**ADAMP: SLOWING DOWN THE SLOWDOWN FOR MOMENTUM OPTIMIZERS ON SCALE-INVARINT WEIGHTS**

Byeongho Heo\(^{*}\), Sanghyuk Chun\(^{*},1,2\)  Seong Joon Oh\(^1\)  Dongyoon Han\(^1\)
Sangdoon Yun\(^3\)  Gyuwan Kim\(^1,2\)  Youngjung Uh\(^3,†\)  Jung-Woo Ha\(^1,2\)

Equal contribution *, Works done at NAVER AI LAB \(^1\),
NAVER AI LAB \(^1\), NAVER CLOVA \(^2\)
Applied Information Engineering, Yonsei University \(^3\)

**ABSTRACT**

Normalization techniques, such as batch normalization (BN), are a boon for modern deep learning. They let weights converge more quickly with often better generalization performances. It has been argued that the normalization-induced scale invariance among the weights provides an advantageous ground for gradient descent (GD) optimizers: the effective step sizes are automatically reduced over time, stabilizing the overall training procedure. It is often overlooked, however, that the additional introduction of momentum in GD optimizers results in a far more rapid reduction in effective step sizes for scale-invariant weights, a phenomenon that has not yet been studied and may have caused unwanted side effects in the current practice. This is a crucial issue because arguably the vast majority of modern deep neural networks consist of (1) momentum-based GD (e.g. SGD or Adam) and (2) scale-invariant parameters (e.g. more than 90% of the weights in ResNet are scale-invariant due to BN). In this paper, we verify that the widely-adopted combination of the two ingredients lead to the premature decay of effective step sizes and sub-optimal model performances. We propose a simple and effective remedy, SGDP and AdamP: get rid of the radial component, or the norm-increasing direction, at each optimizer step. Because of the scale invariance, this modification only alters the effective step sizes without changing the effective update directions, thus enjoying the original convergence properties of GD optimizers. Given the ubiquity of momentum GD and scale invariance in machine learning, we have evaluated our methods against the baselines on 13 benchmarks. They range from vision tasks like classification (e.g. ImageNet), retrieval (e.g. CUB and SOP), and detection (e.g. COCO) to language modelling (e.g. WikiText) and audio classification (e.g. DCASE) tasks. We verify that our solution brings about uniform gains in performances in those benchmarks. Source code is available at https://github.com/clovaai/AdamP.

1 **INTRODUCTION**

Normalization techniques, such as batch normalization (BN) (Ioffe & Szegedy, 2015), layer normalization (LN) (Ba et al., 2016), instance normalization (IN) (Ulyanov et al., 2016), and group normalization (GN) (Wu & He, 2018), have become standard tools for training deep neural network models. Originally proposed to reduce the internal covariate shift (Ioffe & Szegedy, 2015), normalization methods have proven to encourage several desirable properties in deep neural networks, such as better generalization (Santurkar et al., 2018) and the scale invariance (Hoffer et al., 2018).

Prior studies have observed that the normalization-induced scale invariance of weights stabilizes the convergence for the neural network training (Hoffer et al., 2018; Arora et al., 2019; Kohler et al., 2019; Dukler et al., 2020). We provide a sketch of the argument here. Given weights \( w \) and an input \( x \), we observe that the normalization makes the weights become scale-invariant:

\[
\text{Norm}(w ^ \top x) = \text{Norm}(cw ^ \top x) \quad \forall c > 0.
\]

The resulting equivalence relation among the weights lets us consider the weights only in terms of their \( \ell_2 \)-normalized vectors \( \bar{w} := \frac{w}{\|w\|_2} \) on the sphere \( S^{d-1} = \{v \in \mathbb{R}^d : \|v\|_2 = 1\} \). We refer to
\( S^{d-1} \) as the effective space, as opposed to the nominal space \( \mathbb{R}^d \) where the actual optimization algorithms operate. The mismatch between these spaces results in the discrepancy between the gradient descent steps on \( \mathbb{R}^d \) and their effective steps on \( S^{d-1} \). Specifically, for the gradient descent updates, the effective step sizes \( \| \Delta \hat{w}_{t+1} \|_2 := \| \hat{w}_{t+1} - \hat{w}_t \|_2 \) are the scaled versions of the nominal step sizes \( \| \Delta w_{t+1} \|_2 := \| w_{t+1} - w_t \|_2 \) by the factor \( \frac{1}{\| w_t \|_2} \) (Hoffer et al., 2018). Since \( \| w_t \|_2 \) increases during training (Soudry et al., 2018; Arora et al., 2019), the effective step sizes \( \| \Delta \hat{w}_t \|_2 \) decrease as the optimization progresses. The automatic decrease in step sizes stabilizes the convergence of gradient descent algorithms applied on models with normalization layers: even if the nominal learning rate is set to a constant, the theoretically optimal convergence rate is guaranteed (Arora et al., 2019).

In this work, we show that the widely used momentum-based gradient descent optimizers (e.g. SGD and Adam (Kingma & Ba, 2015)) decreases the effective step size \( \Delta \hat{w}_t \) even more rapidly than the momentum-less counterparts considered in (Arora et al., 2019). This leads to a slower effective convergence for \( \hat{w}_t \) and potentially sub-optimal model performances. We illustrate this effect on a 2D toy optimization problem in Figure 1. Compared to “GD”, “GD+momentum” speeds up the norm growth, resulting in a slower effective convergence in \( S^1 \), though being faster in the nominal space \( \mathbb{R}^2 \). This phenomenon is not confined to the toy setup, for example, 95.5% and 91.8% of the parameters of the widely-used ResNet18 and ResNet50 (He et al., 2016) are scale-invariant due to BN. The majority of deep models nowadays are trained with SGD or Adam with momentum. And yet, our paper is first to delve into the issue in the widely-used combination of scale-invariant parameters and momentum-based optimizers.

We propose a simple solution to slow down the decay of effective step sizes while maintaining the step directions of the original optimizer in the effective space. At each iteration of a momentum-based gradient descent optimizer, we propose to project out the radial component (i.e. component parallel to \( w \)) from the update, thereby reducing the increase in the weight norm over time. Because of the scale invariance, the procedure does not alter the update direction in the effective space; it only changes the effective step sizes. We can observe the benefit of our optimizer in the toy setting in Figure 1. “Ours” suppresses the norm growth and thus slows down the effective learning rate decay, allowing the momentum-accelerated convergence in \( \mathbb{R}^2 \) to be transferred to the actual space \( S^1 \). “Ours” converges most quickly and achieves the best terminal objective value.

The projection algorithm is simple and readily applicable to various optimizers for deep neural networks. We apply this technique on SGD and Adam (SGDP and AdamP, respectively) and verify the slower decay of effective learning rates as well as the resulting performance boosts over a diverse set of practical machine learning tasks including image classification, image retrieval, object detection, robustness benchmarks, audio classification, and language modelling.

2 Problem

Widely-used normalization techniques (Ioffe & Szegedy, 2015; Salimans & Kingma, 2016; Ba et al., 2016; Ulyanov et al., 2016; Wu & He, 2018) in deep networks result in the scale invariance for weights. We show that the introduction of momentum in gradient-descent (GD) optimizers, when applied on such scale-invariant parameters, decreases the effective learning rate much more rapidly. This phenomenon has not yet been studied in literature, despite its ubiquity. We suspect the resulting early convergence may have introduced sub-optimality in many SGD and Adam-trained models across machine learning tasks. The analysis motivates our optimizer in §3.
2.1 Normalization Layer and Scale Invariance

For a tensor \( x \in \mathbb{R}^{n_1 \times \cdots \times n_r} \) of rank \( r \), we define the normalization operation along the axes \( k \in \{1, \ldots, r\} \) as \( \text{Norm}_k(x) = \frac{x - \mu_k(x)}{\sigma_k(x)} \) where \( \mu_k, \sigma_k \) are the mean and standard deviation functions along the axes \( k \), without axes reduction (to allow broadcasted operations with \( x \)). Depending on \( k \), \( \text{Norm}_k \) includes special cases like batch normalization (BN) (Ioffe & Szegedy, 2015).

