (1 -$

$a_{t+1})(\mathbf{d}_t - \nabla f(\mathbf{x}_t, \xi_{t+1}))$. This is effectively the exponential moving average of past stochastic gradients *plus* the additional gradient noise "correction" term introduced by the authors.

3. **Return $\hat{\mathbf{x}}$:** The paper states that $\hat{\mathbf{x}}$ is chosen uniformly at random from all $\mathbf{x}_1, \dots, \mathbf{x}_T$ but in practice $\hat{\mathbf{x}}$ is simply set to $\mathbf{x}_T$ (i.e. the final iterate). The reason for the random sampling strategy is that the final iterate for non-convex optimization problems is not guaranteed to be a stationary point. The convergence metric for non-convex optimization problems is simply that the expected gradient norm is decreasing over time down to some ϵ gap.

3 Underlying Algorithmic Design Principles

The architecture of STORM builds upon SGD with momentum by incorporating a variance-reduction technique into the continuous updates of the momentum vector along with adaptive parameters that enable the algorithm to efficiently navigate a non-convex landscape. Notably, STORM achieves the optimal $\mathcal{O}(1/T^{1/3})$ convergence rate for this class of optimization problems without knowledge of the true gradient at any time step or the variance of stochastic gradients. The underlying algorithm design principles that enable this outcome are detailed as follows.

3.1 Recursive Variance Reduction and No Checkpoint Gradients

STORM cleverly integrates stochastic gradient variance reduction continuously along the sequence of iterates by adding the term $(1 - a_t)(\nabla f(\mathbf{x}_t, \xi_t) - \nabla f(\mathbf{x}_{t-1}, \xi_t))$ to the typical EMA momentum vector update formula. Notice that this difference involves the stochastic gradients of the current and previous points but with the *same* sample ξ_t . If we decompose the stochastic gradient at any given point $\mathbf{x}$ as $\nabla f(\mathbf{x}, \xi) = \nabla F(\mathbf{x}) + \epsilon(\mathbf{x}, \xi)$ where $\epsilon(\mathbf{x}, \xi)$ represents the noise or error between the stochastic gradient and the true gradient, then we can rewrite the difference $(\nabla f(\mathbf{x}_t, \xi_t) - \nabla f(\mathbf{x}_{t-1}, \xi_t))$ as $(\nabla F(\mathbf{x}_t) - \nabla F(\mathbf{x}_{t-1})) + (\epsilon(\mathbf{x}_t, \xi_t) - \epsilon(\mathbf{x}_{t-1}, \xi_t))$. Since the stochastic gradients are computed using the *same* sample ξ_t , the noise between the two is therefore highly correlated and so the second term $(\epsilon(\mathbf{x}_t, \xi_t) - \epsilon(\mathbf{x}_{t-1}, \xi_t)) \approx 0$, which leaves us with a close approximation to the change in the true gradient between $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$. By adding a scaled version of this vector that roughly tracks the change in the true gradients to a scaled version of the newly sampled stochastic gradient on every step, we are effectively better aligning the momentum vector $\mathbf{d}_t$ with the true gradient path and therefore reducing the variance of the path that is being followed.

Another way of looking at this is by considering the formula for the variance of the difference between two random variables, X and Y :

$$\text{Var}(X - Y) = \text{Var}(X) + \text{Var}(Y) - 2 \text{Cov}(X, Y)$$

If X and Y are highly correlated, like $\nabla f(\mathbf{x}_t, \xi_t)$ and $\nabla f(\mathbf{x}_{t-1}, \xi_t)$, because they are computed from the *same* random sample, then $\text{Cov}(X, Y) \approx \text{Var}(X) \approx \text{Var}(Y)$ which means that $\text{Var}(X - Y) \approx 0$. Thus, the difference between $\nabla f(\mathbf{x}_t, \xi_t)$ and $\nabla f(\mathbf{x}_{t-1}, \xi_t)$ yields a low variance vector that closely tracks the change in true gradients between consecutive iterates, $(\nabla F(\mathbf{x}_t) - \nabla F(\mathbf{x}_{t-1}))$. Furthermore, since $f(\cdot)$ is characterized by L -Lipschitz gradients, $\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|$, we know that $\|\nabla f(\mathbf{x}_t, \xi_t) - \nabla f(\mathbf{x}_{t-1}, \xi_t)\| \leq L\|\mathbf{x}_t - \mathbf{x}_{t-1}\|$. If we then square both sides and take the expectation of both sides of this inequality, we get the following:

$$\text{Var}(\nabla f(\mathbf{x}_t, \xi_t) - \nabla f(\mathbf{x}_{t-1}, \xi_t)) \leq \mathbb{E}[\|\nabla f(\mathbf{x}_t, \xi_t) - \nabla f(\mathbf{x}_{t-1}, \xi_t)\|^2] \leq L^2\|\mathbf{x}_t - \mathbf{x}_{t-1}\|^2$$

Therefore, we can conclude that the variance of the difference between these two stochastic gradients taken on the same sample ξ_t is upper bounded by L^2 times the squared magnitude of the distance between the two points. Since this distance ultimately decreases over time, the variance will also decrease automatically over time. Notably, STORM was intentionally designed to avoid periodic massive and computationally intensive checkpoint gradients to reduce stochastic gradient variance which, in turn, removes the complexity of tuning hyperparameters such as checkpoint gradient batch size and frequency of checkpoints.

3.2 Adaptive Learning Rate and Momentum Coefficient

STORM differs from SVRG in that it utilizes an adaptive learning rate η_t , and it differs from standard SGD with momentum by utilizing an adaptive momentum coefficient a_t . The algorithm was further designed in a way that tightly couples these two parameters so that the path traveled adapts to the geometry of the objective function landscape without knowledge of the stochastic gradient noise variance σ^2 . The learning rate equation (5) shows that the denominator of η_t monotonically increases as $t \rightarrow \infty$ since the $\|\nabla f(\mathbf{x}_i, \xi_i)\|^2$ positive values accumulate with each iteration which, in turn, causes η_t itself to monotonically *decrease* as $t \rightarrow \infty$. As a result, if the algorithm encounters any high gradient regions, the step size will automatically be scaled down to mitigate overshooting problems. Moreover, equation (6) shows how the momentum coefficient will automatically be scaled down as well as $t \rightarrow \infty$ since it is proportional to η_t^2 . The observation here when reviewing equation (3) is that current stochastic gradients $\nabla f(\mathbf{x}_t, \xi_t)$ get weighted less and less as $t \rightarrow \infty$ and the historical gradient variance reduction terms get weighted more heavily over time. This architecture reveals that STORM places more weight on the initial stochastic gradients during its early iterations to establish a general direction, but then it shifts its focus more to the recursive stochastic gradient variance reduction terms which ultimately enables the algorithm to converge.

