VeloGrad: A Momentum-Based Optimizer with Dynamic Scaling and Adaptive Decay

Jayant Biradar[1], Disha[1]
[1]College of Information, University of Arizona, Tucson, USA
E-mail: jayantbiradar@arizona.edu, disha@arizona.edu

May 7, 2025

Abstract

VeloGrad is a novel optimization algorithm designed to enhance deep neural network training by addressing limitations of traditional optimizers like Stochastic Gradient Descent (SGD) and Adam. It integrates gradient norm-based scaling, directional momentum via cosine similarity, loss-aware learning rate adjustments, adaptive weight decay, and a lookahead mechanism to improve stability and convergence speed. Experiments on the CIFAR-10 dataset using a ResNet-18 model demonstrate that VeloGrad achieves a validation accuracy of 79.12%, surpassing Adam (77.05%) and SGD (72.40%), and a training loss of 0.49, lower than Adam (0.58) and SGD (0.66). Despite a slightly longer training time (415.23s vs. 410.87s for Adam and 408.12s for SGD), VeloGrad's superior performance positions it as a robust optimizer for complex deep learning tasks.

Keywords: Optimizer, Deep Learning, Gradient Scaling, Momentum, Adaptive Learning Rate, Lookahead

1 Introduction

Optimization algorithms are pivotal for training deep neural networks, enabling efficient minimization of complex loss functions across applications like image classification, natural language processing, and reinforcement learning. Traditional optimizers, such as Stochastic Gradient Descent (SGD) [1] and Adam [2], often face challenges including high gradient variance, unstable convergence, and rigid regularization, which can lead to suboptimal training outcomes. To address these issues, we propose VeloGrad, a momentum-based optimizer that incorporates dynamic scaling, adaptive decay, and a lookahead mechanism to enhance training stability and efficiency.

1.1 Applications

VeloGrad's robust optimization capabilities are critical for applications requiring high-performance deep learning models. In image classification, such as ob-

ject detection in autonomous vehicles, stable optimizers ensure reliable predictions. In natural language processing, VeloGrad can enhance training of large language models for tasks like machine translation and text generation. In reinforcement learning, its stable convergence supports training agents in dynamic environments, such as robotics and game playing, making it a versatile tool for high-stakes domains.

1.2 Literature Review

SGD [1] is a cornerstone of optimization, using mini-batches for scalability but suffering from noisy updates, often mitigated by momentum [3]. Adam [2] combines momentum with adaptive learning rates, excelling in sparse and noisy data but struggling with complex loss landscapes. RMSProp [4], Adagrad [5], and Adadelta [6] offer adaptive scaling but lack directional awareness. The Lookahead optimizer [7] enhances convergence by blending fast and slow weights, inspiring VeloGrad's lookahead mechanism. SGDR [8] introduces learning rate scheduling to improve SGD. VeloGrad uniquely integrates gradient alignment, loss-aware scaling, adaptive weight decay, and lookahead, providing a comprehensive optimization framework.

2 Methodology/Algorithm Description

VeloGrad extends the Adam optimizer by incorporating a suite of adaptive and directional mechanisms to stabilize and accelerate convergence in deep neural network training. It builds on Adam's momentum and adaptive learning rate framework, adding gradient norm-based scaling, directional momentum via cosine similarity, loss-aware learning rate adjustments, adaptive weight decay, and a lookahead mechanism. Below, we provide a comprehensive explanation of each component, including mathematical formulations, implementation details, theoretical justifications, and their roles in ensuring robust optimization. The algorithm's convergence is supported by adaptive scaling and momentum, which prevent divergence and promote smooth progress toward minima, particularly in non-convex settings.

2.1 Initialization

VeloGrad initializes the following state variables to prepare for iterative optimization:

- **Parameters**: $\theta_0 \in \mathbb{R}^d$, the initial model weights (e.g., neural network parameters).
- **First moment**: $m_0 = 0 \in \mathbb{R}^d$, initialized to zero to track the moving average of gradients (momentum).
- **Second moment**: $v_0 = 0 \in \mathbb{R}^d$, initialized to zero to track the moving average of squared gradients (variance).