For a function \( g(u) \), we say that \( g \) is scale invariant if \( g(cu) = g(u) \) for any \( c > 0 \). We then observe that \( \text{Norm}(\cdot) \) is scale invariant. In particular, under the context of neural networks,

\[
\text{Norm}(w^\top x) = \text{Norm}((cw)^\top x)
\]

for any \( c > 0 \), leading to the scale invariance against the weights \( w \) preceding the normalization layer. The norm of such weights \( \|w\|_2 \) does not affect the forward \( f_w(x) \) or the backward \( \nabla_w f_w(x) \) computations of a neural network layer \( f_w \) parameterized by \( w \). We may represent the scale-invariant weights via their \( \ell_2 \)-normalized vectors \( \tilde{w} := \frac{w}{\|w\|_2} \in \mathbb{S}^{d-1} \) (i.e. \( c = \frac{1}{\|w\|_2} \)).

2.2 Notations for the Optimization Steps

See the illustration on the right for the summary of notations describing an optimization step. We write a gradient descent (GD) algorithm as:

\[
w_{t+1} \leftarrow w_t - \eta p_t
\]

where \( \eta > 0 \) is the user-defined learning rate. The norm of the difference \( \|\Delta w_{t+1}\|_2 := \|w_{t+1} - w_t\|_2 = \eta \|p_t\|_2 \) is referred to as the step size.

When \( p = \nabla_w f(w) \), equation 3 is the vanilla GD algorithm. Momentum-based variants have more complex forms for \( p \).

In this work, we study the optimization problem in terms of the \( \ell_2 \) normalized weights in \( \mathbb{S}^{d-1} \), as opposed to the nominal space \( \mathbb{R}^d \). As the result of equation 3, an effective optimization step takes place in \( \mathbb{S}^{d-1} \):

\[ \Delta \tilde{w}_{t+1} := \tilde{w}_{t+1} - \tilde{w}_t \]

We refer to the effective step size \( \|\Delta \tilde{w}_{t+1}\|_2 \).

2.3 Effective Step Sizes for Vanilla Gradient Descent (GD)

We approximate the effective step sizes for the scale-invariant \( w \) under the vanilla GD algorithm.

We observe that the scale invariance \( f(cw) = f(w) \) leads to the orthogonality:

\[
0 = \frac{\partial f(cw)}{\partial c} = w^\top \nabla_w f(w).
\]

For example, the vanilla GD update step \( p = \nabla_w f(w) \) is always perpendicular to \( w \). Based on this, we establish the effective step size for \( w \) on \( \mathbb{S}^{d-1} \):

\[
\|\Delta \tilde{w}_{t+1}\|_2 := \left\| \frac{w_{t+1} - w_t}{\|w_{t+1}\|_2} \right\|_2 \approx \left\| \frac{w_{t+1} - w_t}{\|w_{t+1}\|_2} - \frac{w_t}{\|w_t\|_2} \right\|_2 = \frac{\|\Delta w_{t+1}\|_2}{\|w_{t+1}\|_2}
\]

where the approximation assumes \( \frac{1}{\|w_{t+1}\|_2^2} - \frac{1}{\|w_t\|_2^2} = o(\eta) \), which holds when \( p_t \perp w_t \) as in the vanilla GD. We have thus derived that the effective step size on \( \mathbb{S}^{d-1} \) is inversely proportional to the weight norm, in line with the results in Hoffer et al. (2018).

Having established the relationship between the effective step sizes and the weight norm of a scale-invariant parameters (s.i.p), we derive the formula for its growth under the vanilla GD optimization.

**Lemma 2.1 (Norm growth by GD).** For a s.i.p. \( w \) and the vanilla GD, where \( p_t = \nabla_w f(w_t) \),

\[
\|w_{t+1}\|_2^2 = \|w_t\|_2^2 + \eta^2 \|p_t\|_2^2.
\]

The lemma follows from the orthogonality in equation 4. It follows that the norm of a scale-invariant parameter \( \|w\|_2 \) is monotonically increasing and consequently decreases the effective step size for \( w \). Arora et al. (2019) has further shown that GD with the above adaptive step sizes converges to a stationary point at the theoretically optimal convergence rate \( O(T^{-1/2}) \) under a fixed learning rate.
2.4 RAPID DECAY OF EFFECTIVE STEP SIZES FOR MOMENTUM-BASED GD

Momentum is designed to accelerate the convergence of gradient-based optimization by letting \( \mathbf{w} \) escape high-curvature regions and cope with small and noisy gradients. It has become an indispensable ingredient for training modern deep neural networks. A momentum update follows:

\[
\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta \mathbf{p}_t, \quad \mathbf{p}_t \leftarrow \beta \mathbf{p}_{t-1} + \nabla_{\mathbf{w}} f(\mathbf{w}_t)
\]

(7)

for steps \( t \geq 0 \), where \( \beta \in (0, 1) \) and \( \mathbf{p}_0 \) is initialized at \( \mathbf{0} \). Note that the step direction \( \mathbf{p}_t \) and the parameter \( \mathbf{w}_t \) may not be perpendicular anymore. We show below that momentum increases the weight norm under the scale invariance, even more so than does the vanilla GD.

**Lemma 2.2** (Norm growth by momentum). For a s.i.p. \( \mathbf{w} \) updated via equation 7, we have

\[
\| \mathbf{w}_{t+1} \|^2 = \| \mathbf{w}_t \|^2 + \eta^2 \| \mathbf{p}_t \|^2 + 2\eta^2 \sum_{k=0}^{t-1} \beta^{t-k} \| \mathbf{p}_k \|^2.
\]

(8)

Proof is in the Appendix §A. Comparing Lemma 2.1 and 2.2, we notice that the formulation is identical, except for the last term on the right hand side of Lemma 2.2. This term is not only non-negative, but also is an accumulation of the past updates. This additional term results in the significantly accelerated increase of weight norms when the momentum is used. We derive a more precise asymptotic ratio of the weight norms for the GD with and without momentum below.

**Corollary 2.3** (Asymptotic norm growth comparison). Let \( \| \mathbf{w}^G_t \|_2 \) and \( \| \mathbf{w}^{\text{GDM}}_t \|_2 \) be the weight norms at step \( t \geq 0 \), following the recursive formula in Lemma 2.1 and 2.2, respectively. We assume that the norms of the updates \( \| \mathbf{p}_t \|_2 \) for GD with and without momentum are identical for every \( t \geq 0 \). We further assume that the sum of the update norms is non-zero and bounded: \( 0 < \sum_{t \geq 0} \| \mathbf{p}_t \|^2 < \infty \). Then, the asymptotic ratio between the two norms is given by:

\[
\lim_{t \to \infty} \frac{\| \mathbf{w}^\text{GDM}_t \|^2 - \| \mathbf{w}_0 \|^2}{\| \mathbf{w}^G_t \|^2 - \| \mathbf{w}_0 \|^2} = 1 + \frac{2\beta}{1 - \beta} \quad \text{as} \quad t \to \infty.
\]

(9)

Proof in the Appendix §A. While the identity assumption for \( \| \mathbf{p}_t \|_2 \) between GD with and without momentum is strong, the theory is designed to illustrate an approximate norm growth ratios between the algorithms. For a popular choice of \( \beta = 0.9 \), the factor is as high as \( 1 + 2\beta/(1 - \beta) = 19 \). Our observations are also applicable to Nesterov momentum and momentum-based adaptive optimizers like Adam. We later verify that the momentum induce the increase in weight norms and thus rapidly reduces the effective learning rates in many realistic setups of practical relevance (§3.2 and §4).

3 METHOD

We have studied the accelerated decay of effective learning rates for scale-invariant weights (e.g. those preceding a normalization layer) under the momentum. In this section, we propose a projection-based solution that prevents the momentum-induced effective step size decreases while not changing the update directions in the effective weight space \( \mathbb{S}^{d-1} \).

3.1 OUR METHOD: PROJECTED UPDATES

We remove the accumulated error term in Lemma 2.2, while retaining the benefits of momentum, through a simple modification. Let \( \Pi_w(\cdot) \) be a projection onto the tangent space of \( \mathbf{w} \):

\[
\Pi_w(\mathbf{x}) := \mathbf{x} - (\mathbf{w} \cdot \mathbf{x}) \mathbf{w}.
\]

(10)

We apply \( \Pi_w(\cdot) \) to the momentum update \( \mathbf{p} \) (equation 7) to remove the radial component, which accumulates the weight norms without contributing to the optimization. Our modified update rule is:

\[
\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \mathbf{q}_t,
\]

\[
\mathbf{q}_t = \begin{cases} 
\Pi_w(\mathbf{p}_t) & \text{if } \cos(\mathbf{w}_t, \nabla_{\mathbf{w}} f(\mathbf{w}_t)) < \delta / \sqrt{\dim(\mathbf{w})} \\
\mathbf{p}_t & \text{otherwise}
\end{cases}
\]

(11)
where \( \cos(a, b) := \frac{a^T b}{\|a\| \|b\|} \) is the cosine similarity. Instead of manually registering weights preceding normalization layers, our algorithm automatically detects scale invariances with the cosine similarity for user convenience. In all experiments considered, we found \( \delta = 0.1 \) to be sufficiently small to precisely detect orthogonality and sufficiently large to recall all scale-invariant weights (Appendix §C): we suggest future users to use the same value. The proposed update rule makes a scale-invariant parameter \( w \) perpendicular to its update step \( q \). It follows then that the rapid weight norm accumulation shown in Lemma 2.2 is alleviated back to the vanilla gradient descent growth rate in Lemma 2.1 due to the orthogonality:
\[
\|w_{t+1}\|^2 = \|w_t\|^2 + \eta^2\|q_t\|^2.
\]

We emphasize that this modification only alters the effective learning rate while not changing the effective update directions, as shown in the below proposition.