4 Summary of Theoretical Analysis

Having outlined the architecture and key design principles of the STORM algorithm, the goal of the theoretical analysis in the paper is to prove that it achieves the optimal convergence rate of $\mathbb{E}[\|\nabla F(\mathbf{x}_t)\|] = \mathcal{O}(1/T^{1/2} + \sigma^{1/3}/T^{1/3})$ over T iterations and σ^2 variance in the gradients.

4.1 Lyapunov Function

The Lyapunov function Φ_t defined by the STORM authors is the starting point of the convergence rate proof since it measures the progress of the algorithm. In simple convex optimization problems, the optimality gap $F(\mathbf{x}_t) - F(\mathbf{x}^*)$ serves as the Lyapunov function since it monotonically decreases down to zero and always stays non-negative. However, in stochastic non-convex optimization problems, the optimality gap does *not* always decrease on every iteration due to gradient noise induced in the pathway and so a second term to account for "pathway noise" $\hat{\Phi}_t$ is commonly added to the Lyapunov function, $\Phi_t = [F(\mathbf{x}_t) - F(\mathbf{x}^*)] + \hat{\Phi}_t$. For STORM, the Lyapunov function is $\Phi_t = F(\mathbf{x}_t) + z_t \|\epsilon_t\|^2$ where $\|\epsilon_t\|^2 = \|\mathbf{d}_t - \nabla F(\mathbf{x}_t)\|^2$ represents the variance of the momentum-based gradient estimate and z_t is a coefficient that is time-varying and inversely proportional to the learning rate η_{t-1} . This implies that as the algorithm progresses and the learning rate decreases, the precision of the gradient estimator $\mathbf{d}_t$ becomes increasingly more important. To determine the tightest bound on this Lyapunov function, we next determine the tightest bound for each of the two components of this function.

4.2 Bound on Objective Function Value

Lemma 1 in the paper uses the first-order definition of L-smoothness, the definition of $\mathbf{x}_{t+1} - \mathbf{x}_t = -\eta_t \mathbf{d}_t$ in the algorithm, the tracking error definition of $\mathbf{d}_t = \nabla F(\mathbf{x}_t) + \epsilon_t$, Young's inequality $\langle \mathbf{x}, \mathbf{y} \rangle \leq \frac{1}{2}\|\mathbf{x}\|^2 + \frac{1}{2}\|\mathbf{y}\|^2$ and $\|\mathbf{x} + \mathbf{y}\|^2 \leq 2\|\mathbf{x}\|^2 + 2\|\mathbf{y}\|^2$ along with algebraic simplification and taking the expectation of both sides to arrive at its tightest bound on the objective function value with a step size $\eta_t \leq \frac{1}{4L}$ for all t as follows:

$$\mathbb{E}[F(\mathbf{x}_{t+1}) - F(\mathbf{x}_t)] \leq \mathbb{E}[-\eta_t/4 \|\nabla F(\mathbf{x}_t)\|^2 + 3\eta_t/4 \|\epsilon_t\|^2] \quad (7)$$

4.3 Bound on Variance of Gradient Estimator Error

Lemma 2 in the paper details how the bound on the second term of the Lyapunov function (i.e. $\mathbb{E}[\|\epsilon_t\|^2/\eta_{t-1}]$) is computed, and it starts by replacing the ϵ_t in this expectation with the definition of $\epsilon_t = \mathbf{d}_t - \nabla F(\mathbf{x}_t)$ and then further expanding this out using the STORM definition of $\mathbf{d}_t = \nabla f(\mathbf{x}_t, \xi_t) + (1 - a_t)(\mathbf{d}_{t-1} - \nabla f(\mathbf{x}_{t-1}, \xi_t))$. From here, a series of algebraic manipulations are performed that utilize, for instance, norm expansion, the definition of L-Lipschitz gradients, and Young's inequality to arrive at the final bound which highlights a key relationship in that the expected tracking error variance actually contracts over time (i.e. $\mathbb{E}[\|\epsilon_t\|^2] \leq \mathbb{E}[\|\epsilon_{t-1}\|^2]$):

$$\mathbb{E}\left[\frac{\|\epsilon_t\|^2}{\eta_{t-1}}\right] \leq \mathbb{E}\left[2c^2\eta_{t-1}^3\sigma^2 + (1-a_t)^2(1+4L^2\eta_{t-1}^2)\frac{\|\epsilon_{t-1}\|^2}{\eta_{t-1}} + 4(1-a_t)^2L^2\eta_{t-1}\|\nabla F(\mathbf{x}_{t-1})\|^2\right] \quad (8)$$

4.4 Bound on Lyapunov Function Rate of Contraction for Single Iteration

With the bounds determined on the two components of the Lyapunov function in (7) and (8), the next step is to substitute these bounds to determine the bound of the Lyapunov function value difference across one iteration (i.e. $\mathbb{E}[\Phi_{t+1} - \Phi_t] = \mathbb{E}[F(\mathbf{x}_{t+1}) - F(\mathbf{x}_t)] + \mathbb{E}[\|\epsilon_{t+1}\|^2/\eta_t - \|\epsilon_t\|^2/\eta_{t-1}]$). This creates a very large mathematical expression with many terms, so various techniques are used to combine and simplify terms, starting with grouping terms that have the same vector norms (e.g. $\|\nabla F(\mathbf{x}_t)\|^2, \|\epsilon_t\|^2$). It is here when the authors also introduce and choose values for constants, like a_t, c , and w , to strategically eliminate any positive error terms which ensures the Lyapunov function monotonically decreases over time. For instance, choosing to set $a_{t+1} = c\eta_t^2$ and $c = 28L^2 + G^2/(7Lk^3)$ enables this outcome and causes constants to cancel out for various terms leaving the expected drop in Lyapunov function value over a single iteration to have the bound:

$$\mathbb{E}[\Phi_{t+1} - \Phi_t] \leq \mathbb{E}\left[-\frac{\eta_t}{4}\|\nabla F(\mathbf{x}_t)\|^2 + \frac{3\eta_t}{4}\|\epsilon_t\|^2 + \frac{1}{32L^2\eta_t}\|\epsilon_{t+1}\|^2 - \frac{1}{32L^2\eta_{t-1}}\|\epsilon_t\|^2\right] \quad (9)$$

