When and How Much to Perturb? Decoding Radius-Timing-Scale(RTS) in PUGD Optimization

Anonymous CVPR submission

Paper ID xxxx

Abstract

8-10lines We demonstrate 'Radius-Timing-Scale(RTS)' as algorithmic enhancements to the Perturbated Unit Gradient Descent (PUGD). Optimization algorithms are pivotal in deep learning, particularly for image classification tasks. Perturbated Unit Gradient Descent (PUGD) [21] introduces a novel update rule with limitations of high computational costs that cant reach the better Top-1 accuracy when compared with benchmark SGD when SGD reached convergence.

This work tries to alleviate such gap by investigating the limitations of Perturbated Unit Gradient Descent (PUGD) optimizer, proposing a novel additional parameter tuning strategy that adjusts both the perturbation radius, timing of using it and the scale of gradient. It is analogous to learning rate scheduling, systematic adjustment of perturbation radius and scale of gradient boosts PUGD's performance.Meanwhile, our proposed approaches addressed the question—within a limited scope—of whether perturbation radius and gradient scale should be large or small during different training phases. Code and results are available under `https://github.com/eeyzs1`.

1. Introduction

40lines+fig/1page Stochastic Gradient Descent (SGD) [19] remains a cornerstone for iterative model optimization, yet it faces one limitation: While theoretical analysis in Kawaguchi' study [9] demonstrates that deep learning models rarely become trapped in strict saddle points or local minima, empirical evidence shows performance variance across different model architectures and training protocols for different tasks. In other word, sharp minima hinder generalization, as shown in the work from Foret et al.[3], causing poor performance on new scenarios. These challenges have spurred numerous algorithmic variants, each aiming to mitigate specific drawbacks of vanilla GD [18].

The *Perturbated Unit Gradient Descent (PUGD)* [21] introduces a novel update rule: gradient perturbation with unit normalization by scaling the combined (original + perturbed) gradient with unit dual norm, which ensures stable updates. This algorithm addresses generalization improvement and saddle point mitigation simultaneously.

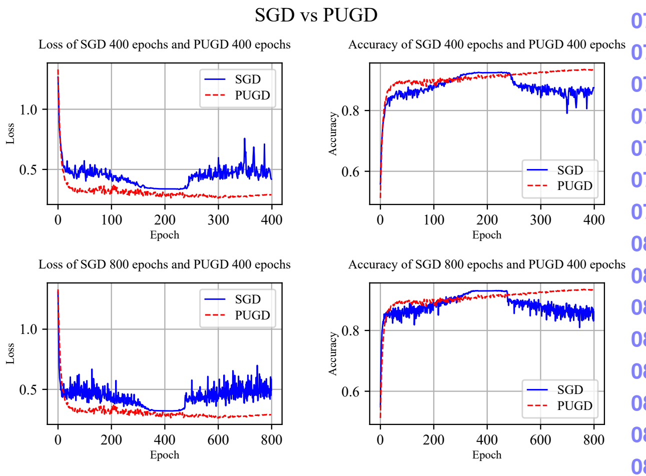


Figure 1: Training histories of SGD and PUGD **Top row**: Loss and accuracy of both optimizers (400 epochs) **Bottom row**: SGD (800 epochs) vs PUGD (400 epochs)

Although Tseng et al. [21] reported that PUGD outperformed *Stochastic Gradient Descent (SGD)* [19] under matched epoch budgets, my CIFAR-10 experiments (Figure 1) reveal a divergence: PUGD fails to match SGD's convergence speed in early training phases, though it eventually achieves higher peak accuracy after extended optimization. This suggests a trade-off between initial convergence rate and final performance, which means potential optimization possibilities. Inspired by this finding and *cosine annealing* [14], which is a learning rate scheduling technique that dynamically adjusts the learning rate (eta_t) during training following a cosine-shaped decay curve. Mathematically, it is defined as: $\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})\left(1 + \cos\left(\frac{T_{\text{cur}}}{T_{\max}}\pi\right)\right)$. Therefore we want to propose enhancements from three aspects: 1. An adapted perturbation scheduler for PUGD that dynamically adjusts the exploration radius through cyclical temperature decay, enabling phase-wise trade-offs between exploration and exploitation. 2. An adaptive SGD-PUGD hybrid that leverages SGD's rapid initial convergence in early training stages, then transitions to PUGD's perturbation-based refinement for sharpness-aware generalization, achieving both training efficiency and flat-minima convergence. 3. Tunning the scale of gradient so that influence the gradient descent direction reasonablely. The integration of these three enhancements is formally designated as 'Radius-Timing Scale(RTS)'. In brief, the main contributions in this work include:

(1) Perturbation Radius Tuning: Analogous to cosine annealing learning rate scheduling, the systematic adjustment of perturbation radiuss boosts PUGD performance.

(2) Computational Efficiency: PUGD is applied efficiently at an appropriate time rather than at the beginning of training.

(3) Scale of gradient Tunning: the systematic adjustment of gradient boosts PUGD performance.

