Optimization of deep neural networks

Milan Straka

📅 Oct 26, 2023

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

$p(y|x;\theta)$

$\rightarrow \ell(x,y;\theta) =$

$= \boxed{-\log} p(y|x;\theta)$

MLE

$x,y \sim D$

$S = \{(x_1,y_1),(x_2,y_2),\ldots,(x_n,y_n)\}$

$\rightarrow \ell(S;\theta) = \frac{1}{n}\sum_{x,y\in S} \ell(x,y;\theta)$

$\ell(\theta)$

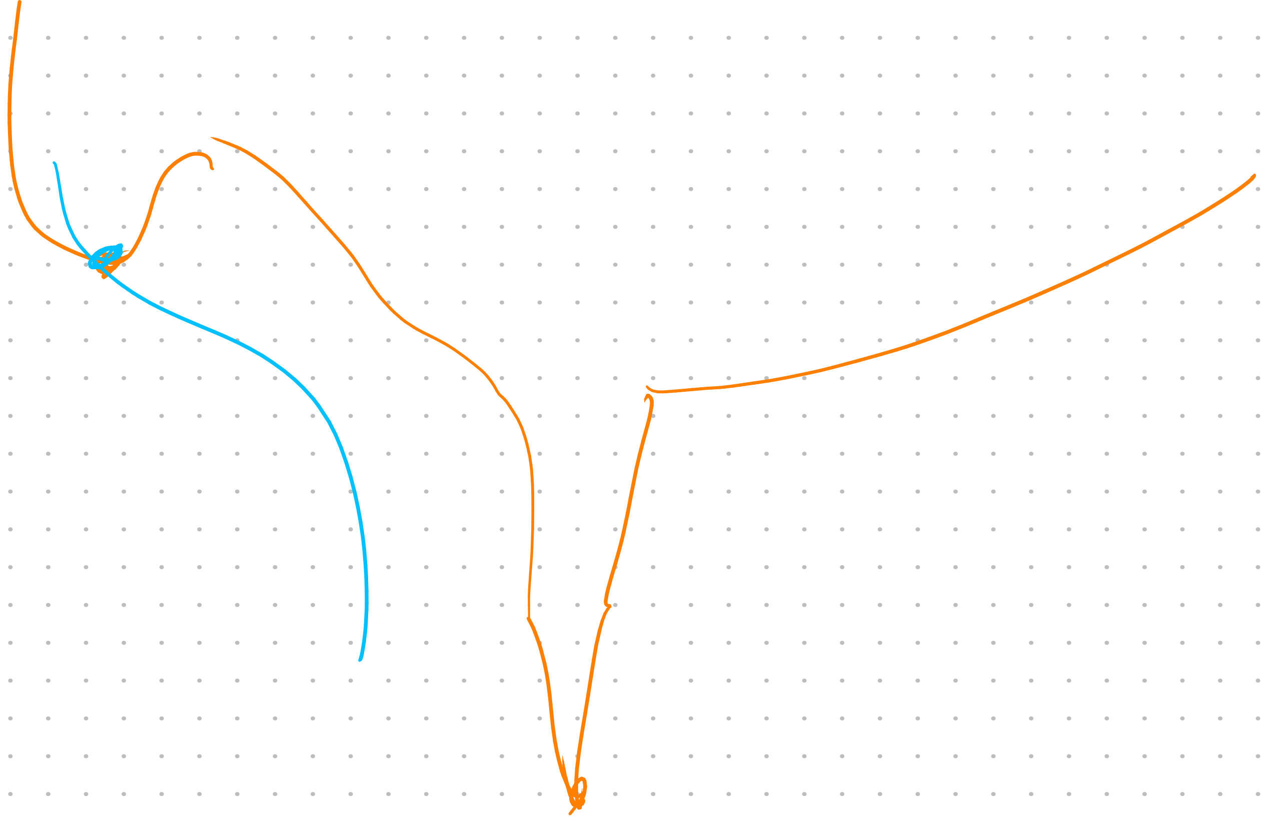
$f(x;\theta) \rightarrow \overset{175}{z}$

$R(y|z)$

$\uparrow$

$N \quad N(175;1)$

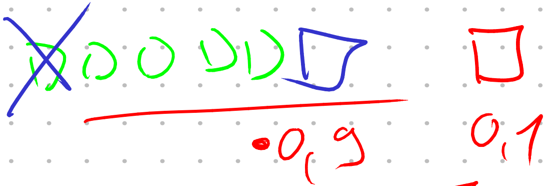
$Cat(y|z)$

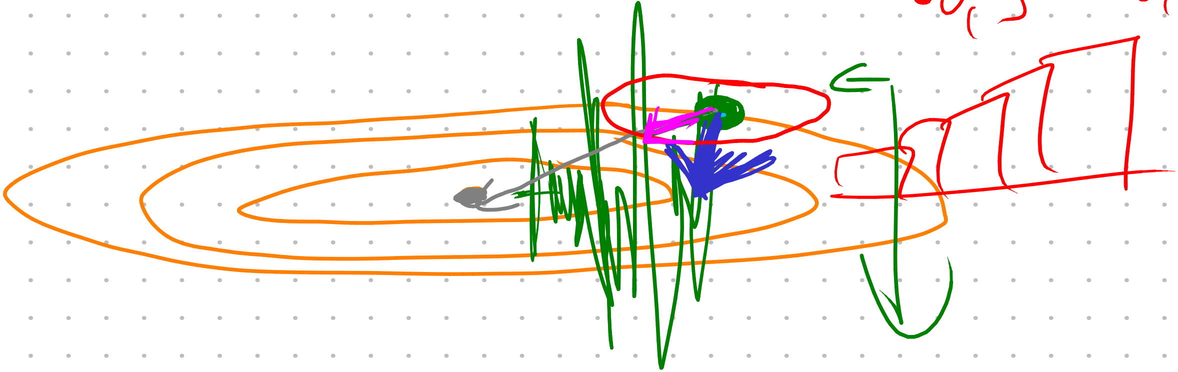


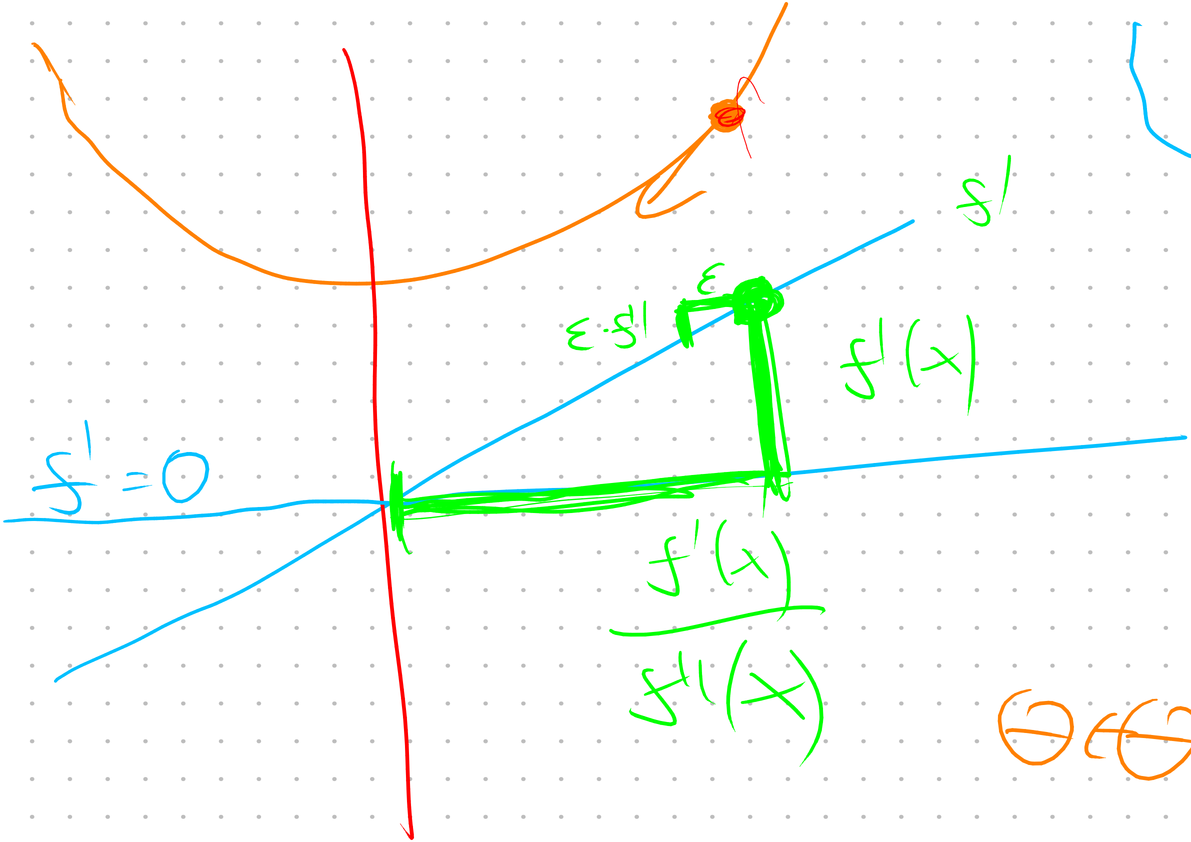
(5) GD

$$\nabla_\theta L(S_i;\theta) = \left(\frac{\partial L}{\partial \theta_1}, \dots, \frac{\partial L}{\partial \theta_d}\right)$$

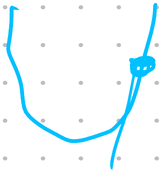
$$\theta \leftarrow \theta - \alpha \nabla_\theta L$$







$$\frac{f'(x)}{f''(x)}$$



$$\theta \leftarrow \theta - \boxed{\frac{f''(x)}{f''(x)}}$$

$|\theta|$ $\nabla_\theta \ell$ $\in \mathbb{R}^d$

$$f(\theta+\varepsilon) \approx f(\theta) + \varepsilon \cdot f'(\theta) + \frac{1}{2}\varepsilon^2 \cdot f''(\theta) + O(\varepsilon^3)$$

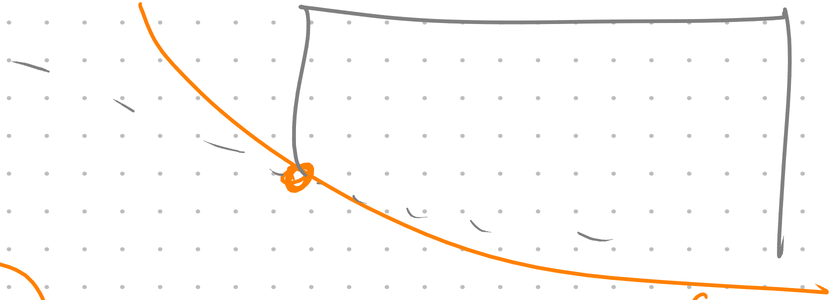
$$f(\theta) + \varepsilon \cdot \nabla_\theta f(\theta)^T + \underline{\frac{1}{2} \cdot \varepsilon^T H \varepsilon}$$

$$H(f)_{i,j} = \frac{\partial^2 f}{\partial\theta_i \partial\theta_j}$$

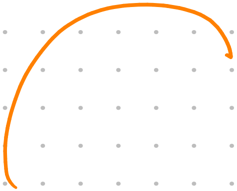
$$\frac{\partial}{\partial \varepsilon} = \nabla_\theta f(\theta) + H \cdot \varepsilon$$

$$-H^{-1} \nabla_\theta f(\theta) = \varepsilon$$

$=$



$/ H$



$$(H + \lambda I)^{0.5}$$

FIM

$p(y|x;\theta)$

score $\boxed{\nabla_\theta \overset{\downarrow}{\log} \, p(y|x;\theta)}$

$$\boxed{\frac{\nabla_\theta \, p(y|x;\theta)}{p(y|x;\theta)}}$$

$$\mathbb{E}_{y|x \sim p}[s(x,y)] = 0$$

$$\sum \cancel{p(y|x)} \cdot \frac{\nabla_\theta p(y|x)}{\cancel{p(y|x)}} =$$

$$\nabla_\theta \underbrace{\sum p(y|x)}_{1} = 0$$

Empirical FIM

$\bar{F}$

$y|x \sim D$

FIM

$$F = \underbrace{\mathbb{E}_{x \sim D}}_{\text{Empirical FIM}} \underbrace{\mathbb{E}_{y|x \sim p}}_{y|x \sim D} \left[(s - \mathbb{E}s)(s - \mathbb{E}s)^T \right]$$

$$\left[s \cdot s^T \right]$$

$$\boxed{g \cdot g^T}$$

$$H_{\ell|S;\theta} = \boxed{\mathbb{E}_{x,y \in S}} \, H_{\ell(x,y;\theta)}$$

$$H \; -\log p(y|x;\theta)$$

$$\left(g \cdot \left(g^T x\right)\right)$$

$$H_{-\log p(y|x;\theta)} = \nabla_\theta \boxed{\nabla_\theta \log p(y|x)}$$

$$= \nabla_\theta \left(-\frac{\nabla_\theta p(y|x;\theta)}{p(y|x;\theta)} \right)$$

$$\left(\frac{f}{g}\right)' = \frac{f'g - fg'}{g^2}$$

$$= \frac{-\left[\nabla_\theta \nabla_\theta p(y|x)\right] \cdot \cancel{p(y|x)} + \nabla p (\nabla p)^T}{p(y|x)\,\cancel{p(y|x)}}$$

$$= -\frac{H_{p(y|x)}}{p(y|x)} + \left(\frac{\nabla p}{p}\right)\left(\frac{\nabla p}{p}\right)^T$$

$$\mathbb{E}_{x\sim D} \boxed{\mathbb{E}_{y|x\sim p}}\, H = \sum \cancel{p(y|x)} \cdot \frac{H_{p(y|x)}}{\cancel{p(y|x)}} = \mathbb{E}_x \mathbb{E}_{y|x} \left[s \cdot s^T \right]$$

$$D_{KL}\left(\boxed{p(y|x;\theta)} \;\|\; \boxed{p(y|x;\theta+d)}\right)$$

$$E\; 0 + d\cdot\nabla_\theta \; -\log p(y|x;\theta) + \frac{1}{2} d^T H_{-\log p(y|x;\theta)}\, d$$

$$\approx \boxed{\frac{1}{2} d^T F d}$$

$$D_{KL}(P \,\|\, Q) \geq 0$$

$$= 0 \iff P = Q$$

$$\mathbb{E}_{p(x)}\left[-\log q(x)\right]$$

$$\boxed{\sum_x -p(x)\cdot\left(\log \frac{q(x)}{p(x)}\right)}$$

$$\lim_{\varepsilon\to0}\frac{1}{\varepsilon}\underset{d;\ \boxed{\|d\|\le\varepsilon}}{\operatorname{argmin}}\ \ell(s;\theta+d) = \frac{-\nabla_\theta \ell(s;\theta)}{\|\nabla_\theta \ell(s;\theta)\|}$$

$$\uparrow$$

$$\frac{\partial}{\partial d}\quad \cancel{\ell(s;\theta)} + d\cdot\underbrace{\nabla_\theta \ell(s;\theta)}_{\downarrow} + \lambda\cdot(d^T d - \varepsilon^2)$$

$$0 = \nabla_\theta \ell(s;\theta) + 2\lambda\boxed{d}$$

$$d \propto -\nabla_\theta \ell(s;\theta)$$

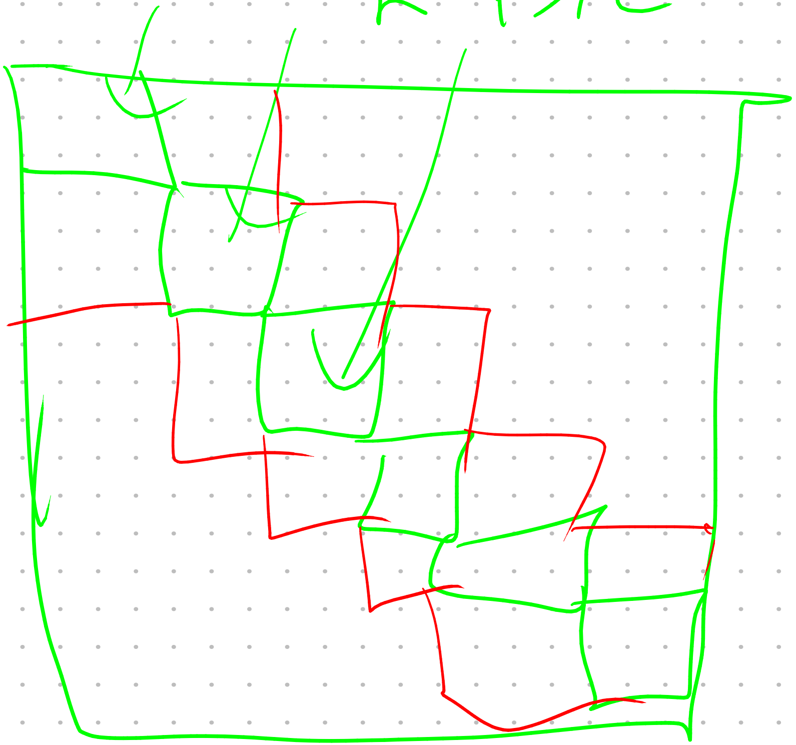
$$\underset{d \,:\, \boxed{D_{KL}}\left(p(y|x_i;\theta) \,\|\, p(y|x_i;\theta+d)\right) \leq \varepsilon}{\arg\min} \boxed{\ell(S_i;\theta+d)}$$