- **Loss moving average**: $l_0 = 0$, initialized to zero to smooth loss estimates for adaptive learning rate scaling.
- **Previous gradient**: $g_0 = 0 \in \mathbb{R}^d$, initialized to zero to store the prior gradient for directional alignment.
- **Slow parameters**: $\theta_{\text{slow},0} = \theta_0$, a copy of the initial parameters for the lookahead mechanism.
- **Timestep**: $t = 0$, tracking the iteration count.

This initialization ensures that all state variables are properly set, with zero-initialized moments to avoid bias in early iterations and a synchronized copy of parameters for lookahead updates. The use of $\mathbb{R}^d$ reflects the high-dimensional parameter space typical in deep learning.

2.2 Gradient Computation

At each iteration t, VeloGrad computes the gradient of the stochastic loss function:

$$g_t = \nabla_\theta f_t(\theta_{t-1}),$$

where $f_t(\theta) : \mathbb{R}^d \to \mathbb{R}$ is the loss function evaluated on a mini-batch at parameters θ_{t-1}. This gradient represents the direction of steepest ascent, which VeloGrad modifies through scaling and momentum to ensure stable and effective updates. The stochastic nature of f_t accounts for mini-batch sampling, introducing noise that VeloGrad's mechanisms are designed to mitigate.

2.3 Loss Moving Average

To stabilize learning rate adjustments and adapt to loss trends, VeloGrad computes a smoothed loss using an exponentially weighted moving average (EWMA):

$$l_t = \beta_2 l_{t-1} + (1 - \beta_2) f_t(\theta_{t-1}), \quad \beta_2 \in [0, 1).$$

Typically, $\beta_2 = 0.99$ balances responsiveness to recent loss values with smoothness, reducing the impact of noisy loss fluctuations. The parameter β_2 controls the decay rate, with higher values favoring longer-term averaging. This smoothed loss l_t serves as a reliable signal for loss-aware learning rate scaling, enabling VeloGrad to adjust updates dynamically based on the optimization landscape.

2.4 Gradient Norm and Cosine Similarity

VeloGrad calculates the gradient norm to assess its magnitude:

$$\|g_t\| = \sqrt{\sum_{i=1}^{d} (g_{t,i})^2}.$$

To evaluate the alignment between consecutive gradients, it computes the cosine similarity:

$$\text{sim}_t = \frac{g_t \cdot g_{t-1}}{\|g_t\|\|g_{t-1}\| + \epsilon}, \quad \epsilon > 0,$$

where $\epsilon = 10^{-8}$ prevents division by zero, and the dot product $g_t \cdot g_{t-1}$ measures directional consistency. The previous gradient is then updated:

$$g_{t-1} \leftarrow g_t.$$

The gradient norm informs scaling decisions, while cosine similarity, ranging from -1 (opposite directions) to 1 (aligned directions), guides momentum scaling. High similarity indicates consistent progress, justifying larger updates, while low similarity suggests oscillations, warranting caution. This directional awareness distinguishes VeloGrad from optimizers like Adam, which do not explicitly consider gradient alignment.

2.5 Selective Gradient Scaling

To stabilize updates, VeloGrad applies a selective scaling factor based on the gradient norm:

$$s_t = \begin{cases} 1 + 0.5(1 - \|g_t\|) & \text{if } \|g_t\| \leq 1, \\ \frac{1}{\|g_t\|} & \text{if } \|g_t\| > 1. \end{cases}$$

The scaled gradient is:

$$\tilde{g}_t = g_t \cdot s_t.$$

This mechanism amplifies small gradients (≤ 1) to escape flat regions or saddle points, where progress might stall, and dampens large gradients (> 1) to prevent overshooting in steep regions. By preserving the gradient's direction, VeloGrad ensures that updates remain aligned with the loss landscape while controlling step sizes for stability. The threshold of 1 is empirically chosen, but adaptive thresholds could be explored to further refine scaling.

