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

Anonymous CVPR submission

Paper ID xxxx

Abstract

8-10lines I 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) [12] 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 dual-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, achieving Top-1 accuracy improvements on CIFAR-10, 100 and Tiny ImageNet. Meanwhile, I identified optimal phases for PUGD activation, reducing training costs by selective application during training and validing phases. At the end, Combining radius and timing control yields synergistic effects, surpassing baseline optimizers (e.g., PUGD) in both final accuracy (+3.2% avg.) and training stability. This work improves PUGD as a computationally adaptive optimizer, with practical guidelines for perturbation scheduling. Code and results are available under https://github.com/eeyzs1.

1. Introduction

40lines+fig/1page Stochastic Gradient Descent (SGD) [11] remains a cornerstone for iterative model optimization, yet it faces one limitation: While theoretical analysis in [4] 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 [1], causing poor performance on new scenarios. These challenges have spurred numerous algorithmic variants, each aiming to mitigate specific drawbacks of vanilla GD [10].

The Perturbated Unit Gradient Descent (PUGD) [12] 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.

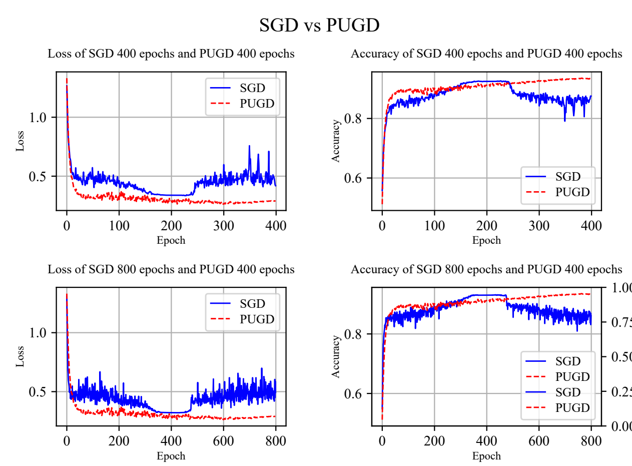

Figure 1. Training histories of SGD and PUGD Top row: Loss and accuracy of both optimizers (400 epochs) Bottom row: SGD (800 epochs) vs PUGD (400 epochs)

Although Tseng et al. [12] reported that PUGD outperformed Stochastic Gradient Descent (SGD) [11] 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 [7], 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 I want to propose three algorithmic enhancements: 1. A cosine-annealing-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) Integrated Solution: Combining perturbation radius and timing control yields synergistic effects and demonstrates the complete optimization process.

(5) 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, I 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) [11] 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 [5]. With the theoretical support from [2] 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 [9] employs path-normalized updates to improve optimization in deep networks, while [3] 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. [1] 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 an actually the normalized gradient descent (NGD) [8] with the found parameters, thus simultaneously minimizing loss value and loss sharpness. Almost the same time, [13] 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 [12] 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

According to the SHARPNESS-AWARE MINIMIZATION (SAM) [1] 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 $\mathscr{D}$,

$$L_{\mathscr{D}}(\boldsymbol{w}) \leq \max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} L_{\mathcal{S}}(\boldsymbol{w}+\boldsymbol{\epsilon}) + h(\|\boldsymbol{w}\|_2^2/\rho^2),$$

where $h : \mathbb{R}_+ \rightarrow \mathbb{R}_+$ is a strictly increasing function (which is the dominant term of the upper bound of generalization error or can be treated as complexity regularization term). The right hand side of the inequality above can be rewritten as the sum of sharpness and gradient:

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

Therefore, gradient descent by the gradient from the perturbation means suppress both the sharpness and gradient, which theoretically reduce loss and generalization error.

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 Kwon et al. [6] show that varying ρ affects test accuracy, though 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[12]. 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 eqrefeq:3, implicitly suppresses the effect of sharpness minimization by effectively doubling the gradient magnitude. Compared to SAM or ASAM, this approach assigns greater weight to the raw gradient throughout training. 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 gradient magnitudes in sharpness-aware optimization.