$$\ell(S_i;\theta) + d \cdot \nabla_\theta \ell(\theta) + \lambda\left(\frac{1}{2} d^T \cdot \mathbf{F} \cdot d\right)$$

$$\boxed{d \propto -\mathbf{F}^{-1} \nabla_\theta L(S_i;\theta)}$$

$$F_{diag}^{-1}$$

F^{-1} K-FAC

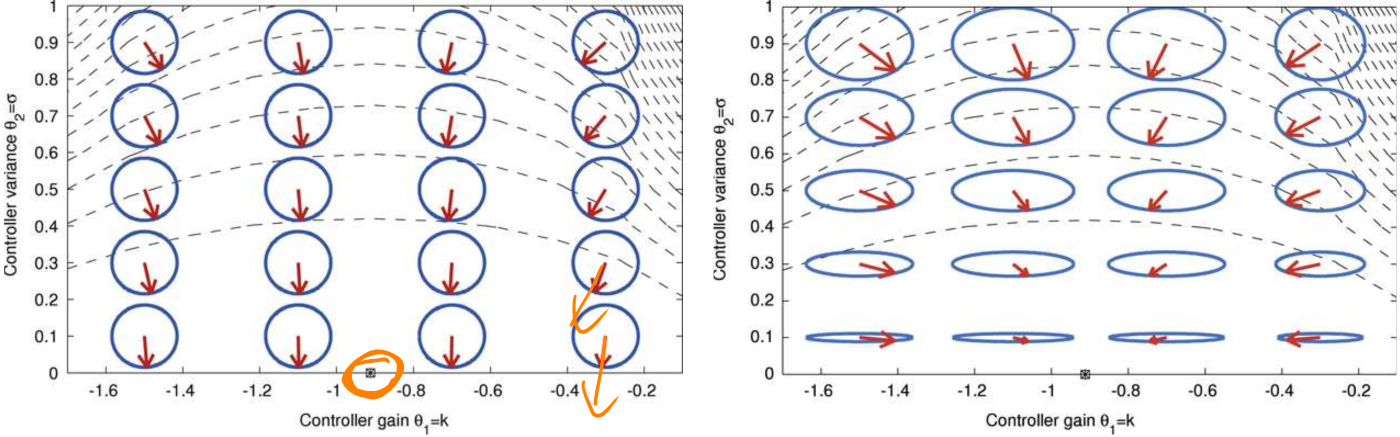


$$s = \beta_1 \cdot s + (1-\beta_1) \cdot g$$

$$r = \beta_2 \cdot r + (1-\beta_2) \cdot g \odot g$$

$$\frac{1}{t^{0,5}}$$

$$\theta = \theta - \frac{\alpha}{\sqrt{r} + \varepsilon} \cdot \boxed{s}$$



(a) Vanilla policy gradient.

(b) Natural policy gradient.

Fig. 1. The classical example of LQR can be used to illustrate why 'vanilla' policy gradients reduce the exploration to zero while natural policy gradients go for the optima solution. The main difference is how the two approaches punish the change in parameters, i.e., the distance between current and next policy parameters. This distance is indicated by the blue ellipses in the contour plot while the dashed lines show the expected return. Obtaining a gradient then corresponds to finding a vector pointing from the center of the ellipses to the location with maximum expected return on the ellipse. A vanilla policy gradient (a) considers a change in all parameters as equally distant thus, it is a search for a maximum on a circle while the natural gradient (b) uses scales determined by the Fisher information which results in a reduction in exploration The slower reduction in exploration results into a faster convergence to the optimal policy. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Journal of Machine Learning Research 21 (2020) 1-76 Submitted 11/17; Revised 6/20; Published 8/20

New Insights and Perspectives on the Natural Gradient Method

James Martens JAMES.MARTENS@GMAIL.COM
DeepMind
London, United Kingdom

Editor: Léon Bottou

Abstract

Natural gradient descent is an optimization method traditionally motivated from the perspective of information geometry, and works well for many applications as an alternative to stochastic gradient descent. In this paper we critically analyze this method and its properties, and show how it can be viewed as a type of 2nd-order optimization method, with the Fisher information matrix acting as a substitute for the Hessian. In many important cases, the Fisher information matrix is shown to be equivalent to the Generalized Gauss-Newton matrix, which both approximates the Hessian, but also has certain properties that favor its use over the Hessian. This perspective turns out to have significant implications for the design of a practical and robust natural gradient optimizer, as it motivates the use of techniques like trust regions and Tikhonov regularization. Additionally, we make a series of contributions to the understanding of natural gradient and 2nd-order methods, including: a thorough analysis of the convergence speed of stochastic natural gradient descent (and more general stochastic 2nd-order methods) as applied to convex quadratics, a critical examination of the oft-used "empirical" approximation of the Fisher matrix, and an analysis of the (approximate) parameterization invariance property possessed by natural gradient methods (which we show also holds for certain other curvature matrices, but notably not the Hessian).

Keywords: natural gradient methods, 2nd-order optimization, neural networks, convergence rate, parameterization invariance

This kind of objective function fits into the general supervised learning framework described in Section 3 as follows. We define the learned conditional distribution $P_{y|x}(\theta)$ to be the composition of the deterministic prediction function $f(x, \theta)$ (which may be a neural network), and an "output" conditional distribution $R_{y|z}$ (with associated density function $r(y|z)$), so that

$$P_{y|x}(\theta) = R_{y|f(x,\theta)} \, .$$

We then define the loss function as $L(y, z) = -\log r(y|z)$.

Given a loss function L which is not explicitly defined this way one can typically still find a corresponding R to make the definition apply. In particular, if $\exp(-L(y, z))$ has the same finite integral w.r.t. y for each z, then one can define R by taking $r(y|z) \propto \exp(-L(y, z))$, where the proportion is w.r.t. both y and z.

5. Various Definitions of the Natural Gradient and the Fisher Information Matrix

The Fisher information matrix F of $P_{x,y}(\theta)$ w.r.t. θ (aka the "Fisher") is given by

$$\begin{aligned} F &= \mathrm{E}_{P_{x,y}}\left[\nabla \log p(x, y|\theta) \nabla \log p(x, y|\theta)^\top\right] && (3) \\ &= -\mathrm{E}_{P_{x,y}}\left[H_{\log p(x,y|\theta)}\right] \, . && (4) \end{aligned}$$

where gradients and Hessians are taken w.r.t. θ. It can be immediately seen from the first of these expressions for F that it is positive semi-definite (PSD) (since it's the expectation of something which is trivially PSD, a vector outer-product). And from the second expression we can see that it also has the interpretation of being the negative expected Hessian of $\log p(x, y|\theta)$.

The usual definition of the natural gradient (Amari, 1998) which appears in the literature is

$$\tilde{\nabla} h = F^{-1} \nabla h \, ,$$

where F is the Fisher and h is the objective function.

Because $p(x, y|\theta) = p(y|x, \theta)q(x)$, where $q(x)$ doesn't depend on θ, we have

$$\nabla \log p(x, y|\theta) = \nabla \log p(y|x, \theta) + \nabla \log q(x) = \nabla \log p(y|x, \theta) \, ,$$

and so F can also be written as the expectation (w.r.t. Q_x) of the Fisher information matrix of $P_{y|x}(\theta)$ as follows:

$$F = \mathrm{E}_{Q_x}\left[\mathrm{E}_{P_{y|x}}\left[\nabla \log p(y|x, \theta) \nabla \log p(y|x, \theta)^\top\right]\right] \quad \text{or} \quad F = -\mathrm{E}_{Q_x}\left[\mathrm{E}_{P_{y|x}}\left[H_{\log p(y|x,\theta)}\right]\right] \, .$$

In Amari (1998), this version of F is computed explicitly for a basic perceptron model (basically a neural network with 0 hidden layers) in the case where $Q_x = N(0, I)$. However, in practice the real $q(x)$ may not be directly available, or it may be difficult to integrate $H_{\log p(y|x,\theta)}$ over Q_x. For example, the conditional Hessian $H_{\log p(y|x,\theta)}$ corresponding to a multi-layer neural network may be far too complicated to be analytically integrated, even for a very simple Q_x. In such situations Q_x may be replaced with its empirical version $\hat{Q}_x$, giving

$$F = \frac{1}{|S|} \sum_{x \in S_x} \mathrm{E}_{P_{y|x}} \left[\nabla \log p(y|x,\theta) \nabla \log p(y|x,\theta)^\top \right] \quad \text{or} \quad F = -\frac{1}{|S|} \sum_{x \in S_x} \mathrm{E}_{P_{y|x}} \left[H_{\log p(y|x,\theta)} \right] .$$

This is the version of F considered in Park et al. (2000).

From these expressions we can see that when $L(y,z) = -\log r(y|z)$ (as in Section 4), the Fisher has the interpretation of being the expectation under $P_{x,y}$ of the Hessian of $L(y, f(x,\theta))$:

$$F = \frac{1}{|S|} \sum_{x \in S_x} \mathrm{E}_{P_{y|x}} \left[H_{L(y,f(x,\theta))} \right] .$$

Meanwhile, the Hessian H of h is also given by the expected value of the Hessian of $L(y, f(x,\theta))$, except under the distribution $\hat{Q}_{x,y}$ instead of $P_{x,y}$ (where $\hat{Q}_{x,y}$ is given by the density function $\hat{q}(x,y) = \hat{q}(y|x)\hat{q}(x)$). In other words

$$H = \frac{1}{|S|} \sum_{x \in S_x} \mathrm{E}_{\hat{Q}_{x,y}} \left[H_{L(y,f(x,\theta))} \right] .$$

Thus F and H can be seen as approximations of each other in some sense.

6. Geometric Interpretation

The negative gradient $-\nabla h$ can be interpreted as the steepest descent direction for h in the sense that it yields the greatest instantaneous rate of reduction in h per unit of change in θ, where change in θ is measured using the standard Euclidean norm $\|\cdot\|$. More formally we have

$$\frac{-\nabla h}{\|\nabla h\|} = \lim_{\epsilon \to 0} \frac{1}{\epsilon} \operatorname*{arg\,min}_{d : \|d\| \leq \epsilon} h(\theta + d) .$$

This interpretation highlights the strong dependence of the gradient on the Euclidean geometry of the parameter space (as defined by the norm $\|\cdot\|$).

One way to motivate the natural gradient is to show that it (or more precisely its negation) can be viewed as a steepest descent direction, much like the negative gradient can be, except with respect to a metric that is intrinsic to the distributions being modeled, as opposed to the default Euclidean metric which is tied to the given parameterization. In particular, the natural gradient can be derived by adapting the steepest descent formulation to use an alternative definition of (local) distance based on the "information geometry" (Amari and Nagaoka, 2000) of the space of probability distributions. The particular distance function[7] which gives rise to the natural gradient turns out to be

$$\mathrm{KL}(P_{x,y}(\theta + d) \| P_{x,y}(\theta)) .$$

---

[7] Note that this is not a formal "distance" function in the usual sense since it is not symmetric.

To formalize this, one can use the well-known connection between the KL divergence and the Fisher, given by the Taylor series approximation

$$\mathrm{KL}(P_{x,y}(\theta+d) \| P_{x,y}(\theta)) = \frac{1}{2} d^\top F d + O(d^3)\,,$$

where "$O(d^3)$" is short-hand to mean terms that are order 3 or higher in the entries of d. Thus, F defines the local quadratic approximation of this distance, and so gives the mechanism of *local* translation between the geometry of the space of distributions, and that of the original parameter space with its default Euclidean geometry.

To make use of this connection, Arnold et al. (2011) proves for general PSD matrices A that

$$\frac{-A^{-1}\nabla h}{\|\nabla h\|_{A^{-1}}} = \lim_{\epsilon \to 0} \frac{1}{\epsilon} \operatorname*{arg\,min}_{d : \|d\|_A \leq \epsilon} h(\theta + d)\,,$$

where the notation $\|v\|_B$ is defined by $\|v\|_B = \sqrt{v^\top B v}$. Taking $A = \frac{1}{2}F$ and using the above Taylor series approximation to establish that

$$\mathrm{KL}(P_{x,y}(\theta+d) || P_{x,y}(\theta)) \to \frac{1}{2} d^\top F d = \frac{1}{2}\|d\|_F^2$$

as $\epsilon \to 0$, (Arnold et al., 2011) then proceed to show that

$$-\sqrt{2}\frac{\tilde{\nabla} h}{\|\nabla h\|_{F^{-1}}} = \lim_{\epsilon \to 0} \frac{1}{\epsilon} \operatorname*{arg\,min}_{d\,:\, \mathrm{KL}(P_{x,y}(\theta+d) \| P_{x,y}(\theta)) \leq \epsilon^2} h(\theta + d)\,,$$

(where we recall the notation $\tilde{\nabla} h = F^{-1}\nabla h$).

Thus the negative natural gradient is indeed the steepest descent direction in the space of distributions where distance is measured in small local neighborhoods by the KL divergence.

Note that both F and $\tilde{\nabla} h$ are defined in terms of the standard basis in θ-space, and so obviously depend on the parameterization of h. But the KL divergence does not, and instead only depends on the form of the predictive distribution $P_{y|x}$. Thus, the direction in distribution space defined implicitly by $\tilde{\nabla} h$ will be invariant to our choice of parameterization (whereas the direction defined by ∇h will not be, in general).

By using the smoothly varying PSD matrix F to locally define a metric tensor at every point in parameter space, a Riemannian manifold can be generated over the space of distributions. Note that the associated metric of this space won't be the square root of the KL divergence (this isn't even a valid metric), although it will be "locally equivalent" to it in the sense that the two functions will approximate each other within a small local neighborhood.

7. 2nd-order Optimization

The basic idea in 2nd-order optimization is to compute the update δ to $\theta \in \mathbb{R}^n$ by minimizing some local quadratic approximation or "model" $M_k(\delta)$ of $h(\theta_k + \delta)$ centered around the of $M_k(\delta)$. Examples include Tikhonov regularization/damping and the closely related trust-region method (e.g. Tikhonov, 1943; Moré and Sorensen, 1983; Conn et al., 2000; Nocedal and Wright, 2006), and other ones such as the "structural damping" approach of Martens and Sutskever (2011), or the approach present in Krylov Subspace Descent (Vinyals and Povey, 2012). See Martens and Sutskever (2012) for an in-depth discussion of these and other damping techniques in the context of neural network optimization.

This idea is supported by practical experience in neural network optimization. For example, the Hessian-free optimization approach of Martens (2010) generates its updates using a Tikhonov damping scheme applied to the exact GGN matrix (which was equivalent to the Fisher in that work). These updates, which can be applied with a step-size of 1, make a lot more progress optimizing the objective than updates computed without any damping (which must instead rely on a carefully chosen step-size to even be feasible).

It is worth pointing out that other interpretations of natural gradient descent can also motivate the use of damping/regularization terms. In particular, Ollivier et al. (2018) has shown that online natural gradient descent, with a particular flavor of Tikhonov regularization, closely resembles a certain type of extended Kalman filter-based training algorithm for neural networks (Singhal and Wu, 1989; Ruck et al., 1992), where θ is treated as an evolving hidden state that is estimated by the filter (using training targets as noisy observations and inputs as control signals).

11. The Empirical Fisher

An approximation of the Fisher known as the "empirical Fisher" (Schraudolph, 2002), which we denote by $\bar{F}$, is commonly used in practical natural gradient methods. It is obtained by taking the inner expectation of eqn. 3 over the target distribution $Q_{x,y}$ (or its empirical surrogate $\hat{Q}_{x,y}$) instead of the model's distribution $P_{x,y}$.

In the case where one uses $\hat{Q}_{x,y}$, this yields the following simple form:

$$\begin{aligned}\bar{F} &= \mathrm{E}_{\hat{Q}_{x,y}}\left[\nabla \log p(x,y|\theta) \nabla \log p(x,y|\theta)^\top\right] \\ &= \mathrm{E}_{\hat{Q}_x}\left[\mathrm{E}_{\hat{Q}_{y|x}}\left[\nabla \log p(y|x,\theta) \nabla \log p(y|x,\theta)^\top\right]\right] \\ &= \frac{1}{|S|}\sum_{(x,y)\in S} \nabla \log p(y|x,\theta) \nabla \log p(y|x,\theta)^\top .\end{aligned}$$

This matrix is often incorrectly referred to as the Fisher, or even the Gauss-Newton, even though it is not equivalent to either of these matrices in general.

11.1 Comparisons to the Standard Fisher

Like the Fisher F, the empirical Fisher $\bar{F}$ is PSD. But unlike F, it is essentially free to compute, provided that one is already computing the gradient of h. And it can also be applied to objective functions which might not involve a probabilistic model in any obvious way.

Compared to F, which is of rank $\leq |S| \operatorname{rank}(F_R)$, $\bar{F}$ has a rank of $\leq |S|$, which can make it easier to work with in practice. For example, the problem of computing the diagonal (or various blocks) is easier for the empirical Fisher than it is for higher rank matrices like the standard Fisher (Martens et al., 2012). This has motivated its use in optimization methods such as TONGA (Le Roux et al., 2008), and as the diagonal preconditioner of choice in the Hessian-free optimization method (Martens, 2010). Interestingly however, there are stochastic estimation methods (Chapelle and Erhan, 2011; Martens et al., 2012) which can be used to efficiently estimate the diagonal (or various blocks) of the standard Fisher F, and these work quite well in practice. (These include the obvious method of sampling y's from the model's conditional distribution and computing gradients from them, but also includes methods based on matrix factorization and random signs. See Martens et al. (2012) for comparative analysis of the variance of these methods.)

