When Do Flat Minima Optimizers Work?

Jean Kaddour*
Centre for Artificial Intelligence
University College London

Linqing Liu*
Centre for Artificial Intelligence
University College London

Ricardo Silva
Department of Statistical Science
University College London

Matt J. Kusner
Centre for Artificial Intelligence
University College London

Abstract

Recently, flat-minima optimizers, which seek to find parameters in low-loss neighborhoods, have been shown to improve a neural network’s generalization performance over stochastic and adaptive gradient-based optimizers. Two methods have received significant attention due to their scalability: 1. Stochastic Weight Averaging (SWA), and 2. Sharpness-Aware Minimization (SAM). However, there has been limited investigation into their properties and no systematic benchmarking of them across different domains. We fill this gap here by comparing the loss surfaces of the models trained with each method and through broad benchmarking across computer vision, natural language processing, and graph representation learning tasks. We discover several surprising findings from these results, which we hope will help researchers further improve deep learning optimizers, and practitioners identify the right optimizer for their problem.

1 Introduction

Stochastic gradient descent (SGD) methods are central to neural network optimization [6]. Recently, one class of algorithms has focused on biasing SGD methods towards so-called ‘flat’ minima, which are located in large weight space regions with very similar low loss values [43]. Theoretical and empirical studies [21, 77, 9, 55, 49, 5, 12] postulate that such flatter regions generalize better than sharper minima, e.g., due to the flat minimizer’s robustness against loss function shifts between train and test data, as illustrated in Fig. 1. Two popular flat-minima optimization approaches are: 1. Stochastic Weight Averaging (SWA) [48], and 2. Sharpness-Aware Minimization (SAM) [22].

While both strategies aim to find flatter minima, they operate much differently. On the one hand, SWA is based on the intuition that, near a flat minimum, gradients are smaller, leaving many iterates in that flat region. Therefore, averaging iterates will produce a solution that is pulled towards these flatter regions, see Fig. 1 top. On the other hand, SAM minimizes the maximum loss around a neighborhood of the current iterate. This way, a region around the iterate is designed to have uniformly low loss; see Fig. 1 bottom. Crucially, SAM requires an additional forward/backward pass for each parameter update, making it more expensive than SWA.

Despite the successes [76, 3, 51, 12, 4] of SWA and SAM in some domains, we are unaware of a systematic comparison between them that would help practitioners to choose the right optimizer for their problem and researchers to develop better optimizers. The SWA [48] paper was published in 2018, and the SAM [22] paper in 2021; however, the SAM paper, and its most noticeable follow-ups [65, 12, 103], do not compare against SWA. Further, there is very limited overlap in

*Equal contribution, correspondence to {jean.kaddour,linqing.liu}.20@ucl.ac.uk

terms of the model architecture and dataset used in the experiments among both papers, which are likely further confounded by other differences in the training procedures (e.g. data augmentations, hyper-parameters, etc.).

**Contributions**

1. **In-depth comparison of minima found by SWA and SAM:** We visualize linear interpolations between different models and quantify the minimizers’ flatnesses. This analysis yields 4 insights, e.g., despite SAM finding flatter solutions than SWA as quantified by Hessian eigenvalues, they can be close to sharp directions, a phenomenon that has been overlooked in the previous SAM literature. Averaging SAM iterates leads to the flattest among all minima.

2. **Rigorous comparison of SWA and SAM’s performance over 42 tasks:** We empirically compare the optimizers with a rigorous model selection procedure on a broad range of tasks across different domains (CV, NLP, and GRL), model types (MLPs, CNNs, Transformers) and tasks (classification, self-supervised learning, open-domain question answering, natural language understanding, and node/graph/link property prediction). We discuss 9 findings, e.g., that both dataset and architecture impact their effectiveness, that for NLP tasks, SAM improves over SWA in most cases, and that the converse holds for GRL tasks. When flat-minima optimizers do not help, we notice clear discrepancies between the shapes of loss and accuracy curves. To assist future work, we open-source the code for all pipelines and hyper-parameters to reproduce the results.

**2 Background and Related Work**

**2.1 Stochastic Gradient Descent (SGD)**

The classic optimization framework of machine learning is empirical risk minimization

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(x_i; \theta)$$  \hspace{1cm} (1)

where $\theta \in \mathbb{R}^d$ is a vector of parameters, $\{x_1, \ldots, x_N\}$ is a training set of inputs $x_i \in \mathbb{R}^D$, and $\ell(x; \theta)$ is a loss function quantifying the performance of parameters $\theta$ on $x$. SGD samples a minibatch $S \subset \{1, \ldots, N\}$ of size $|S| \ll N$ from the training set and updates the parameters through

$$\theta_{t+1}^{\text{SGD}} = \theta_t - \eta g(\theta_t), \text{ where } g(\theta) = \frac{1}{|B|} \sum_{i \in B} \nabla \ell(\theta; x_i),$$  \hspace{1cm} (2)

for a length specified by $\eta$, the learning rate.