4.5 Summation of Bound on Lyapunov Function Decrease Across All Iterations

Calculating the final convergence rate bound for STORM amounts to now summing both sides of (9) from $t = 1$ to T . Due to the author's choice of constants and variables, it should be apparent that the final two terms in (9) will telescope (i.e. intermediate terms in the summation all cancel out) just leaving the first and last terms for that portion of the inequality. Likewise, the summation on the left side of (9) telescopes as well, $\sum_{t=1}^T \mathbb{E}[\Phi_{t+1} - \Phi_t] = \mathbb{E}[\Phi_{T+1} - \Phi_1] \geq F^* - F(\mathbf{x}_1)$, which allows us to remove any references to Φ_t from the inequality. With additional algebraic manipulation, including the use of Cauchy-Schwarz, the subadditivity inequality $(a+b)^{1/3} \leq a^{1/3} + b^{1/3}$, Young's inequality, abstracting certain terms that repeat into a new variable $M = \frac{1}{k}[8(F(\mathbf{x}_1) - F^*) + \frac{w^{1/3}\sigma^2}{4L^2k} + \frac{k^3c^2}{2L^2} \ln(T+2)]$, and moving the expected gradient norm to the left side, we arrive at the final convergence bound for STORM:

$$\mathbb{E}[\|\nabla F(\hat{\mathbf{x}})\|] \leq \frac{w^{1/6}\sqrt{2M} + 2M^{3/4}}{\sqrt{T}} + \frac{2\sigma^{1/3}}{T^{1/3}} = \mathcal{O}(1/T^{1/2} + \sigma^{1/3}/T^{1/3}) \quad (10)$$

Thus, by starting with the unique Lyapunov function designed by the STORM authors, bounding each of the two components of this function, and summing the per-iteration Lyapunov difference bound over all t from 1 to T , we confirm the final $\mathcal{O}(1/T^{1/2} + \sigma^{1/3}/T^{1/3})$ convergence bound.

5 Limitation and Proposed Enhancement to STORM Adaptive Learning Rate

One key design feature of STORM that caught my attention during the review of this paper has to do with the fact that the denominator of η_t accumulates **all** previous stochastic gradient norm squared values which suggests that the algorithm may not adapt well to local changes in geometry of the objective function landscape in later iterations, especially if many high gradient values are observed in early iterations. As such, my hypothesis is that STORM may converge faster for loss function scenarios that match this profile by applying an exponential forgetting factor (EFF), λ^p (where $\lambda \in (0, 1)$ and $p \geq 0$), in the adaptive learning rate η_t definition as follows.

$$\eta_t = \frac{k}{\left(w + \sum_{i=1}^t \lambda^{t-i} \|\nabla f(\mathbf{x}_i, \xi_i)\|^2\right)^{1/3}} \quad (11)$$

My empirical results confirm that this is indeed the case when tested using the non-convex Rosenbrock function for varying degrees of λ against the performance of the original STORM algorithm. For lack of space, the details of this analysis and empirical results are shown in the Appendix.

References

- [1] Ashok Cutkosky and Francesco Orabona. Momentum-based variance reduction in non-convex SGD. In *Advances in Neural Information Processing Systems 33*, pages 15236–15246, 2020.
- [2] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems*, volume 26, pages 315–323, 2013.
- [3] Lam M. Nguyen, Jie Liu, Katya Scheinberg, and Martin Takáč. SARAH: A novel method for machine learning problems using stochastic recursive gradient. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2613–2621, 2017.
- [4] Cong Fang, Chris Junchi Li, Zhouchen Lin, and Tong Zhang. SPIDER: Near-optimal non-convex optimization via stochastic path-integrated differential estimator. In *Advances in Neural Information Processing Systems 31*, volume 31 of *NeurIPS 2018*, pages 689–699, 2018.
- [5] Aaron Defazio and Léon Bottou. On the ineffectiveness of variance reduced optimization for deep learning. In *Advances in Neural Information Processing Systems*, volume 32, pages 9454–9464, 2019.
- [6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, volume 3, pages 1–15, 2015.
- [7] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(64):2121–2159, 2011.
- [8] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *International Conference on Learning Representations (ICLR)*, 2018.

Appendix

In studying the architecture of STORM, one inherent design feature that caught my attention was the fact that the adaptive learning rate η_t monotonically decreases as the algorithm progresses, regardless of the local geometry or topography of the loss function landscape in later iterations. While the adaptive learning rate itself is a novel aspect of STORM, given its inverse relationship with the stochastic gradients seen thus far, it unfortunately gives equal weight to gradient values seen far in the past versus more recently observed gradients. My intuition is that this actually makes the algorithm *less adaptive* (in terms of step size) and less responsive to the current objective function landscape than it otherwise could be as it makes progress toward a stationary point. Additionally, my theory is that this behavior may also cause STORM to stall out well before it reaches a stationary point if high magnitude gradient values are observed in early iterations.

Proposed Modification to Adaptive Learning Rate

To test this hypothesis, my proposal is to simply add an exponential forgetting factor (EFF), λ^p , within the summation of the squared norms of the historical stochastic gradients in the denominator of η_t and analyze the impact on convergence rate compared to the original STORM for varying degrees of λ in the range $(0, 1)$. The modified adaptive learning rate formula is listed as follows:

$$\eta_t = \frac{k}{\left(w + \sum_{i=1}^t \lambda^{t-i} \|\nabla f(\mathbf{x}_i, \xi_i)\|^2\right)^{1/3}} \quad (12)$$

This proposed modification to the adaptive learning rate of STORM, i.e. making it more sensitive and responsive to more recently observed gradient values, borrows a key design feature of the popular ADAM algorithm [6] but preserves the variance-reduction design features of STORM. This adjustment is also simple to implement and adds virtually no computational overhead to the existing algorithm aside from a trivial multiplicative factor. It can also be noted that when the exponential forgetting factor λ is set to 1, the adaptive learning rate term collapses into the original STORM definition.