Despite the various practical advantages of using $\bar{F}$, there are good reasons to use true Fisher F instead of $\bar{F}$ whenever possible. In addition to Amari's extensive theory developed for the exact natural gradient (which uses F), perhaps the best reason for using F over $\bar{F}$ is that F turns out to be a reasonable approximation/substitute to the Hessian H of h in certain important special cases, which is a property that $\bar{F}$ lacks in general.

For example, as discussed in Section 5, when the loss is given by $-\log p(y|x)$ (as in Section 4), F can be seen as an approximation of H, because both matrices have the interpretation of being the expected Hessian of the loss under some distribution. Due to the similarity of the expression for F in eqn. 3 and the one above for $\bar{F}$, it might be tempting to think that $\bar{F}$ is given by the expected Hessian of the loss under $\hat{Q}_{x,y}$ (which is actually the formula for H) in the same way that F is given by eqn. 4. But this is not the case in general.

And as we saw in Section 9, given certain assumptions about how the GGN is computed, and some additional assumptions about the form of the loss function L, F turns out to be equivalent to the GGN. This is very useful since the GGN can be used to define a local quadratic approximation of h, whereas F normally doesn't have such an interpretation. Moreover, Schraudolph (2002) and later Martens (2010) compared $\bar{F}$ to the GGN and observed that the latter performed much better as a curvature matrix within various neural network optimization methods.

As concrete evidence for why the empirical Fisher is, at best, a questionable choice for the curvature matrix, we will consider the following example. Set $n = 1$, $f(x, \theta) = \theta$, $R_{y|z} = \mathcal{N}(z, 1)$, and $S = \{(0, 0)\}$, so that $h(\theta)$ is a simple convex quadratic function of θ, given by $h(\theta) = \frac{1}{2}\theta^2$. In this example we have that $\nabla h = \theta$, $\bar{F} = \theta^2$, while $F = 1$. If we use $\bar{F}^\xi$ as our curvature matrix for some exponent $\frac{1}{2} \leq \xi \leq 1$, then it is easy to see that an iteration of the form

$$\theta_{k+1} = \theta_k - \alpha_k (\bar{F}(\theta_k)^\xi)^{-1} \nabla h(\theta_k) = \theta_k - \alpha_k (\theta_k^2)^{-\xi} \theta_k = (1 - \alpha_k |\theta_k|^{-2\xi}) \theta_k$$

will fail to converge to the minimizer (at $\theta = 0$) unless $\xi < 1$ and the step-size α_k goes to 0 sufficiently fast. And even when it does converge, it will only be at a rate comparable to the speed at which α_k goes to 0, which in typical situations will be either $\mathcal{O}(1/k)$ or $\mathcal{O}(1/\sqrt{k})$. Meanwhile, a similar iteration of the form

$$\theta_{k+1} = \theta_k - \alpha_k F^{-1} \nabla h(\theta_k) = \theta_k - \alpha_k \theta_k = (1 - \alpha_k) \theta_k \, ,$$

which uses the exact Fisher F as the curvature matrix, will experience very fast linear convergence[13] with rate $|1 - \alpha|$, for any fixed step-size $\alpha_k = \alpha$ satisfying $0 < \alpha < 2$.

---

[13] Here we mean "linear" in the classical sense that $|\theta_k - 0| \leq |\theta_0 - 0| |1 - \alpha|^k$.

It is important to note that this example uses a noise-free version of the gradient, and that this kind of linear convergence is (provably) impossible in most realistic stochastic/online settings. Nevertheless, we would argue that a highly desirable property of any stochastic optimization method should be that it can, in principle, revert to an optimal (or nearly optimal) behavior in the deterministic setting. This might matter a lot in practice, since the gradient may end up being sufficiently well estimated in earlier stages of optimization from only a small amount of data (which is a common occurrence in our experience), or in later stages provided that larger mini-batches or other variance-reducing procedures are employed (e.g. Le Roux et al., 2012; Johnson and Zhang, 2013). More concretely, the pre-asymptotic convergence rate of stochastic 2nd-order optimizers can still depend strongly on the choice of the curvature matrix, as we will show in Section 14.

11.2 A Discussion of Recent Diagonal Methods Based on the Empirical Fisher

Recently, a spate of stochastic optimization methods have been proposed that are all based on diagonal approximations of the empirical Fisher $\bar{F}$. These include the diagonal version of AdaGrad (Duchi et al., 2011), RMSProp (Tieleman and Hinton, 2012), Adam (Ba and Kingma, 2015), etc. Such methods use iterations of the following form (possibly with some slight modifications):

$$\theta_{k+1} = \theta_k - \alpha_k (B_k + \lambda I)^{-\xi} g_k(\theta_k) \, , \tag{10}$$

where the curvature matrix B_k is taken to be a diagonal matrix $\mathrm{diag}(u_k)$ with u_k adapted to maintain some kind of estimate of the diagonal of $\bar{F}$ (possibly using information from previous iterates/mini-batches), $g_k(\theta_k)$ is an estimate of $\nabla h(\theta_k)$ produced from the current mini-batch, α_{kk} is a schedule of step-sizes, and $0 < \lambda$ and $0 < \xi \leq 1$ are hyperparameters (discussed later in this section).

There are also slightly more sophisticated methods (Schaul et al., 2013; Zeiler, 2013) which use preconditioners that combine the diagonal of $\bar{F}$ with other quantities (such as an approximation of the diagonal of the Gauss-Newton/Fisher in the case of Schaul et al. (2013)) in order to correct for how the empirical Fisher doesn't have the right "scale" (which is ultimately the reason why it does poorly in the example given at the end of Section 11.1).

A diagonal preconditioner (Nash, 1985) of the form used in eqn. 10 was also used by (Martens, 2010) to accelerate the conjugate gradient (CG) sub-optimizations performed within a truncated-Newton method (using the GGN matrix). In the context of CG, the improper scale of $\bar{F}$ is not as serious an issue due to the fact that CG is invariant to the overall scale of its preconditioner (since it computes an optimal "step-size" at each step which automatically adjusts for the scale). However, it still makes more sense to use the diagonal of the true Fisher F as a preconditioner, and thanks to the method proposed by Chapelle and Erhan (2011), this can be estimated efficiently and accurately.

The idea of using the diagonal of F, $\bar{F}$, or the Gauss-Newton as a preconditioner for stochastic gradient descent (SGD) and was likely first applied to neural networks with the work of Lecun and collaborators (Becker and LeCun, 1989; LeCun et al., 1998), who proposed an iteration of the form in eqn. 10 with $\xi = 1$ where u_k approximates the diagonal of the Hessian or the Gauss-Newton matrix (which as shown in Section 9, is actually equivalent to F for the common squared-error loss). Following this work, various neural network

arXiv:2006.00719v3 [cs.LG] 29 Apr 2021

AdaHessian: An Adaptive Second Order Optimizer for Machine Learning

Zhewei Yao[*,1], **Amir Gholami**[*,1], **Sheng Shen**[1], **Mustafa Mustafa**[2], **Kurt Keutzer**[1],
Michael W. Mahoney[1]
[1]**University of California, Berkeley,** [2]**NERSC, Lawrence Berkeley National Laboratory**
{zheweiy, amirgh, sheng.s, keutzer, mahoneymw}@berkeley.edu, mmustafa@lbl.gov

Abstract

We introduce AdaHessian, a second order stochastic optimization algorithm which dynamically incorporates the curvature of the loss function via ADAptive estimates of the Hessian. Second order algorithms are among the most powerful optimization algorithms with superior convergence properties as compared to first order methods such as SGD and Adam. The main disadvantage of traditional second order methods is their heavier per-iteration computation and poor accuracy as compared to first order methods. To address these, we incorporate several novel approaches in AdaHessian, including: (i) a fast Hutchinson based method to approximate the curvature matrix with low computational overhead; (ii) a root-mean-square exponential moving average to smooth out variations of the Hessian diagonal across different iterations; and (iii) a block diagonal averaging to reduce the variance of Hessian diagonal elements. We show that AdaHessian achieves new state-of-the-art results by a large margin as compared to other adaptive optimization methods, including variants of Adam. In particular, we perform extensive tests on CV, NLP, and recommendation system tasks and find that AdaHessian: (i) achieves 1.80%/1.45% higher accuracy on ResNets20/32 on Cifar10, and 5.55% higher accuracy on ImageNet as compared to Adam; (ii) outperforms AdamW for transformers by 0.13/0.33 BLEU score on IWSLT14/WMT14 and 2.7/1.0 PPL on PTB/Wikitext-103; (iii) outperforms AdamW for SqueezeBert by 0.41 points on GLUE; and (iv) achieves 0.032% better score than Adagrad for DLRM on the Criteo Ad Kaggle dataset. Importantly, we show that the cost per iteration of AdaHessian is comparable to first-order methods, and that it exhibits robustness towards its hyperparameters. The code for AdaHessian is open-sourced and publicly-available [1].

Introduction

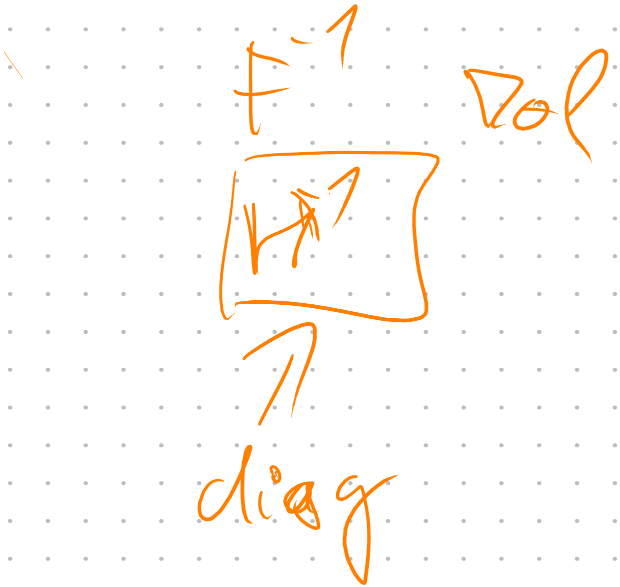
The high dimensional and non-convex nature of many machine learning tasks has rendered many classical optimization methods inefficient for training and/or evaluating Neural Network (NN) models. After decades of research, first order methods, and in particular variants of Stochastic Gradient Descent (SGD), have become the main workhorse for training NN models. However, they are by no means an ideal solution for training NN models. There are often a lot of ad-hoc rules that need to be followed very precisely to converge (hopefully) to a point with good generalization properties. Even the choice of the first order optimizer has become an ad-hoc rule which can significantly affect the performance. For example, SGD with momentum is typically used in Computer Vision (CV); Adam is used for training transformer models for Natural Language Processing (NLP); and Adagrad is used for Recommendation Systems (RecSys). Using the wrong SGD variant can lead to significant performance degradation. Another challenging ad-hoc rule is the choice of hyperparameters and hyperparameter tuning methods, even after an optimizer is chosen. Hyperparameters include learning rate, decay schedule, choice of momentum parameters, number of warmup iterations, etc. As a result of these and other issues, one has to *babysit* the optimizer to make sure that training converges to an *acceptable* training loss, without any guarantee that a given number of iterations is enough to reach a local minima.

Importantly, one may *not* observe the above problems for certain popular learning tasks, such as ResNet50 training on ImageNet. The reason is that, for these tasks, years of industrial scale hyperparameter tuning has lead to what may be called *ideal SGD behaviour*. That is, for this problem, hyperparameters have been brute-force engineered to compensate for the deficiencies of first order methods. Such a brute force approach is computationally and financially not possible for many large scale learning problems—certainly it is not possible to do routinely—and this has made it challenging to train and apply NN models reliably.

Many of these issues stem from the fact that first order methods only use gradient information and do not consider the curvature properties of the loss landscape, thereby leading to their suboptimal behaviour. Second order methods, on the other hand, are specifically designed to capture and exploit the curvature of the loss landscape and to incorporate both gradient and Hessian information. They are among the most powerful optimization algorithms, and they have many favorable

---

[*] Equal contribution.


F^{-1}
Pol
diag

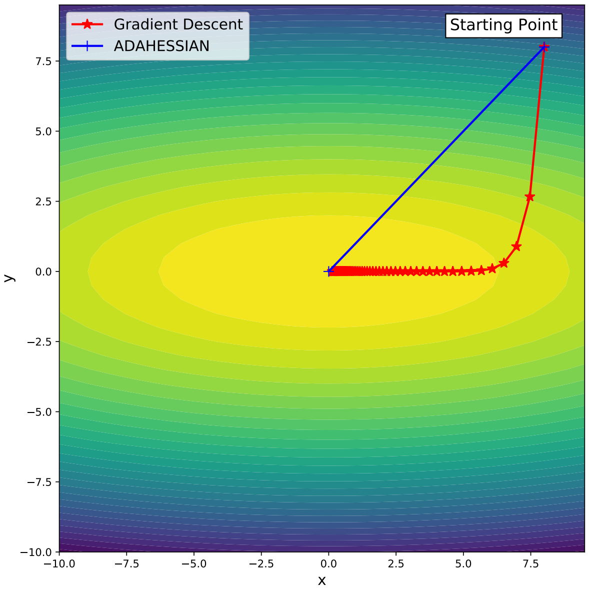


Figure 1: *The trajectory of gradient descent and* AdaHessian *on a simple 2D quadratic function $f(x, y) = 10x^2 + y^2$. Gradient descent converges very slowly, even though this problem has a reasonable condition number. However,* AdaHessian *converges to the optimum in just one step. This is because second order methods normalize the curvature difference between x and y axis by preconditioning the gradient vector before the weight update (by rescaling and rotating the gradient vector).*

number of data points in the training dataset. Furthermore, we denote the gradient of the loss w.r.t. model parameters as $\mathbf{g} = \frac{1}{N_B}\sum_{i=1}^{N_B}\frac{\partial l_i}{\partial \theta}$, and the corresponding second derivative (i.e., Hessian) as $\mathbf{H} = \frac{1}{N_B}\sum_{i=1}^{N_B}\frac{\partial^2 l_i}{\partial \theta^2}$, where N_B is the size of one mini-batch.

Solving Eq. 1 for a real learning problem (and not a simple model) is a very challenging task. Despite years of research, we have not yet been able to resolve several seemingly ad-hoc tricks that are required to converge (hopefully) to a *good* solution. Next, we briefly discuss the different popular optimization methods proposed in recent years to address the challenges associated with solving Eq. 1. This is by no means a comprehensive review, and we refer the interested reader to [8] for a thorough review.

Adaptive First Order Methods

Due to their simplicity and effectiveness, first order optimization methods [20, 29, 34, 41, 49, 71] have become the de-facto algorithms used in deep learning. There are multiple variations, but these methods can be represented using the following general update formula:

$$\theta_{t+1} = \theta_t - \eta_t m_t / v_t, \tag{2}$$

where η_t is the learning rate, and m_t, and v_t denote the so called first and second moment terms, respectively. A simple and popular update method is SGD, originally proposed in 1951 as a root-solving algorithm [49]:

$$m_t = \beta m_{t-1} + (1-\beta)\mathbf{g}_t \quad \text{and} \quad v_t \equiv 1. \tag{3}$$

Here, $\mathbf{g}_t$ is the gradient of a mini-batch at t-th iteration and β is the momentum hyperparameter.

Using SGD to solve Eq. 1 is often very challenging, as the convergence of the iterative formulae in Eq. 2 is very sensitive to the right choice of the learning rate, its decay schedule, and the momentum parameter. To address this, several methods have been proposed to take into account the knowledge of the geometry of the data by scaling gradient coordinates, using the past gradient information. This can be viewed in one of two equivalent ways: either as automatically adjusting the learning rate in Eq. 2; or as an adaptive *preconditioning* of the gradient. One notable method is Adagrad [20, 37], which accumulates all the gradients from the first iteration and applies the square root of the result to precondition the current gradient. The update formulae in this case become[1]:

$$m_t = \mathbf{g}_t \qquad \text{and} \qquad v_t = \sqrt{\sum_{i=1}^{t} \mathbf{g}_i \mathbf{g}_i}. \tag{4}$$

While Adagrad works well for sparse settings, its performance significantly degrades for dense settings, which is the case for many machine learning tasks. In particular, this stems from the accumulation of all previous gradients for the preconditioner Eq. 4. This results in a monotonic increase in the magnitude of the second moment, v_t, which effectively translates into a rapid decay of the learning rate. To address this, several methods have been proposed where the intuition is to limit the accumulation to a small window of past iterations, and in particular exponentially reduce the weight of earlier iterations. Notable works incorporating this method are RMSProp, ADADelta, and Adam [29, 56, 71]. In particular, for Adam [29], the two moments for the update rule are the following:

$$\begin{aligned} m_t &= \frac{(1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}\mathbf{g}_i}{1-\beta_1^t}, \\ v_t &= \sqrt{\frac{(1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i}\mathbf{g}_i\mathbf{g}_i}{1-\beta_2^t}}, \end{aligned} \tag{5}$$

where $0 < \beta_1,\ \beta_2 < 1$ are two hyperparameters sometimes referred to as first and second moment coefficients. In particular, note that the sum over past gradients is scaled by β_2 which exponentially reduces the contribution of early gradients. A summary of the different m_t and v_t used by common first-order optimizers is given in Table 1. A notable variant here is AdamW [34], which shows that decoupling weight decay

---

[1] Throughout the paper, without further notification, for two vectors, e.g., a and b, we use both ab and $a \odot b$ to denote the element-wise product, and $\langle a, b\rangle$ denotes the inner product.