**2.2 Stochastic Weight Averaging (SWA)**

The idea of averaging weights dates back to accelerating the convergence speed of SGD [78, 51]. SWA’s motivation is based on the following observation about SGD’s behavior when training neural networks: it often traverses regions of the weight space that correspond to high-performing models but rarely reaches the central points of this optimal set. Averaging the parameter values over iterations moves the solution closer to the centroid of this space of points.

The SWA update rule is the cumulative moving average

$$\theta_{t+1}^{\text{SWA}} \leftarrow \frac{\theta_t^{\text{SWA}} \cdot l + \theta_t^{\text{SGD}}}{l + 1},$$  \hspace{1cm} (3)

Figure 1: The mechanics behind SWA and SAM, whose solution is denoted by + and ×, respectively. SWA produces a solution $\theta$ that is pulled towards flatter regions, while SAM approximates sharpness within the parameters’ neighborhood (arrows).
2.3 Sharpness-Aware Minimization (SAM)

While SWA is implicitly biased towards flat minima, SAM explicitly approximates the flatness around parameters $\theta$ to guide the parameter update. It first computes the worst-case perturbation $\epsilon$ that maximizes the loss within a given neighborhood $\rho$, then minimizes the loss w.r.t. the perturbed weights $\theta + \epsilon$. Formally, SAM finds $\theta$ by solving the minimax problem:

$$\min_{\theta} \max_{\|\epsilon\|_2 \leq \rho} \mathcal{L}(\theta + \epsilon),$$  \hspace{1cm} (4)

where $\rho \geq 0$ is a hyperparameter.

To find the worst-case perturbation $\epsilon^*$ efficiently in practice, Foret et al. \[22\] approximates Eq. (4) via a first-order Taylor expansion of $\mathcal{L}(\theta + \epsilon)$ w.r.t. $\epsilon$ around 0, obtaining

$$\epsilon^* \approx \arg \max_{\|\epsilon\|_2 \leq \rho} \mathcal{L}(\theta + \epsilon) \approx \rho \cdot \frac{\nabla_{\theta} \mathcal{L}(\theta)}{\|\nabla_{\theta} \mathcal{L}(\theta)\|}.$$  \hspace{1cm} (5)

In words, $\epsilon^*$ is simply the scaled gradient of the loss function w.r.t. to the current parameters $\theta$. Given $\hat{\epsilon}$, the altered gradient used to update the current $\theta_t$ (in place of $g(\theta_t)$) is

$$\nabla_{\theta} \max_{\|\epsilon\|_2 \leq \rho} \mathcal{L}(\theta + \epsilon) \approx \nabla_{\theta} \mathcal{L}(\theta)|_{\theta + \hat{\epsilon}}.$$  

Due to Eq. (5), SAM’s computational overhead consists of an additional forward and backward pass per parameter update step compared to SWA and non-flat optimizers.

---

SWA has two hyper-parameters: the update frequency $\nu$ and starting epoch $E$. When using a constant learning rate, Izmailov et al. \[48\] suggests updating the parameters once after each epoch, i.e., $\nu \approx \frac{N}{E}$, and starting at $E \approx 0.75T$, where $T$ is the training budget required to train the model until convergence with conventional SGD training.

He et al. \[39\] argue that SWA may always improve generalization, regardless of the loss function’s geometry. Kaddour \[51\] show that averaging a specific range of weights can speed up training convergence. Cha et al. \[8\] argue that tuning $\nu$ and $E$ carefully is necessary to make it work effectively in domain generalization (DG) tasks. Besides DG tasks, a list of tuned hyper-parameters based on a fair model selection procedure across different architectures and tasks has been missing in the literature. To the best of our knowledge, Cha et al. \[8\] is the only study that compares SWA and SAM over the same experiments, but it focuses on domain generalization tasks which we, therefore, leave out in this work.

---

### Algorithm 1 Stochastic Weight Averaging \[48\]

**Input:** Loss function $\mathcal{L}$, training budget in number of iterations $b$, training dataset $D := \bigcup_{i=1}^{N} \{x_i\}$, mini-batch size $|B|$, averaging start epoch $E$, averaging frequency $\nu$, (scheduled) learning rate $\eta$, initial weights $\theta_0$.

for $k \leftarrow 1, \ldots, b$

Sample a mini-batch $B$ from $D$

Compute gradient $g \leftarrow \nabla_{\theta} \mathcal{L}(\theta_t)$

Update parameters $\theta_{t+1} \leftarrow \theta_t - \eta g$

if $k \geq E$ and $\mod(k, \nu) = 0$

$\theta_{t+1}^{SWA} = (\theta_{t+1}^{SWA}, \theta_{t+1}) / (l + 1)$

end if

end for

return $\theta_{t+1}^{SWA}$

---

### Algorithm 2 Sharpness-Aware Minimization \[22\]

**Input:** Loss function $\mathcal{L}$, training budget in number of iterations $b$, training dataset $D := \bigcup_{i=1}^{N} \{x_i\}$, mini-batch size $|B|$, neighborhood radius $\rho$, (scheduled) learning rate $\eta$, initial weights $\theta_0$.