Objective Function and Trajectory for Empirical Analysis

In the original STORM paper, the authors only test their algorithm on a single labeled dataset of images, CIFAR-10, and on a (non-convex) ResNet-32 deep neural network that uses a standard Cross-Entropy Loss objective function for their empirical results which they compare against the popular ADAM [6] and AdaGrad [7] algorithms.

To test my hypothesis for the modified adaptive learning rate, the commonly used non-convex Rosenbrock ("banana") benchmark function is selected as a starting point in order to more easily guarantee the conditions where it is suspected that STORM's original adaptive learning rate falls short. The Rosenbrock function $F : \mathbb{R}^2 \rightarrow \mathbb{R}$ takes $(x, y) \in \mathbb{R}^2$ as input parameters, is defined by constants a and b , and has a global minimum at $(x^*, y^*) = (a, a^2)$ where $F(x^*, y^*) = 0$:

$$F(x, y) = (a - x)^2 + b(y - x^2)^2$$

Commonly selected values of $a = 1$ and $b = 100$ are utilized for this experiment which gives us the following function and true gradient vector definitions for the simulations:

$$F(x, y) = (1 - x)^2 + 100(y - x^2)^2 \quad (13)$$

$$\nabla F(x, y) = \begin{bmatrix} -2(1 - x) - 400x(y - x^2) \\ 200(y - x^2) \end{bmatrix} \quad (14)$$

The geometry of this instance of the Rosenbrock function is defined by a global minimum at $(x^*, y^*) = (1, 1)$ and a shallow, parabolic-shaped valley with deep walls as demonstrated by the contour plot in Figure 1. The starting point, $\mathbf{x}_1 = (-2, 2)$ is intentionally chosen so that the optimization algorithm first has to descend a region of high gradient values before then navigating the more shallow valley floor with much lower gradient values.

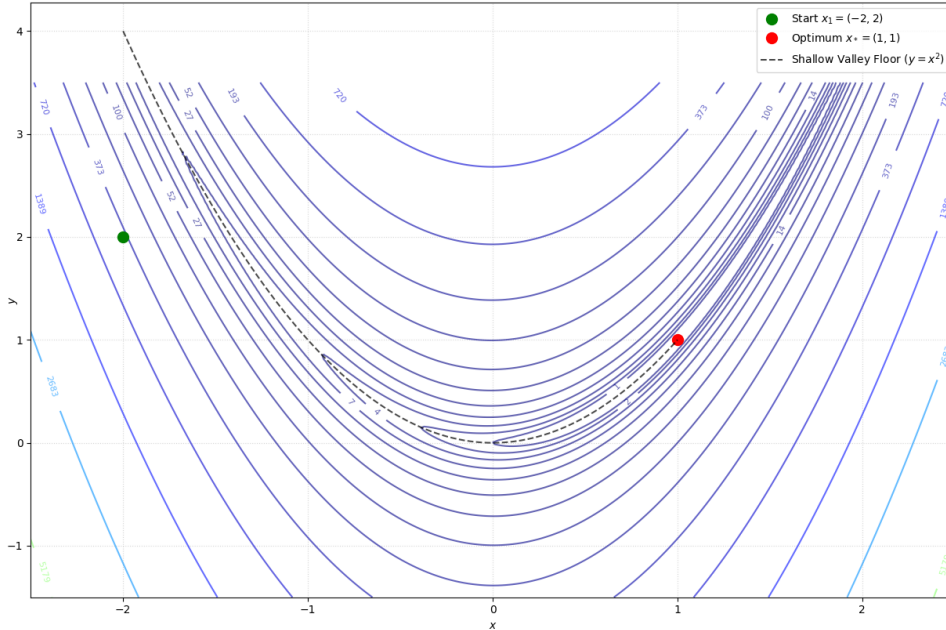


Figure 1: Rosenbrock function contour plot and testing trajectory.

Since we can compute the true gradient $\nabla F(\mathbf{x}_t)$ at each iteration as defined by Equation (14) and the STORM implementation instead uses a stochastic gradient value $\nabla f(\mathbf{x}_t, \xi_t)$, my simulation Python code in Listing 1 synthesizes a stochastic gradient by adding some random noise ϵ to the true gradient value as seen in a previous equation, $\nabla f(\mathbf{x}, \xi) = \nabla F(\mathbf{x}) + \epsilon(\mathbf{x}, \xi)$.

Constant EFF Empirical Results and Analysis

My simulations start by comparing the performance of the standard STORM algorithm using the non-convex Rosenbrock objective function defined by Equation (13) against my modified version of STORM, which is labeled as **EFF-STORM**, that uses the modified adaptive learning rate with an EFF listed in Equation (11). Four different values of the EFF are tested in these experiments where $\lambda \in \{0.9, 0.99, 0.999, 0.9999\}$ and the stopping threshold is set to a standard value of $\epsilon = 10^{-4}$. The results after running each simulation for 16,000 iterations can be seen in Figures 2, 3, 4, and 5.