**Proposition 3.1** (Effective update direction after projection). Let \( w^e_{t+1} := w_t - \eta p_t \) and \( w^p_{t+1} := w_t - \eta \Pi_w(p_t) \) be original and projected updates, respectively. Then, the effective update after the projection \( w^p_{t+1} \) lies on the geodesic on \( S^{d-1} \) defined by \( \tilde{w}_t \) and \( w^p_{t+1} \).

Proof in Appendix. As such, we expect that our algorithm inherits the convergence guarantees of GD. As to the convergence rate, we conjecture that a similar analysis by Arora et al. (2019) holds.

The proposed method is readily adaptable to existing gradient-based optimization algorithms like SGD and Adam. Their modifications, SGDP and AdamP, are shown in Algorithms 1 and 2, respectively ( Modifications are colorized). In practice, we consider two types of scale-invariance: layer-wise (e.g., by LN) and channel-wise (e.g., by BN, IN) invariance. Hence, the weight projection in equation 11 are performed either channel- or layer-wise. Our optimizers incur only 8% extra training time on top of the baselines (ResNet18 on ImageNet classification).

### Algorithm 1: SGDP

**Require:** Learning rate \( \eta > 0 \), momentum \( \beta > 0 \), thresholds \( \delta, \varepsilon > 0 \).

1: while \( w_t \) not converged do
2: \( p_t \leftarrow \beta p_{t-1} + \nabla_w f_t(w_t) \)
3: Compute \( q_t \) with equation 11.
4: \( w_{t+1} \leftarrow w_t - \eta q_t \)
5: end while

### Algorithm 2: AdamP

**Require:** Learning rate \( \eta > 0 \), momentum \( 0 < \beta_1, \beta_2 < 1 \), thresholds \( \delta, \varepsilon > 0 \).

1: while \( w_t \) not converged do
2: \( m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) \nabla_w f_t(w_t) \)
3: \( v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) (\nabla_w f_t(w_t))^2 \)
4: \( p_t \leftarrow m_t / (\sqrt{v_t} + \varepsilon) \)
5: Compute \( q_t \) with equation 11.
6: \( w_{t+1} \leftarrow w_t - \eta q_t \)
7: end while

### 3.2 Empirical Analysis of Effective Step Sizes and the Proposed Projection

So far, we have studied the problem (§2), namely that the momentum accelerates effective step sizes for scale-invariant weights, as well as the corresponding solution (§3.1) only at a conceptual level. Here, we verify that the problem does indeed exist in practice and that our method successfully addresses it on both synthetic and real-world optimization problems.

**Synthetic simulation.** While examples of scale-invariant optimization problems abound in modern deep learning (e.g. BN), they are scant in popular toy optimization objectives designed for sanity checks. We use Rosenbrock function (Rosenbrock, 1960): \( h(x_1, x_2) = (1 - x_1)^2 + 300(x_2 - x_1^2)^2 \) (Figure 3). As our verification requires scale invariance, we define a 3D Rosenbrock function by adding a redundant radial axis \( r \), while treating the original coordinates \((x_1, x_2)\) as the polar angles \((\psi, \phi)\) of the spherical coordinates, resulting in the function \( h(r, \psi, \phi) = h(\psi, \phi) \). It is optimized in the 3D space with the Cartesian coordinates. We describe the full details in Appendix §B.

In Figure 3, we compare the trajectories for optimizers on the spherical coordinates \((\psi, \phi)\). We compare the baseline momentum GD and our projection solution. We additionally examine the impact of weight decay (WD), since a careful choice of WD is another way to regularize the norm growth. We observe that the momentum GD does not converge sufficiently to the optimum. The slowdown is explained by the decreased effective step sizes on \( S^2 \) due to the increase in the parameter norm \((r = 1.0 \rightarrow 3.2)\). Careful tuning of WD partially addresses the slow convergence (momentum GD + WD trajectory) by regularizing the norm growth, but WD is still unable to preclude the initial surge
in the weight norm \(r = 1.0 \rightarrow 2.53\). In practice, addressing the problem with WD is even less attractive because WD is often a sensitive hyperparameter (see the following experiments). On the other hand, our projection solution successfully subdues the weight norm growth \(r = 1.0 \rightarrow 1.12\), ensuring undiminished effective step sizes and a faster convergence to the optimum.

### Real-world experiments

We verify the surge of weight norm and sub-optimality of model performances in momentum-trained deep networks on real-world datasets: ImageNet classification with ResNet-18 and music tagging (Law et al., 2009) with Harmonic CNN (Won et al., 2020). See Table 1 for the analysis. In all experiments, our projection solutions (SGDP, AdamP) restrain the weight norm growth much better than the vanilla momentum methods. For example, Adam-induced norm increase (+4.21) is 13.2 times greater than that of AdamP (+0.32) in the music tagging task. In ImageNet SGD experiments, we observe that a careful choice of weight decay (WD) mitigates the norm increases, but the final norm values and performances are sensitive to the WD value. On the other hand, our SGDP results in stable final norm values and performances across different WD values, even at WD = 0. We observe that under the same learning setup models with smaller terminal weight norms tend to obtain improved performances. Though it is difficult to elicit a causal relationship, we verify in §4 that SGDP and AdamP bring about performance gains in a diverse set of real-world tasks. More analysis around the norm growth and the learning curves are in Appendix §F.

### 4 Experiments

In this section, we demonstrate the effectiveness of our projection module for training scale-invariant weights with momentum-based optimizers. We experiment over various real-world tasks and datasets. From the image domain, we show results on ImageNet classification (§4.1, §D.3, §D.4), object detection (§D.1), and robustness benchmarks (§4.2, §D.2). From the audio domain, we study music tagging, speech recognition, and sound event detection (§4.3). We further show the results when the scale invariance is artificially introduced to a network with no scale-invariant parameters (e.g., Transformer-XL (Dai et al., 2019)) in §4.4. To diversify the root cause of scale invariances, we consider the case where it stems from the \(\ell_2\) projection of the features, as opposed to the statistical
normalization done in e.g. BN, in the image retrieval experiments (§4.5). In the above set of experiments totaling > 10 setups, our proposed modifications (SGDP and AdamP) bring about consistent performance gains against the baselines (SGD (Sutskever et al., 2013) and AdamW (Loshchilov & Hutter, 2019)). We provide the implementation details in the Appendix §E.

4.1 IMAGE CLASSIFICATION

Batch normalization (BN) and momentum-based optimizer are standard techniques to train state-of-the-art image classification models (He et al., 2016; Han et al., 2017; Sandler et al., 2018; Tan & Le, 2019; Han et al., 2020). We evaluate the proposed method with ResNet (He et al., 2016), one of the most popular and powerful architectures on ImageNet, and MobileNetV2 (Sandler et al., 2018), a relatively lightweight model with ReLU6 and depthwise convolutions, on the ImageNet-1K benchmark (Russakovsky et al., 2015). For ResNet, we employ the training hyperparameters in (He et al., 2016). For MobileNetV2, we have searched for the best setting, as it is generally difficult to train it with the usual settings. We use the cosine annealed learning rates for all experiments.

Our optimizers are compared against their corresponding baselines in Table 2. Note that AdamP is compared against AdamW (Loshchilov & Hutter, 2019), which has closed the gap between Adam and SGD performances on large-scale benchmarks. Across the spectrum of network sizes, our optimizers outperform the baselines. Even when the state-of-the-art CutMix (Yun et al., 2019) regularization is applied, our optimizers introduce further gains. We provide two additional experiments on EfficientNet (Tan & Le, 2019) (§D.3) and the large-batch training scenario (§D.4) to demonstrate the benefit of AdamP on diverse training setups. There, again, our methods outperform the baselines.