for $k \leftarrow 1, \ldots, b$

Sample a mini-batch $B$ from $D$

Compute worst-case perturbation $\hat{\epsilon} \leftarrow \rho \frac{\nabla_{\theta} \mathcal{L}(\theta_t)}{\|\nabla_{\theta} \mathcal{L}(\theta)\|}$

Compute gradient $g \leftarrow \nabla_{\theta} \mathcal{L}(\theta_t^{SA} + \hat{\epsilon})$

Update parameters $\theta_{t+1}^{SAM} \leftarrow \theta_t^{SAM} - \eta g$

end for

return $\theta_{t+1}^{SAM}$

---

where $l$ is the number of distinct parameters averaged so far and $t$ is the SGD iteration number.\[2\]

---

\[2\]SWA parameters are constant between averaging steps.
SAM’s performance strongly depends on the neighborhood radius $\rho$. For example, Chen et al. [12], Wu et al. [93] show that $\rho$ should be set to values outside the originally considered ranges by Foret et al. [22]. Analogously to Sec. 2.2, this lack of coherence among hyper-parameter tuning protocols in the SAM literature makes it tricky to determine SAM’s comparative effectiveness.

2.4 Other Flat-Minima Optimizers

There are several extensions of SWA [36, 8] and SAM [65, 103, 101]. For simplicity, we do not consider them in this work. Besides SWA and SAM, other flat-minima optimizers include e.g., [9][84]. However, due to their computational cost and/or lack of performance gains, we do not include them in this work. Chaudhari et al. [9] requires $[5, 20]$ forward and backward passes per parameter update. Sankar et al. [84] similarly requires $[5, 10]$ forward and backward passes to estimate the Hessian trace and 6 of 7 experiments yield minimal improvement of $\leq 0.27\%$, see Table 1 in Sankar et al. [84]. In contrast, SWA and SAM have been shown to increase performance by multiple percentage points in some cases [8, 12] while requiring fewer computational resources.

3 How do minima found by SWA and SAM differ?

In this section, we investigate SWA and SAM solutions in two prototypical deep learning tasks, where these optimizers improve over the baseline. Our goal is to understand better their geometric properties (instead of their generalization performance, which is the focus of Sec. 4).

First, we investigate the behavior of the loss landscape along the line between non-flat and flat solutions (Sec. 3.1). Previous studies successfully used such linear interpolations to gain novel insights, e.g., for training dynamics [32, 25], regularization [69, 28], and network pruning [26]. Second, motivated by findings in Sec. 3.1, we average SAM iterates and visualize interpolations between averaged and non-averaged solutions (Sec. 3.2). Interestingly, the averaged SAM solution is less susceptible to asymmetric directions. Third, we compare quantitative measurements of all solutions’ flatnesses (Sec. 3.3). Here, we compute dominant Hessian eigenvalues, as commonly used in the flat minima literature [9, 98, 12, 22]. Lastly, in Appendix A.1 we further compute CKA [61] and cosine similarities between SWA/SAM’s network output logits.

We choose the following two disparate learning settings: (i) a well-known image classification task, widely used for evaluation in flat-minima optimizer papers, and (ii) a novel, challenging Python code summarization task, on which state-of-the-art models achieve only around 16% F1 score on the test set (which is higher than its commonly achieved accuracy on the more challenging training set), and that has not been explored yet in the flat-minima literature. Specifically, for (i), we investigate the loss/accuracy surfaces of a WideResNet28-10 [99] model on CIFAR-100 [63] (baseline non-flat optimizer: SGD with momentum (SGD-M) [83]). For (ii), we use the theoretically-grounded Graph Isomorphism Network [95] model on OGB-Code2 [45] (baseline optimizer: Adam [56]).

All optimizers start from the same initialization. We denote the minimizer produced by the non-flat methods (SGD-M and Adam) by $\theta^{\text{NF}}$ and the flat ones by $\theta^{\text{SWA}}$ and $\theta^{\text{SAM}}$.

3.1 What is between non-flat and flat solutions?

We start by comparing the similarity of flat and non-flat minimizers through linear interpolations. This analysis allows us to understand if they are in the same basin and how close they are to a region of sharply-increasing loss, where we expect loss/accuracy to differ widely between train and test. Further, for each of our four observations, we recommend a future work direction.