(4) The tuning coefficient α: four adaptive update strategies.

(5) Integrated Solution: Combining perturbation radius and timing control yields synergistic effects and demonstrates the complete optimization process.

(6) Results comparisons: The results compare the proposed method with PUGD and SGD, showing improvements from Radius-Timing Scale(RTS).

This paper is divided into five parts. First, the background, motivation and a summary of Radius-Timing Scale(RTS) have been present in this. Then, in Section 2, the mechanism and its limitations. After, we present the explanation of Radius-Timing Scale(RTS) enhancement in Section 3. Finally, a series of experiments on PUGD and enhancement is shown in Section 4, with the conclusion in Section 5 and supplements in Appendix.

2. Related Work

Since Stochastic Gradient Descent (SGD) [19] first emerged as an optimization technique, it has gradually become the de facto standard optimizer across machine learning paradigms, owing to its computational efficiency and proven empirical success in large-scale learning scenarios. Whereas modern neural networks exhibit complex, non-convex loss landscapes with multiple global minima that demonstrate distinct generalization capabilities [10]. With the theoretical support from Ji and Telgarsky[7] that the local Lipschitz condition ensures gradient flow(infinitesimal gradient descent) trajectories avoid oscillatory paths, while SGD noise helps escape sharp basins-jointly contributing to the flat minima. As well as Empirical evidence suggests that gradient normalization can enhance generalization, as demonstrated in prior work. For instance, Path-SGD [17] employs path-normalized updates to improve optimization in deep networks, while [6, 8] further links normalized gradients to favorable generalization properties. These findings support the hypothesis that gradient normalization per step promotes stable and well-behaved training dynamics, leading to better generalization. Foret et al.[3] does further generalization analysis and shows the SGD converged to a sharp minimum which cause bad generalization. Then it provides one method called *SHARPNESS-AWARE MINIMIZATION (SAM)* to handle it by seeking parameters that lie in neighborhoods having uniformly low loss, which is the core idea of perturbation, and then dose the *normalized gradient descent (NGD)* [16] with the found parameters, thus simultaneously minimizing loss value and loss sharpness. In addition, Zheng et al. [22] raised *Adversarial Model Perturbation (AMP)* with a similar idea that add perturbation iteratively to increase the robustness of the model. Both Sharpness-Aware Minimization (SAM) and Adversarial Model Perturbation (AMP) enhance model robustness by introducing perturbations to model parameters, yet they target distinct goals: SAM seeks flat minima for better generalization, while AMP directly defends against parameter-space adversarial attacks. Inspired by the effort listed above, PUGD [21] was created to eliminate the landscape noise generated by using dual-norm as a high dimensional space scaler for sharpness detectin, it was demonstrated as below:

$$\hat{\epsilon_t} = \frac{|w_t| \cdot g_t}{\| |w_t| \cdot g_t \|} \tag{1}$$

$$g_{t^*} = \bigtriangledown f(w_t + \hat{\epsilon_t}) \tag{2}$$

$$w_{t+1} = w_t - \eta_t \frac{(g_{t^*} + g_t)}{\|g_{t^*} + g_t\|} = w_{c,t} - \eta_t U_t \tag{3}$$

Notation explanation: ϵ_t is the unit perturbation, U_t is the unit gradient at t where the "unit gradient" in PUGD came from, $g_t = \bigtriangledown f(w_t)$ is the gradients of the loss function at t, g_{t^*} is the gradients from the unit perturbation ϵ_t with adaptive steps toward each component in a unit ball within the norm of total perturbation radius $\|\epsilon_t\|$, $U_t = \frac{(g_{t^*}+g_t)}{\|g_{t^*}+g_t\|}$ is the final unit gradient at t by which combined the original gradient and the gradient from perturbation, η_t is the learning rate.

3. Radius-Timing Scale(RTS)

This section discusses the limitations of PUGD caused by perturbation radius, double computational cost and the influence from final gradient U_t. In order to eliminates these three limitations, three methods based on empirical observations was proposed.

3.1. Limitations of PUGD

SHARPNESS-AWARE MINIMIZATION (SAM) [3] defined its core algorithm that used to minimize the PAC-Bayesian generalization error upper bound as: For any $\rho > 0$, with high probability over training set $\mathcal{S}$ generated from distribution $\mathscr{D}$,

$$[\max_{\|\boldsymbol{\epsilon}\|_2 \le \rho} L_{\mathcal{S}}(\boldsymbol{w}+\boldsymbol{\epsilon}) - L_{\mathcal{S}}(\boldsymbol{w})] + L_{\mathcal{S}}(\boldsymbol{w}) + h(\|\boldsymbol{w}\|_2^2/\rho^2)$$