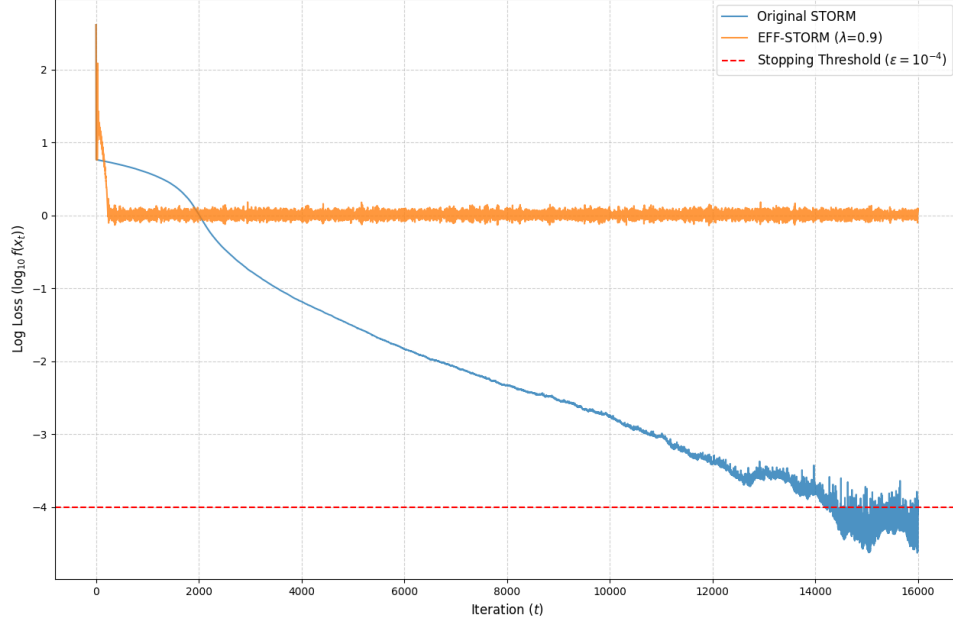


Figure 2: EFF-STORM with $\lambda = 0.9$.

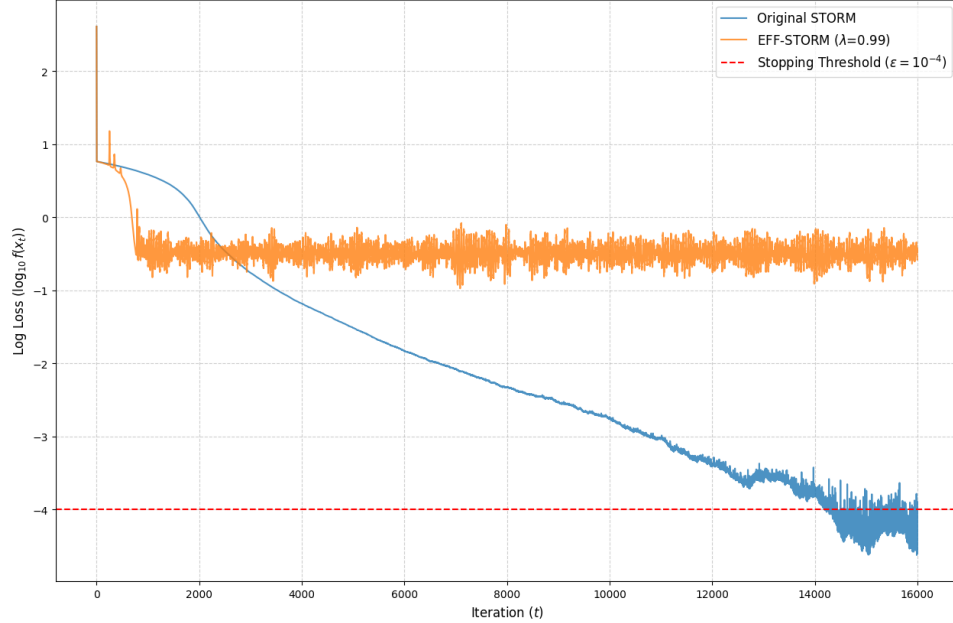


Figure 3: EFF-STORM with $\lambda = 0.99$.

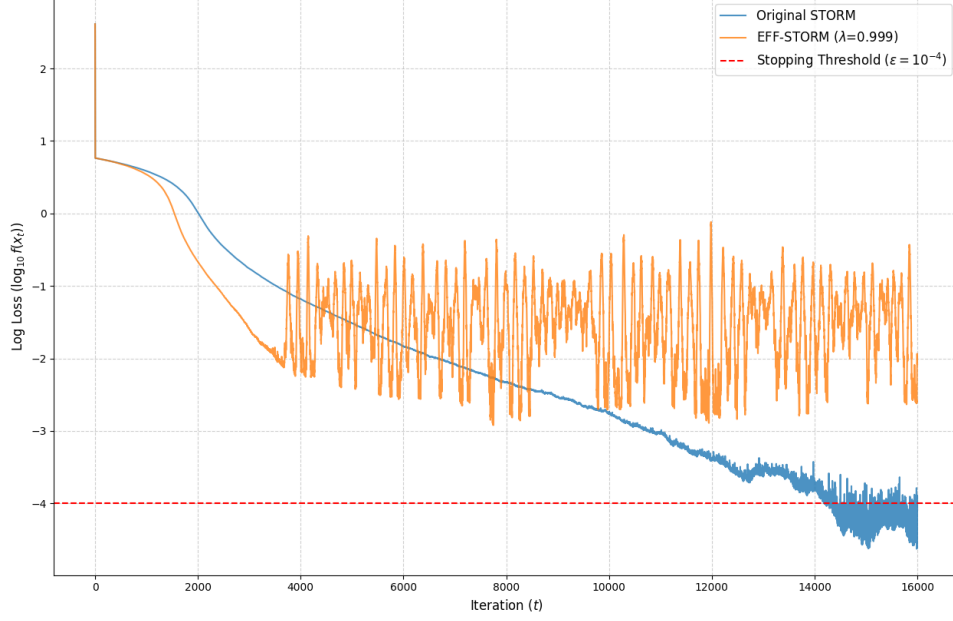


Figure 4: EFF-STORM with $\lambda = 0.999$.