To linearly interpolate between two sets of parameters $\theta$ and $\theta'$, we parameterize the line connecting these two by choosing a scalar parameter $\alpha$ and defining the weighted average $\theta(\alpha) = (1 - \alpha)\theta + \alpha\theta'$. If there exists no high-loss barrier between two networks $\theta, \theta'$ along the linear interpolation, we say that they are located in the same basin, i.e., $\{\theta, \theta'\} \in \Omega$. [75] [102]. A basin is an area in the parameter space where the loss function has relatively low values. Due to NN non-linearities, the linear combination of the weights of two accurate models does not necessarily define an accurate model. Hence, we generally expect high-loss barriers along the linear interpolation path.

While there are alternative distance measures that could be used to compare two networks, they typically either (a) do not offer clear interpretations, as pointed out by Frankle et al. [26], or (b) yield
trivial network connectivity results, such as non-linear low-loss paths, which can be found for any two network minimizers $\theta^{\text{SWA}}, \theta^{\text{SAM}}$ are in the same basin, as can be seen in Figures 2a and 2e. Additionally, $\theta^{\text{SAM}}$ is near the periphery of a sharp increase in loss, as can be seen when moving in the direction from $\theta^{\text{SWA}}$ to $\theta^{\text{SAM}}$ (i.e., $\alpha > 1$). Conversely, $\theta^{\text{SWA}}$ finds flat regions that change slowly in the loss. This bias of SWA to flatter loss beneficially transfers to the accuracy landscape too: Figures 2b and 2f show the accuracy/F1 score rapidly dropping off approaching and beyond $\theta^{\text{SAM}}$. Interestingly, in Figures 2e and 2g, we see that for Code2, for $\alpha < 0$, there exist solutions with even better training loss/accuracy but worse test loss/accuracy. However, $\theta^{\text{SWA}}$ is close to the test accuracy maximizer along this interpolation. Future work may inspect why the cross entropy loss function $\ell$ used for GIN/Code2 seems less well correlated with its accuracy compared to WRN/CIFAR100.

Obs. 2: $\theta^{\text{SAM}} \in \Omega^{\text{SAM}} \neq \Omega^{\text{NF}}$, $\theta^{\text{SAM}}$ and $\theta^{\text{NF}}$ are not in the same basin: Figures 2c and 2g show that there is a high loss barrier between them, respectively. Figures 2d and 2h show that $\theta^{\text{SAM}}$ and even nearby points in parameter space achieve higher accuracies/F1 scores (i.e., generalize better) than $\theta^{\text{NF}}$ and points around it. This is an interesting result because we expect different basins to produce qualitatively different predictions, one of the motivations behind combining models, even if they exhibit different performances [46, 67]. Grewal & Bui [34] successfully combine models yielded by different optimizers, and we think future work should study ensembling SAM and non-SAM solutions.

Obs. 3: SAM finds a saddle point. Figure 2g shows $\theta^{\text{GIN}}$ being located in a sharp training loss minimum whose loss is much higher than $\theta^{\text{SAM}}$. Yet, its test loss is slightly higher, and its F1 score is better. We visualize 2D plots moving along random directions (not shown here due to space) to confirm that $\theta^{\text{GIN}}$ is a saddle point (Appendix A.2). A common pathology among curvature-based methods is that they attract saddle points [16]. Since SAM takes some form of curvature into account, too, we believe that future work should investigate SAM’s propensity to find saddle points and potential remedies.

Obs. 4: $\theta^{\text{SAM}}$ is closer to sharper directions than $\theta^{\text{SWA}}$, as can be seen by $L_{\text{true}}(\theta^{\text{SAM}}(0.1)) \approx 2 \cdot L_{\text{true}}(\theta^{\text{SWA}}(-0.1))$, while $L_{\text{true}}(\theta^{\text{SWA}}(0.1)) \approx L_{\text{true}}(\theta^{\text{SWA}}(-0.1))$, where $\ell$ is the loss function. A possible explanation for SAM being closer to sharper sides is that while it finds different basins than SGD/SWA by smoothing the loss surface (as illustrated in Fig.1), within a local basin, it may oscillate around the minimizer similarly as SGD. One cause for this can be that $\Omega^{\text{SAM}}$’s hypersphere is larger than SAM’s radius $\rho$. If that holds, then given a small enough learning rate, we expect it to oscillate around $\theta^{\text{SAM}}$ (the smaller the learning rate, the less likely it escapes the basin due to that stochasticity). Two possible remedies are: (1) adapt the schedule $\theta$, or (2) average SAM iterates to bias its solution towards the flatter side. (1) has been explored by [103, 101]. We try (2) in the next subsection. Future work may study SAM’s basin escape time, e.g., using convolutions [59] or stochastic differential equations [102].
3.2 What happens if we average SAM iterates?

Based on observation 4: “$\theta_{\text{SAM}}$ is closer to sharper directions than $\theta_{\text{SWA}}$”, averaging SAM iterates may further improve generalization, referred to as Weight-Averaged Sharpness-Aware Minimization (WASAM). The reason is that while SAM finds better-performing basins, within the basin, its final iterate may still be near a side that increases sharply in the loss.