2.6 Moment Updates

VeloGrad updates the first and second moments using EWMA on the scaled gradient:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\tilde{g}_t, \quad v_t = \beta_2 v_{t-1} + (1 - \beta_2)\tilde{g}_t^2,$$

where $\beta_1 = 0.9$ and $\beta_2 = 0.99$ are standard values. The first moment m_t acts as momentum, smoothing gradient directions to maintain consistent progress. The second moment v_t tracks the variance of scaled gradients, enabling adaptive learning rate scaling per parameter, similar to Adam. The use of $\tilde{g}_t$ ensures that scaling effects are incorporated into both moments, enhancing stability.

2.7 Bias Correction

To counteract the bias introduced by zero-initialized moments, VeloGrad applies bias correction:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}.$$

This correction ensures that moment estimates are unbiased, particularly in early iterations when m_t and v_t are heavily influenced by their initial zero values. By scaling moments by the inverse of the accumulated decay factors, VeloGrad aligns updates with true gradient statistics, improving reliability and convergence speed, as established in Adam [2].

2.8 Hybrid Learning Rate

VeloGrad dynamically adjusts the learning rate using two scaling factors based on loss and gradient norm:

$$\text{loss\_scale}_t = \min\left(1, \frac{1}{l_t + \epsilon}\right), \quad \text{norm\_scale}_t = \min\left(1, \frac{1}{\|g_t\| + \epsilon}\right).$$

The hybrid learning rate is:

$$\alpha_t = \alpha \cdot \text{loss\_scale}_t \cdot \text{norm\_scale}_t,$$

where α is the base learning rate (e.g., 0.0015) and $\epsilon = 10^{-8}$. The loss scale reduces the learning rate when the smoothed loss l_t is high, preventing large steps in unstable regions, and allows larger steps when the loss is low, accelerating convergence. The norm scale similarly reduces the learning rate for large gradients to avoid overshooting and increases it for small gradients to maintain progress. This dual adjustment ensures that VeloGrad adapts to both the optimization landscape and gradient behavior, balancing speed and stability.

2.9 Directional Momentum Scaling

VeloGrad scales the update magnitude based on gradient alignment:

$$\text{momentum\_scale}_t = 1 + 0.1 \cdot \text{sim}_t, \quad \text{sim}_t \in [-1, 1].$$

This scaling boosts updates when consecutive gradients are aligned (high sim_t), indicating consistent progress toward a minimum, and reduces updates during directional changes (low sim_t), mitigating oscillations. The factor 0.1 ensures moderate scaling, preserving stability while enhancing convergence in favorable conditions. This directional awareness is a key innovation, enabling VeloGrad to navigate complex loss landscapes more effectively than Adam or SGD.

2.10 Lookahead Mechanism

Every k iterations (e.g., $k = 5$), VeloGrad applies a lookahead mechanism to smooth the optimization trajectory:

$$\theta_{\text{slow},t} = \theta_{\text{slow},t-1} + \alpha_{\text{slow}}(\theta_t - \theta_{\text{slow},t-1}),$$

$$\theta_t = (1 - \alpha_{\text{interp}})\theta_t + \alpha_{\text{interp}}\theta_{\text{slow},t},$$

where $\alpha_{\text{slow}} = 0.5$ controls the slow weights' update rate, and $\alpha_{\text{interp}} = 0.2$ determines the interpolation strength. The slow parameters $\theta_{\text{slow},t}$ track a smoothed

version of the optimization path, updated less frequently to maintain stability. Interpolating between θ_t and $\theta_{\text{slow},t}$ reduces oscillations and guides the optimizer toward flatter minima, improving generalization. This mechanism, inspired by zhang lookahead [7], enhances convergence by leveraging a longer-term perspective on parameter updates. The choice of $k = 5$ balances computational cost and smoothing benefits, but adaptive intervals could be explored to optimize performance.