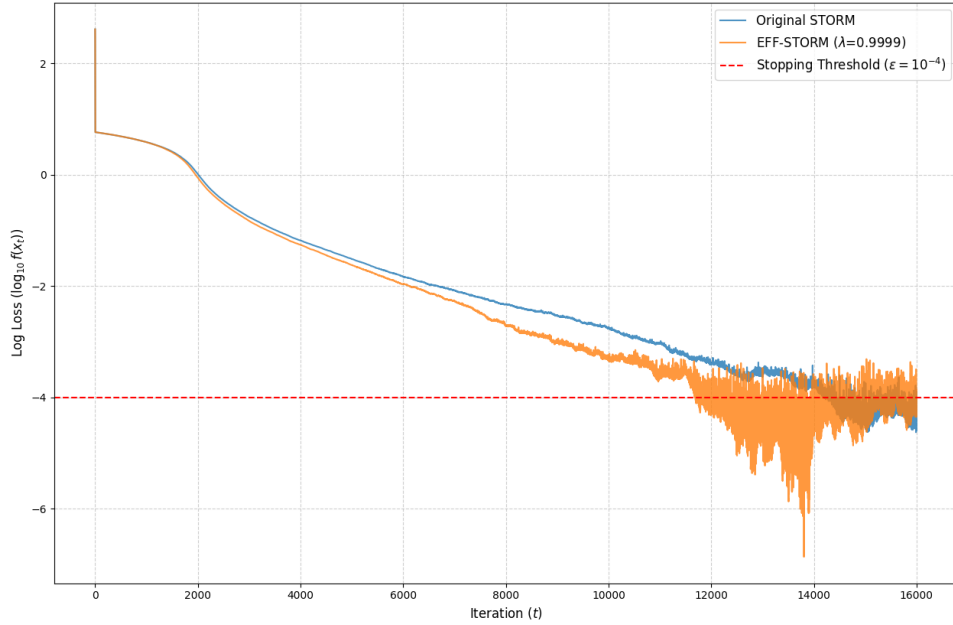


Figure 5: EFF-STORM with $\lambda = 0.9999$.

Interestingly enough, the closer the EFF λ is to 1, the better it performs on this particular benchmark. For the first three trials in Figures 2, 3, and 4 where $\lambda \in \{0.9, 0.99, 0.999\}$, the EFF-STORM algorithm never reaches the stopping threshold of $\epsilon = 10^{-4}$ within the 16,000 iterations while original STORM ultimately does. My assessment of this outcome is that a lower value of λ means that EFF-STORM has a shorter window of applying the effects of previous gradients, and so when the algorithm gets into the shallow valley portion of the Rosenbrock function surface fairly early on, those gradients becomes very small which, in turn, makes the denominator of the learning rate smaller and the actual learning rate larger. Since this function has a narrow valley with steep walls, the larger learning rate essentially causes it to oscillate violently off the walls instead of making

forward progress toward the optimal point which is classic overshooting behavior. This characteristic is confirmed by plotting the progress of these two algorithms on a contour plot of the objective function with markers at selected iteration milestones (i.e. $t \in \{75, 250, 16000\}$) as seen in Figure 6 where we observe the more extreme case when $\lambda = 0.9$.

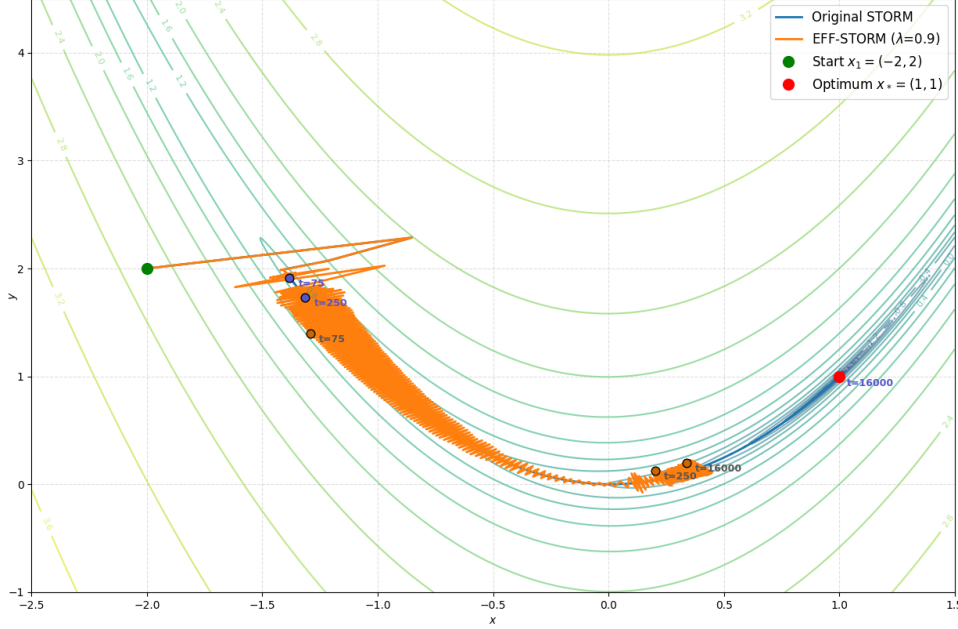


Figure 6: EFF-STORM ($\lambda = 0.9$) vs Original STORM pathway progress on Rosenbrock objective function contour plot with selected iteration index progress markers.

It was not until the exponential forgetting factor was set to $\lambda = 0.9999$ that the EFF-STORM implementation ultimately reached the stopping threshold, and it actually did so in fewer iterations than original STORM! In this fourth trial in Figure 5, EFF-STORM reaches the loss threshold of 10^{-4} in roughly 11,700 iterations vs. roughly 14,200 iterations for original STORM, so it is approximately 21.4% faster than original STORM. This is a promising result that suggests a properly tuned exponential forgetting factor applied to the squared gradient norm summation in the denominator of STORM's adaptive learning rate formula may result in faster convergence with essentially no additional computational overhead for certain objective function landscapes.

Time-Varying Exponential Forgetting Factor Simulation

After carefully observing the progress made by EFF-STORM using these four different values for λ , it was also noticeable and interesting to see that the *rate of convergence* was very impressive the *lower* the value of λ until it reached its "noise floor" and failed to make more progress toward the optimal point or ϵ stopping threshold. In Figure 2, for instance, the convergence rate is incredibly fast (steep) in the first roughly 200 iterations before its progress bottoms out. This convergence rate then becomes less aggressive as the value of λ increases but at the benefit of allowing the algorithm to get closer and closer to the optimal point. This observation led me to consider the possibility that a *time-varying* exponential forgetting factor λ_t could potentially yield better results. My strategy is that the time-varying λ_t would start on the lower end of the $[0, 1]$ spectrum to reap the benefits of the more aggressive convergence rate early on but then steadily increase to essentially push out the "noise floor" phenomenon that plagued the static smaller λ trials. A simple means of accomplishing this goal is to start with the desired upper bound, 1, and subtract the reciprocal of t (the iteration index) as seen in Equation (15) so that as $t \rightarrow \infty$ the value $\lambda_t \rightarrow 1$:

$$\lambda_t = 1 - \left(\frac{1}{t}\right) \quad (15)$$

Let us call this modification of the original algorithm **TVEFF-STORM** for its unique Time-Varying Exponential Forgetting Factor (TVEFF) term λ_t in the denominator of the adaptive learning rate η_t formula as denoted in Equation (16).