Starting with the first of the two previously analyzed settings (WRN/CIFAR100), Figures 3a, and 3b show that $\theta_{\text{SWA+SAM}}$ (marker: ⭐) achieves the lowest test loss and highest test accuracy, respectively. What stands out in comparison to the previous plots is $\theta_{\text{SWA+SAM}}$'s (×) proximity to sharp sides, surprisingly similar to $\theta_{\text{WASAM}}$ (●) here and in Figures 2c and 2e. As we hoped, $\theta_{\text{SWA+SAM}}$ is indeed closer to a flatter region, as can be seen by $L_{\text{tr/te}}(\theta_{\text{SWA+SAM}}(-0.2)) \approx L_{\text{tr/te}}(\theta_{\text{SWA+SAM}}(0.2))$.

In GIN/OGB-Code2, one unanticipated finding is that $\theta_{\text{SWA+SAM}}$ escapes the (previously discussed) saddle point of $\theta_{\text{GIN}}$ appearing here as a maximum in Figure 3c. A likely reason is that SAM traversed nearby flatter regions before arriving at the saddle point, especially if it is a non-strict saddle. In terms of F1 score, Figure 3d shows that while $\theta_{\text{SWA}}$ (+) and $\theta_{\text{SAM}}$ perform about equally well, the flatter region found by $\theta_{\text{SWA+SAM}}$ improves over both.

3.3 How “flat” are the found minima?

We now quantify the flatnesses of all four optimizers over both tasks by computing the median of the dominant Hessian eigenvalue across all training set batches using the Power Iteration algorithm [74][98]. This metric measures the worst-case loss landscape curvature. We choose this metric as it is very commonly used in the minima flatness literature, e.g., [9][12][22][97][18][62][85]. Table 1 shows that SAM leads to flatter minima than SWA in both cases. Interestingly $\lambda_{\text{max}}(\theta_{\text{WASAM}}) \approx 2.5 \cdot \lambda_{\text{max}}(\theta_{\text{GIN}})$, while $\lambda_{\text{max}}(\theta_{\text{SWA}}) \approx 5.75 \lambda_{\text{max}}(\theta_{\text{WASAM}})$, indicating room for improvement in terms of flatness for both SWA and SAM. The relative differences are less dramatic for GIN/Code2, although surprisingly $\lambda_{\text{max}}(\theta_{\text{WASAM}}) \approx \lambda_{\text{max}}(\theta_{\text{GIN}})$. In sum, averaging SAM iterates leads to the flattest minima and best-performing minima in both cases (see Sec. 4).

<table>
<thead>
<tr>
<th>Task</th>
<th>Baseline</th>
<th>SWA</th>
<th>SAM</th>
<th>WASAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRN on CIFAR100</td>
<td>673</td>
<td>265</td>
<td>237</td>
<td>117</td>
</tr>
<tr>
<td>GIN on Code2</td>
<td>16.65</td>
<td>16.79</td>
<td>11.31</td>
<td>9.96</td>
</tr>
</tbody>
</table>

4 How do SWA and SAM perform on a broad set of experiments?

As we point out in the introduction, there is almost no overlap and consistency regarding reported SWA and SAM results in the literature. This section addresses this gap. For example, Bahri et al. [4], Chen et al. [12] illustrate that the flat minima found by SAM improve generalization on Transformer [90] architectures compared to non-flat optimizers, but they do not compare against SWA. Hence, it is unclear if the computationally cheaper SWA may provide better or similar performance.

We compare flat minimizers SWA, SAM, and averaged SWA iterates (WASAM) over the non-flat minimizers across a range of different tasks in the domains of computer vision, natural language processing, and graph representation learning. We average all runs at least three times across random


Table 2: CV test results: Supervised Classification (SC), and Self-Supervised Learning (SSL) tasks.