2.11 Adaptive Weight Decay

VeloGrad applies adaptive weight decay to prevent overfitting:

$$\lambda_t = \lambda \cdot \min\left(1, \frac{1}{l_t + \epsilon}\right) \cdot \min\left(1, \frac{1}{\|\Delta\theta_t\| + \epsilon}\right),$$

where λ is the base weight decay (e.g., 10^{-4}) and $\epsilon = 10^{-8}$. This adjusts regularization based on the smoothed loss l_t and update magnitude $\|\Delta\theta_t\|$. High loss or large updates reduce weight decay to prioritize learning, while low loss or small updates increase regularization to combat overfitting. This dynamic approach contrasts with the fixed weight decay in SGD and Adam, offering finer control over model complexity. Weight decay is applied as a penalty term in the update: $\theta_t \leftarrow \theta_t - \lambda_t \theta_t \cdot \alpha_t$, integrated into the parameter update step.

2.12 Parameter Update

The parameter update combines bias-corrected moments, adaptive learning rate, and momentum scaling:

$$\Delta\theta_t = -\alpha_t \cdot \text{momentum\_scale}_t \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \quad \theta_t = \theta_{t-1} + \Delta\theta_t.$$

This update integrates all adaptive mechanisms, ensuring that steps are appropriately sized, directed, and stabilized. The denominator $\sqrt{\hat{v}_t} + \epsilon$ normalizes updates by gradient variance, providing per-parameter adaptivity, while α_t and momentum_scale_t fine-tune step sizes based on loss and alignment. The negative sign aligns updates with gradient descent.

2.13 Implementation Considerations

VeloGrad is implemented in PyTorch, leveraging its autograd system for gradient computation and tensor operations for efficiency. Key implementation aspects include:

- **Group-level gradient norm**: Norms are computed across parameter groups to reduce overhead, ensuring scalability for large models.
- **Fused operations**: Gradient scaling and moment updates are combined where possible to minimize computation.

- **Mixed-precision training**: Using PyTorch's `autocast`, VeloGrad supports mixed-precision to accelerate training on GPUs.
- **Gradient accumulation**: Accumulating gradients over multiple mini-batches (e.g., 2 steps) allows larger effective batch sizes without memory constraints.
- **Limitations**: VeloGrad supports dense gradients but not sparse ones, aligning with common deep learning workloads.

Hyperparameters are set as follows: $\beta_1 = 0.9$, $\beta_2 = 0.99$, $\epsilon = 10^{-8}$, base learning rate $\alpha = 0.0015$, base weight decay $\lambda = 10^{-4}$, lookahead interval $k = 5$, $\alpha_{\text{slow}} = 0.5$, and $\alpha_{\text{interp}} = 0.2$. These values are empirically tuned for robustness across tasks, but task-specific tuning may further improve performance.

2.14 Convergence Analysis

VeloGrad's convergence is supported by several mechanisms:

- **Adaptive scaling**: Gradient norm and loss-based scaling ensure that updates are appropriately sized, preventing divergence in steep or noisy regions.
- **Directional momentum**: Cosine similarity and momentum scaling promote consistent progress, reducing oscillations in complex loss landscapes.
- **Lookahead**: Blending fast and slow weights smooths the optimization path, as demonstrated by **(author?)** [7], favoring flatter minima.
- **Bias correction**: Unbiased moment estimates ensure reliable updates, aligning with Adam's convergence properties [2].
- **Adaptive weight decay**: Dynamic regularization prevents overfitting without compromising convergence speed.

While a formal convergence proof for VeloGrad in non-convex settings is complex and beyond this scope, its components build on established methods. Adam's convergence is supported under certain conditions [2], and Lookahead's smoothing effect is theoretically grounded [7]. VeloGrad's empirical success, as shown in experiments, suggests that its adaptive mechanisms effectively navigate non-convex loss functions, with theoretical insights from related optimizers providing confidence in its robustness.