$$\eta_t = \frac{k}{\left(w + \sum_{i=1}^t \left(1 - \frac{1}{t}\right)^{t-i} \|\nabla f(\mathbf{x}_i, \xi_i)\|^2\right)^{1/3}} \quad (16)$$

The results of this TVEFF-STORM vs. original STORM simulation for the same 16,000 iterations can be seen in Figure 7:

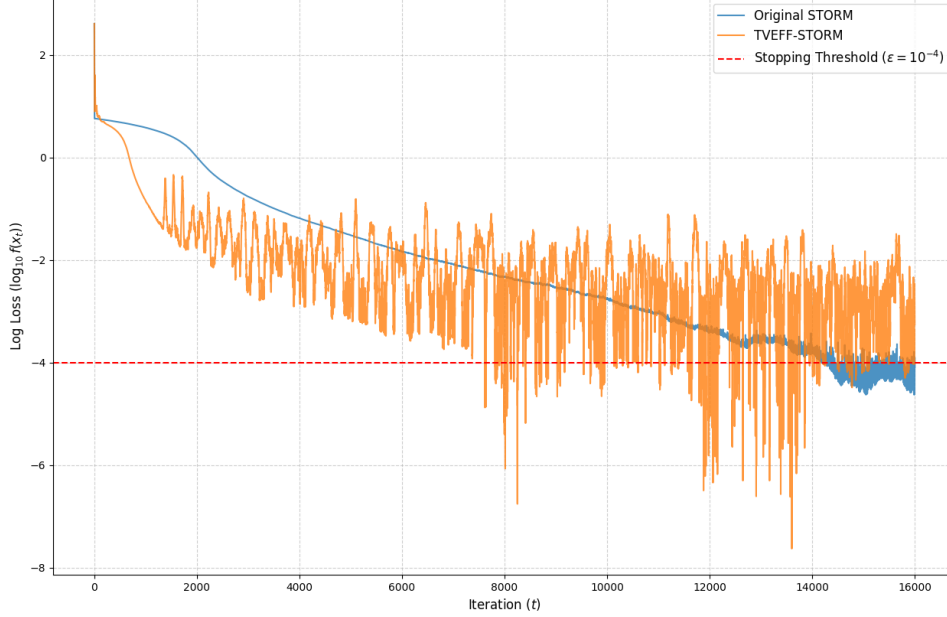


Figure 7: TVEFF-STORM convergence behavior vs. original STORM.

Impressively, it is observed that TVEFF-STORM, despite its rather noisy loss curve, actually reaches the ϵ stopping threshold well before original STORM. In fact, TVEFF-STORM reaches the loss threshold of 10^{-4} in roughly 7,600 iterations vs. roughly 14,200 iterations for original STORM, so it is *approximately 86.8% faster than original STORM* under this test scenario using a widely adopted non-convex objective function benchmark!

Such a significant improvement in convergence performance over the existing STORM algorithm for this use case with virtually no additional computational overhead is a very optimistic result in terms of the potential value of incorporating a time-varying exponential forgetting factor in the adaptive learning rate. The additional benefit of the time-varying EFF λ_t is that it also eliminates the need to tune and identify an optimal constant λ which was the focus of the earlier simulations.

Acknowledgment of Potential Drawbacks and Future Opportunities

While the empirical results of the EFF-STORM (for specific λ) and TVEFF-STORM variants of the original algorithm demonstrated strong promise in these experiments, the potential drawbacks of this approach are also noted.

For instance, with this modification, the learning rate is no longer guaranteed to monotonically decrease which impacts the steps taken in the original STORM mathematical proof for deriving its convergence rate bound of $\mathcal{O}(1/T^{1/3})$. In short, a consequence of the addition of an EFF term to η_t in the original STORM are the downstream impacts to the momentum coefficient a_t and the momentum vector $\mathbf{d}_t$ updates and its variance reduction capabilities. For instance, since the momentum coefficient a_t is directly proportional to the square of the learning rate, a more erratic

learning rate could potentially counteract the variance reduction benefits of STORM in later iterations by putting more emphasis on the current stochastic gradient values over the historical EMA and variance reduction components of the $\mathbf{d}_t$ momentum vector.

My theory is that part of the large noise envelope we observed in the above simulations, especially in Figures 4 and 7, could be partly explained by these downstream impacts to the momentum coefficient and momentum vector updates. It's almost as if any positive benefits obtained from a potentially more accurate learning rate on the local geometry of the objective function surface were overpowered by the downstream loss in variance reduction capabilities. Updating the lengthy mathematical proof of STORM's convergence bounds to try and verify this would be an interesting and worthy exercise, but it's possible to at least identify the specific area of the proof where non-monotonicity of η_t breaks the existing proof logic.

Recall that the Lyapunov function for STORM and the starting point of the convergence proof is $\Phi_t = F(\mathbf{x}_t) + z_t \|\epsilon_t\|^2$ where z_t is a coefficient that is time-varying and inversely proportional to the learning rate η_{t-1} . After we determine the bounds on the objective function value and the tracking error and combine these bounds to arrive at the Lyapunov function rate of contraction bound for a single iteration as seen in (9), the fact that the last two terms on the right hand side of (9) will telescope when we sum both sides from $t = 1$ to T is a key feature of this proof. Let us isolate and inspect the telescoping squared error norm terms in the summation as follows:

$$\sum_{t=1}^T \mathbb{E} \left[\frac{1}{32L^2\eta_t} \|\epsilon_{t+1}\|^2 - \frac{1}{32L^2\eta_{t-1}} \|\epsilon_t\|^2 \right] \quad (17)$$

Notice the consecutive adaptive learning rate variables, η_t and η_{t-1} , in the denominators of this expression. For the telescoping to work and ensure that the coefficient of the squared norm error term in the final convergence bound is less than or equal to zero, the difference $\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}}$ needs to be *non-negative* and this can only happen if $\eta_t \leq \eta_{t-1}$ which is the definition of a monotonically decreasing learning rate.