Table 2. ImageNet classification. Accuracies of state-of-the-art networks trained with SGDP and AdamP.

<table>
<thead>
<tr>
<th>Architecture</th>
<th># params</th>
<th>SGD</th>
<th>SGDP (ours)</th>
<th>Adam</th>
<th>AdamW</th>
<th>AdamP (ours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNetV2</td>
<td>3.5M</td>
<td>71.61</td>
<td><strong>72.18</strong> (+0.57)</td>
<td>69.27</td>
<td>72.12</td>
<td><strong>72.57</strong> (+0.45)</td>
</tr>
<tr>
<td>ResNet18</td>
<td>11.7M</td>
<td>70.28</td>
<td><strong>70.73</strong> (+0.45)</td>
<td>68.05</td>
<td>70.41</td>
<td><strong>70.81</strong> (+0.40)</td>
</tr>
<tr>
<td>ResNet50</td>
<td>25.6M</td>
<td>76.64</td>
<td><strong>76.71</strong> (+0.07)</td>
<td>71.89</td>
<td>76.65</td>
<td><strong>76.94</strong> (+0.29)</td>
</tr>
<tr>
<td>ResNet50 + CutMix</td>
<td>25.6M</td>
<td>77.61</td>
<td><strong>77.72</strong> (+0.11)</td>
<td>76.41</td>
<td>78.00</td>
<td><strong>78.31</strong> (+0.31)</td>
</tr>
</tbody>
</table>

4.2 ROBUSTNESS

Model robustness is an emerging problem in the real-world applications and the training of robust models often involve complex optimization problems (e.g. minimax). We examine how our optimizers stabilize the complex optimization. We consider two types of robustness: adversarial (below) and cross-bias (Bahng et al., 2020) (Appendix §D.2).

Adversarial training alternatively optimizes a minimax problem where the inner optimization is an adversarial attack and the outer optimization is the standard classification problem. Adam is commonly employed, in order to handle the complexity of the optimization (Tsipras et al., 2019; Chun et al., 2019).

We consider solving the minimax optimization problem with our proposed optimizers. We train Wide-ResNet (Zagoruyko & Komodakis, 2016) with the projected gradient descent (PGD) (Madry et al., 2018) on CIFAR-10. We use 10 inner PGD iterations and ε = 80/255 for the $L_2$ PGD and ε = 4/255 for the $L_{\infty}$ PGD. Figure 4 shows the learning curves of Adam and AdamP. By handling the effective step sizes, AdamP achieves a faster convergence than Adam (less than half the epochs required). AdamP brings more than +9.3 pp performance gap in all settings.

Figure 4. Adversarial training. Learning curves by Adam and AdamP.
4.3 AUDIO CLASSIFICATION

We evaluate the proposed optimizer on three audio classification tasks with different physical properties: music clips, verbal audios, and acoustic signals. For automatic music tagging, we use the MagnaTagATune (MTAT) benchmark (Law et al., 2009) with 21k samples and 50 tags. Each clip contains multiple tags. We use the Speech Commands dataset (Warden, 2018) for the keyword spotting task (106k samples, 35 classes, single label). For acoustic signals, we use the DCASE sound event detection benchmark (Mesaros et al., 2017) (53k samples, 17 tags, multi-labeled).

We train the Harmonic CNN (Won et al., 2020) on the three benchmarks. Harmonic CNN consists of data-driven harmonic filters and stacked convolutional filters with BN. Audio datasets are usually smaller than the image datasets and are multi-labeled, posing another set of challenges for the optimization. Won et al. (2020) have trained the network with a mixture of Adam and SGD (Won et al., 2019). Instead of the mixture solution, we have searched the best hyperparameters for the Adam and AdamP on a validation set. The results are given in Table 3. AdamP shows better performances than the baselines, without having to adopt the complex mixture solution. The results signify the superiority of AdamP for training scale-invariant weights on the audio domain.

<table>
<thead>
<tr>
<th>Optimizer</th>
<th>Music Tagging</th>
<th>Keyword Spotting</th>
<th>Sound Event Tagging</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ROC-AUC</td>
<td>PR-AUC</td>
<td>Accuracy</td>
</tr>
<tr>
<td>Adam + SGD (Won et al., 2019)</td>
<td>91.27</td>
<td>45.67</td>
<td>96.08</td>
</tr>
<tr>
<td>AdamW</td>
<td>91.12</td>
<td>45.61</td>
<td>96.47</td>
</tr>
<tr>
<td>AdamP (ours)</td>
<td>91.35 (+0.23)</td>
<td>45.79 (+0.18)</td>
<td>96.89 (+0.42)</td>
</tr>
</tbody>
</table>

Table 3. Audio classification. Results on three audio classification tasks with Harmonic CNN.

4.4 LANGUAGE MODELING

Models in language domains, such as the Transformer (Vaswani et al., 2017), often do not have any scale-invariant weight. The use of layer normalization (LN) in the Transformer does not result in the scale invariance because LN is applied right after the skip connection $f(w, x) := LN(g_w(x) + x)$ (note the difference from equation 2). The skip connection makes $f$ scale-variant with respect to $w$. To allow our optimizers to effectively operate on Transformer, we introduce scale invariance artificially through the weight normalization (WN) (Salimans & Kingma, 2016).

We have trained Transformer-XL (Dai et al., 2019) on WikiText-103 (Merity et al., 2016). As shown in Table 4, AdamP does not significantly outperform AdamW (23.33→23.26) without the explicit enforcement of scale invariance. WN on its own harms the performance (23.33→23.90 for Adam). When AdamP is applied on Transformer-XL + WN, it significantly improves over the AdamW baseline (23.90→22.73), beating the original performance 23.33 by Adam on the vanilla network.

<table>
<thead>
<tr>
<th>Model</th>
<th>AdamW</th>
<th>AdamP (ours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer-XL</td>
<td>23.33</td>
<td>23.26 (+0.07)</td>
</tr>
<tr>
<td>Transformer-XL + WN</td>
<td>23.90</td>
<td>22.73 (-1.17)</td>
</tr>
</tbody>
</table>

Table 4. Language Modeling. Perplexity on WikiText103. Lower is better.

4.5 RETRIEVAL

In the previous experiments, we have examined the scale invariances induced by statistical normalization methods (e.g. BN). Here, we consider another source of scale invariance, $\ell_2$ projection of features, which induces a scale invariance in the preceding weights. It is widely used in retrieval tasks for more efficient distance computations and better performances. We fine-tune the ImageNet-pretrained ResNet-50 network on CUB (Wah et al., 2011), Cars-196 (Krause et al., 2013), In-Shop Clothes (Liu et al., 2016b), and Stanford Online Products (SOP) (Oh Song et al., 2016) benchmarks with the triplet (Schroff et al., 2015) and the ProxyAnchor (Kim et al., 2020) losses. In Table 5, we observe that AdamP outperforms Adam over all four image retrieval datasets. The results support the superiority of AdamP for networks with $\ell_2$ normalized embeddings.
Table 5. **Image retrieval.** Optimizers are tested on networks with \( \ell_2 \)-induced scale invariant weights. Recall@1.

<table>
<thead>
<tr>
<th>Optimizer</th>
<th>CUB</th>
<th>Cars-196</th>
<th>InShop</th>
<th>SOP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Triplet</td>
<td>PA</td>
<td>Triplet</td>
<td>PA</td>
</tr>
<tr>
<td>AdamW</td>
<td>57.9</td>
<td>69.3</td>
<td>59.8</td>
<td>86.7</td>
</tr>
<tr>
<td>AdamP (ours)</td>
<td><strong>58.2</strong> (+0.3)</td>
<td><strong>69.5</strong> (+0.2)</td>
<td><strong>59.9</strong> (+0.2)</td>
<td><strong>86.9</strong> (+0.2)</td>
</tr>
</tbody>
</table>

5 CONCLUSION

Momentum-based optimizers induce an excessive growth of the scale-invariant weight norms. The growth of weight norms prematurely decays the effective optimization steps, leading to sub-optimal performances. The phenomenon is prevalent in many commonly-used setup. Momentum-based optimizers (e.g. SGD and Adam) are used for training the vast majority of deep models. The widespread use of normalization layers make a large proportion of network weights scale-invariant (e.g. ResNet). We propose a simple and effective solution: project out the radial component from the optimization updates. The resulting SGDP and AdamP successfully suppress the weight norm growth and train a model at an unobstructed speed. Empirically, our optimizers have demonstrated their superiority over the baselines on more than 10 real-world learning tasks.