Therefore, gradient descent by the perturbation gradient means suppress both the sharpness and gradient, which theoretically reduce loss and generalization error. Through a series of mathematical deductions and simplifications, the original SAM inequality can be updated to gradient descent with

$$\nabla_{\boldsymbol{w}} L_{\mathcal{S}}^{SAM}(\boldsymbol{w}) \approx \nabla_{\boldsymbol{w}} L_{\mathcal{S}}(\boldsymbol{w} + \hat{\boldsymbol{\epsilon}}(\boldsymbol{w})).$$

where

$$L_{\mathcal{S}}^{SAM}(\boldsymbol{w}) \triangleq \max_{||\boldsymbol{\epsilon}||_p \le \rho} L_{\mathcal{S}}(\boldsymbol{w}+\boldsymbol{\epsilon}),$$

Returning to PUGD, its perturbation radius (ρ in the SAM's formula) is fixed to 1 as shown in equation 1, unlike SAM/ASAM where ρ is tunable. This invariance may stem from PUGD's implicit adaptive correction of perturbations through utility-based gradient statistics, bypassing the need to explicitly optimize ρ-dependent terms like $h(\|\boldsymbol{w}\|_2^2/\rho^2)$ in generalization bounds. While Foret et al.[3] and Kwon et al. [11] inidicates in their works that ρ with different values can also generate competitive performance. Show that varying ρ affects generalization ability, despite ASAM used the similar method as PUGD to bypass $h(\cdot)$. No empirical or theoretical evidence supports $\rho = 1$ as the optimal perturbation radius across all scenarios.

Meanwhile, PUGD faces two inherent challenges:

(1) Computational Cost: Persistent sharpness minimization throughout training incurs doubled computational overhead due to repeated gradient calculations. As shown in Figure 1, the loss decrease and accuracy increase didn't show significant differences in initial epochs, and the SGD converged faster than PUGD with a higher accuracy. No matter PUGD used the same computational cost or half the computational cost as SGD.

(2) Dynamic Perturbation Effect: The learning trajectories of different optimizers has similar paths during the initial epochs[21]. This suggests that the optimal timing for applying different optimizers may potentially influence the optimization outcomes, provided we can identify such timing through a measurable criterion.

This necessitates a strategic discussion on when to activate perturbation-based sharpness control, rather than enforcing it indiscriminately across all training phases.

The final gradient update direction in PUGD, defined as $U_t = \frac{(g_{t^*}+g_t)}{\|g_{t^*}+g_t\|}$ from equation 3, implicitly suppresses the effect of sharpness minimization by effectively increasing the scale of gradient. However, similar to the lack of consensus on an optimal perturbation radius, there exists no empirical or theoretical justification for assuming that doubling the gradient is universally optimal across all scenarios. This observation suggests the need to further evaluate:

(1) Gradient Scaling: Whether the current heuristic (e.g., $g_{t^*} + g_t$) provides the most effective balance between sharpness control and convergence.

(2) Scenario Adaptivity: How gradient scaling should be dynamically adjusted based on problem-specific geometry (e.g., loss landscape curvature or batch statistics).

Further research is warranted to establish guidelines for calibrating scale of gradient in sharpness-aware optimization.