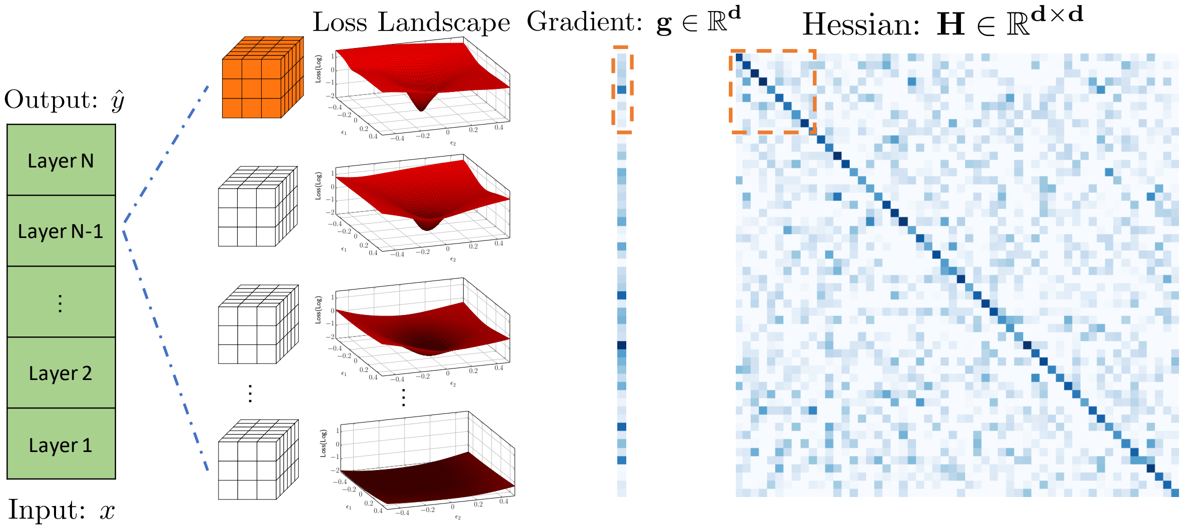

Loss Landscape
Gradient: $\mathbf{g} \in \mathbb{R}^{\mathbf{d}}$
Hessian: $\mathbf{H} \in \mathbb{R}^{\mathbf{d}\times\mathbf{d}}$
Output: $\hat{y}$
Layer N
Layer N-1
⋮
Layer 2
Layer 1
Input: x
Loss(Log)
ϵ_1
ϵ_2

Figure 2: *A simple model with N layers (first column); with the convolutional blocks of the N-1 layer shown (second column); and the loss landscape of each block (third column), which can be calculated by perturbing the convolutions's parameters in two different eigendirections. (See [66] for details of how to construct loss landscape.) Note the different loss landscape topologies. First order methods do not explicitly capture this difference. The entries (3D tensors) colored in orange show the components used for calculating the spatial average of Hessian. The part of the gradient (fourth panel) highlighted in the orange box is the corresponding gradient of the orange convolution kernel; and the part of the Hessian diagonal (fifth panel) highlighted in the orange box is used to compute the spatial average.*

since we can efficiently incorporate moving averages and momentum. Ideally, if there was a way to apply the same moving average method to the Hessian, then that would help smooth out local curvature noise to get a better approximation to the non-noisy curvature of the loss landscape. However, such an approximation is challenging since the Hessian is a matrix that cannot be explicitly formed to be averaged, whereas it is easy to form the gradient vector.

As we show below, AdaHessian addresses this problem by incorporating the Hutchinson's method along with spatial averaging to reduce the impact of the stochastic noise. The result exceeds the performance of all the above methods for machine learning tasks. Next, we formally introduce the AdaHessian algorithm.

Methodological Approach

Here, we first provide the formulation for the full Newton method in Section . Then, we describe the three components of AdaHessian, namely Hessian diagonal approximation (Section), spatial averaging (Section), and Hessian momentum (Section). Finally, we discuss the overall formulation of AdaHessian in Section .

A General Hessian Based Descent Direction

For the loss function $f(w) : \mathbb{R}^d \to \mathbb{R}$, let us denote the corresponding gradient and Hessian of $f(w_t)$ at iteration t as $\mathbf{g}_t$, and $\mathbf{H}_t$, respectively.[3] A general descent direction can then be written as follows for a positive-definite Hessian:

$$\Delta w_t = \mathbf{H}_t^{-k}\mathbf{g}_t, \quad \text{where} \quad \mathbf{H}_t^{-k} = U_t^T \Lambda_t^{-k} U_t. \tag{6}$$

Here, we refer to $0 \le k \le 1$ as *Hessian power*, and $U_t^T \Lambda_t U_t$ is the eigendecomposition of $\mathbf{H}_t$. Note that for $k = 0$, we recover the gradient descent method; and for $k = 1$, we recover the Newton method. In our empirical tests we consider non-convex machine learning problems, but we provide a standard convergence behaviour of Eq. 6 in Appendix for a simple strongly convex and strictly smooth function $f(w)$. (We emphasize that the proof is very standard and we are only including it for completeness.)

The basic idea of Hessian based methods is to *precondition* the gradient with the $\mathbf{H}^{-k}$ and use $\mathbf{H}^{-k}\mathbf{g}$ for the update direction, instead of using the *bare* gradient $\mathbf{g}$ vector. The preconditioner automatically rotates and rescales the gradient vector. This is important since the loss landscape curvature is generally different across different directions/layers and since these directions need not correspond to the canonical axes. This is illustrated in Figure 2, where we show a 2D schematic plot of the loss landscape for different convolution channels [66]. Each channel can have a different loss landscape topology. For example, the last channel has a much flatter loss landscape, as compared to other layers. As a result, it is preferable to take a larger step size for the last channel than for the first channel, which has a very "sharp" loss landscape. Problems that exhibit this behaviour are *ill-conditioned.* The role of the Hessian is to automatically normalize this ill-conditionedness by stretching and

---

[3] Without confusion, we use the same gradient and Hessian notations for $f(w)$ and $\mathcal{L}(\theta)$. Furthermore, when there is no confusion we will drop subscript t.

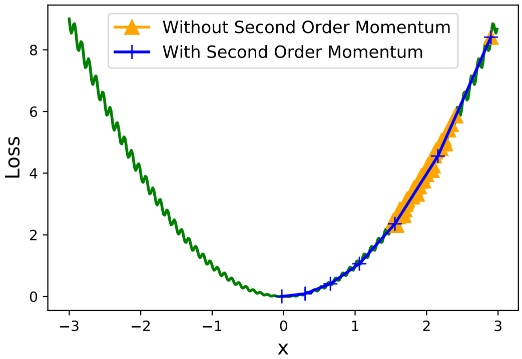


Figure 3: *Local versus global curvature. Illustration of the local curvature which can be noisy, and the global curvature with a simple 1D problem* $f(x) = x^2 + 0.1x\sin(20\pi x)$*. Using the exponential moving average of Eq. 12 is key to avoid the misleading local curvature information. To demonstrate this we test* AdaHessian *without moving average (orange trajectory) which does not converge even after 1000 iterations. On the other hand,* AdaHessian *converges in 7 iterations with the moving average enabled.*

contracting different directions to accommodate for the curvature differences (full Newton method also rotates the gradient vector along with adjusting the step size).

However, there are two major problems with this approach. The first problem is that a naïve use of the Hessian preconditioner comes at the prohibitively high cost of applying Hessian inverse to the gradient vector at every iteration ($\mathbf{H}^{-k}\mathbf{g}$ term). The second and more challenging problem is that local Hessian (curvature) information can be very misleading for a noisy loss landscape. A simple example is illustrated in Figure 3, where we plot a simple parabola with a small sinusoidal noise as the loss landscape (shown in green). As one can see, the local Hessian (curvature) information is completely misleading, as it computes the curvature of the sinusoidal noise instead of global Hessian information for the parabola. Applying such misleading information as the preconditioner would actually result in very small steps to converge to one of the many local minima created by the sinusoidal noise. The same problem exists for the gradient as well, but that can be alleviated by using gradient momentum instead of local gradient information. However, as mentioned before it is computationally infeasible to compute (naïvely) a Hessian momentum. The reason is that we cannot form the Hessian matrix and average it throughout different iterations, as such an approach has quadratic memory complexity in the number of parameters along with a prohibitive computational cost. However, one could use Randomized Numerical Linear Algebra to get a sketch of the Hessian matrix [22, 66, 67]. In particular, we show how this can be done to approximate the Hessian diagonal. However, as we discuss next, both problems can be resolved by using Hessian diagonal instead of the full Hessian.

Hessian Diagonal Approximation

To address the issue that applying the inverse Hessian to the gradient vector at every iteration is computationally infeasible, one could use an inexact Newton method, where an approximate Hessian operator is used instead of the full Hessian [6, 16, 63, 64, 69]. The most simple and computationally efficient approach is to approximate the Hessian as a diagonal operator in Eq. 6:

$$\Delta w = diag(\mathbf{H})^{-k}\mathbf{g}, \tag{7}$$

where $diag(\mathbf{H})$ is the Hessian diagonal, which we denote as $\boldsymbol{D}$.[4] We show that using Eq. 7 has the same convergence rate as using Eq. 6 for simple strongly convex and strictly smooth function $f(w)$ (see Appendix). Note that we only include the proof for completeness, and our algorithm AdaHessian can be applied for general machine learning problems.

The Hessian diagonal $\boldsymbol{D}$ can be efficiently computed using the Hutchinson's method. The two techniques we use for this approximation are: (i) a Hessian-free method [67]; and (ii) a randomized numerical linear algebra (RandNLA) method [5, Figure 1]. In particular, the Hessian-free method is an oracle to compute the multiplication between the Hessian matrix $\mathbf{H}$ with a random vector z, i.e.,

$$\frac{\partial \mathbf{g}^T z}{\partial \theta} = \frac{\partial \mathbf{g}^T}{\partial \theta} z + \mathbf{g}^T \frac{\partial z}{\partial \theta} = \frac{\partial \mathbf{g}^T}{\partial \theta} z = \mathbf{H}z. \tag{8}$$

Here, the first equality is the chain rule, and the second equality is since z is independent of θ. Eq. 8 effectively allows us to compute the Hessian times a vector z, without having to form explicitly the Hessian, by backpropotating the $\mathbf{g}^T z$ term. This has the same cost as ordinary gradient backpropogation [67]. Then, with the Hessian matvec oracle, one can compute the Hessian diagonal using Hutchinson's method:

$$\boldsymbol{D} = diag(\mathbf{H}) = \mathbb{E}[z \odot (\mathbf{H}z)], \tag{9}$$

where z is a random vector with Rademacher distribution, and $\mathbf{H}z$ is computed by the Hessian matvec oracle given in Eq. 8. This process is illustrated in Figure 4. It can be proved that the expectation of $z \odot (\mathbf{H}z)$ is the Hessian diagonal [5].

Another important advantage, besides computational efficiency, of using the Hessian diagonal is that we can compute its moving average to resolve the local noisy Hessian as mentioned at the end of Section . This allows us to smooth out noisy local curvature information, and to obtain estimates that use global Hessian information instead. We incorporate both spatial averaging and momentum (temporal averaging) to smooth out this noisy Hessian estimate as described next.

Spatial Averaging

The Hessian diagonal can vary significantly for each single parameter dimension of the problem. We found it

---

[4] Note that $\boldsymbol{D}$ can be viewed as a vector, in which case $\boldsymbol{D}^{-k}\mathbf{g}$ is an element-wise product of vectors. Without clarification, $\boldsymbol{D}$ is treated as a vector for the rest of the paper.

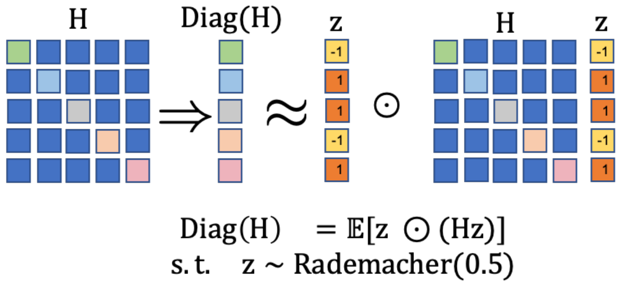


Figure 4: *Illustration of the diagonal Hessian estimation with Hutchinson's method.*

helpful to perform spatial averaging of Hessian diagonal and use the average to smooth out spatial variations. For example, for a convolutional layer, each convolution parameter can have a very different Hessian diagonal. In ADAHESSIAN we compute the average of the Hessian diagonal for each convolution kernel (3×3) as illustrated in Figure 5. Mathematically, we perform a simple spatial averaging on the Hessian diagonal as follows:

$$\boldsymbol{D}^{(s)}[ib+j] = \frac{\sum_{k=1}^{b} \boldsymbol{D}[ib+k]}{b}, \text{for } 1 \le j \le b, 0 \le i \le \frac{d}{b}-1, \tag{10}$$

where $\boldsymbol{D} \in \mathbb{R}^d$ is the Hessian diagonal, $\boldsymbol{D}^{(s)} \in \mathbb{R}^d$ is the spatially averaged Hessian diagonal, $\boldsymbol{D}[i]$ ($\boldsymbol{D}^{(s)}[i]$) refers to the i-th element of $\boldsymbol{D}$ ($\boldsymbol{D}^{(s)}$), b is the spatial average block size, and d is the number of model parameters divisible by b. We show that replacing $\boldsymbol{D}$ in Eq. 7 by $\boldsymbol{D}^{(s)}$ in Eq. 10, the update direction has the same convergence rate as using Eq. 6 for simple strongly convex and strictly smooth function $f(w)$ (see Appendix). Note that we only include the proof for completeness, and our algorithm ADAHESSIAN can be applied for general machine learning problems.

Figure 5 provides illustration of spatial averaging for both convolutional and matrix kernels. In general, the block size b is a hyperparameter that can be tuned for different tasks. While this is a new hyperparameter that can help the performance, the performance of ADAHESSIAN is not very sensitive to it (we provide sensitivity results in Section).

Next we describe momentum which is another useful method to smooth out Hessian noise over different iterations.

Hessian Momentum

We can easily apply momentum to Hessian diagonal since it is a vector instead of a quadratically large matrix. This enables us to adopt momentum for Hessian diagonal in ADAHESSIAN. More specifically, let $\bar{\boldsymbol{D}}_t$ denote the Hessian diagonal with momentum that is calculated as:

$$\bar{\boldsymbol{D}}_t = \sqrt{\frac{(1-\beta_2)\sum_{i=1}^{t} \beta_2^{t-i} \boldsymbol{D}_i^{(s)} \boldsymbol{D}_i^{(s)}}{1-\beta_2^t}}, \tag{11}$$

where $\boldsymbol{D}^{(s)}$ is the spatially averaged Hessian diagonal (defined in Eq. 10), and $0 < \beta_2 < 1$ is the second moment

Algorithm 1: ADAHESSIAN

```
Require: Initial Parameter: $\theta_0$
Require: Learning rate: $\eta$
Require: Exponential decay rates: $\beta_1$, $\beta_2$
Require: Block size: $b$
Require: Hessian Power: $k$
Set: $m_0 = 0$, $v_0 = 0$
for $t = 1, 2, \ldots$ do    // Training Iterations
    $\mathbf{g}_t \leftarrow$ current step gradient
    $\boldsymbol{D}_t \leftarrow$ current step estimated diagonal Hessian
    Compute $\boldsymbol{D}_t^{(s)}$ based on Eq. 10
    Update $\bar{\boldsymbol{D}}_t$ based on Eq. 11
    Update $m_t$, $v_t$ based on Eq. 12
    $\theta_t = \theta_{t-1} - \eta m_t / v_t$
```

hyperparameter. Note that this is exactly the same as the momentum term in Adam [29] or RMSProp [56] except that we are using the spatial averaging Hessian diagonal instead of the gradient.

To illustrate the importance of Hessian momentum, we provide a simple example in 1D by considering $f(x) = x2 + 0.1xsin(20\pi x)$, as shown in Figure 3. It can be clearly seen that the method without the second order momentum gets trapped at a local minima even with more than 1000 iterations (orange trajectory). On the contrary, the optimization converges within 7 iterations with Hessian momentum (blue trajectory). (While this example is over-simplified in certain ways, we are using it here only to convey the importance of momentum.)