ACKNOWLEDGEMENT

We thank NAVER AI Research team for discussion and advice, especially Junsuk Choe for the internal review. Naver Smart Machine Learning (NSML) platform (Kim et al., 2018) has been used in the experiments.

REFERENCES


Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In International Conference on Learning Representations (ICLR), 2020.


APPENDIX

This document provides additional materials for the main paper. Content includes the proofs (§A), detailed experimental setups (§B and §E), and the additional analysis on the learning rate scheduling and weight decay (§F).

A PROOFS FOR THE CLAIMS

We provide proofs for Lemma 2.2, Corollary 2.3, and Proposition 3.1 in the main paper.

Lemma A.1 (Monotonic norm growth by the momentum). For a scale-invariant parameter $w$ updated via equation 7, we have

$$\|w_{t+1}\|^2_2 = \|w_t\|^2_2 + \eta^2 \|p_t\|^2_2 + 2\eta^2 \sum_{k=0}^{t-1} \beta^{t-k} \|p_k\|^2_2. \quad (A.1)$$

Proof. From equation 7, we have

$$\|w_{t+1}\|^2_2 = \|w_t\|^2_2 + \eta^2 \|p_t\|^2_2 - 2\eta w_t \cdot p_t \quad (A.2)$$

It remains to prove that $w_t \cdot p_t = -\eta \sum_{k=0}^{t-1} \beta^{t-k} \|p_k\|^2_2$. We prove by induction on $t \geq 0$.

First, when $t = 0$, we have $w_0 \cdot p_0 = w_0 \cdot \nabla_w f(w_0) = 0$ because of equation 4.

Now, assuming that $w_r \cdot p_r = -\eta \sum_{k=0}^{r-1} \beta^{r-k} \|p_k\|^2_2$, we have

$$w_{r+1} \cdot p_{r+1} = w_{r+1} \cdot (\beta p_r + \nabla_w f(w_{r+1})) = \beta w_{r+1} \cdot p_r = \beta (w_r - \eta p_r) \cdot p_r \quad (A.3)$$

$$= -\beta \eta \sum_{k=0}^{r-1} \beta^{r-k} \|p_k\|^2_2 - \beta \eta \|p_r\|^2_2 = -\eta \sum_{k=0}^{r} \beta^{r-k+1} \|p_k\|^2_2 \quad (A.4)$$

which completes the proof. □

Corollary A.2 (Asymptotic norm growth comparison). Let $\|w^G_t\|^2_2$ and $\|w^{GDM}_t\|^2_2$ be the weight norms at step $t \geq 0$, following the vanilla gradient descent growth (Lemma 2.1) and momentum-based gradient descent growth (Lemma 2.2), respectively. We assume that the norms of the updates $\|p_t\|^2_2$ for GD with and without momentum are identical for every $t \geq 0$. We further assume that the sum of the update norms is non-zero and bounded: $0 < \sum_{t=0}^{\infty} \|p_t\|^2_2 < \infty$. Then, the asymptotic ratio between the two norms is given by:

$$\frac{\|w^G_t\|^2_2 - \|w_0\|^2_2}{\|w^{GDM}_t\|^2_2 - \|w_0\|^2_2} \to 1 + \frac{2\beta}{1 - \beta} \text{ as } t \to \infty. \quad (A.5)$$

Proof. From Lemma 2.1 and Lemma 2.2, we obtain

$$\|w^G_t\|^2_2 - \|w_0\|^2_2 = \eta^2 \sum_{k=0}^{t-1} \|p_k\|^2_2 \quad (A.6)$$

$$\|w^{GDM}_t\|^2_2 - \|w_0\|^2_2 = \eta^2 \sum_{k=0}^{t-1} \|p_k\|^2_2 + 2\eta^2 \sum_{k=0}^{t-1} \left( \sum_{l=1}^{t-1-k} \beta^l \right) \|p_k\|^2_2. \quad (A.7)$$

Thus, the corollary boils down to the claim that

$$F_t := \frac{\sum_{k=0}^{t} \left( \sum_{l=1}^{t-k} \beta^l \right) A_k}{\sum_{k=0}^{t} A_k} \to \frac{\beta}{1 - \beta} \text{ as } t \to \infty \quad (A.8)$$

where $A_k := \|p_k\|^2_2$.

Let $\epsilon > 0$. We will find a large-enough $t$ that bounds $F_t$ around $\frac{\beta}{1 - \beta}$ by a constant multiple of $\epsilon$. 

13
We first let $T$ be large enough such that
\[
\sum_{k \geq T+1} A_k \leq \epsilon \tag{A.9}
\]
which is possible because $\sum_{t \geq 0} A_t < \infty$. We then let $T'$ be large enough such that
\[
\frac{\beta}{1-\beta} - \sum_{l=1}^{T'} \beta^l \leq \frac{\epsilon}{T \max_k A_k} \tag{A.10}
\]
which is possible due to the convergence of the geometric sum and the boundedness of $A_k$ (because its infinite sum is bounded).

We then define $t = T + T'$ and break down the sums in $F_t$ as follows:
\[
F_t = \sum_{k=0}^{T} \left( \sum_{t=1}^{T+T'-k} \beta^t \right) A_k + \sum_{k=T+T'}^{T+T+1} \left( \sum_{t=1}^{T+T'-k} \beta^t \right) A_k \tag{A.11}
\]
\[
= \sum_{k=0}^{T} \left( \frac{\beta}{1-\beta} + r_1(\epsilon) \right) A_k + r_2(\epsilon) \tag{A.12}
\]
\[
\leq \frac{\beta}{1-\beta} \sum_{k=0}^{T} A_k + T \max_k A_k r_1(\epsilon) + r_2(\epsilon) \tag{A.13}
\]
where $r_1, r_2,$ and $r_3$ are the residual terms that are bounded as follows:
\[
|r_1(\epsilon)| \leq \frac{\epsilon}{T \max_k A_k} \tag{A.14}
\]
by equation A.10 and
\[
|r_2(\epsilon)| \leq \frac{(1-\beta)\epsilon}{\beta} \quad \text{and} \quad |r_3(\epsilon)| \leq \epsilon \tag{A.15}
\]
by equation A.9.

It follows that
\[
\left| F_t - \frac{\beta}{1-\beta} \right| \leq \left| -\frac{\beta}{1-\beta} r_3(\epsilon) + T \max_k A_k r_1(\epsilon) + r_2(\epsilon) \right| \sum_{k=0}^{T} A_k + r_3(\epsilon) \tag{A.16}
\]
\[
\leq \frac{1}{\sum_{k=0}^{T} A_k} \left( \frac{\beta}{1-\beta} + \frac{1-\beta}{\beta} + 1 \right) \epsilon \tag{A.17}
\]
\[
\leq \frac{1}{M} \left( \frac{\beta}{1-\beta} + \frac{1-\beta}{\beta} + 1 \right) \epsilon \tag{A.18}
\]
due to the triangular inequality and the positivity of $r_3$. $M > 0$ is a suitable constant independent of $T$.

**Proposition A.3** (Effective update direction after projection). Let $w_{t+1}^o := w_t - \eta p_t$ and $w_{t+1}^p := w_t - \eta \Pi_{w_t}(p_t)$ be original and projected updates, respectively. Then, the effective update after the projection $w_{t+1}^p$ lies on the geodesic on $\mathbb{S}^{d-1}$ defined by $\hat{w}_t$ and $w_{t+1}^o$.

**Proof.** The geodesic defined by $\hat{w}_t$ and $w_{t+1}^o$ can be written as $\mathbb{S}^{d-1} \cap \text{span}(w_t, p_t)$. Thus, it suffices to show that $w_{t+1}^p \in \text{span}(w_t, p_t)$. Indeed, we observe that $w_{t+1}^p := \frac{w_{t+1}^o}{\|w_{t+1}^o\|} \in \text{span}(w_t, p_t)$ because $\Pi_{w_t}(p_t) = p_t - (w_t \cdot p_t)w_t$. \qed
B **TOY EXAMPLE DETAILS**

**2D toy example in Figure 1** We describe the details of the toy example in Figure 1. We solve the following optimization problem:

\[
\min_w \frac{w}{\|w\|_2}, \quad \frac{w^*}{\|w^*\|_2} \tag{B.1}
\]

where \(w\) and \(w^*\) are 2-dimensional vectors. The problem is identical to the maximization of the cosine similarity between \(w\) and \(w^*\). We set the \(w^*\) to \((0, -1)\) and the initial \(w\) to \((0.001, 1)\).