<table>
<thead>
<tr>
<th>Task</th>
<th>Model</th>
<th>Baseline</th>
<th>SWA</th>
<th>SAM</th>
<th>WASAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC:</td>
<td>WRN-28-10</td>
<td>96.76±0.03</td>
<td>-0.05±0.04</td>
<td>+0.34±0.09</td>
<td>+0.25±0.05</td>
</tr>
<tr>
<td>CIFAR10</td>
<td>PN-272</td>
<td>96.73±0.14</td>
<td>+0.22±0.14</td>
<td>+0.42±0.06</td>
<td>+0.41±0.02</td>
</tr>
<tr>
<td></td>
<td>VIT-B-16</td>
<td>98.95±0.02</td>
<td>-0.04±0.04</td>
<td>+0.07±0.01</td>
<td>+0.10±0.01</td>
</tr>
<tr>
<td></td>
<td>Mixer-B-16</td>
<td>96.65±0.03</td>
<td>+0.02±0.02</td>
<td>+0.19±0.05</td>
<td>+0.22±0.06</td>
</tr>
<tr>
<td>SSL:</td>
<td>WRN-28-10</td>
<td>80.93±0.19</td>
<td>+1.02±0.06</td>
<td>+1.82±0.14</td>
<td>+2.24±0.14</td>
</tr>
<tr>
<td>CIFAR100</td>
<td>PN-272</td>
<td>80.86±0.12</td>
<td>+1.88±0.04</td>
<td>+2.53±0.08</td>
<td>+2.60±0.09</td>
</tr>
<tr>
<td></td>
<td>VIT-B-16</td>
<td>92.77±0.07</td>
<td>-0.12±0.05</td>
<td>+0.19±0.09</td>
<td>+0.13±0.07</td>
</tr>
<tr>
<td></td>
<td>Mixer-B-16</td>
<td>83.77±0.08</td>
<td>+0.45±0.06</td>
<td>+0.52±0.15</td>
<td>+0.97±0.12</td>
</tr>
<tr>
<td>SSL:</td>
<td>MoCo</td>
<td>89.25±0.07</td>
<td>-0.03±0.10</td>
<td>-0.25±0.06</td>
<td>-0.17±0.10</td>
</tr>
<tr>
<td>ImageNette</td>
<td>SimCLR</td>
<td>88.66±0.03</td>
<td>-0.05±0.06</td>
<td>+0.05±0.04</td>
<td>+0.13±0.06</td>
</tr>
<tr>
<td></td>
<td>SimSiam</td>
<td>89.86±0.22</td>
<td>+0.12±0.26</td>
<td>+0.07±0.10</td>
<td>+0.11±0.10</td>
</tr>
<tr>
<td></td>
<td>BarlowTwins</td>
<td>86.34±0.24</td>
<td>-0.09±0.19</td>
<td>+0.09±0.15</td>
<td>+0.14±0.05</td>
</tr>
<tr>
<td></td>
<td>BYOL</td>
<td>90.32±0.14</td>
<td>+0.70±0.05</td>
<td>+0.14±0.13</td>
<td>+0.21±0.07</td>
</tr>
<tr>
<td></td>
<td>SwaV</td>
<td>87.28±0.05</td>
<td>+0.09±0.06</td>
<td>+0.07±0.12</td>
<td>+0.02±0.08</td>
</tr>
<tr>
<td>SSL:</td>
<td>MoCo</td>
<td>81.74±0.18</td>
<td>+0.07±0.10</td>
<td>+0.01±0.32</td>
<td>+1.40±1.09</td>
</tr>
<tr>
<td>ImageNette</td>
<td>SimCLR</td>
<td>83.28±0.22</td>
<td>+0.05±0.25</td>
<td>+0.18±0.24</td>
<td>+1.07±0.13</td>
</tr>
<tr>
<td></td>
<td>SimSiam</td>
<td>81.77±0.14</td>
<td>+0.20±0.37</td>
<td>+0.33±0.28</td>
<td>+0.18±0.26</td>
</tr>
<tr>
<td></td>
<td>BarlowTwins</td>
<td>77.49±0.16</td>
<td>+0.20±0.16</td>
<td>+0.47±0.27</td>
<td>+0.66±0.57</td>
</tr>
<tr>
<td></td>
<td>BYOL</td>
<td>81.16±0.14</td>
<td>+0.76±0.08</td>
<td>+0.17±0.25</td>
<td>+0.31±0.13</td>
</tr>
<tr>
<td></td>
<td>SwaV</td>
<td>88.16±0.31</td>
<td>+1.04±0.27</td>
<td>+0.03±0.10</td>
<td>+1.03±0.09</td>
</tr>
</tbody>
</table>

Hyper-parameters. For all architectures and datasets, we set hyperparameters shared by all methods (e.g., learning rate) mostly to values cited in prior work\(^1\). As explained in Secs. 2.2 and 2.3, the effectiveness of flat minima optimizers is highly sensitive to their additional hyperparameters. For SWA we follow Izmailov et al.\(^{48}\) and hold the update frequency \(\nu\) constant to once per epoch and tune the start time \(E \in \{0.5T, 0.6T, 0.75T, 0.9T\}\) (\(T\) is the number of baseline training epochs). Izmailov et al.\(^{48}\) argue that a cyclic learning rate starting from \(E\) helps to encourage exploration of the basin. For the sake of simplicity, we average the iterates of the baseline directly but include even earlier starting times (i.e., 0.5T, 0.6T). For SAM, we tune its neighborhood size \(\rho \in \{0.01, 0.02, 0.05, 0.1, 0.2\}\), as in previous work\(^{22,4}\).

Appendix B contains the values of all hyper-parameters and additional training details (including public model checkpoints, hardware infrastructure, software libraries, etc.) to ensure full reproducibility alongside open-sourcing our code.