AdaHessian

To summarize, instead of only applying momentum for gradient, ADAHESSIAN uses *spatial averaging* and *Hessian momentum* to smooth out local variations in Hessian diagonal. More specifically, the first and second order moments (m_t and v_t) for ADAHESSIAN are computed as follows:

$$m_t = \frac{(1-\beta_1)\sum_{i=1}^{t} \beta_1^{t-i} \mathbf{g}_i}{1-\beta_1^t},$$
$$v_t = (\bar{\boldsymbol{D}}_t)^k = \left(\sqrt{\frac{(1-\beta_2)\sum_{i=1}^{t} \beta_2^{t-i} \boldsymbol{D}_i^{(s)} \boldsymbol{D}_i^{(s)}}{1-\beta_2^t}}\right)^k, \tag{12}$$

where $0 < \beta_1, \beta_2 < 1$ are the first and second moment hyperparameters that are also used in Adam. Note that Adam uses the same formulation except that the spatial averaging Hessian diagonal $\boldsymbol{D}_i^{(s)}$ is replaced with gradient.

The main overhead of ADAHESSIAN is the Hutchinson's method to approximate Hessian diagonal, $\boldsymbol{D}$. We use one Hutchinson step per iteration to approximate the Hessian diagonal (i.e., one random Rademacher vector z in Eq. 9). The cost of this estimation is one Hessian matvec (to compute Hz), which is equivalent to one gradient backpropagation [66, 67].

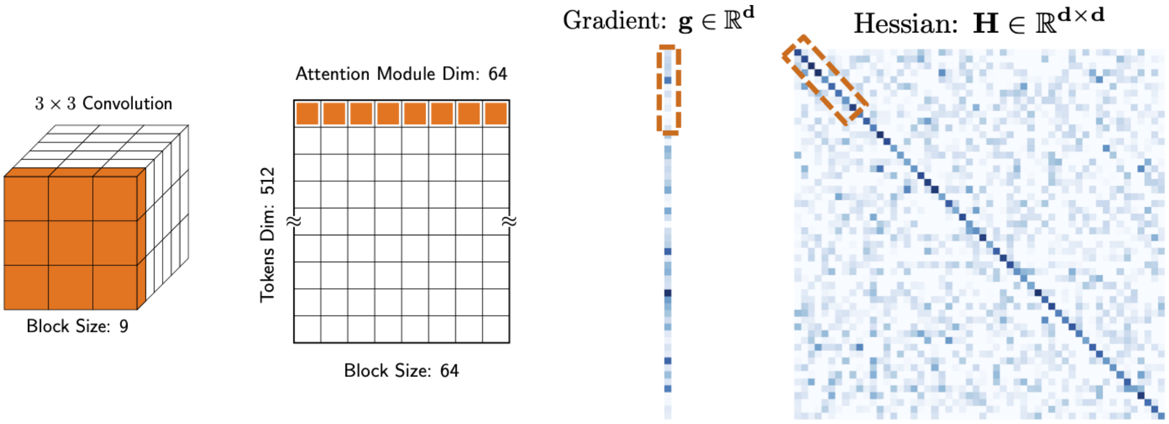

3 × 3 Convolution
Block Size: 9
Attention Module Dim: 64
Tokens Dim: 512
Block Size: 64
Gradient: $\mathbf{g} \in \mathbb{R}^{\mathbf{d}}$
Hessian: $\mathbf{H} \in \mathbb{R}^{\mathbf{d}\times\mathbf{d}}$

Figure 5: *Illustration of the block size used to average the Hessian diagonal to smooth spatial variations. For a convolution layer, we average each channel (groups of 9 parameters); and for multi-head attention, we average consecutive elements along the rows (attention dimension). We found that using block averaging helps, although* AdaHessian *is not very sensitive to this hyperparameter as illustrated in Table 7.*

Also note that it is possible to get a more accurate approximation to Hessian diagonal by using more Hutchinson steps per iteration. However, we found that one step per iteration performs well in practice since the multiple calculations could be performed as Hessian momentum (Section). In fact, as we discuss in Section , it is possible to skip the Hutchinson calculation for few iterations to reduce further its computational overhead, without significant impact on final accuracy.

Results

Experiment Setup

One of the problems with several formerly proposed optimization methods is that the methods were originally tested with very simple models on very few tasks. When those methods were later tested by the community on more complex models the results were often worse than popular optimization methods. To avoid such a scenario, we extensively test AdaHessian on a wide range of learning tasks, including image classification, neural machine translation (NMT), language modeling (LM), and recommendation system (RecSys). We compare the AdaHessian performance with SGD, Adam, AdamW [34], and Adagrad. Moreover, to enable a fair comparison we will use the same β_1 and β_2 parameters in AdaHessian as in Adam/AdamW for each task, even though those default values may favor Adam (or AdamW) and disfavor AdaHessian. Furthermore, we will use the exact same weight decay and learning rate schedule in AdaHessian as that used by other optimizers. Below we briefly explain each of the learning tasks tested.

Table 2: *Results of ResNet20/32 on Cifar10 (left two columns) and ResNet18 on ImageNet (last column). On Cifar10: Adam performs consistently worse than SGD; AdamW has slightly worse performance than SGD; and* AdaHessian *outperforms AdamW and even gets accuracy comparable to SGD. On ImageNet:* AdaHessian *has significantly better accuracy than Adam (5.53%), AdamW (2.67%), and has similar performance to SGD.*

Dataset	**Cifar10**		**ImageNet**
	`ResNet20`	`ResNet32`	`ResNet18`
SGD [51]	92.08 ± 0.08	**93.14 ± 0.10**	70.03
Adam [29]	90.33 ± 0.13	91.63 ± 0.10	64.53
AdamW [34]	91.97 ± 0.15	92.72 ± 0.20	67.41
AdaHessian	**92.13 ± 0.18**	93.08 ± 0.10	**70.08**

Image Classification We experiment on both Cifar10 (using ResNet20/32) and ImageNet (using ResNet18) datasets. Cifar10 consists of 50k training images and 10k testing images. ImageNet has 1.2M training images and 50k validation images. We follow the settings described in [25] for training. We run each experiment 5 times on Cifar10 and report the mean and standard deviation of the results.

Neural Machine Translation (NMT) We use IWSLT14 German-to-English (De-En) and WMT14 English-to-German (En-De) datasets. Transformer `base` architecture is used for WMT14 (4.5M sentence pairs), and `small` architecture is used for IWSLT14 (0.16M sentence pairs). We follow the settings reported in [43] and use pre-normalization described in [59]. The length penalty is set to 0.6/1.0 and the beam size is set to 4/5 for WMT/IWSLT [42]. We report the average results of the last 10/5 checkpoints respectively. For NMT, BLEU score is used [44]. In particular, we report tokenized case-sensitive BLEU on WMT14 En-De and case-insensitive BLEU IWSLT14 De-En. Furthermore, we use AdamW for this task instead of Adam since the former is the standard optimizer (Adam consistently

Table 3: *NMT performance (BLEU) on IWSLT14 De-En and WMT14 En-De testsets (higher is better). Unlike in Table 2, SGD has significantly worse results than AdamW. Note that* AdaHessian *outperforms the default and heavily tuned optimizer AdamW by 0.13 and 0.33 on IWSLT14 and WMT14, which is significant for this task.*

Model	**IWSLT14** `small`	**WMT14** `base`
SGD	28.57 ± .15	26.04
AdamW [34]	35.66 ± .11	28.19
AdaHessian	**35.79 ± .06**	**28.52**

Table 4: *LM performance (PPL) on PTB and Wikitext-103 test datasets (lower is better). The PPL of* AdaHessian *is 2.7 and 1.0 lower than that of AdamW.*

Model	**PTB** `Three-Layer`	**Wikitext-103** `Six-Layer`
SGD	59.9 ± 3.0	78.5
AdamW [34]	54.2 ± 1.6	20.9
AdaHessian	**51.5 ± 1.2**	**19.9**

scores lower).

Language Modeling We use PTB [39] and Wikitext-103 [38] datasets, which contain 0.93M and 100M tokens, respectively. Following [35], a three-layer tensorized transformer core-1 for PTB and a six-layer tensorized transformer core-1 for Wikitext-103 are used in the experiments. We apply the multi-linear attention mechanism with masking and report the perplexity (PPL) on the test set with the best validation model.

Natural Language Understanding We use the GLUE task [58] to evaluate the fine-tuning performance of SqueezeBERT [27]. More specifically, we use 8 different tasks in GLUE and report and final average performance on the validation dataset.

Recommendation System The Criteo Ad Kaggle dataset contains approximately 45 million samples over 7 days. We follow the standard setting and use the first 6 days as the training set and the last day as the test set. Furthermore, we use DLRM, a novel recommendation model that has been recently released by Facebook [40]. The testing metric for Recommendation Systems is Click Through Rate (CTR), measured on training and test sets.

We refer the interested reader to Appendix for more detailed experimental settings. Next we report the experimental results on each of these tasks.

Image Classification

The results on Cifar10 are shown in Table 2. First, note the significantly worse performance of Adam, as compared to SGD even on this simple image classification dataset. Particularly, Adam has 1.75%/1.51% lower accuracy for ResNet20/32 than SGD. AdamW achieves better results than Adam, but its performance is still slightly worse than SGD. However, AdaHessian achieves significantly better results as compared to Adam (1.80%/1.45% for ResNet20/32), even though we use the same β_1 and β_2 parameters in AdaHessian as in Adam. That is, we did not tune these two hyperparameters, even though tuning them could potentially lead to even better performance.[5] Compared with SGD, AdaHessian achieves comparable accuracy for both ResNet20 (0.05% higher) and ResNet32 (0.06% lower). The training and testing curves of different optimizers for ResNet20/32 on Cifar10 are shown in Figure .7.

Next, we use the best learning rate obtained by training ResNet20/32 on Cifar10 to optimize ResNet18 on ImageNet for all four optimizers. We try two different learning rate schedules for all four optimizers, and we use the one with the better result. The two learning rate schedules are quite standard, i.e., the step decay schedule and the plateau based schedule [46]. The final result is reported in Table 2. Again note that the final performances of Adam and AdamW are much worse than that of SGD and AdaHessian. We plot the training and testing curve in Figure .8.

It is worthwhile to note that our learning rate tuning is performed at an academic scale, but AdaHessian still significantly exceeds other adaptive methods and reaches the same performance level as SGD which has been tuned at the industrial scale.

Neural Machine Translation

We use BLEU [44] as the evaluation metric for NMT. Following standard practice, we measure tokenized case-sensitive BLEU and case-insensitive BLEU for WMT14 En-De and IWSLT14 De-En, respectively. For a fair comparison, we do not include other external datasets.

The NMT results are shown in Table 3. The first interesting observation is that here SGD performs much worse than AdamW (which is opposite to its behaviour for image classification problems where SGD has superior performance; see Appendix). As pointed out in the introduction, even the choice of the optimizer has become another hyperparameter. In particular, note that the BLEU scores of SGD are 7.09 and 2.15 lower than AdamW on IWSLT14 and WMT14, which is quite significant. Similar observations about SGD were also reported in [72].

Despite this, AdaHessian achieves state-of-the-art performance for NMT with transformers. In particular, AdaHessian outperforms AdamW by 0.13 BLEU score

---

[5] In fact, in Table 8 we achieve 92.40 for ResNet20 which is higher than what we report in Table 2. This is to emphasize that we only tuned learning rate in Table 2. Still AdaHessian achieves significantly better results than Adam.

Table 5: *Comparison of AdamW and* AdaHessian *for SqueezeBERT on the development set of the GLUE benchmark. As can be seen, the average performance of* AdaHessian *is 0.41 higher as compared to AdamW. The result of AdamW$^+$ is directly from [27] and the result of AdamW* is reproduced by us.*

	RTE	MPRC	STS-B	SST-2	QNLI	QQP	MNLI-m	MNLI-mm	Avg.
AdamW$^+$ [27]	71.8	89.8	89.4	**92.0**	**90.5**	89.4	82.9	82.3	86.01
AdamW*	79.06	90.69	90.00	91.28	90.30	**89.49**	82.61	81.84	86.91
AdaHessian	**80.14**	**91.94**	**90.59**	91.17	89.97	89.33	**82.78**	**82.62**	**87.32**

on IWSLT14. Furthermore, the accuracy of AdaHessian on WMT14 is 28.52, which is 0.33 higher than that of AdamW. We also plot the training losses of AdamW and AdaHessian on IWSLT14/WMT14 in Figure .9. As one can see, AdaHessian consistently achieves lower training loss. These improvements are quite significant for NMT, and importantly these are achieved even though AdaHessian directly uses the same β_1 and β_2, as well as the same number of warmup iterations as in AdamW.

Language Modeling

We report the language modeling results in Table 4, using the tensorized transformer proposed in [35]. Similar to NMT, note that the perplexity (PPL) of SGD is more than 57 points worse than AdamW on Wikitext-103. That is, similar to the NMT task, SGD performs worse than AdamW. However, AdaHessian achieves more than 1.8/1.0 better PPL than that of AdamW on PTB/Wikitext-103, respectively.

We also show the detailed training loss curves in Figure .10. AdaHessian achieves consistently lower loss values than AdamW throughout the training process on both PTB and Wikitext-103. Similar to NMT, the β_1/β_2 as well as the warmup phase of AdaHessian are kept the same as AdamW.

Natural Language Understanding

We report the NLU results in Table 5, using the SqueezeBERT model [26] tested on GLUE datasets [58]. As can be seen, AdaHessian has better performance than AdamW on 5 out of 8 tasks. Particularly, on RTE and MPRC, AdaHessian achieves more than 1 point as compared to AdamW. On average, AdaHessian outperforms AdamW by 0.41 points. Note that similar to NMT and LM, except learning rate and block size, AdaHessian directly uses the same hyperparameters as AdamW. Interestingly, note that these results are better than those reported in SqueezeBERT [27], even though we only change the optimizer to AdaHessian instead of AdamW.

Recommendation System

We solely focus on modern recommendation systems, and in particular on the DLRM model widely adopted in industry [40]. These systems include a large embedding layer followed by a series of dense FC layers. In training, a sparse set of rows of the embedding layer is used and only those rows are updated. These rows do change from one iteration to the next. For such a sparse setting, we use Adagrad to update the embedding table, and we use AdaHessian to update the rest of the FC network in the experiments. (Pytorch currently does not support second order backpropagation for the sparse gradient to the embedding.) AdaHessian uses the same hyperparameters for updating the embedding table as in the Adagrad experiment without tuning. The training and testing accuracy curves are reported in Figure 6. The testing accuracy of AdaHessian is 79.167%, which is 0.032% higher than Adagrad. It should be noted that this is a quite significant accuracy increase for Recommendation Systems [60].

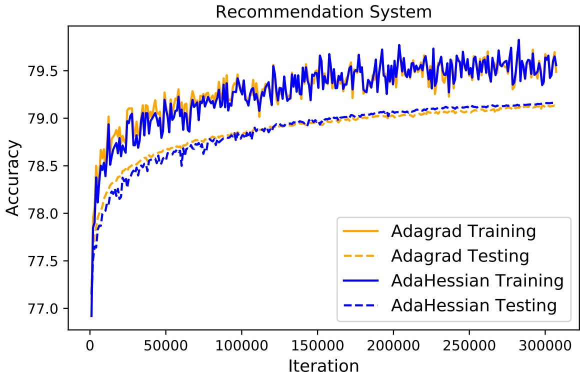


Figure 6: *Training and Testing Accuracy curves of Adagrad and* AdaHessian *on Criteo Ad Kaggle dataset. As can be seen, the test accuracy of* AdaHessian *is better (0.032%) than that of Adagrad. This is quite significant for this task.*

Discussion

As reported in the previous section, AdaHessian achieves state-of-the-art performance on a wide range of tasks. Two important issues are the sensitivity of AdaHessian to the hyperparameters of learning rate and block size. This is discussed next.

Learning Rate and Block Size Effects

Here, we explore the effects of the learning rate and block size b on AdaHessian. We first start with the effect of learning rate, and test the performance of AdaHessian and AdamW with different learning rates. The results

Table 6: *Robustness of AdamW and AdaHessian to the learning rate on IWSLT14. We scale the base learning rate used in Section . As can be seen, AdaHessian is much more robust to large learning rate variability as compared to AdamW.*

LR Scaling	**0.5**	**1**	**2**	**3**	**4**	**5**	**6**	**10**
AdamW	**35.42 ± .09**	35.66 ± .11	**35.37 ± .07**	**35.18 ± .07**	**34.79 ± .15**	14.41 ± 13.25	0.41 ± .32	Diverge
AdaHessian	35.33 ± .10	**35.79 ± .06**	35.21 ± .14	34.74 ± .10	34.19 ± .06	**33.78 ± .14**	**32.70 ± .10**	**32.48 ± .83**

Table 7: *Block Size effect of AdaHessian on IWSLT14. With various block sizes, the performance of AdaHessian is very stable and no worse than that of AdamW (35.66 ± .11).*

Block Size	**1**	**2**	**4**	**8**	**16**	**32**	**64**	**128**
AdaHessian	35.67 ± .10	35.66 ± .07	35.78 ± .07	35.77 ± .08	35.67 ± .08	**35.79 ± .06**	35.72 ± .06	35.67 ± .11

are reported in Table 6 for IWSLT14 dataset, where we scale the original learning rate with a constant factor, ranging from 0.5 to 20 (the original learning rate is the same as in Section). It can be seen that AdaHessian is more robust to the large learning rates. Even with 10× learning rate scaling, AdaHessian still achieves 32.48 BLEU score, while AdamW diverges even with 6× learning rate scaling. This is a very desirable property of AdaHessian, as it results in reasonable performance for such a wide range of learning rates.