This toy example has two interesting properties. First, the normalization term makes the optimal \(w\) for the problem not unique: if \(w^*\) is optimal, then \(cw^*\) is optimal for any \(c > 0\). In fact, the cost function is scale-invariant. Second, the cost function is not convex.

As demonstrated in Figure 1 and videos attached in our submitted code, the momentum gradient method fails to optimize equation B.1 because of the excessive norm increases. In particular, our simulation results show that a larger momentum induces a larger norm increase (maximum norm 2.93 when momentum is 0.9, and 27.87 when momentum is 0.99), as we shown in the main paper § 2.4. On the other hand, our method converges most quickly, among the compared methods, by taking advantage of the momentum-induced accelerated convergence, while avoiding the excessive norm increase.

**3D spherical toy example in Figure 3** We employ 3D Rosenbrock function \((\text{Rosenbrock}, 1960)\):

\[
\tilde{h}(r, \psi, \phi) = (1 - \psi)^2 + 300(\phi - \psi^2)^2
\]

in the spherical coordinate \((r, \psi, \phi)\). Since we are mapping 2D space to a spherical surface, \((\psi, \phi)\) is defined on the hemisphere. Thus, \(-\pi/2 \leq (\psi, \phi) \leq \pi/2\). Based on the required range of problem \(f\), scalar \(c\) can be multiplied to each angles \(f(c\psi, c\phi)\) to adjust angle scale to problem scale, where we choose \(c = 1.5\) in the experiments. The initial point is \((c\psi, c\phi) = (-2, 2)\) above the unit sphere \(r = 1\), and the minimum point is \((c\psi, c\phi) = (1, 1)\).

Instead of optimizing \(h\) in the spherical coordinate, we optimize the toy example in the Cartesian coordinate \((x, y, z)\) by computing \(\min_{x,y,z} \tilde{h}(r, \psi, \phi) = \min_{x,y,z} \tilde{h}(T(x, y, z))\). We employ the spherical transform \(T : (x, y, z) \rightarrow (r, \psi, \phi)\) as follows:

\[
r = \sqrt{x^2 + y^2 + z^2}, \quad \psi = \cos^{-1}(x/z), \quad \phi = \sin^{-1}(y/r). \tag{B.2}
\]

For all optimizers, we set the momentum to 0.9, and we have exhaustively searched the optimal initial learning rates between 0.001 and 0.1. The learning rates are decayed by linearly at every iteration.

**C δ SENSITIVITY**

Based on ImageNet training with ResNet18, we analyzed the cosine similarity of scale-invariant parameter and scale-variant parameter, and verified the sensitivity of δ. We first measured the cosine similarity between the gradient and weights (Eq. 11). As a results, the cosine similarities for scale-variant weights are \([0.0028, 0.5660]\), compared to \([5.5 \times 10^{-10}, 4.4 \times 10^{-6}]\) for scale-invariant ones (99% confidence intervals). Because of this large gap, our methods are stable on a wide range of δ values. We also measured scale-variant and scale-invariant parameter detection accuracy based on Eq. 11 with various δ values. The results are shown in Table B.1. In a wide range of delta values, AdamP perfectly discriminates the scale-variant and scale-invariant parameters. Also, AdamP consistently shows high performance. Therefore, AdamP is not sensitive to the δ value and δ = 0.1 is suitable to separate scale-variant and scale-invariant parameters.

<table>
<thead>
<tr>
<th>δ</th>
<th>Scale-variant detection</th>
<th>Scale-invariant detection</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>86.64%</td>
<td>100%</td>
<td>70.74</td>
</tr>
<tr>
<td>0.2</td>
<td>100%</td>
<td>100%</td>
<td>70.83</td>
</tr>
<tr>
<td>0.1</td>
<td>100%</td>
<td>100%</td>
<td>70.81</td>
</tr>
<tr>
<td>0.05</td>
<td>100%</td>
<td>100%</td>
<td>70.78</td>
</tr>
<tr>
<td>0.02</td>
<td>100%</td>
<td>99.94%</td>
<td>70.81</td>
</tr>
<tr>
<td>0.01</td>
<td>100%</td>
<td>97.73%</td>
<td>70.66</td>
</tr>
</tbody>
</table>
D ADDITIONAL EXPERIMENTS

D.1 OBJECT DETECTION

Object detection is another widely-used real-world task where the models often include normalization layers and are trained with momentum-based optimizers. We study the two detectors CenterNet (Zhou et al., 2019) and SSD (Liu et al., 2016a) to verify that the proposed optimizers are also applicable to various objective functions beyond the classification task. The detectors are either initialized with the ImageNet-pretrained network (official PyTorch models) or trained from scratch, in order to separate the effect of our method from that of the pretraining. ResNet18 (He et al., 2016) and VGG16 BN (Simonyan & Zisserman, 2015) are used for the CenterNet and SSD backbones, respectively. In Table D.1, we report average precision performances based on the MS-COCO (Lin et al., 2014) evaluation protocol. We observe that AdamP boosts the performance against the baselines. It demonstrates the versatility of our optimizers.

<table>
<thead>
<tr>
<th>Model</th>
<th>Initialize</th>
<th>Adam</th>
<th>AdamP (ours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CenterNet</td>
<td>Random</td>
<td>26.57</td>
<td><strong>27.11</strong> (+0.54)</td>
</tr>
<tr>
<td>CenterNet</td>
<td>ImageNet</td>
<td>28.29</td>
<td><strong>29.05</strong> (+0.76)</td>
</tr>
<tr>
<td>SSD</td>
<td>Random</td>
<td>27.10</td>
<td><strong>27.97</strong> (+0.87)</td>
</tr>
<tr>
<td>SSD</td>
<td>ImageNet</td>
<td>28.39</td>
<td>28.67 (+0.28)</td>
</tr>
</tbody>
</table>

D.2 CROSS-BIAS ROBUSTNESS

Cross-bias generalization problem (Bahng et al., 2020) tackles the scenario where the training and test distributions have different real-world biases. This often occurs when the training-set biases provide an easy shortcut to solve the problem: e.g., using the snow cues for recognizing snowmobiles. Bahng et al. (2020) has proposed the ReBias scheme based on the minimax optimization, where the inner problem maximizes the independence between an intentionally biased representation and the target model of interest and the outer optimization solves the standard classification problem. As for the adversarial training, Bahng et al. (2020) has employed the Adam to handle the complex optimization.

We follow the two cross-bias generalization benchmarks proposed by Bahng et al. (2020). The first benchmark is the Biased MNIST, the dataset synthesized by injecting colors on the MNIST background pixels. Each sample is colored according to a pre-defined class-color mapping with probability ρ. The color is selected at random with 1 − ρ chance. For example, ρ = 1.0 leads a completely biased dataset and ρ = 0.1 leads to an unbiased dataset. Each model is trained on the ρ-biased MNIST and tested on the unbiased MNIST. We train a stacked convolutional network with BN and ReLU. The second benchmark is the 9-Class ImageNet representing the real-world biases, such as textures (Geirhos et al., 2019). The unbiased accuracy is measured by pseudo-labels generated by the texture clustering. We also report the performance on ImageNet-A (Hendrycks et al., 2019), the collection of failure samples of existing CNNs.

In Table D.2, we observe that AdamP consistently outperforms Adam in all the benchmarks. AdamP is a good alternative to Adam for difficult optimization problems applied on scale-invariant parameters.

<table>
<thead>
<tr>
<th>Optimizer</th>
<th>Biased MNIST</th>
<th>Unbiased acc. at ρ</th>
<th>9-Class ImageNet</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>.999</td>
<td>.997</td>
<td>.995</td>
</tr>
<tr>
<td>Adam</td>
<td>22.9</td>
<td>63.0</td>
<td>74.9</td>
</tr>
<tr>
<td>AdamP (ours)</td>
<td><strong>30.5</strong> (+7.5)</td>
<td><strong>70.9</strong> (+7.9)</td>
<td><strong>80.9</strong> (+6.0)</td>
</tr>
</tbody>
</table>

D.3 EFFICIENTNET

EfficientNet (Tan & Le, 2019) is a recently proposed high-performance network that is trained using various techniques such as data augmentation (Cubuk et al., 2018; 2020), stochastic depth (Huang
et al., 2016), and dropout (Srivastava et al., 2014). We measured the performance of AdamP on EfficientNets to verify it can be used with other training techniques. The experiment were conducted on the well-known image classification codebase\(^1\). Table D.3 shows performance of the original paper, our reproduced results, and AdamP. The results show that AdamP is still effective in training with various techniques, and can contribute to improving the best performance of the network.