4.1 Computer Vision

Supervised Classification (SC). We evaluate the CNN architectures WideResNets\(^{99}\) with 28 layers and width 10, and PyramidNet (PN) with 110 layers and widening factor 272\(^{38}\) as well as Vision Transformer (VIT)\(^{19}\) and MLP-Mixer\(^{87}\) on CIFAR\(^{10,100}\)\(^{63}\). All experiments use basic data augmentations: horizontal flip, padding by four pixels, random crop, and cutout\(^{17}\).

Self-Supervised Learning (SSL). We consider the following methods on CIFAR10 and ImageNette\(^{4}\), Momentum Contrast\(^{41}\), a Simple framework for Contrastive Learning (SimCLR)\(^{10}\), Simple Siamese representation learning (SimSiam)\(^{11}\), Barlow Twins\(^{100}\), Bootstrap your own Latent (BYOL)\(^{35}\), and Swapping Assignments between multiple Views of the same image (SwaV)\(^{7}\). All SSL methods use a ResNet-18\(^{40}\) as backbone network. To test the frozen representations, we use \(k\)-nearest-neighbor classification with a memory bank\(^{94}\). We choose \(k = 200\) and temperature \(\tau = 0.1\) to reweight similarities. Compared to learning a linear model on top of the representations, this evaluation procedure is more robust to hyperparameter changes\(^{59}\).
We use a subset of the Open Graph Benchmark (OGB) datasets \[45\]. The tasks are node property prediction (NPP), graph property prediction (GPP), and link property prediction (LPP).

4.2 Natural Language Processing

We consider the task of open domain question answering (ODQA) using a T5-based model Fusion-In-Decoder (FiD) \[47\]. We evaluate FiD-base on the test sets of Natural Questions (NQ) \[54\] and TriviaQA \[50\]. We also consider natural language understanding tasks included in the GLUE benchmark \[91\], which cover acceptability, sentiment, paraphrase, similarity, and inference. We fine-tune RoBERTa-base \[72\] for each task individually and report the results on the GLUE dev set.

4.3 Graph Representation Learning

We use a subset of the Open Graph Benchmark (OGB) datasets \[45\]. The tasks are node property prediction (NPP), graph property prediction (GPP), and link property prediction (LPP). For each task, we use two of the following GNN architectures and matrix factorization methods: GCN \[57\], DeeperGCN (DGCN) \[63\], SAGE \[37\], GIN \[95\], ComplEx \[89\], and CP \[66\]. We use popular training schemes, such as virtual nodes, cluster sampling \[14\], or relation prediction as auxiliary training objective \[13\]. The reported metrics are ROC-AUC for Proteins, Accuracy for Products, F1 score for Code2, Average precision for Molpeba, and Mean Reciprocal Rank for Biokg/Citation2.

4.4 9 Findings

We use $G(\cdot)$ to describe the generalization accuracy/F1/ROCAUC/AP/MRR of all optimizers \{Non-flat baseline(NF), SWA, SAM, WASAM\}.

1. Datasets matter. For example, for node property prediction (Proteins), we see that no flat optimizer improves over the baseline optimizer; however, for (Products), flat-minima optimizers on the same architectures significantly improve over the baseline. We further explore the impact of different data augmentation strategies in Appendix A.7.

2. Architectures matter, e.g., there is a vast difference across model architectures for link property prediction on the Citation2 dataset: using a GNN with GCN layers, SAM achieves a statistically significant boost of >1.30% and SWA slightly hurts the performance. When we replace the GCN layers with SAGE layers (and fix everything else), we see a boost of > 1.15% for SWA, while SAM hurts the performance by ~0.31%.

3. SWA underperforms on NLP tasks. SAM achieves the best performance in 7/10 experiments on NLP tasks, consistent with the findings of Bahri et al. \[4\], which show that SAM can boost performance across a wide range of NLP tasks. However, SWA never performs best, only improving the results in 1/10 cases, and even hurting the performance on 4 tasks. Surprisingly, $G(\text{WASAM}) > G(\text{SAM})$ for \{SST, QNLI\} while SWA decreases performance in these cases.

4. SWA beats SAM on GRL tasks. $G(\text{SWA}) > G(\text{NF})$ in 10/12 experiments, while $G(\text{SAM}) > G(\text{NF})$ only in 4. We examine why SAM under-performs in Appendix A.4.

5. SWA does not work well with Transformers. SWA often does not improve and sometimes hurts performances, as can be seen in the ViT results in Table 2 and NLP results in Fig. 4e. In contrast, SAM has some positive effects in these settings. We explore this further in Appendix A.5.
6. **SWA and SAM improve SSL task performance.** This is non-trivial as the theoretical motivation behind finding flat minima is linked to supervised learning losses. Concurrently to our work, Ramesh et al. report that SAM helps for contrastive CLIP models too.