3.2. Perturbation Radius Tuning

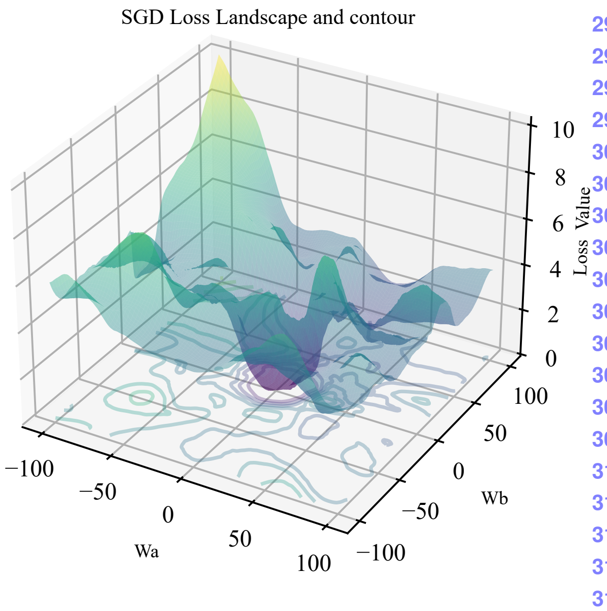


Figure 2: Loss landscape and contour of SGD

With respect to the loss landscape as shown in Figure 2, which generated based on the loss landscape work [12]. In the light of Figure 1 and the ideas that "flat" minimizers that generalize well [2]. Based on the explorations from Li et al.[12] and Jastrzebski et al.[6]. The training process can be illustrated as optimizer generally navigates the loss landscape by starting at random initial weights, iteratively moving opposite to noisy mini-batch gradients (which help escape saddle points), oscillating in narrow valleys due to high curvature, and eventually converging toward flat regions (minima) as the learning rate decays—behaving like a marble rolling down a bumpy slope with stochastic nudges. This motivates a two-way exploration into how perturbation radii (large vs. small) interact with training phases (early vs. late) — an underexplored dimension that may reveal optimal noise scheduling strategies. To explain it explicitly, The one is considered as (1) early training benefits from larger perturbation radii to escape poor initializations (e.g., saddle points in high-curvature regions), while later stages require smaller radii to fine-tune within flat minima basins. The another is (2) early training benefits from smaller perturbation radii to reach flat minimizers efficiently, while later stages require larger radii to expand the area of flat basins. These two approaches declared the background philosophy as whether to prioritize global exploration in early training (escaping poor initializations) or late training (expanding flat basins), constituting a continuum of noise-driven optimization strategies. Therefore, four tuning strategies were proposed in 3.5, by which multiply the $\epsilon_t (the unit perturbation)$ with an adaptive variable α to tuning the perturbation radius.

3.3. Timing of application for PUGD

Given that (a) perturbation may provide negligible benefits during early training stages (as hypothesized in 3.2), and (b) SGD and PUGD exhibit nearly identical performance in initial phases Figure 1, we propose a phase-aware switching mechanism to dynamically activate PUGD only when its gradient modification proves statistically meaningful. Adapting the methodologies of Maclaurin et al. [15] that stop training early based on specific criterion, two statistically-grounded ideas were proposed for adaptive optimization control: Dynamic triggering conditions based on 1. gradient statistics, 2. generalization gap Derived from these two ideas, there are two methods to measure when to use PUGD as:

(1) Gradient variance threshold: Calculate the variance of the L2 norm the current batch gradient: $\sigma^2 = Var(\|g_t\|_2)$ (Sliding window statistics, such as past k=10 batches). Trigger condition: $\sigma^2 < \gamma \cdot \sigma^2_{init}$, σ^2_{init} is the mean gradient variance in the first k steps of training, γ is attenuation coefficient (such as 0.2).

(3) generalization gap monitoring: Calculate the generalization gap of resampling every T (like 3) epoch: $\Delta = |L_{val} - L_{train}|$. Trigger condition: $\Delta > \mu \cdot \Delta_{base}$, Δ_{base} is initial gap, μ is proportional threshold (such as 2.0), $\Delta_{base} = \frac{1}{\xi}\sum_{i=1}^{\xi} \Delta_i$, which is the mean gap during the initial ξ epochs.

3.4. Scale of gradient Tunning

This section introduces a method for tuning the scale of $g_t = \bigtriangledown f(w_t)$ (gradient of the loss function at t) when compared with g_{t^*} (gradient from the unit perturbation ϵ_t). As shown in equation 3, PUGD enforces a fixed ratio of 1 between these gradients, whereas SAM implicitly treats this ratio as 1:0. Empirical results from Tseng et al. [19] demonstrate that PUGD outperforms SAM, suggesting that aggregating the original gradient and perturbation gradient (i.e., $g_{t^*} + g_t$) is empirically effective. This observation raises a critical question: What is the theoretically justified ratio between these gradients? Foret et al. [3] proved that gradient descent via perturbation gradients can simultaneously suppress g_t and sharpness. However, PUGD amplifies gradient effects through simple summation. The optimality of the ratio 1 remains uncertain—it may be a theoretically justified value or merely a heuristic choice. From a theoretical perspective, it remains ambiguous whether applying larger scaling factors to g_t during early or late training stages would enhance generalization performance. The fundamental challenge lies in the lack of conclusive evidence about the correlation between gradient magnitude at specific training phases and final model robustness. Inspired by the considerations from tuning the perturbation radius 3.2, we propose to dynamically rescale the gradient g_t during training through an adaptive parameter α, employing tuning strategies 3.5 similar to those used for perturbation radius adjustment. This approach enhances the contribution of g_t to gradient updates and reveals phase-dependent scaling laws that govern the tripartite relationship between training phases, generalization performance, and scale of g_t.

3.5. Four tuning strategies with α

Adapting the cosine annealing paradigm proposed by Loshchilov and Hutter [14], we implemented and compared four distinct tuning strategies during model training to evaluate their relative performance. The proposed algorithms and corresponding conceptual diagrams are illustrated as follows:

(1) cosine annealing:

$$\alpha = \beta_{min} + (\beta_{max} - \beta_{min}) \cos(\frac{\pi t}{2T})$$

(2) inverse sine annealing:

$$\alpha = \left[\beta_{min} + (\beta_{max} - \beta_{min}) \sin(\frac{\pi t}{2T})\right]^{-1}$$

(3) sine heating:

$$\alpha = \beta_{min} + (\beta_{max} - \beta_{min}) \sin(\frac{\pi t}{2T})$$

(4) inverse cosine heating:

$$\alpha = \left[\beta_{min} + (\beta_{max} - \beta_{min}) \cos(\frac{\pi t}{2T})\right]^{-1}$$

Where T is the total num of epochs, t accounts for how many epochs have been performed since the last start. Max and min β decide the uppper bound and lower bound of the α. Heating contrasts with annealing, inducing monotonic increase in α during training. Inverse denotes the algorithmic inversion operation.