Table D.3. EfficientNet training. We evaluated performance of AdamP on EfficientNet (Tan & Le, 2019) training. EfficientNet was trained using the latest training techniques, and this experiment shows that AdamP can also contribute to learning with these techniques.

<table>
<thead>
<tr>
<th>Network</th>
<th>Image size</th>
<th># of params</th>
<th>Paper</th>
<th>Reproduce</th>
<th>AdamP</th>
</tr>
</thead>
<tbody>
<tr>
<td>EfficientNet-B0</td>
<td>224</td>
<td>5.29M</td>
<td>77.1</td>
<td>77.7</td>
<td>78.1 (+0.4)</td>
</tr>
<tr>
<td>EfficientNet-B1</td>
<td>240</td>
<td>7.79M</td>
<td>79.1</td>
<td>78.7</td>
<td>79.9 (+1.2)</td>
</tr>
<tr>
<td>EfficientNet-B2</td>
<td>260</td>
<td>9.11M</td>
<td>80.1</td>
<td>80.4</td>
<td>80.5 (+0.1)</td>
</tr>
<tr>
<td>EfficientNet-B3</td>
<td>300</td>
<td>12.23M</td>
<td>81.6</td>
<td>81.5</td>
<td>81.9 (+0.4)</td>
</tr>
</tbody>
</table>

D.4 LARGE-BATCH TRAINING

In order to efficiently use multiple machines and huge computational resources, large-batch training is essential. However, a general optimizer suffers from a significant performance decrease in a large-batch training, so large-batch training is another challenge for a deep learning optimizer. (You et al., 2017; 2019; Goyal et al., 2017) We conducted experiments to verify the performance of AdamP in such large-batch training, and the results are shown in Table D.4. The performance improvement of AdamP in large-batch training is greater than that of regular batch-size (Table 2). Therefore, the decrease of effective learning rate due to momentum can be considered as one of the causes of performance degradation in large-batch training. However, AdamP does not show as much performance as the large-batch optimizers (You et al., 2017; 2019; Goyal et al., 2017), and therefore, applying AdamP to the large-batch optimizer should be studied as a future work.

E EXPERIMENTS SETTINGS

We describe the experimental settings in full detail for reproducibility.

E.1 COMMON SETTINGS

All experiments are conducted based on PyTorch. SGDP and AdamP are implemented to handle channel-wise (e.g. batch normalization (Ioffe & Szegedy, 2015) and instance normalization (Ulyanov et al., 2016)) and layer-wise normalization (Ba et al., 2016). Based on the empirical measurement of the inner product between the weight vector and the corresponding gradient vector for scale-invariant parameters (they are supposed to be orthogonal), we set the \( \delta \) in Algorithms 1 and 2 to 0.1. We use the decoupled weight decay (Loshchilov & Hutter, 2019) for SGDP and AdamP in order to separate the gradient due to the weight decay from the gradient due to the loss function. Please refer to the attached codes: sgdp.py and adamp.py for further details.

E.2 IMAGE CLASSIFICATION

Experiments on ResNet (He et al., 2016) are conducted based on the standard settings : learning rate 0.1, weight decay \( 10^{-4} \), batch-size 256, momentum 0.9 with Nesterov (Sutskever et al., 2013) for SGD and SGDP. For Adam series, we use the learning rate 0.001, weight decay \( 10^{-4} \), batch-size 256, \( \beta_1 0.9, \beta_2 0.999, \epsilon 10^{-8} \).

\(^1\)https://github.com/rwightman/pytorch-image-models
Table E.1. **Dataset statistics.** Summary of the dataset specs used in the experiments. ImageNet-1k (Rus-sakovs’k et al., 2015), MS-COCO (Lin et al., 2014), CIFAR-10 (Krizhevsky, 2009), Biased-MNIST (LeCun et al., 1998; Bahng et al., 2020), ImageNet-A (Hendrycks et al., 2019), 9-Class ImageNet (Bahng et al., 2020), 9-Class ImageNet-A (Bahng et al., 2020), MagnaTagATune (Law et al., 2009), Speech Commands (Warden, 2018), DCASE 2017 task 4 (Mesaros et al., 2017), CUB (Wah et al., 2011), In-Shop Clothes (Liu et al., 2016b) and SOP (Oh Song et al., 2016) are used in experiments.

<table>
<thead>
<tr>
<th>Task</th>
<th>Dataset</th>
<th>#classes</th>
<th>#samples</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image classification</td>
<td>ImageNet-1k</td>
<td>1,000</td>
<td>≈ 1.33M</td>
<td></td>
</tr>
<tr>
<td>Object detection</td>
<td>MS-COCO</td>
<td>80</td>
<td>≈ 123k</td>
<td></td>
</tr>
<tr>
<td>Robustness</td>
<td>CIFAR-10</td>
<td>10</td>
<td>≈ 60k</td>
<td>colors are injected to be biased</td>
</tr>
<tr>
<td></td>
<td>Biased-MNIST</td>
<td>10</td>
<td>≈ 60k</td>
<td></td>
</tr>
<tr>
<td></td>
<td>9-Class ImageNet</td>
<td>9</td>
<td>≈ 57k</td>
<td>a subset of ImageNet-1k</td>
</tr>
<tr>
<td></td>
<td>9-Class ImageNet-A</td>
<td>9</td>
<td>617</td>
<td>a subset of ImageNet-A</td>
</tr>
<tr>
<td>Audio classification</td>
<td>MagnaTagATune</td>
<td>50</td>
<td>≈ 21k</td>
<td>mutl-labeled dataset</td>
</tr>
<tr>
<td></td>
<td>Speech Commands</td>
<td>35</td>
<td>≈ 106k</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DCASE 2017 task 4</td>
<td>17</td>
<td>≈ 53k</td>
<td>mutl-labeled dataset</td>
</tr>
<tr>
<td>Language Modeling</td>
<td>WikiText-103</td>
<td>-</td>
<td>≈ 103M tokens</td>
<td>vocabulary size (267,735)</td>
</tr>
<tr>
<td>Image retrieval</td>
<td>CUB</td>
<td>200</td>
<td>≈ 12k</td>
<td>tr classes (100), te classes (100)</td>
</tr>
<tr>
<td></td>
<td>Cars-196</td>
<td>196</td>
<td>≈ 16k</td>
<td>tr classes (98), te classes (98)</td>
</tr>
<tr>
<td></td>
<td>In-Shop Clothes</td>
<td>7,982</td>
<td>≈ 53k</td>
<td>tr classes (3,997), te classes (3985)</td>
</tr>
<tr>
<td></td>
<td>SOP</td>
<td>22,634</td>
<td>≈ 120k</td>
<td>tr classes (11,318), te classes (11,316)</td>
</tr>
</tbody>
</table>

For training MobileNetV2 (Sandler et al., 2018), we have additionally used label-smoothing and large batch size 1024, and have searched the best learning rates and weight decay values for each optimizer.

The training sessions are run for 100 epochs (ResNet18, ResNet50) or 150 epochs (MobileNetV2, ResNet50 + CutMix) with the cosine learning rate schedule (Loshchilov & Hutter, 2016) on a machine with four NVIDIA V100 GPUs.

### E.3 Object detection

Object detection performances have been measured on the MS-COCO dataset (Lin et al., 2014) with two popular object detectors: CenterNet (Zhou et al., 2019) and SSD (Liu et al., 2016a). We adopt the CenterNet with ResNet18 (He et al., 2016) backbone and the SSD with VGG16 BN (Simonyan & Zisserman, 2015) backbone as baseline detectors. CenterNet has been trained for 140 epochs with learning rate $2 \times 10^{-4}$, weight decay $10^{-5}$, batch size 64, and the cosine learning rate schedule. SSD has been trained for 110 epochs with learning rate $10^{-4}$, weight decay $10^{-5}$, batch size 64, and the step learning rate schedule which decays learning rates by $1/10$ at 70% and 90% of training.

### E.4 Robustness

#### E.4.1 Adversarial training

Adversarial robustness benchmark results have been reproduced using the unofficial PyTorch implementation of the adversarial training of Wide-ResNet (Zagoruyko & Komodakis, 2016)\(^2\) for the CIFAR-10 attack challenge\(^3\). Projected gradient descent (PGD) attack variants (Madry et al., 2018) have been used as the threat model for all the experiments. We employed 10 inner PGD iterations and $\varepsilon = 80/255$ for the $L_2$ PGD attack and $\varepsilon = 4/255$ for the $L_\infty$ PGD attack. In all the experiments, Wide-ResNet-34-10 have been trained with the PGD threat model. The models have been trained for 200 epochs with learning rate 0.01, weight decay 0.0002, batch size 128, and the step learning rate schedule which decays learning rates by $1/10$ at epochs 100 and 150. Table E.2 shows the detailed results.