We also test the effect of the spatial averaging block size (parameter b in Eq. 10). As a reminder, this parameter is used for spatially averaging the Hessian diagonal as illustrated in Figure 5. The sensitivity results are shown in Table 7 where we vary the block size from 1 to 128. While the best performance is achieved for the block size of 32, the performance variation for other block sizes is rather small. Moreover, all the results are still no worse than the result with AdamW.

AdaHessian Overhead

Here, we discuss and measure the overhead of AdaHessian. In terms of computational complexity, AdaHessian requires twice the flops as compared to SGD. This 2× overhead comes from the cost of computing the Hessian diagonal, when one Hutchinson step is performed per optimization iteration. Each Hutchinson step requires computing one Hessian matvec (the $\mathbf{H}z$ term in Eq. 9). This step requires one more gradient backpropagation, hence leading to twice the theoretical complexity.

We have also measured the actual runtime of AdaHessian in PyTorch on a single RTX Titan GPU machine, as reported in the second column of Table 8. For ResNet20, AdaHessian is 2.42× slower than SGD (and 2.27× slower than Adam). As one can see, AdaHessian is not orders of magnitude slower than first order methods. The gap between the measured and theoretical speed is likely due to the fact that Pytorch [45] (and other existing frameworks) are highly optimized for first order methods. Even then, if one considers the fact that SGD needs a lot of tuning, this overhead may not be large.

It is also possible to reduce the AdaHessian overhead. One simple idea is to reduce the Hutchinson calculation frequency from 1 Hessian matvec per iteration to every multiple iterations. For example, for a frequency of 2, we perform the Hutchinson step at every other optimization iteration. This reduces the theoretical computational cost to 1.5× from 2×. One can also further reduce the frequency to 5, for which this cost reduces to 1.2×.

We studied how such reduced Hutchinson calculation frequency approach would impact the performance. We report the results for training ResNet20/ResNet32 on the Cifar10 in Table 8, when we vary the Hutchinson frequency from 1 to 5. As one can see, there is a small performance variation, but the AdaHessian overhead significantly decreases as compared to SGD and Adam.

Conclusions

In this work, we proposed AdaHessian, an adaptive Hessian based optimizer. AdaHessian incorporates an approximate Hessian diagonal, with spatial averaging and momentum to precondition the gradient vector. This automatically rescales the gradient vector resulting in better descent directions. One of the key novelties in our approach is the incorporation spatial averaging for Hessian diagonal along with an exponential moving average in time. These enable us to smooth noisy local Hessian information which could be highly misleading.

We extensively tested AdaHessian on various datasets and tasks, using state-of-the-art models. These include IWSLT14 and WMT14 for neural machine translation, PTB and Wikitext-103 for language modeling, GLUE for natural language understanding, Cifar10 and ImageNet for image classification (provided in Appendix), and Criteo Ad Kaggle for recommendation system (provided in Appendix). AdaHessian consistently achieves comparable or higher generalization performance as compared to the highly tuned default optimizers used for these different tasks.

Stepping back, it is important for every work to state its limitations (in general, but in particular in this area). The current limitation of AdaHessian is that it is $2-3\times$ slower than first order methods such as SGD and Adam. We briefly explored how this overhead could be reduced, but more work is needed in this area. However, AdaHessian consistently achieves comparable or better accuracy. For example, for LM task, AdaHessian

Table 8: *Comparison between* AdaHessian *theoretical and measured speed, as compared to Adam and SGD, tested on Cifar10. We also measured the speed up for different Hessian computation frequencies. As one can see,* AdaHessian *is* not *orders of magnitude slower than SGD, despite the widely-held incorrect belief about the efficiency of Hessian based methods. Furthermore, by increasing the Hessian computation frequency, the run time can improve from* 3.23× *to* 1.45×*, as compared to SGD for ResNet32. The real measurement is performed on one RTX Titan GPU.*

Hessian Computation Frequency	**1**	**2**	**3**	**4**	**5**
Theoretical Per-iteration Cost (×SGD)	2×	1.5×	1.33×	1.25×	1.2×
ResNet20 (Cifar10)	92.13 ± .08	92.40 ± .04	92.06 ± .18	92.17 ± .21	92.16 ± .12
Measured Per-iteration Cost (×SGD)	2.42×	1.71×	1.47×	1.36×	1.28×
Measured Per-iteration Cost (×Adam)	2.27×	1.64×	1.42×	1.32×	1.25×
ResNet32 (Cifar10)	93.08 ± .10	92.91 ± .14	92.95 ± .17	92.93 ± .24	93.00 ± .10
Measured Per-iteration Cost (×SGD)	3.23×	2.12×	1.74×	1.56×	1.45×
Measured Per-iteration Cost (×Adam)	2.91×	1.96×	1.64×	1.48×	1.38×

achieves up to 2.7 better PPL, as compared to AdamW, which is significant for this task.

Finally, from a higher-level perspective, we should note that there has been significant development within second order methods, both theory and practice, even though these methods were widely viewed as being inapplicable for machine learning even just a few years ago. Some examples include Hessian based model compression [18, 19, 24, 31], adversarial attacks [68], and studies of the loss landscape topology for different NN architectures [52, 66], to name just a few. AdaHessian is an important step in this area, and we expect that it will enable still further progress. We have open sourced AdaHessian and we hope that it would help this progress [1].

Acknowledgments

This work was supported by a gracious fund from Amazon Machine Learning Research Award (MLRA). The UC Berkeley team also acknowledges gracious support from Intel corporation, Intel VLAB team, Google Cloud, Google TFTC team, and Nvidia. Amir Gholami was supported through funding from Samsung SAIT. Michael Mahoney would like to acknowledge the DARPA, NSF, and ONR via its BRC on RandNLA for providing partial support of this work. Our conclusions do not necessarily reflect the position or the policy of our sponsors, and no official endorsement should be inferred.

References

[1] 2020. https://github.com/amirgholami/ADAHESSIAN.git.

[2] Agarwal, N.; Allen-Zhu, Z.; Bullins, B.; Hazan, E.; and Ma, T. 2016. Finding approximate local minima for nonconvex optimization in linear time. *arXiv preprint arXiv:1611.01146* .

[3] Agarwal, N.; Bullins, B.; and Hazan, E. 2016. Second-order stochastic optimization in linear time. *Journal of Machine Learning Research* 1050: 15.

[4] Aghazadeh, A.; Gupta, V.; DeWeese, A.; Koyluoglu, O. O.; and Ramchandran, K. 2020. BEAR: Sketching BFGS Algorithm for Ultra-High Dimensional Feature Selection in Sublinear Memory. *arXiv preprint arXiv:2010.13829* .

[5] Bekas, C.; Kokiopoulou, E.; and Saad, Y. 2007. An estimator for the diagonal of a matrix. *Applied numerical mathematics* 57(11-12): 1214–1229.

[6] Bollapragada, R.; Byrd, R. H.; and Nocedal, J. 2019. Exact and inexact subsampled Newton methods for optimization. *IMA Journal of Numerical Analysis* 39(2): 545–578.

[7] Bollapragada, R.; Mudigere, D.; Nocedal, J.; Shi, H.-J. M.; and Tang, P. T. P. 2018. A progressive batching L-BFGS method for machine learning. *arXiv preprint arXiv:1802.05374* .

[8] Bottou, L.; Curtis, F. E.; and Nocedal, J. 2018. Optimization methods for large-scale machine learning. *SIAM Review* 60(2): 223–311.

[9] Boyd, S.; and Vandenberghe, L. 2004. *Convex optimization.* Cambridge university press.

[10] Byrd, R. H.; Lu, P.; Nocedal, J.; and Zhu, C. 1995. A limited memory algorithm for bound constrained optimization. *SIAM Journal on scientific computing* 16(5): 1190–1208.

[11] Carmon, Y.; Duchi, J. C.; Hinder, O.; and Sidford, A. 2018. Accelerated methods for nonconvex optimization. *SIAM Journal on Optimization* 28(2): 1751–1772.

[12] Chaudhari, P.; Choromanska, A.; Soatto, S.; LeCun, Y.; Baldassi, C.; Borgs, C.; Chayes, J.; Sagun, L.; and Zecchina, R. 2019. Entropy-sgd: Biasing gradient descent into wide valleys. *Journal of Statistical Mechanics: Theory and Experiment* 2019(12): 124018.

[13] Chen, C.; Reiz, S.; Yu, C.; Bungartz, H.-J.; and Biros, G. 2019. Fast Evaluation and Approximation of the Gauss-Newton Hessian Matrix for the Multilayer Perceptron. *arXiv preprint arXiv:1910.12184* .

[14] Conn, A. R.; Gould, N. I.; and Toint, P. L. 2000. *Trust region methods.* Series on Optimization. SIAM.

[15] Dai, Z.; Yang, Z.; Yang, Y.; Carbonell, J. G.; Le, Q.; and Salakhutdinov, R. 2019. Transformer-XL: Attentive Language Models beyond a Fixed-Length Context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2978–2988.

flat minima generalize well

arXiv:2010.01412v2 [cs.LG] 4 Dec 2020

Sharpness-Aware Minimization for Efficiently Improving Generalization

Pierre Foret* Ariel Kleiner Hossein Mobahi Behnam Neyshabur

{pierreforet,akleiner,hmobahi,neyshabur}@google.com
Google Research, Mountain View, CA, USA

Abstract

In today's heavily overparameterized models, the value of the training loss provides few guarantees on model generalization ability. Indeed, optimizing only the training loss value, as is commonly done, can easily lead to suboptimal model quality. Motivated by the connection between geometry of the loss landscape and generalization—including a generalization bound that we prove here—we introduce a novel, effective procedure for instead simultaneously minimizing loss value and loss sharpness. In particular, our procedure, Sharpness-Aware Minimization (SAM), seeks parameters that lie in neighborhoods having uniformly low loss; this formulation results in a min-max optimization problem on which gradient descent can be performed efficiently. We present empirical results showing that SAM improves model generalization across a variety of benchmark datasets (e.g., CIFAR-{10, 100}, ImageNet, finetuning tasks) and models, yielding novel state-of-the-art performance for several. Additionally, we find that SAM natively provides robustness to label noise on par with that provided by state-of-the-art procedures that specifically target learning with noisy labels. We open source our code at `https://github.com/google-research/sam`.

1 Introduction

Modern machine learning's success in achieving ever better performance on a wide range of tasks has relied in significant part on ever heavier overparameterization, in conjunction with developing ever more effective training algorithms that are able to find parameters that generalize well. Indeed, many modern neural networks can easily memorize the training data and have the capacity to readily overfit (Zhang et al., 2016). Such heavy overparameterization is currently required to achieve state-of-the-art results in a variety of domains (Tan and Le, 2019; Kolesnikov et al., 2020; Huang et al., 2018). In turn, it is essential that such models be trained using procedures that ensure that the parameters actually selected do in fact generalize beyond the training set.

Unfortunately, simply minimizing commonly used loss functions (e.g., cross-entropy) on the training set is typically not sufficient to achieve satisfactory generalization. The training loss landscapes of today's models are commonly complex and non-convex, with a multiplicity of local and global minima, and with different global minima yielding models with different generalization abilities (Shirish Keskar et al., 2016). As a result, the choice of optimizer (and associated optimizer settings) from among the many available (e.g., stochastic gradient descent (Nesterov, 1983), Adam (Kingma and Ba, 2014), RMSProp (Hinton et al., 2012), and others (Duchi et al., 2011; Dozat, 2016; Martens and Grosse, 2015)) has become an important design choice, though understanding of its relationship to model generalization remains nascent (Shirish Keskar et al., 2016; Wilson et al., 2017; Shirish Keskar and Socher, 2017; Agarwal et al., 2020; Jacot et al., 2018). Relatedly, a panoply of methods for modifying

---

[*] Work done as part of the Google AI Residency program.

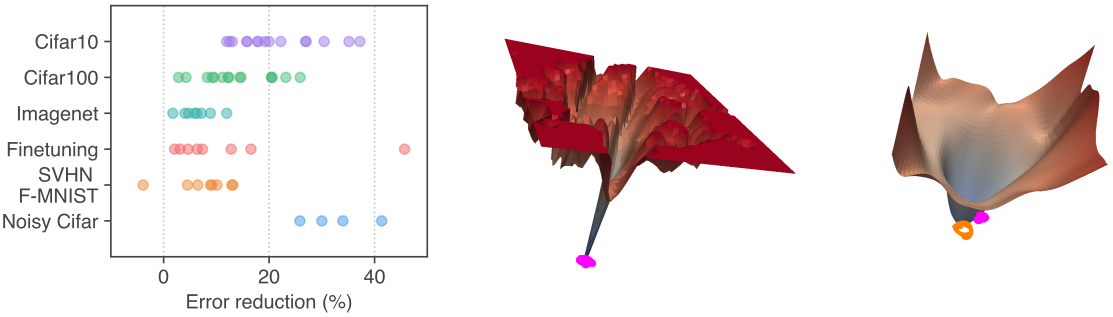


Figure 1: (left) Error rate reduction obtained by switching to SAM. Each point is a different dataset / model / data augmentation. (middle) A sharp minimum to which a ResNet trained with SGD converged. (right) A wide minimum to which the same ResNet trained with SAM converged.

the training process have been proposed, including dropout (Srivastava et al., 2014), batch normalization (Ioffe and Szegedy, 2015), stochastic depth (Huang et al., 2016), data augmentation (Cubuk et al., 2018), and mixed sample augmentations (Zhang et al., 2017; Harris et al., 2020).

The connection between the geometry of the loss landscape—in particular, the flatness of minima—and generalization has been studied extensively from both theoretical and empirical perspectives (Shirish Keskar et al., 2016; Dziugaite and Roy, 2017; Jiang et al., 2019). While this connection has held the promise of enabling new approaches to model training that yield better generalization, practical efficient algorithms that specifically seek out flatter minima and furthermore effectively improve generalization on a range of state-of-the-art models have thus far been elusive (e.g., see (Chaudhari et al., 2016; Izmailov et al., 2018); we include a more detailed discussion of prior work in Section 5).

We present here a new efficient, scalable, and effective approach to improving model generalization ability that directly leverages the geometry of the loss landscape and its connection to generalization, and is powerfully complementary to existing techniques. In particular, we make the following contributions:

- We introduce Sharpness-Aware Minimization (SAM), a novel procedure that improves model generalization by simultaneously minimizing loss value and loss sharpness. SAM functions by seeking parameters that lie in neighborhoods having uniformly low loss value (rather than parameters that only themselves have low loss value, as illustrated in the middle and righthand images of Figure 1), and can be implemented efficiently and easily.

- We show via a rigorous empirical study that using SAM improves model generalization ability across a range of widely studied computer vision tasks (e.g., CIFAR-{10, 100}, ImageNet, finetuning tasks) and models, as summarized in the lefthand plot of Figure 1. For example, applying SAM yields novel state-of-the-art performance for a number of already-intensely-studied tasks, such as CIFAR-100, SVHN, Fashion-MNIST, and the standard set of image classification finetuning tasks (e.g., Flowers, Stanford Cars, Oxford Pets, etc).

- We show that SAM furthermore provides robustness to label noise on par with that provided by state-of-the-art procedures that specifically target learning with noisy labels.

- Through the lens provided by SAM, we further elucidate the connection between loss sharpness and generalization by surfacing a promising new notion of sharpness, which we term *m-sharpness*.

Section 2 below derives the SAM procedure and presents the resulting algorithm in full detail. Section 3 evaluates SAM empirically, and Section 4 further analyzes the connection between loss sharpness and generalization through the lens of SAM. Finally, we conclude with an overview of related work and a discussion of conclusions and future work in Sections 5 and 6, respectively.

2 Sharpness-Aware Minimization (SAM)

Throughout the paper, we denote scalars as a, vectors as $\boldsymbol{a}$, matrices as $\boldsymbol{A}$, sets as $\mathcal{A}$, and equality by definition as $\triangleq$. Given a training dataset $\mathcal{S} \triangleq \cup_{i=1}^{n}\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$ drawn i.i.d. from distribution $\mathscr{D}$, we seek to learn a model that generalizes well. In particular, consider a family of models parameterized by $\boldsymbol{w} \in \mathcal{W} \subseteq \mathbb{R}^d$; given a per-data-point loss function $l : \mathcal{W} \times \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}_+$, we define the training set loss $L_\mathcal{S}(\boldsymbol{w}) \triangleq \frac{1}{n}\sum_{i=1}^{n} l(\boldsymbol{w}, \boldsymbol{x}_i, \boldsymbol{y}_i)$ and the population loss $L_\mathscr{D}(\boldsymbol{w}) \triangleq \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim D}[l(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y})]$. Having observed only $\mathcal{S}$, the goal of model training is to select model parameters $\boldsymbol{w}$ having low population loss $L_\mathscr{D}(\boldsymbol{w})$.

Utilizing $L_\mathcal{S}(\boldsymbol{w})$ as an estimate of $L_\mathscr{D}(\boldsymbol{w})$ motivates the standard approach of selecting parameters $\boldsymbol{w}$ by solving $\min_{\boldsymbol{w}} L_\mathcal{S}(\boldsymbol{w})$ (possibly in conjunction with a regularizer on $\boldsymbol{w}$) using an optimization procedure such as SGD or Adam. Unfortunately, however, for modern overparameterized models such as deep neural networks, typical optimization approaches can easily result in suboptimal performance at test time. In particular, for modern models, $L_\mathcal{S}(\boldsymbol{w})$ is typically non-convex in $\boldsymbol{w}$, with multiple local and even global minima that may yield similar values of $L_\mathcal{S}(\boldsymbol{w})$ while having significantly different generalization performance (i.e., significantly different values of $L_\mathscr{D}(\boldsymbol{w})$).