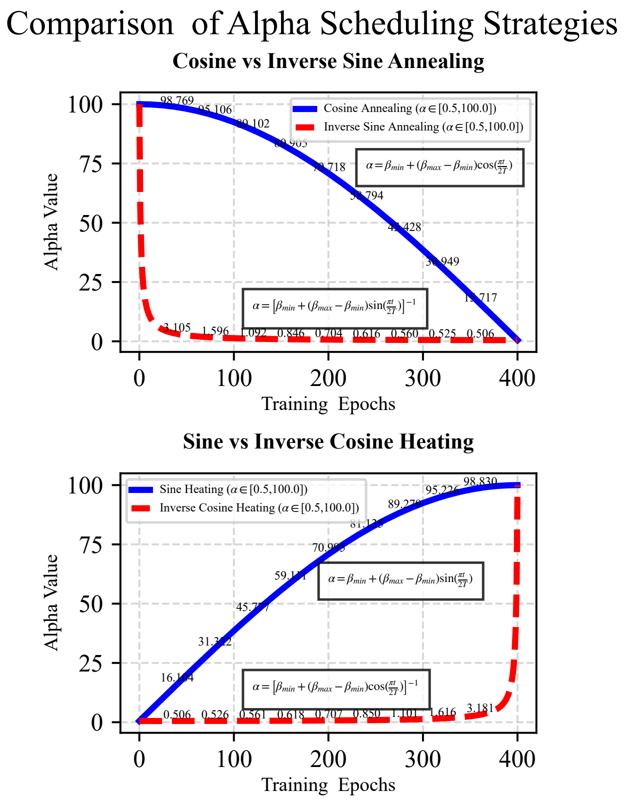


Figure 3: Comparison of alpha scheduling strategies **Top subfigure**: Cosine vs inverse sine annealing **Bottom subfigure**: Sine vs inverse cosine annealing. The annotations are the decile-sampled values

As illustrated in Figure 3, the main distinction between cosine annealing and inverse sine annealing lies in their decay dynamics: Cosine annealing exhibits a gradual decay of α throughout the training process. Inverse sine annealing rapidly decreases α during the initial 10% of epochs (labeled with 3.105 in the figure), followed by a slower decay phase until converging to the minimum value β_{min} (set to 0.5 in this example). Similarly, the main distinction between sine heating and inverse cosine heating lies in their growth dynamics: Sine heating exhibits a gradual growth of α throughout the training process. Inverse cosine heating increases α slowly during the last 90% of epochs (labeled with 3.181 in the figure), followed by a rapid growth phase until converging to the maximum value β_{min} (set to 100 in this example).

The benefits of inversion lie in the rapid adjustment of α (used for tuning both perturbation radius and gradient scale) during specific training phases. This dynamic behavior aligns more closely with the loss variation patterns observed in Figure 1 and Figure 2, suggesting that abrupt changes—rather than gradual ones—may better capture underlying optimization patterns. While empirical evidence for this hypothesis remains to be established (including whether α should be larger or smaller for perturbation radius and gradient scaling), I additionally propose testing cosine annealing and sine heating strategies to systematically investigate their effects on model generalization.

Scope Clarification: our objective is not to present universally optimal tuning strategies, as Gastpar et al. [4] theoretically proved, uniformly tight generalization bounds are unattainable in overparameterized regimes without distribution-dependent or algorithmic priors. Rather, I demonstrate that the four strategies remain effective under specific conditions for tuning the perturbation radius and scale of gradient.

4. Experiments

In this section, we implemented two experiments, including (1) optimizing behaviours comparison and (2) fine-tuning comparison. The experiments were implemented on open datasets in the equipment with GTX 1060 Mobile, then were implemented on open datasets in the equipment with GTX 5090 Mobile.

In addition, every experiment followed the hyperparameters setting in the learning rate (LR) = 0.1, weight decay = 0.0005, momentum = 0.9, Nesterov = False, and the learning schedule of cosine annealing [14]. Every subject followed the same Pytorch default initial weights policy, a random uniform distribution with seed = 42.

4.1. Comparative analysis of optimization strategies

To observe the behaviours of PUGD with different strategies for the CIFAR-10 and CIFAR-100 dataset, this subsection of simulation has been organized into 3-parts: 1. Perturbation radius Tuning, 2. Timing of application, 3. Scale of gradient Tunning. Through these observations, we can get a picture of what these strategies can do with models.