\(^2\)https://github.com/louis2889184/pytorch-adversarial-training

\(^3\)https://github.com/MadryLab/cifar10_challenge
### Table E.2. Adversarial training

Standard and attacked accuracies of PGD-adversarially trained Wide-ResNet on CIFAR-10.

<table>
<thead>
<tr>
<th>Attack Method</th>
<th>Optimizer</th>
<th>Standard Acc</th>
<th>Attacked Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ell_\infty$ ($\epsilon = 4/255$)</td>
<td>Adam</td>
<td>80.12</td>
<td>56.58</td>
</tr>
<tr>
<td></td>
<td>AdamP</td>
<td><strong>89.85 (+9.73)</strong></td>
<td><strong>66.28 (+9.70)</strong></td>
</tr>
<tr>
<td>$\ell_2$ ($\epsilon = 80/255$)</td>
<td>Adam</td>
<td>84.14</td>
<td>70.33</td>
</tr>
<tr>
<td></td>
<td>AdamP</td>
<td><strong>93.46 (+9.32)</strong></td>
<td><strong>83.59 (+13.26)</strong></td>
</tr>
</tbody>
</table>

### E.4.2 Robustness against real-world biases

We follow the two cross-bias generalization benchmarks proposed by (Bahng et al., 2020). We refer (Bahng et al., 2020) for interested readers. For all experiments, the batch size is 256 and 128 for Biased MNIST and 9-Class ImageNet, respectively. For Biased MNIST, the initial learning rate is 0.001, decayed by factor 0.1 every 20 epochs. For 9-Class ImageNet, the learning rate is 0.001, decayed by cosine annealing. We train the fully convolutional network and ResNet18 for 80 and 120 epochs, respectively. The weight decay is $10^{-4}$ for all experiments.

### E.5 Audio classification

**Dataset.** Three datasets with different physical properties are employed as the audio benchmarks. We illustrate the statistics in Table E.1. The music tagging is a multi-label classification task for the prediction of user-generated tags, e.g., genres, moods, and instruments. We use a subset of MagnaTagATune (MTAT) dataset (Law et al., 2009) which contains ≈21k audio clips and 50 tags. The average of tag-wise Area Under Receiver Operating Characteristic Curve (ROC-AUC) and Area Under Precision-Recall Curve (PR-AUC) are used as the evaluation metrics. Keyword spotting is a primitive speech recognition task where an audio clip containing a keyword is categorized among a list of limited vocabulary. We use the Speech Commands dataset (Warden, 2018) which contains ≈106k samples and 35 command classes such as “yes”, “no”, “left”, “right”. The accuracy metric is used for the evaluation. Acoustic sound detection is a multi-label classification task with non-music and non-verbal audios. We use the “large-scale weakly supervised sound event detection for smart cars” dataset used for the DCASE 2017 challenge (Mesaros et al., 2017). It has ≈53k audio clips with 17 events such as “Car”, “Fire truck”, and “Train horn”. For evaluation, we use the F1-score by setting the prediction threshold as 0.1.

**Training setting.** We use the 16kHz sampling rate for the all experiments, and all hyperparameters, e.g., the number of harmonics, trainable parameters, are set to the same as in (Won et al., 2020). The official implementation by (Won et al., 2020) is used for all the experiments. We compare three different optimizers, Adam, AdamP (ours), and the complex mixture of Adam and SGD proposed by (Won et al., 2019).

For the mixture of Adam and SGD, we adopt the same hyperparameters as in the previous papers (Won et al., 2019; 2020). The mixed optimization algorithm first runs Adam for 60 epochs with learning rate $10^{-4}$. After 60 epochs, the model with the best validation performance is selected as the initialization for the second phase. During the second phase, the model is trained using SGD for 140 epochs with the learning rate $10^{-4}$, decayed by 1/10 at epochs 20 and 40. We use the weight decay $10^{-4}$ for the optimizers.

To show the effectiveness of our method, we have searched the best hyperparameters for the Adam optimizer on the MTAT validation dataset and have transferred them to AdamP experiments. As the result of our search, we set the weight decay as 0 and the initial learning rate as 0.0001 decayed by the cosine annealing scheduler. The number of training epochs are set to 100 for MTAT dataset and 30 for SpeechCommand and DCASE dataset. As a result, we observe that AdamP shows superior performances compared to the complex mixture, with a fewer number of training epochs (200 → 30).

---

4https://github.com/minzwon/data-driven-harmonic-filters
E.6 Retrieval

Dataset. We use four retrieval benchmark datasets. For the CUB (Wah et al., 2011) dataset which contains bird images with 200 classes, we use 100 classes for training and the rest for evaluation. For evaluation, we query every test image to the test dataset, and measure the recall@1 metric. The same protocol is applied to Cars-196 (Krause et al., 2013) (196 classes) and SOP (Oh Song et al., 2016) (2,634 classes) datasets. For InShop (Liu et al., 2016b) experiments, we follow the official benchmark setting proposed by (Liu et al., 2016b). We summarize the dataset statistics in Table E.1.

Training setting. For the all experiments, we use the same backbone network and the same training setting excepting the optimizer and the loss function. The official implementation by (Kim et al., 2020) is used for the all experiments.

We use the Pytorch official ImageNet-pretrained ResNet50 model as the initialization. During the training, we freeze the BN statistics as the ImageNet statistics (eval mode in PyTorch). We replace the global average pooling (GAP) layer of ResNet with the summation of GAP and global max pooling layer as in the implementation provided by (Kim et al., 2020). Pooled features are linearly mapped to the 512 dimension embedding space and $\ell_2$-normalized.

We set the initial learning rate $10^{-4}$, decayed by the factor 0.5 for every 5 epochs. Every mini-batch contains 120 randomly chosen samples. For the better stability, we train only the last linear layer for the first 5 epochs, and update all the parameters for the remaining steps. The weight decay is set to 0.

F Analysis with Learning Rate Schedule and Weight Decay

We analyze the norm growth of scale-invariant parameters and the corresponding change in effective step-size. We provide extended results of this experiment by measuring norm growth and effective step-size for SGD, SGDP, Adam and AdamP under various weight decay values. The experiment is based on ResNet18 trained on the ImageNet dataset, and the network was trained for 100 epoch in the standard setting as in E.2. We have analyzed the impact of learning rate schedule and weight decay for the scale-invariant parameters. Figure F.1 and Figure F.2 show the results of SGD and SGDP under the step-decay and cosine-annealing learning rate schedules, respectively. The same results for Adam and AdamP are shown in Figures F.3 and Figure F.4. We have used the optimal weight decay value of the baseline as the reference point and changed the weight decay values. We write the weight decay in each experiment in relative values with respect to the corresponding optimal values.

In all considered settings, SGDP and AdamP effectively prevent the norm growth, which prevents the rapid decrease of the effective step sizes. SGDP and AdamP shows better performances than the baselines. Another way to prevent the norm growth is to control the weight decay. However, this way of norm adjustment is sensitive to the weight decay value and results in poor performances as soon as non-optimal weight decay values are used. Figure F.1 and F.3 shows that the learning curves are generally sensitive to the weight decay values even showing abnormalities such as the gradual increase of the effective step sizes. On the other hand, SGDP and AdamP prevent rapid norm growth without weight decay, leading to smooth effective step size reduction. SGDP and AdamP are not sensitive to the weight decay values.

G Related Work

We provide a brief overview of related prior work. A line of work is dedicated to the development general and effective optimizers, such as Adagrad (Duchi et al., 2011), Adam (Kingma & Ba, 2015), and RMSprop. Researchers have sought strategies to improve Adam through e.g. improved convergence (Reddi et al., 2018), warmup learning rate (Liu et al., 2020), moving average (Zhang et al., 2019b), Nesterov momentum (Dozat, 2016), and rectified weight decay (Loshchilov & Hutter, 2019). Another line of researches studies existing optimization algorithms in greater depth. For
example, (Hoffer et al., 2018; Arora et al., 2019; Zhang et al., 2019a) have delved into the effective learning rates on scale-invariant weights. This paper at the intersection between the two. We study the issues when momentum-based optimizers are applied on scale-invariant weights. We then propose a new optimization method to address the problem.