Motivated by the connection between sharpness of the loss landscape and generalization, we propose a different approach: rather than seeking out parameter values $\boldsymbol{w}$ that simply have low training loss value $L_\mathcal{S}(\boldsymbol{w})$, we seek out parameter values whose entire neighborhoods have uniformly low training loss value (equivalently, neighborhoods having both low loss and low curvature). The following theorem illustrates the motivation for this approach by bounding generalization ability in terms of neighborhood-wise training loss (full theorem statement and proof in Appendix A):

Theorem (stated informally) 1. *For any $\rho > 0$, with high probability over training set $\mathcal{S}$ generated from distribution $\mathscr{D}$,*

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

where $h : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ is a strictly increasing function (under some technical conditions on $L_\mathscr{D}(\boldsymbol{w})$).

To make explicit our sharpness term, we can rewrite the right hand side of the inequality above as

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

The term in square brackets captures the sharpness of $L_\mathcal{S}$ at $\boldsymbol{w}$ by measuring how quickly the training loss can be increased by moving from $\boldsymbol{w}$ to a nearby parameter value; this sharpness term is then summed with the training loss value itself and a regularizer on the magnitude of $\boldsymbol{w}$. Given that the specific function h is heavily influenced by the details of the proof, we substitute the second term with $\lambda\|\boldsymbol{w}\|_2^2/\rho^2$ for a hyperparameter λ, yielding a standard L2 regularization term. Thus, inspired by the terms from the bound, we propose to select parameter values by solving the following Sharpness-Aware Minimization (SAM) problem:

$$\min_{\boldsymbol{w}} L_\mathcal{S}^{SAM}(\boldsymbol{w}) + \lambda||\boldsymbol{w}||_2^2 \qquad \text{where} \qquad L_\mathcal{S}^{SAM}(\boldsymbol{w}) \triangleq \max_{||\boldsymbol{\epsilon}||_p \leq \rho} L_S(\boldsymbol{w} + \boldsymbol{\epsilon}), \tag{1}$$

where $\rho \geq 0$ is a hyperparameter and $p \in [1, \infty]$ (we have generalized slightly from an L2-norm to a p-norm in the maximization over $\boldsymbol{\epsilon}$, though we show empirically in appendix C.5 that $p = 2$ is typically optimal). Figure 1 shows[1] the loss landscape for a model that converged to minima found by

---

[1] Figure 1 was generated following Li et al. (2017) with the provided ResNet56 (no residual connections) checkpoint, and training the same model with SAM.

minimizing either $L_{\mathcal{S}}(\boldsymbol{w})$ or $L_{\mathcal{S}}^{SAM}(\boldsymbol{w})$, illustrating that the sharpness-aware loss prevents the model from converging to a sharp minimum.

In order to minimize $L_{\mathcal{S}}^{SAM}(\boldsymbol{w})$, we derive an efficient and effective approximation to $\nabla_{\boldsymbol{w}} L_{\mathcal{S}}^{SAM}(\boldsymbol{w})$ by differentiating through the inner maximization, which in turn enables us to apply stochastic gradient descent directly to the SAM objective. Proceeding down this path, we first approximate the inner maximization problem via a first-order Taylor expansion of $L_{\mathcal{S}}(\boldsymbol{w}+\boldsymbol{\epsilon})$ w.r.t. $\boldsymbol{\epsilon}$ around $\mathbf{0}$, obtaining

$$\boldsymbol{\epsilon}^*(\boldsymbol{w}) \triangleq \underset{\|\boldsymbol{\epsilon}\|_p \le \rho}{\arg\max}\, L_{\mathcal{S}}(\boldsymbol{w}+\boldsymbol{\epsilon}) \approx \underset{\|\boldsymbol{\epsilon}\|_p \le \rho}{\arg\max}\, L_{\mathcal{S}}(\boldsymbol{w}) + \boldsymbol{\epsilon}^T \nabla_{\boldsymbol{w}} L_{\mathcal{S}}(\boldsymbol{w}) = \underset{\|\boldsymbol{\epsilon}\|_p \le \rho}{\arg\max}\, \boldsymbol{\epsilon}^T \nabla_{\boldsymbol{w}} L_{\mathcal{S}}(\boldsymbol{w}).$$

In turn, the value $\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})$ that solves this approximation is given by the solution to a classical dual norm problem ($|\cdot|^{q-1}$ denotes elementwise absolute value and power) [2]:

$$\hat{\boldsymbol{\epsilon}}(\boldsymbol{w}) = \rho \; \text{sign}\left(\nabla_{\boldsymbol{w}} L_{\mathcal{S}}(\boldsymbol{w})\right) \frac{|\nabla_{\boldsymbol{w}} L_{\mathcal{S}}(\boldsymbol{w})|^{q-1}}{\left(\|\nabla_{\boldsymbol{w}} L_{\mathcal{S}}(\boldsymbol{w})\|_q^q\right)^{1/p}}, \tag{2}$$

where $1/p + 1/q = 1$. Substituting back into equation (1) and differentiating, we then have

$$\begin{aligned} \nabla_{\boldsymbol{w}} L_{\mathcal{S}}^{SAM}(\boldsymbol{w}) &\approx \nabla_{\boldsymbol{w}} L_{\mathcal{S}}(\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})) = \frac{d(\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}(\boldsymbol{w}))}{d\boldsymbol{w}} \nabla_{\boldsymbol{w}} L_{\mathcal{S}}(\boldsymbol{w})|_{\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}(w)} \\ &= \nabla_{w} L_{\mathcal{S}}(\boldsymbol{w})|_{\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})} + \frac{d\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})}{d\boldsymbol{w}} \nabla_{\boldsymbol{w}} L_{\mathcal{S}}(\boldsymbol{w})|_{\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})}. \end{aligned}$$

This approximation to $\nabla_{\boldsymbol{w}} L_{\mathcal{S}}^{SAM}(\boldsymbol{w})$ can be straightforwardly computed via automatic differentiation, as implemented in common libraries such as JAX, TensorFlow, and PyTorch. Though this computation implicitly depends on the Hessian of $L_{\mathcal{S}}(\boldsymbol{w})$ because $\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})$ is itself a function of $\nabla_{\boldsymbol{w}} L_{\mathcal{S}}(\boldsymbol{w})$, the Hessian enters only via Hessian-vector products, which can be computed tractably without materializing the Hessian matrix. Nonetheless, to further accelerate the computation, we drop the second-order terms. obtaining our final gradient approximation:

$$\nabla_{\boldsymbol{w}} L_{\mathcal{S}}^{SAM}(\boldsymbol{w}) \approx \nabla_{\boldsymbol{w}} L_{\mathcal{S}}(w)|_{\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})}. \tag{3}$$

As shown by the results in Section 3, this approximation (without the second-order terms) yields an effective algorithm. In Appendix C.4, we additionally investigate the effect of instead including the second-order terms; in that initial experiment, including them surprisingly degrades performance, and further investigating these terms' effect should be a priority in future work.

We obtain the final SAM algorithm by applying a standard numerical optimizer such as stochastic gradient descent (SGD) to the SAM objective $L_{\mathcal{S}}^{SAM}(\boldsymbol{w})$, using equation 3 to compute the requisite objective function gradients. Algorithm 1 gives pseudo-code for the full SAM algorithm, using SGD as the base optimizer, and Figure 2 schematically illustrates a single SAM parameter update.

3 Empirical Evaluation

In order to assess SAM's efficacy, we apply it to a range of different tasks, including image classification from scratch (including on CIFAR-10, CIFAR-100, and ImageNet), finetuning pretrained models, and learning with noisy labels. In all cases, we measure the benefit of using SAM by simply replacing the optimization procedure used to train existing models with SAM, and computing the resulting effect on model generalization. As seen below, SAM materially improves generalization performance in the vast majority of these cases.

---

[2] In the case of interest $p = 2$, this boils down to simply rescaling the gradient such that its norm is ρ.

```
Input: Training set $\mathcal{S} \triangleq \cup_{i=1}^{n}\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$, Loss function
    $l : \mathcal{W} \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_+$, Batch size $b$, Step size
    $\eta > 0$, Neighborhood size $\rho > 0$.
Output: Model trained with SAM
Initialize weights $\boldsymbol{w}_0$, $t = 0$;
while not converged do
    Sample batch $\mathcal{B} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), ...(\boldsymbol{x}_b, \boldsymbol{y}_b)\}$;
    Compute gradient $\nabla_{\boldsymbol{w}} L_{\mathcal{B}}(\boldsymbol{w})$ of the batch's training loss;
    Compute $\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})$ per equation 2;
    Compute gradient approximation for the SAM objective (equation 3): $\boldsymbol{g} = \nabla_w L_{\mathcal{B}}(\boldsymbol{w})|_{\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})}$;
    Update weights: $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \boldsymbol{g}$;
    $t = t + 1$;
end
return $\boldsymbol{w}_t$
```

Algorithm 1: SAM algorithm

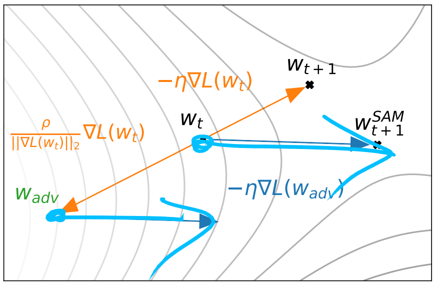


Figure 2: Schematic of the SAM parameter update.

3.1 Image Classification From Scratch

We first evaluate SAM's impact on generalization for today's state-of-the-art models on CIFAR-10 and CIFAR-100 (without pretraining): WideResNets with ShakeShake regularization (Zagoruyko and Komodakis, 2016; Gastaldi, 2017) and PyramidNet with ShakeDrop regularization (Han et al., 2016; Yamada et al., 2018). Note that some of these models have already been heavily tuned in prior work and include carefully chosen regularization schemes to prevent overfitting; therefore, significantly improving their generalization is quite non-trivial. We have ensured that our implementations' generalization performance in the absence of SAM matches or exceeds that reported in prior work (Cubuk et al., 2018; Lim et al., 2019)

All results use basic data augmentations (horizontal flip, padding by four pixels, and random crop). We also evaluate in the setting of more advanced data augmentation methods such as cutout regularization (Devries and Taylor, 2017) and AutoAugment (Cubuk et al., 2018), which are utilized by prior work to achieve state-of-the-art results.

SAM has a single hyperparameter ρ (the neighborhood size), which we tune via a grid search over $\{0.01, 0.02, 0.05, 0.1, 0.2, 0.5\}$ using 10% of the training set as a validation set. Please see appendix C.1 for the values of all hyperparameters and additional training details. We observed that a default value of $\rho = 0.05$ is often satisfactory, and we also report in appendix C.3 the test accuracy obtained for this value of ρ without additional tuning. As each SAM weight update requires two backpropagation operations (one to compute $\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})$ and another to compute the final gradient), we allow each non-SAM training run to execute twice as many epochs as each SAM training run, and we report the best score achieved by each non-SAM training run across either the standard epoch count or the doubled epoch count [3]. We run five independent replicas of each experimental condition for which we report results (each with independent weight initialization and data shuffling), reporting the resulting mean error (or accuracy) on the test set, and the associated 95% confidence interval. Our implementations utilize JAX (Bradbury et al., 2018), and we train all models on a single host having 8 Nvidia V100 GPUs. To compute the SAM update when parallelizing across multiple accelerators, we divide each data batch evenly among the accelerators, independently compute the SAM gradient on each accelerator, and average the resulting sub-batch SAM gradients to obtain the final SAM update.

As seen in Table 1, SAM improves generalization across all settings evaluated for CIFAR-10 and CIFAR-100. For example, SAM enables a simple WideResNet to attain 1.6% test error, versus 2.2% error without SAM. Such gains have previously been attainable only by using more complex model

---

[3] Training for longer generally did not improve accuracy significantly, except for the models previously trained for only 200 epochs and for the largest, most regularized model (PyramidNet + ShakeDrop).

architectures (e.g., PyramidNet) and regularization schemes (e.g., Shake-Shake, ShakeDrop); SAM provides an easily-implemented, model-independent alternative. Furthermore, SAM delivers improvements even when applied atop complex architectures that already use sophisticated regularization: for instance, applying SAM to a PyramidNet with ShakeDrop regularization yields 10.3% error on CIFAR-100, which is, to our knowledge, a new state of the art on this dataset without the use of additional data.

		CIFAR-10		CIFAR-100	
Model	Augmentation	SAM	SGD	SAM	SGD
WRN-28-10 (200 epochs)	Basic	**2.7**$_{\pm 0.1}$	3.5$_{\pm 0.1}$	**16.5**$_{\pm 0.2}$	18.8$_{\pm 0.2}$
WRN-28-10 (200 epochs)	Cutout	**2.3**$_{\pm 0.1}$	2.6$_{\pm 0.1}$	**14.9**$_{\pm 0.2}$	16.9$_{\pm 0.1}$
WRN-28-10 (200 epochs)	AA	**2.1**$_{\pm <0.1}$	2.3$_{\pm 0.1}$	**13.6**$_{\pm 0.2}$	15.8$_{\pm 0.2}$
WRN-28-10 (1800 epochs)	Basic	**2.4**$_{\pm 0.1}$	3.5$_{\pm 0.1}$	**16.3**$_{\pm 0.2}$	19.1$_{\pm 0.1}$
WRN-28-10 (1800 epochs)	Cutout	**2.1**$_{\pm 0.1}$	2.7$_{\pm 0.1}$	**14.0**$_{\pm 0.1}$	17.4$_{\pm 0.1}$
WRN-28-10 (1800 epochs)	AA	**1.6**$_{\pm 0.1}$	2.2$_{\pm <0.1}$	**12.8**$_{\pm 0.2}$	16.1$_{\pm 0.2}$
Shake-Shake (26 2x96d)	Basic	**2.3**$_{\pm <0.1}$	2.7$_{\pm 0.1}$	**15.1**$_{\pm 0.1}$	17.0$_{\pm 0.1}$
Shake-Shake (26 2x96d)	Cutout	**2.0**$_{\pm <0.1}$	2.3$_{\pm 0.1}$	**14.2**$_{\pm 0.2}$	15.7$_{\pm 0.2}$
Shake-Shake (26 2x96d)	AA	**1.6**$_{\pm <0.1}$	1.9$_{\pm 0.1}$	**12.8**$_{\pm 0.1}$	14.1$_{\pm 0.2}$
PyramidNet	Basic	**2.7**$_{\pm 0.1}$	4.0$_{\pm 0.1}$	**14.6**$_{\pm 0.4}$	19.7$_{\pm 0.3}$
PyramidNet	Cutout	**1.9**$_{\pm 0.1}$	2.5$_{\pm 0.1}$	**12.6**$_{\pm 0.2}$	16.4$_{\pm 0.1}$
PyramidNet	AA	**1.6**$_{\pm 0.1}$	1.9$_{\pm 0.1}$	**11.6**$_{\pm 0.1}$	14.6$_{\pm 0.1}$
PyramidNet+ShakeDrop	Basic	**2.1**$_{\pm 0.1}$	2.5$_{\pm 0.1}$	**13.3**$_{\pm 0.2}$	14.5$_{\pm 0.1}$
PyramidNet+ShakeDrop	Cutout	**1.6**$_{\pm <0.1}$	1.9$_{\pm 0.1}$	**11.3**$_{\pm 0.1}$	11.8$_{\pm 0.2}$
PyramidNet+ShakeDrop	AA	**1.4**$_{\pm <0.1}$	1.6$_{\pm <0.1}$	**10.3**$_{\pm 0.1}$	10.6$_{\pm 0.1}$

Table 1: Results for SAM on state-of-the-art models on CIFAR-{10, 100} (WRN = WideResNet; AA = AutoAugment; SGD is the standard non-SAM procedure used to train these models).

Beyond CIFAR-{10, 100}, we have also evaluated SAM on the SVHN (Netzer et al., 2011) and Fashion-MNIST datasets (Xiao et al., 2017). Once again, SAM enables a simple WideResNet to achieve accuracy at or above the state of the art for these datasets: 0.99% error for SVHN, and 3.59% for Fashion-MNIST. Details are available in appendix B.1.

To assess SAM's performance at larger scale, we apply it to ResNets (He et al., 2015) of different depths (50, 101, 152) trained on ImageNet (Deng et al., 2009). In this setting, following prior work (He et al., 2015; Szegedy et al., 2015), we resize and crop images to 224-pixel resolution, normalize them, and use batch size 4096, initial learning rate 1.0, cosine learning rate schedule, SGD optimizer with momentum 0.9, and weight decay 0.0001. When applying SAM, we use $\rho = 0.05$ (determined via a grid search on ResNet-50 trained for 100 epochs). We train all models on ImageNet for up to 400 epochs using a Google Cloud TPUv3 and report top-1 and top-5 test error rates for each experimental condition (mean and 95% confidence interval across 5 independent runs).

As seen in Table 2, SAM again consistently improves performance, for example improving the ImageNet top-1 error rate of ResNet-152 from 20.3% to 18.4%. Furthermore, note that SAM enables increasing the number of training epochs while continuing to improve accuracy without overfitting. In contrast, the standard training procedure (without SAM) generally significantly overfits as training extends from 200 to 400 epochs.