In terms of potential improvements, since the learning rate is no longer guaranteed to converge to zero as $t \rightarrow \infty$, we may consider incorporating a separate learning rate decay factor to achieve this outcome. This could be accomplished in a number of ways, such as replacing the static k constant in the numerator of η_t with a time-dependent variable that gradually decays over time to counteract any potential intermittent large gradient values encountered in later iterations.

Additionally, to force monotonicity, the adaptive learning rate η_t formula for STORM could keep track of the maximum exponential moving average value of the squared gradient norms seen thus far (i.e. $\hat{G}_t = \max(\hat{G}_{t-1}, G_t)$) similar to the mechanism found in AMSGrad [8]. However, this strategy would still keep older large gradient values alive to a certain extent, so the benefits may be minimal.

Python Source Code

The following Python source code in Listing 1 was written to run the above EFF-STORM simulations for varying constant values of the exponential forgetting factor λ and generate performance comparison plots against the original STORM using the non-convex Rosenbrock objective function.

Listing 1: Python Source Code for EFF-STORM vs. STORM Optimization Performance Plots

```

1
2 # ECE 273 Final Project Simulations
3 # STORM Algorithm Modifications (EFF-STORM)
4 # Alan Nawoj (Spring 2026 Quarter)
5
6 import numpy as np
7 import matplotlib.pyplot as plt
8
9 # Total Iteration Count
10 iterations = 16000
11
12 # Exponential Forgetting Factor for EFF-STORM (0.9, 0.99, 0.999, 0.9999)
13 lambda_eff = 0.9999
14
15 # Ensure Consistency with Randomness

```

```

16 np.random.seed(42)
17
18 # Define the Rosenbrock Objective Function
19 def rosenbrock(x):
20     #  $F(x,y) = (a-x)^2 + b(y-x^2)^2$ 
21     #  $x[0]=x, x[1]=y$ ; with  $a=1, b=100$ 
22     return (1.0 - x[0]) ** 2 + 100.0 * (x[1] - x[0] ** 2) ** 2
23
24 # Define Gradient of Rosenbrock Function Instance ("True Gradient")
25 def rosenbrock_gradient(x):
26     df_dx = -2.0 * (1.0 - x[0]) - 400.0 * x[0] * (x[1] - x[0] ** 2)
27     df_dy = 200.0 * (x[1] - x[0] ** 2)
28     return np.array([df_dx, df_dy])
29
30 # STORM and EFF-STORM Optimization Loop Function
31 def optimize_storm(is_eff_storm=False, lambda_eff=0.9, iterations=10000):
32     # Starting Point  $x_1 = (-2, 2)$ 
33     x = np.array([-2.0, 2.0])
34
35     # STORM Hyperparameters
36     k = 0.1; w = 1.0; c = 10.0
37
38     loss_history = []
39
40     # Simulate Stochastic Gradient as True Gradient Plus Noise
41     noise_level = 0.5
42     g_current = rosenbrock_gradient(x) + np.random.normal(0, noise_level, 2)
43
44     d = g_current.copy() # Initial Momentum Vector = Initial Stochastic Gradient
45     sum_g_squared = 0.0 # Sum of Squared Stochastic Gradient Norms
46
47     # Run Algorithm from  $t = 1$  to  $T+1$  Iterations
48     for t in range(1, iterations + 1):
49         # Compute Optimality Gap (i.e.  $F(x)$ ) for Current  $x$  and Store Value
50         optimality_gap = rosenbrock(x)
51         loss_history.append(optimality_gap)
52
53         # Calculate Squared Norm of Current Stochastic Gradient
54         g_norm_squared = np.sum(g_current ** 2)
55
56         # Apply Exponential Forgetting Factor for EFF-STORM Only
57         if is_eff_storm:
58             sum_g_squared = lambda_eff * sum_g_squared + g_norm_squared
59         else:
60             sum_g_squared += g_norm_squared
61
62         # Calculate Adaptive Learning Rate
63         eta = k / ((w + sum_g_squared) ** (1 / 3))
64
65         # Update Position (Optimization Variable)
66         x_next = x - eta * d
67
68         # Sample Stochastic Gradient at Next Position  $x_{t+1}$ 
69         # Scale Down Noise Level in Randomness to Simulate High Correlation b/w  $G_t, G_{t+1}$ 
70         noise_level = 0.25
71         g_next = rosenbrock_gradient(x_next) + np.random.normal(0, noise_level, 2)
72
73         # Calculate Momentum Coefficient  $a_{t+1} = c * (\eta_t)^2$  Capped at 1.0
74         a = min(1.0, c * (eta ** 2))
75
76         # Update Momentum Vector:  $d_{t+1} = g_{t+1} + (1 - a_{t+1})(d_t - g_t)$ 
77         d = g_next + (1.0 - a) * (d - g_current)
78
79         # Shift Variables for Next Time Step
80         x = x_next
81         g_current = g_next
82
83     return np.array(loss_history)
84
85 # Run Simulations for Original STORM and EFF-STORM
86 loss_original = optimize_storm(is_eff_storm=False, iterations=iterations)
87 loss_eff = optimize_storm(is_eff_storm=True, lambda_eff=lambda_eff, iterations=iterations)
88
89 # Plot Results
90 plt.figure(figsize=(10, 6))
91 plt.plot(np.log10(loss_original), label="Original STORM", color="#1f77b4", alpha=0.8)
92 plt.plot(np.log10(loss_eff), label=f"EFF-STORM ( $\lambda={lambda\_eff}$ )", color="#ff7f0e", alpha=0.8)
93 plt.axhline(y=-4, color='red', linestyle='--', linewidth=1.5, label=f'Stopping Threshold ( $\epsilon = 10^{-4}$ )')

```

```

94
95 plt.title("Optimization Performance of STORM vs. EFF-STORM on Rosenbrock Function",
           fontsize=14, fontweight='bold')
96 plt.xlabel("Iteration (${t$})", fontsize=12)
97 plt.ylabel("Log Loss (${log_{10}} f(x_t)$)", fontsize=12)
98 plt.grid(True, linestyle="--", alpha=0.6)
99 plt.legend(fontsize=12)
100 plt.show()
101
102 # Print Final Convergence Metrics
103 print(f"Final Objective Value (Original STORM): {loss_original[-1]:.4f}")
104 print(f"Final Objective Value (EFF-STORM): {loss_eff[-1]:.4f}")
105
106

```