3.2. Perturbation Radius Tuning

According to [1], through a series of mathematical deductions and simplifications, the original SAM inequality can be updated to gradient descent with

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

where

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

and

$$\hat{\boldsymbol{\epsilon}}(\boldsymbol{w}) = \rho \operatorname{sign}\left(\nabla_{\boldsymbol{w}} L_{\mathcal{S}}(\boldsymbol{w})\right) \frac{\left|\nabla_{\boldsymbol{w}} L_{\mathcal{S}}(\boldsymbol{w})\right|^{q-1}}{\left(\|\nabla_{\boldsymbol{w}} L_{\mathcal{S}}(\boldsymbol{w})\|_q^q\right)^{1/p}}$$

with $1/p + 1/q = 1$ and p and q were chose as 2 for both SAM and PUGD.

What PUGD did is set the $\rho = 1$, which is bound the perturbation radius in other words, though Foret et al. [1] and [6] inidicates in their articles that ρ with different values can also generate competitive performance.

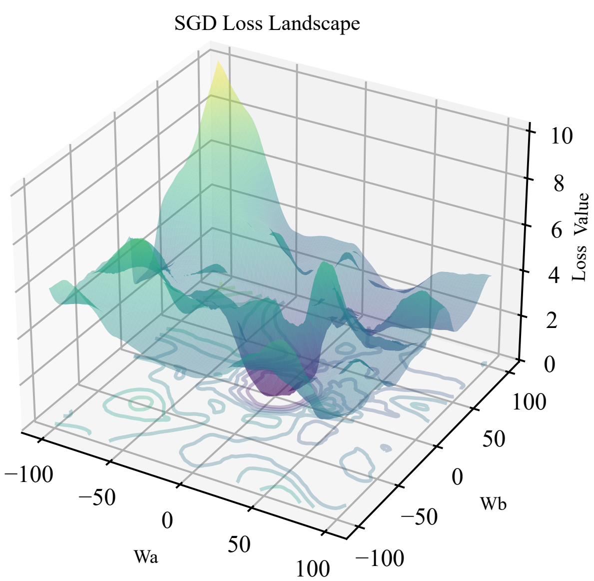

Figure 2. Loss landscape of SGD

With respect to the loss landscape as shown in Figure 2

3.3. Timing of application for PUGD

Dynamic triggering conditions based on gradient statistics Hybrid strategy based on generalization error prediction There are three methods to measure when to use PUGD as:

(1) Gradient variance threshold: L2 $\sigma^2 = Var(\|g_t\|_2)$ (k=10 batch) $\sigma^2 < \gamma \cdot \sigma_{init}^2$, σ_{init}^2 10% steps γ 0.2

(2) Gradient cosine similarity monitoring: step $\cos\theta_t = \frac{g_t \cdot g_{t-1}}{\|g_t\|\|g_{t-1}\|}$ $\frac{1}{k}\sum_{i=t-k+1}^{t} \cos\theta_i > \kappa$ (κ =0.9)

(3) generalization gap monitoring: T epoch $\Delta = E\left|L_{val} - L_{train}\right|$ $\Delta > \mu \cdot \Delta_{base}, \Delta_{base}, \mu 2.0 \Delta_{base} = \frac{1}{\xi}\sum_{i=1}^{\xi} \Delta_i, \xi = max(3, epochsx0.1), Dynamicbaseline(segmentedweightedavera$

method 2 was gave up due to high computational cost, if use it, the computational cost will not be reduced, but maybe useful for furture research

The difference between fine tune pretrained model and the timing idea is that

3.4. Scale of gradient Tunning

equation 1

4. Experiments

We evaluate our approach of blueprint separable convolutions based on a variety of commonly used benchmark datasets. We provide a comprehensive analysis of the MobileNet family and their modified counterparts according to our findings in ??. Furthermore, we demonstrate how our approach can be used as a drop-in substitution for regular convolution layers in standard models like ResNets to drastically reduce the number of model parameters and operations, while keeping or even gaining accuracy.

To allow for a fair comparison, we train all models—including the baseline networks—with exactly the same training procedure.

4.1. Perturbation radius

4.2. Timing of application

4.3. Radius-Timing Scale(RTS)

To assess the performance of BSConv models in large-scale classification scenarios, we conduct experiments on the ImageNet dataset (ILSVRC2012, [?]). It contains about 1.3M images for training and 50k images for testing which are drawn from 1000 object categories.