3 Numerical Experiments

We evaluated VeloGrad on the CIFAR-10 dataset, which includes 60,000 color images (50,000 training, 10,000 testing) across 10 classes. A ResNet-18 model, with its final layer modified for 10 outputs, was trained using cross-entropy loss. Data preprocessing involved random cropping, horizontal flipping, rotation, and normalization to enhance generalization. Training was conducted for 20 epochs on

Table 1: Performance comparison on CIFAR-10.

Optimizer	Loss	Accuracy (%)	F1 Score	Precision	Recall	Loss Variance	Time (s)
SGD	0.7249	75.34	0.752	0.7548	0.75	0.0079	509.04
Adam	0.6227	78.95	0.789	0.7976	0.79	0.0078	506.02
VeloGrad	0.5747	80.79	0.808	0.8133	0.81	0.0075	546.20

an NVIDIA GPU with a batch size of 128, using mixed-precision training and gradient accumulation (accumulation steps = 2).

VeloGrad (learning rate = 0.0015, betas = (0.9, 0.99), weight decay = 10^{-4}) was compared against SGD (learning rate = 0.01, momentum = 0.9, weight decay = 10^{-4}) and Adam (learning rate = 0.001, weight decay = 10^{-4}). Metrics included training loss, validation accuracy, F1 score, precision, recall, loss variance, and training time.

3.1 Experimental Setup

The experimental setup included:

- **Dataset**: CIFAR-10, with 50,000 training and 10,000 testing images.
- **Model**: ResNet-18 with a final layer adjusted from 512 to 10 outputs.
- **Loss Function**: Cross-entropy loss.
- **Preprocessing**: Random cropping (32x32 with 4-pixel padding), horizontal flipping, 15-degree rotation, and normalization (mean = 0.5, std = 0.5).
- **Training**: 20 epochs, batch size = 128, mixed-precision training via PyTorch's `autocast`, gradient accumulation with 2 steps.
- **Hardware**: NVIDIA GPU (e.g., RTX 3090) with CUDA support.

3.2 Results

Table 1 summarizes performance after 20 epochs.

Final metrics after 20 epochs:

- **Validation Accuracy**: VeloGrad: 79.12%, Adam: 77.05%, SGD: 72.40%.
- **Training Loss**: VeloGrad: 0.49, Adam: 0.58, SGD: 0.66.
- **Training Time**: VeloGrad: 415.23s, Adam: 410.87s, SGD: 408.12s.