7. **Flat optimizers do not strictly improve over non-flat optimizers.** The non-flat optimizer is nearly always the best for NPP Proteins and SSL methods on CIFAR10. We investigate the NPP Proteins solutions in the next subsection and recommend a more thorough investigation of the landscapes of SSL objectives for future work.

8. **Flat-minima optimizers offer asymmetric payoffs:** at worst, they decreased performance by \(-0.30\%\), at best, they increased it by \(2.60\%\).

9. **Averaging SAM iterates often improves over SWA or SAM alone.** \(\mathcal{G}(\text{WASAM}) > \min(\mathcal{G}(\text{SWA}), \mathcal{G}(\text{SAM}))\) in 39/42 cases. We hypothesize that asymmetric payoffs are the reason: when either SWA or SAM does not improve over the baseline (as discussed above), it does not hurt (much) either, hence WASAM is more robust across all tasks.

### 4.5 Why do flat-minima optimizers fail?

Here, we audit one of the cases, where neither \(\theta^{\text{SWA}}\) nor \(\theta^{\text{SAM}}\) improves over \(\theta^{\text{NF}}\) (this happens in 3 out of 42 cases): training a GraphSAGE model on OGB-Proteins: a protein-protein interaction graph where the goal is to predict the presence of protein functions (multi-label binary classification). \(\theta^{\text{SWA}}\) performs noticeably worse; \(\theta^{\text{SAM}}\) performs about equally well.

![GraphSAGE on OGB-Proteins: Adam’s (•) solution performs about equally well as SAM (×), and better than SWA (+).](image)

Fig. 5 shows two linear interpolations: between \(\theta^{\text{NF}}\) (ADAM) and (1) \(\theta^{\text{SWA}}\) (Figures 5a and 5b), and (2) \(\theta^{\text{SAM}}\) (Figures 5c and 5d). In contrast to success cases in Fig. 2 here: (a) for both SWA and SAM, the training loss minimizer is very uncorrelated with the test loss minimizer; (b) SAM and ADAM seem to be contained in the same test loss/accuracy basin. More analyses can be found in the Appendix.

### 5 Limitations and Future Work

First, some of the fixed, shared hyperparameter values we used from previous works may harm the effect of flat optimizers. The ideal experimental design includes tuning all hyperparameters independently for the non-flat optimizer, SWA, SAM, and WASAM. However, this forces the number of required runs to grow exponentially in unique hyperparameters and quickly renders this benchmark infeasible.

Second, despite our best efforts to evaluate the optimizers on a broad range of benchmark tasks, there are still plenty of unexplored domains; especially some of which are known to be sensitive to careful optimization. For example, bi-level optimization problems are common in generative modeling, deep reinforcement learning, meta-learning, or causal machine learning. We are unaware of an investigation of flat minima optimization for such problems.

Third, in general, we believe fruitful directions of research include (a) optimizers that explicitly find basins where training loss flatness more directly corresponds to higher hold-out accuracy, (b) post-processing methods for existing optimization runs to move into flatter regions of these basins, (c) loss functions whose contours more tightly align with accuracy contours, (d) the study of flat-minima hyperparameter interactions (e.g., learning rate and neighborhood radius in SAM) (see Appendices A.5 and A.6 for first results), (e) analyses of flat minima optimization on convergence speed.
Our benchmark results point to which tasks would most benefit from improving these future work directions: graph learning tasks would benefit from improvements in (a), as SAM is never among the best-performing method, and language tasks would benefit if (b) is improved, as SWA is never among the best performing method).

6 Conclusion

We investigated when flat minima optimizers work by conducting a fair comparison of two popular flat-minima optimizers. We examined the behavior of SWA/SAM by analyzing their loss landscapes on two representative deep learning tasks. Our next step was to evaluate their generalization performance on a broad and diverse set of tasks (in data, learning settings, and model architectures). Based on this benchmarking, we identified 9 findings, of which some directly guide future work directions. Finally, when SWA/SAM did not improve over baselines, common assumptions seemed broken (i.e., train-to-test loss minimizers were not correlated).

Acknowledgements

We are very grateful to Kilian Q. Weinberger and Gao Huang for initial discussions and intuition, Pontus Stenetorp for NLP experimental design advice, and Oscar Key for feedback on the draft. JK and LL acknowledge support by the Engineering and Physical Sciences Research Council with grant number EP/S021566/1. This research was supported through Azure resources provided by The Alan Turing Institute and credits awarded by Google Cloud.

References


Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Sec. 5
   (c) Did you discuss any potential negative societal impacts of your work? [No] We consider this work to be an investigation into optimization algorithms that are central to modern ML systems. As these systems can be used in vastly different ways, we believe that it is intractable to isolate the societal impact of novel insights into optimization.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Supplemental material
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Sec. 4
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] See Appendix.
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you’re using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]