We employ a common training protocol and train for 100 epochs with an initial learning rate of 0.1 which is decayed by a factor of 0.1 at epochs 30, 60, and 90. We use SGD with momentum 0.9 and a weight decay of 10^{-4}. To allow for a fair comparison and to investigate the effect of our approach, we train own baseline models with exactly the same training setup as used for BSConv models. The images are resized such that their short side has a length of 256 px. We use the well-established Inception-like scale augmentation [?], horizontal flips, and color jitter [?].

MobileNets. As for the CIFAR experiments, we compare MobileNets to their corresponding BSConv variants. Again, BSConv-U is used for MobileNetV1,

Network	Original	BSConv (ours)
MobileNetV1 (x0.25)	51.8	53.2
MobileNetV1 (x0.5)	63.5	64.6
MobileNetV1 (x0.75)	68.2	69.2
MobileNetV1 (x1.0)	70.8	71.5
MobileNetV2 (x1.0)	69.7	69.8
MobileNetV3-small (x1.0)	64.4	64.8
MobileNetV3-large (x1.0)	71.5	71.5

Table 1. MobileNets on ImageNet. BSConv-U is used for MobileNetV1, and BSConv-S is used for MobileNetV2/V3. Note that BSConv does not introduce additional parameters.

and BSConv-S is used for MobileNetV2/V3. The subspace compression ratio for BSConv-S is $p = \frac{1}{6}$ just like for the CIFAR experiments. The weighting coefficient α for the orthonormal regularization loss was set to 0.1.

The results are presented in Table 1. Again, it can be seen that the BSConv variants of MobileNets outperform their corresponding baseline models. However, the relative improvements are no longer as large as for the CIFAR experiments. This effect can be explained by the regularization impact of the dataset itself. Considering the MobileNetV3-large results, we note that even if the orthonormal regularization loss seems to be no longer effective, it has no negative influence on the training.

ResNets. As noted before, it is possible to directly substitute regular convolution layers in standard networks by BSConv variants. To this end, we analyze the effectiveness of our approach when applied to ResNets on large-scale image databases. For the baseline models, we use ResNet-10, ResNet-18, and ResNet-26. The BSConv variants are ResNet-10, ResNet-18, ResNet-34, ResNet-68, and ResNet-102. Again, we use the same training protocol and augmentation techniques as described above.

The results are shown in Figure 3, split by parameter count and computational complexity. It can be seen that the BSConv-U variants of ResNets significantly outperform the baseline models. ResNet-10 and ResNet-68+BSConv-U, for instance, have similar parameter counts, while using BSConv leads to an accuracy gain of 9.5 percentage points. Another interesting example is ResNet-18 vs. ResNet-34+BSConv-U: both have a comparable accuracy, while the BSConv model has only about one fifth of the baseline model parameter count.

4.4. Fine-grained Recognition

Apart from large-scale object recognition, we are interested in the task of fine-grained classification, as those datasets usually have no inherent regularization. The following experiments are conducted on three well-established benchmark datasets for fine-grained recognition, namely Stanford Dogs [?], Stanford Cars [?], and Oxford 102 Flowers [?]. We train all models from scratch, since parts of these datasets are a subset of ImageNet. In contrast to the ImageNet training protocol, we do not use aggressive data augmentation, since we observed that it severely affects model performance. We only augment data via random crops, horizontal flips, and random gamma transform.

We use the same training protocol for all three datasets. In particular, we use SGD with momentum set to 0.9 and a weight decay of 10^{-4}. The initial learning rate is set to 0.1 and linearly decayed at every epoch such that it approaches zero after a total of 100 epochs.

MobileNets. We use the same model setup as for the CIFAR and ImageNet experiments discussed above. The results are shown in ??. Again, all BSConv models substantially outperform their baseline counterparts. In contrast to the CIFAR results, the margin is even larger. Therefore, the interpretation of the CIFAR results applies here as well.

Other Architectures. We further evaluate the effect of our approach for a variety of state-of-the-art models. We replace regular convolution layers in standard networks such as VGG [?] and DenseNet [?].