4.1.1 Perturbation radius Tuning

As shown in fig. 4, both strategies (inverse sine and cosine) consistently outperformed native PUGD. Although the best accuracy improvements were marginal (0.9353 vs. 0.9419/0.9417), two critical patterns emerged:

- The performance gap *w.r.t* PUGD widened progres-

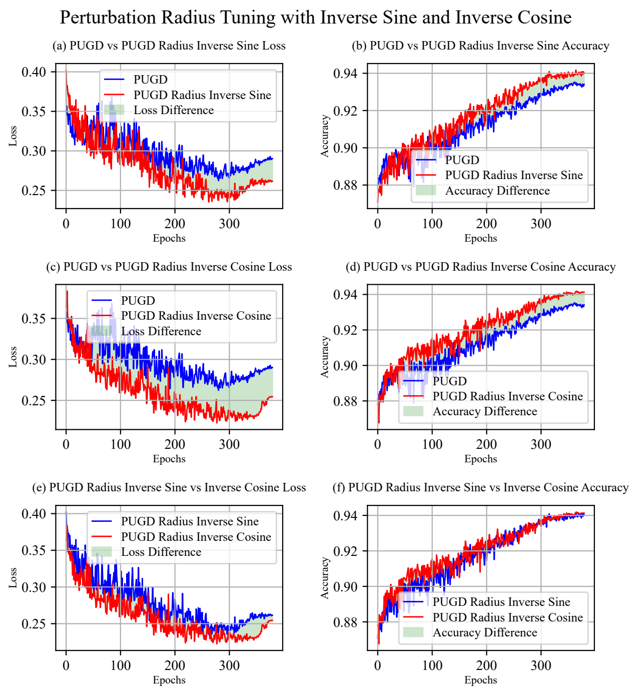


Figure 4: Perturbation Radius Tuning with Inverse Sine and Inverse Cosine **(a)**: Pugd vs Pugd Radius inverse sine loss **(b)**: Pugd vs Pugd Radius inverse sine accuracy **(c)**: Pugd vs Pugd Radius inverse cosine loss **(d)**: Pugd vs Pugd Radius inverse cosine accuracy **(e)**: Pugd Radius inverse sine vs inverse cosine loss **(f)**: Pugd Radius inverse sine vs inverse cosine accuracy. The epochs ranged from 21 to 400, β_{min} set as 0.01 and β_{max} set as 2 for both inverse sine and cosine.

sively with more epochs, evidenced by growing differences in both loss and accuracy metrics (subfigure a, b, c, d).

- Between the two strategies, inverse cosine demonstrated superior loss minimization despite nearly identical accuracy trajectories (subfigure e, f).

Unfortunately, the experimental results only demonstrate that both strategies can optimize model performance beyond the baseline achieved by the native PUGD method. However, they cannot substantiate the theoretical assumptions proposed in section 3.2, since the two strategies produced statistically equivalent results. This equivalence suggests that the futher investigation needed.

appendix A.1

One additional thing has to be noticed is that loss was increased during the accuracy increase after 300 epochs, which was explained by Ishida et al. [5] as the "overfitting" problem. The reason is overfitting or inconsistent distribution of training validation data, that is, in the later stage of training, the predicted results tend to be extreme, causing a small number of incorrectly predicted samples to dominate the loss, but at the same time, a small number of samples do not affect the overall validation accuracy. Due to the time limitation and the fact that the benchmark (native PUGD) didn't solve this problem, this study is unable to address this issue either.. For subsequent researchers or developers, if you want to overcome this problem, you can try (1) Increase training samples, (2) Increase the weight of regularization coefficients, (3) Add early stopping mechanism [1], (4) Adopting Focal Loss [13], (5) use Label Smoothing [20]

Note: The training curves of the initial 20 epochs were excluded from visualization due to their statistically indistinguishable patterns.

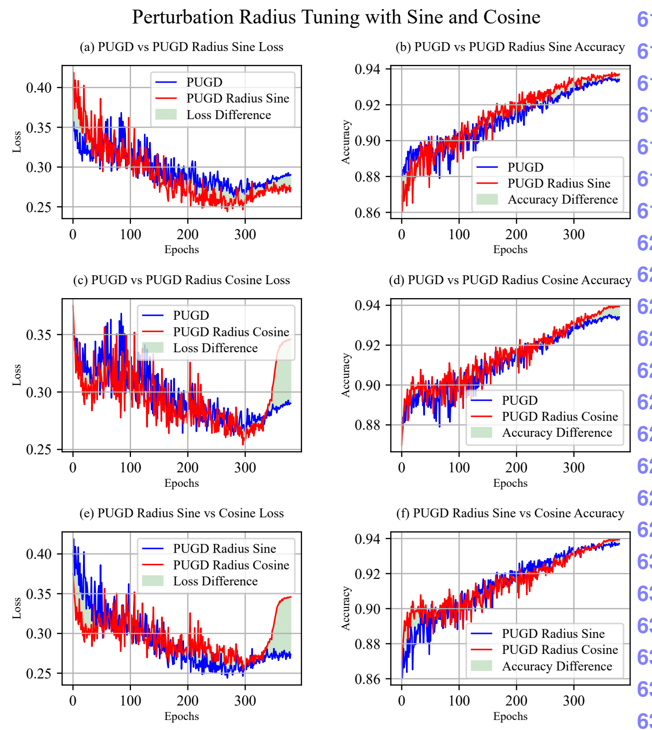


Figure 5: Perturbation Radius Tuning with Sine and Cosine **(a)**: Pugd vs Pugd Radius sine loss **(b)**: Pugd vs Pugd Radius sine accuracy **(c)**: Pugd vs Pugd Radius cosine loss **(d)**: Pugd vs Pugd Radius cosine accuracy **(e)**: Pugd Radius sine vs cosine loss **(f)**: Pugd Radius sine vs cosine accuracy. The epochs ranged from 21 to 400, cosine used β_{min} as 0 and β_{max} as 1, sine used β_{min} as 0 and β_{max} as 2.