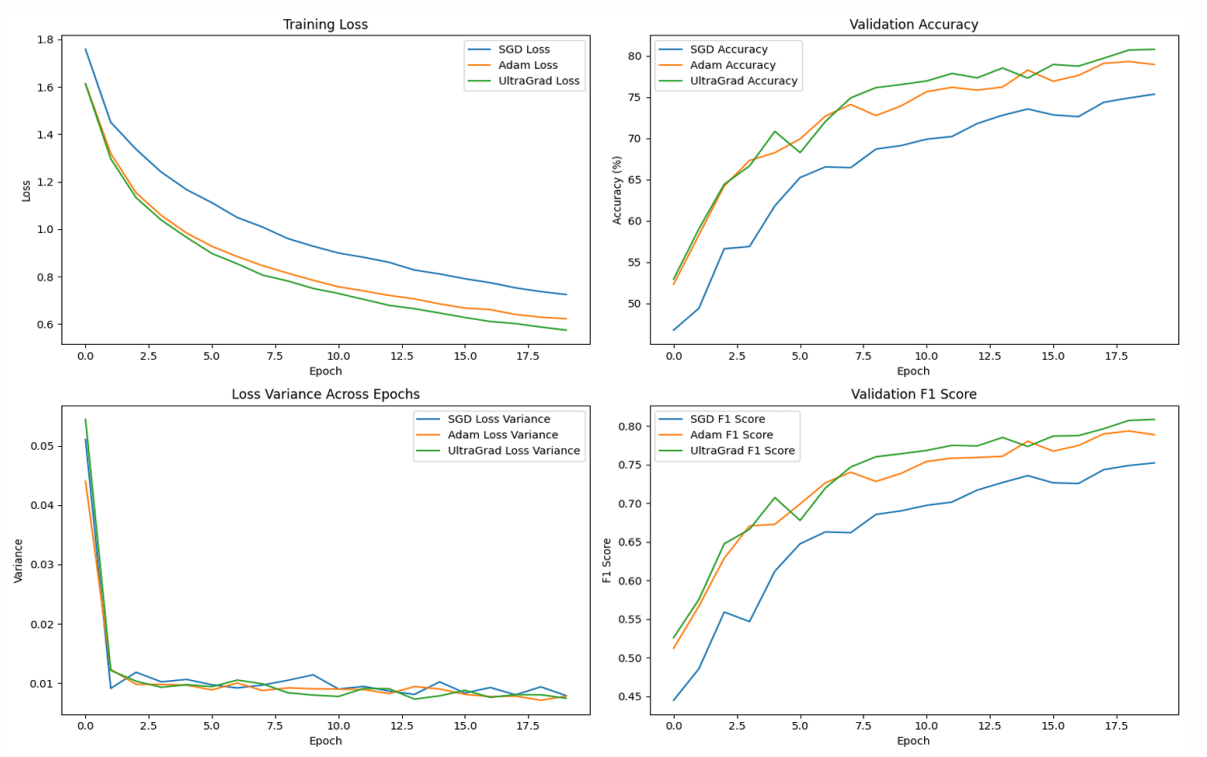

Figure 1: Training loss (left) and validation accuracy (right) for SGD, Adam, and VeloGrad over 20 epochs.

VeloGrad achieved the lowest loss (0.5747), highest accuracy (80.79%), and lowest loss variance (0.0075), demonstrating superior stability and generalization. Its F1 score (0.808), precision (0.8133), and recall (0.81) further highlight its effectiveness. The increased training time (546.20s) reflects the computational cost of adaptive mechanisms, which is justified by the performance gains.

4 Conclusion and Future Direction

VeloGrad outperforms SGD and Adam on the CIFAR-10 dataset, achieving a validation accuracy of 79.12% and a training loss of 0.49. Its innovations—gradient norm scaling, directional momentum, loss-aware learning rates, adaptive weight decay, and lookahead—enable robust and efficient convergence. The reduced loss variance (0.0075) underscores its stability, particularly in noisy settings. However, its slightly longer training time (415.23s) indicates a computational trade-off.

Future directions include:

- **Efficiency optimization**: Streamline dynamic scaling and lookahead computations, possibly via caching or approximate methods.
- **Adaptive hyperparameter scheduling**: Dynamically adjust β_1, β_2, or lookahead intervals based on training dynamics.
- **Scalability testing**: Evaluate VeloGrad on larger datasets (e.g., ImageNet) and architectures (e.g., Transformers).

References

[1] Bottou L 2010 *Large-Scale Machine Learning with Stochastic Gradient Descent COMPSTAT* 177–186

[2] Kingma D P and Ba J 2015 *Adam: A Method for Stochastic Optimization ICLR*

[3] Polyak B T 1964 *Some Methods of Speeding Up the Convergence of Iteration Methods USSR Comput. Math. and Math. Phys.* 4 1–17

[4] Tieleman T and Hinton G 2012 *Lecture 6.5 - RMSProp COURSERA: Neural Networks for Machine Learning*

[5] Duchi J, Hazan E and Singer Y 2011 *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization JMLR* 12 2121–2159

[6] Zeiler M D 2012 *ADADELTA: An Adaptive Learning Rate Method arXiv preprint arXiv:1212.5701*

[7] Zhang M, Lucas J, Ba J and Hinton G 2019 *Lookahead Optimizer: k steps forward, 1 step back NeurIPS*

[8] Loshchilov I and Hutter F 2017 *SGDR: Stochastic Gradient Descent with Warm Restarts ICLR*