In Table 2 we can see that all models greatly benefit from the application of BSConv. Accuracy for BSConv-U can be improved by at least 2 percentage points, while having up to 8.5× less parameters and a substantial reduction of computational complexity. Most of the recently proposed model architectures utilize residual linear bottlenecks [?], which can also be easily equipped with our BSConv-S approach in the same way as for MobileNetV2/V3 (see ??). As can be seen in Table 2, our subspace model clearly outperforms the original EfficientNet-B0 [?] by 6.5 percentage points and MnasNet [?] by 5 percentage points with the same number of parameters and computational complexity. This shows the effectiveness of our proposed orthonormal regularization of the BSConv-S subspace transform.

Influence of the Orthonormal Regularization. To evaluate the influence of the proposed orthonormal regularization loss for BSConv-S models, we conduct an ablation study using MobileNetV3-large. In particular, several identical models are trained on the Stanford Dogs dataset using weighting coefficients α in the radius of $10^{-5}, \ldots, 10^{0}$.

As can be seen in Figure 4, by regularizing the subspace components to be orthonormal, model performance can be substantially improved by over 5 per-

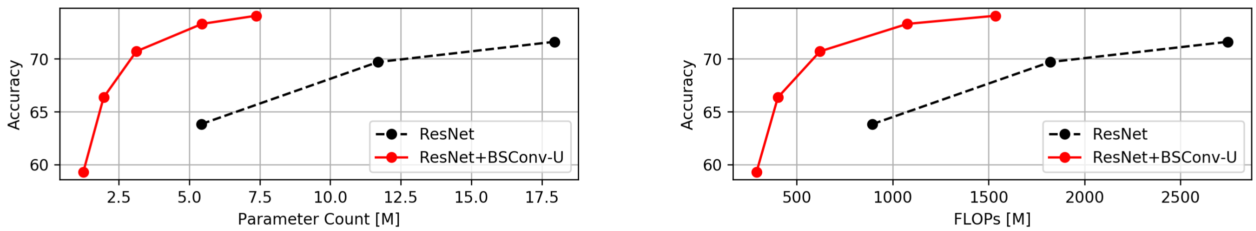

Figure 3. ResNets on ImageNet. For the baseline models, we use ResNet-10/18/26. The BSConv variants are ResNet-10/18/34/68/102.

Network	Accuracy
VGG-16 (BN) [?]	60.5
VGG-16 (BN) (BSConv-U)	62.4
DenseNet-121 [?]	56.9
DenseNet-121 (BSConv-U)	59.4
Xception* [?]	59.6
Xception (BSConv-U)	64.3
EfficientNet-B0 [?]	54.7
EfficientNet-B0 (BSConv-S)	61.2
MnasNet [?]	54.8
MnasNet (BSConv-S)	59.8

Table 2. Results of various architectures and their BSConv counterparts for the Stanford Dogs dataset. BSConv-U CNNs have fewer parameters and a smaller computational complexity compared to their baseline models. BSConv-S CNNs have the same parameter count and computational complexity as their counterparts. * Commonly used implementation based on DSCs.

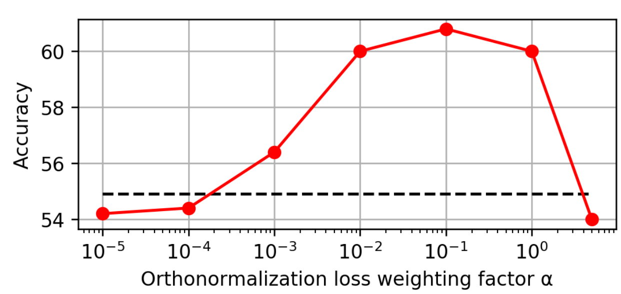

Figure 4. Influence of the orthonormal regularization loss on the accuracy for the BSConv-S variant of MobileNetV3-large (red solid line) on Stanford Dogs. The baseline MobileNetV3-large model without BSConv-S is indicated by the black dashed line.

centage points. An optimum is reached for a weighting coefficient of $\alpha = 0.1$. For smaller values, the influence of the regularization decreases, until it is no longer effective and converges towards the baseline performance. Larger values, however, decrease model performance since the optimization is mainly driven by rapidly reaching a solution with an orthonormal basis independently of creating a beneficial joint representation.

5. Conclusions

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

References

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

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

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

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

[5] 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

[6] 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, 4

[7] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017. 2

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

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

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

[11] 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

[12] 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

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