As shown in fig. 5, (a) and (b) showed that

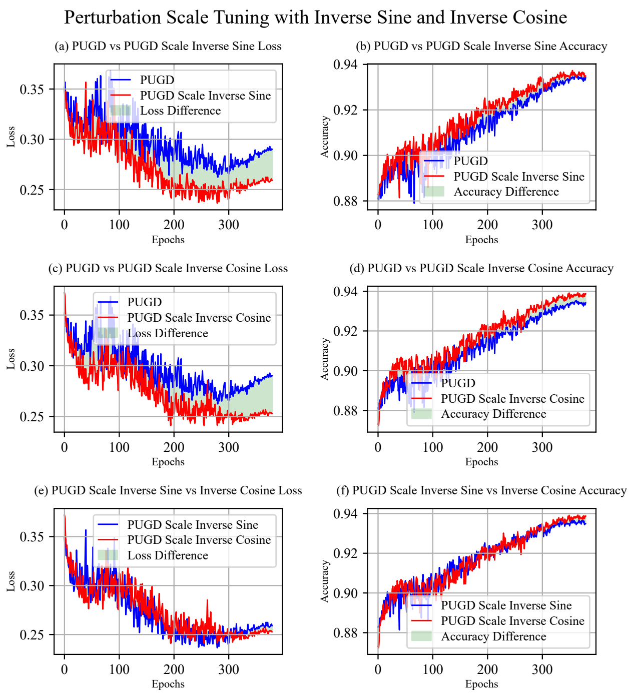


Figure 6: Perturbation Scale Tuning with Inverse Sine and Inverse Cosine **(a)**: Pugd vs Pugd Scale inverse sine loss **(b)**: Pugd vs Pugd Scale inverse sine accuracy **(c)**: Pugd vs Pugd Scale inverse cosine loss **(d)**: Pugd vs Pugd Scale inverse cosine accuracy **(e)**: Pugd Scale inverse sine vs inverse cosine loss **(f)**: Pugd Scale inverse sine vs inverse cosine accuracy The epochs ranged from 21 to 400, inverse cosine used β_{min} as 1 and β_{max} as 2, inverse sine used β_{min} as 0.8 and β_{max} as 3.

4.1.2 Timing of application

4.1.3 Scale of gradient Tunning

4.2. Radius-Timing Scale(RTS)

4.3. Finetuning comparison

5. Conclusions

This article didn't aims to provide the best parameters, but to provide the enhancement methods RTS

References

[1] Yingbin Bai, Erkun Yang, Bo Han, Yanhua Yang, Jiatong Li, Yinian Mao, Gang Niu, and Tongliang Liu. Understanding and improving early stopping for learning with noisy labels, 2021. 6

[2] Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-sgd: Biasing gradient descent into wide valleys, 2017. 3

[3] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization, 2021. 1, 2, 3, 4

[4] Michael Gastpar, Ido Nachum, Jonathan Shafer, and Thomas Weinberger. Fantastic generalization measures are nowhere to be found, 2023. 5

[5] Takashi Ishida, Ikko Yamane, Tomoya Sakai, Gang Niu, and Masashi Sugiyama. Do we need zero training loss after achieving zero training error?, 2021. 6

[6] Stanislaw Jastrzebski, Maciej Szymczak, Stanislav Fort, Devansh Arpit, Jacek Tabor, Kyunghyun Cho, and Krzysztof Geras. The break-even point on optimization trajectories of deep neural networks, 2020. 2, 3

[7] Ziwei Ji and Matus Telgarsky. Directional convergence and alignment in deep learning, 2020. 2

[8] Yiding Jiang, Behnam Neyshabur, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic generalization measures and where to find them, 2019. 2

[9] Kenji Kawaguchi. Deep learning without poor local minima, 2016. 1

[10] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima, 2017. 2

[11] Jungmin Kwon, Jeongseop Kim, Hyunseo Park, and In Kwon Choi. Asam: Adaptive sharpness-aware minimization for scale-invariant learning of deep neural networks, 2021. 3

[12] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets, 2018. 3

[13] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2018. 6

[14] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017. 1, 4, 5

[15] Dougal Maclaurin, David Duvenaud, and Ryan P. Adams. Early stopping is nonparametric variational inference, 2015. 4

[16] Ryan Murray, Brian Swenson, and Soummya Kar. Revisiting normalized gradient descent: Fast evasion of saddle points, 2018. 2

[17] Behnam Neyshabur, Ruslan Salakhutdinov, and Nathan Srebro. Path-sgd: Path-normalized optimization in deep neural networks, 2015. 2

[18] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017. 1

[19] Naresh K. Sinha and Michael P. Griscik. A stochastic approximation method. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-1(4):338–344, 1971. 1, 2, 4

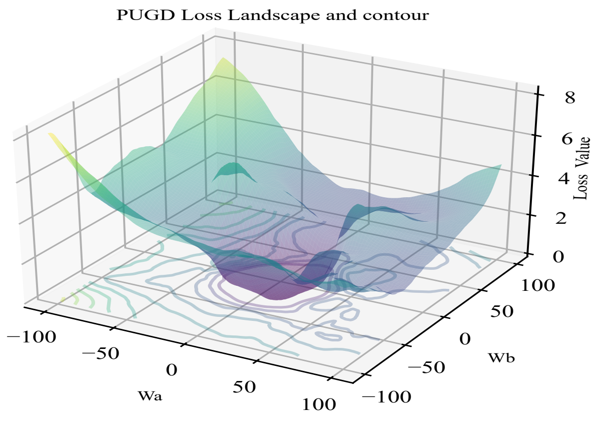
[20] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015. 6

[21] Ching-Hsun Tseng, Hsueh-Cheng Liu, Shin-Jye Lee, and Xiaojun Zeng. Perturbed gradients updating within unit space for deep learning. In *2022 International Joint Conference on Neural Networks (IJCNN)*, page 01–08. IEEE, July 2022. 1, 2, 3

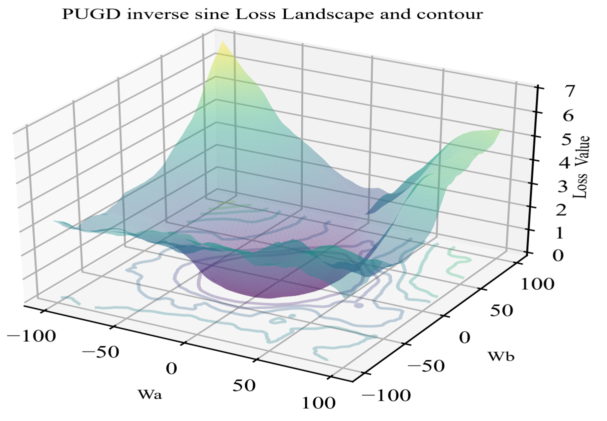
[22] Yaowei Zheng, Richong Zhang, and Yongyi Mao. Regularizing neural networks via adversarial model perturbation, 2021. 2

A. Appendix

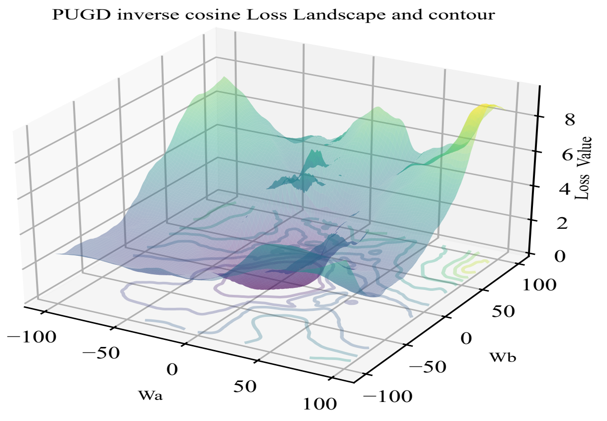
A.1. Loss landscape and contour of PUGD Radius



(a) Loss landscape of PUGD



(b) PUGD Inverse sine



(c) PUGD Inverse cosine

Figure 7: Loss landscape and contour of PUGD Radius

A.2. Comparative analysis of optimization strategies for the CIFAR-10 dataset

Methods	β_{min}	β_{max}	Best Accuracy
PUGD	None	None	**0.9353**
Radius inverse cosine	0.01	2	**0.9419**
Radius inverse sine	0.01	2	**0.9417**
Radius cosine	0.0	1.0	**0.9398**
Radius sine	0.0	2.0	**0.938**
Scale inverse cosine	1	2	**0.9392**
Scale inverse sine	0.8	3	**0.9373**
Scale cosine			
Scale sine	0.0	2	**0.9381**

Table 1: The best accuracy of different strategies for Radius and Scale Tunning

A.3. Comparative analysis of optimization strategies for the CIFAR-100 dataset

Methods	β_{min}	β_{max}	Best Accuracy
PUGD	None	None	**0.7188**
Radius inverse cosine			
Radius inverse sine	0.01	2	**0.7315**
Radius cosine			
Radius sine	0.0	1.0	**0.7199**
Scale inverse cosine	0.8	3	**0.7222**
Scale inverse sine	0.1	2	**0.7267**
Scale cosine	0.0	1.0	**0.7209**
Scale sine			

Table 2: The best accuracy of different strategies for Radius and Scale Tunning