3.2 Finetuning

Transfer learning by pretraining a model on a large related dataset and then finetuning on a smaller target dataset of interest has emerged as a powerful and widely used technique for producing high-quality models for a variety of different tasks. We show here that SAM once again offers considerable benefits in this setting, even when finetuning extremely large, state-of-the-art, already high-performing models.

Model	Epoch	SAM		Standard Training (No SAM)	
		Top-1	Top-5	Top-1	Top-5
ResNet-50	100	**22.5**$_{\pm 0.1}$	6.28$_{\pm 0.08}$	22.9$_{\pm 0.1}$	6.62$_{\pm 0.11}$
	200	**21.4**$_{\pm 0.1}$	5.82$_{\pm 0.03}$	22.3$_{\pm 0.1}$	6.37$_{\pm 0.04}$
	400	**20.9**$_{\pm 0.1}$	5.51$_{\pm 0.03}$	22.3$_{\pm 0.1}$	6.40$_{\pm 0.06}$
ResNet-101	100	**20.2**$_{\pm 0.1}$	5.12$_{\pm 0.03}$	21.2$_{\pm 0.1}$	5.66$_{\pm 0.05}$
	200	**19.4**$_{\pm 0.1}$	4.76$_{\pm 0.03}$	20.9$_{\pm 0.1}$	5.66$_{\pm 0.04}$
	400	**19.0**$_{\pm < 0.01}$	4.65$_{\pm 0.05}$	22.3$_{\pm 0.1}$	6.41$_{\pm 0.06}$
ResNet-152	100	**19.2**$_{\pm < 0.01}$	4.69$_{\pm 0.04}$	20.4$_{\pm < 0.0}$	5.39$_{\pm 0.06}$
	200	**18.5**$_{\pm 0.1}$	4.37$_{\pm 0.03}$	20.3$_{\pm 0.2}$	5.39$_{\pm 0.07}$
	400	**18.4**$_{\pm < 0.01}$	4.35$_{\pm 0.04}$	20.9$_{\pm < 0.0}$	5.84$_{\pm 0.07}$

Table 2: Test error rates for ResNets trained on ImageNet, with and without SAM.

In particular, we apply SAM to finetuning EfficentNet-b7 (pretrained on ImageNet) and EfficientNet-L2 (pretrained on ImageNet plus unlabeled JFT; input resolution 475) (Tan and Le, 2019; Kornblith et al., 2018; Huang et al., 2018). We initialize these models to publicly available checkpoints[4] trained with RandAugment (84.7% accuracy on ImageNet) and NoisyStudent (88.2% accuracy on ImageNet), respectively. We finetune these models on each of several target datasets by training each model starting from the aforementioned checkpoint; please see the appendix for details of the hyperparameters used. We report the mean and 95% confidence interval of top-1 test error over 5 independent runs for each dataset.

As seen in Table 3, SAM uniformly improves performance relative to finetuning without SAM. Furthermore, in many cases, SAM yields novel state of the art performance, including 0.30% error on CIFAR-10, 3.92% error on CIFAR-100, and 11.39% error on ImageNet.

Dataset	EffNet-b7 + SAM	EffNet-b7	Prev. SOTA (ImageNet only)	EffNet-L2 + SAM	EffNet-L2	Prev. SOTA
FGVC_Aircraft	6.80$_{\pm 0.06}$	8.15$_{\pm 0.08}$	**5.3** (TBMSL-Net)	**4.82**$_{\pm 0.08}$	5.80$_{\pm 0.1}$	5.3 (TBMSL-Net)
Flowers	**0.63**$_{\pm 0.02}$	1.16$_{\pm 0.05}$	0.7 (BiT-M)	**0.35**$_{\pm 0.01}$	0.40$_{\pm 0.02}$	0.37 (EffNet)
Oxford_IIIT_Pets	**3.97**$_{\pm 0.04}$	4.24$_{\pm 0.09}$	4.1 (Gpipe)	**2.90**$_{\pm 0.04}$	3.08$_{\pm 0.04}$	4.1 (Gpipe)
Stanford_Cars	5.18$_{\pm 0.02}$	5.94$_{\pm 0.06}$	**5.0** (TBMSL-Net)	4.04$_{\pm 0.03}$	4.93$_{\pm 0.04}$	**3.8** (DAT)
CIFAR-10	**0.88**$_{\pm 0.02}$	0.95$_{\pm 0.03}$	1 (Gpipe)	**0.30**$_{\pm 0.01}$	0.34$_{\pm 0.02}$	0.63 (BiT-L)
CIFAR-100	**7.44**$_{\pm 0.06}$	7.68$_{\pm 0.06}$	7.83 (BiT-M)	**3.92**$_{\pm 0.06}$	4.07$_{\pm 0.08}$	6.49 (BiT-L)
Birdsnap	**13.64**$_{\pm 0.15}$	14.30$_{\pm 0.18}$	15.7 (EffNet)	**9.93**$_{\pm 0.15}$	10.31$_{\pm 0.15}$	14.5 (DAT)
Food101	7.02$_{\pm 0.02}$	7.17$_{\pm 0.03}$	7.0 (Gpipe)	**3.82**$_{\pm 0.01}$	3.97$_{\pm 0.03}$	4.7 (DAT)
ImageNet	15.14$_{\pm 0.03}$	15.3	**14.2** (KDforAA)	**11.39**$_{\pm 0.02}$	11.8	11.45 (ViT)

Table 3: Top-1 error rates for finetuning EfficientNet-b7 (left; ImageNet pretraining only) and EfficientNet-L2 (right; pretraining on ImageNet plus additional data, such as JFT) on various downstream tasks. Previous state-of-the-art (SOTA) includes EfficientNet (EffNet) (Tan and Le, 2019), Gpipe (Huang et al., 2018), DAT (Ngiam et al., 2018), BiT-M/L (Kolesnikov et al., 2020), KDforAA (Wei et al., 2020), TBMSL-Net (Zhang et al., 2020), and ViT (Dosovitskiy et al., 2020).

3.3 Robustness to Label Noise

The fact that SAM seeks out model parameters that are robust to perturbations suggests SAM's potential to provide robustness to noise in the training set (which would perturb the training loss landscape). Thus, we assess here the degree of robustness that SAM provides to label noise.

In particular, we measure the effect of applying SAM in the classical noisy-label setting for CIFAR-10, in which a fraction of the training set's labels are randomly flipped; the test set remains unmodified

---

[4] `https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet`

Method	Noise rate (%)			
	20	40	60	80
Sanchez et al. (2019)	94.0	92.8	90.3	74.1
Zhang and Sabuncu (2018)	89.7	87.6	82.7	67.9
Lee et al. (2019)	87.1	81.8	75.4	-
Chen et al. (2019)	89.7	-	-	52.3
Huang et al. (2019)	92.6	90.3	43.4	-
MentorNet (2017)	92.0	91.2	74.2	60.0
Mixup (2017)	94.0	91.5	86.8	76.9
MentorMix (2019)	**95.6**	**94.2**	91.3	**81.0**
SGD	84.8	68.8	48.2	26.2
Mixup	93.0	90.0	83.8	70.2
Bootstrap + Mixup	93.3	92.0	87.6	72.0
SAM	95.1	93.4	90.5	77.9
Bootstrap + SAM	95.4	**94.2**	**91.8**	79.9

Table 4: Test accuracy on the clean test set for models trained on CIFAR-10 with noisy labels. Lower block is our implementation, upper block gives scores from the literature, per Jiang et al. (2019).

(i.e., clean). To ensure valid comparison to prior work, which often utilizes architectures specialized to the noisy-label setting, we train a simple model of similar size (ResNet-32) for 200 epochs, following Jiang et al. (2019). We evaluate five variants of model training: standard SGD, SGD with Mixup (Zhang et al., 2017), SAM, and "bootstrapped" variants of SGD with Mixup and SAM (wherein the model is first trained as usual and then retrained from scratch on the labels predicted by the initially trained model). When applying SAM, we use $\rho = 0.1$ for all noise levels except 80%, for which we use $\rho = 0.05$ for more stable convergence. For the Mixup baselines, we tried all values of $\alpha \in \{1, 8, 16, 32\}$ and conservatively report the best score for each noise level.

As seen in Table 4, SAM provides a high degree of robustness to label noise, on par with that provided by state-of-the art procedures that specifically target learning with noisy labels. Indeed, simply training a model with SAM outperforms all prior methods specifically targeting label noise robustness, with the exception of MentorMix (Jiang et al., 2019). However, simply bootstrapping SAM yields performance comparable to that of MentorMix (which is substantially more complex).

4 Sharpness and Generalization Through the Lens of SAM

4.1 m-sharpness

Though our derivation of SAM defines the SAM objective over the entire training set, when utilizing SAM in practice, we compute the SAM update per-batch (as described in Algorithm 1) or even by averaging SAM updates computed independently per-accelerator (where each accelerator receives a subset of size m of a batch, as described in Section 3). This latter setting is equivalent to modifying the SAM objective (equation 1) to sum over a set of independent ϵ maximizations, each performed on a sum of per-data-point losses on a disjoint subset of m data points, rather than performing the ϵ maximization over a global sum over the training set (which would be equivalent to setting m to the total training set size). We term the associated measure of sharpness of the loss landscape *m-sharpness*.

To better understand the effect of m on SAM, we train a small ResNet on CIFAR-10 using SAM with a range of values of m. As seen in Figure 3 (middle), smaller values of m tend to yield models having better generalization ability. This relationship fortuitously aligns with the need to parallelize across multiple accelerators in order to scale training for many of today's models.

Intriguingly, the m-sharpness measure described above furthermore exhibits better correlation with models' actual generalization gaps as m decreases, as demonstrated by Figure 3 (right)[5]. In

---

[5] We follow the rigorous framework of Jiang et al. (2019), reporting the mutual information between the m-sharpness

eigenvalues(H) small

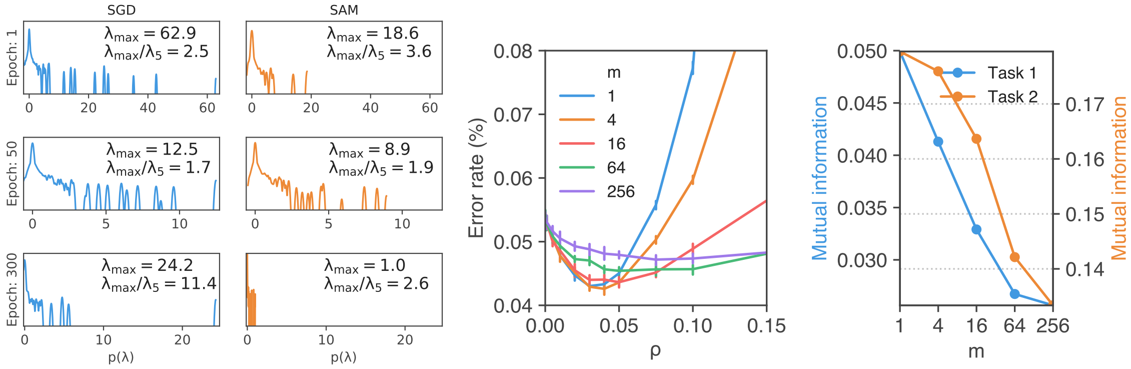


Figure 3: (left) Evolution of the spectrum of the Hessian during training of a model with standard SGD (lefthand column) or SAM (righthand column). (middle) Test error as a function of ρ for different values of m. (right) Predictive power of m-sharpness for the generalization gap, for different values of m (higher means the sharpness measure is more correlated with actual generalization gap).

particular, this implies that m-sharpness with $m < n$ yields a better predictor of generalization than the full-training-set measure suggested by Theorem 1 in Section 2 above, suggesting an interesting new avenue of future work for understanding generalization.

4.2 Hessian Spectra

Motivated by the connection between geometry of the loss landscape and generalization, we constructed SAM to seek out minima of the training loss landscape having both low loss value and low curvature (i.e., low sharpness). To further confirm that SAM does in fact find minima having low curvature, we compute the spectrum of the Hessian for a WideResNet40-10 trained on CIFAR-10 for 300 steps both with and without SAM (without batch norm, which tends to obscure interpretation of the Hessian), at different epochs during training. Due to the parameter space's dimensionality, we approximate the Hessian spectrum using the Lanczos algorithm of Ghorbani et al. (2019).

Figure 3 (left) reports the resulting Hessian spectra. As expected, the models trained with SAM converge to minima having lower curvature, as seen in the overall distribution of eigenvalues, the maximum eigenvalue ($\lambda_{\max}$) at convergence (approximately 25 without SAM, 1.0 with SAM), and the bulk of the spectrum (the ratio $\lambda_{\max}/\lambda_5$, commonly used as a proxy for sharpness (Jastrzebski et al., 2020); up to 11.4 without SAM, and 2.6 with SAM).

5 Related Work

The idea of searching for "flat" minima can be traced back to Hochreiter and Schmidhuber (1995), and its connection to generalization has seen significant study (Shirish Keskar et al., 2016; Dziugaite and Roy, 2017; Neyshabur et al., 2017; Dinh et al., 2017). In a recent large scale empirical study, Jiang et al. (2019) studied 40 complexity measures and showed that a sharpness-based measure has highest correlation with generalization, which motivates penalizing sharpness. Hochreiter and Schmidhuber (1997) was perhaps the first paper on penalizing the sharpness, regularizing a notion related to Minimum Description Length (MDL). Other ideas which also penalize sharp minima include operating on diffused loss landscape (Mobahi, 2016) and regularizing local entropy (Chaudhari et al., 2016), which has also been connected to generalization bounds (Dziugaite and Roy, 2019). The notion of

---

measure and generalization on the two publicly available tasks from the *Predicting generalization in deep learning* NeurIPS2020 competition. `https://competitions.codalab.org/competitions/25301`

A Appendix

A.1 PAC Bayesian Generalization Bound

Below, we state a generalization bound based on sharpness.

Theorem 2. *For any* $\rho > 0$ *and any distribution* $\mathscr{D}$*, with probability* $1-\delta$ *over the choice of the training set* $\mathcal{S} \sim \mathscr{D}$*,*

$$L_{\mathscr{D}}(\boldsymbol{w}) \leq \max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} L_{\mathcal{S}}(\boldsymbol{w}+\boldsymbol{\epsilon}) + \sqrt{\frac{k \log\left(1 + \frac{\|\boldsymbol{w}\|_2^2}{\rho^2}\left(1+\sqrt{\frac{\log(n)}{k}}\right)^2\right) + 4\log\frac{n}{\delta} + \tilde{O}(1)}{n-1}} \tag{4}$$

where $n = |\mathcal{S}|$*,* k *is the number of parameters and we assumed* $L_{\mathscr{D}}(\boldsymbol{w}) \leq \mathbb{E}_{\epsilon_i \sim \mathcal{N}(0,\rho)}[L_{\mathscr{D}}(\boldsymbol{w}+\boldsymbol{\epsilon})]$.

The condition $L_{\mathscr{D}}(\boldsymbol{w}) \leq \mathbb{E}_{\epsilon_i \sim \mathcal{N}(0,\rho)}[L_{\mathscr{D}}(\boldsymbol{w}+\boldsymbol{\epsilon})]$ means that adding Gaussian perturbation should not decrease the test error. This is expected to hold in practice for the final solution but does not necessarily hold for any $\boldsymbol{w}$.

Proof. First, note that the right hand side of the bound in the theorem statement is lower bounded by $\sqrt{k\log(1+\|\boldsymbol{w}\|_2^2/\rho^2)/(4n)}$ which is greater than 1 when $\|\boldsymbol{w}\|_2^2 > \rho^2(\exp(4n/k)-1)$. In that case, the right hand side becomes greater than 1 in which case the inequality holds trivially. Therefore, in the rest of the proof, we only consider the case when $\|\boldsymbol{w}\|_2^2 \leq \rho^2(\exp(4n/k)-1)$.

The proof technique we use here is inspired from Chatterji et al. (2020). Using PAC-Bayesian generalization bound McAllester (1999) and following Dziugaite and Roy (2017), the following generalization bound holds for any prior $\mathscr{P}$ over parameters with probability $1-\delta$ over the choice of the training set $\mathcal{S}$, for any posterior $\mathscr{Q}$ over parameters:

$$\mathbb{E}_{\boldsymbol{w}\sim\mathscr{Q}}[L_{\mathscr{D}}(\boldsymbol{w})] \leq \mathbb{E}_{\boldsymbol{w}\sim\mathscr{Q}}[L_{\mathcal{S}}(\boldsymbol{w})] + \sqrt{\frac{KL(\mathscr{Q}\|\mathscr{P}) + \log\frac{n}{\delta}}{2(n-1)}} \tag{5}$$

Moreover, if $\mathscr{P} = \mathcal{N}(\boldsymbol{\mu}_P, \sigma_P^2\boldsymbol{I})$ and $\mathscr{Q} = \mathcal{N}(\boldsymbol{\mu}_Q, \sigma_Q^2\boldsymbol{I})$, then the KL divergence can be written as follows:

$$KL(\mathscr{P}\|\mathscr{Q}) = \frac{1}{2}\left[\frac{k\sigma_Q^2 + \|\boldsymbol{\mu}_P - \boldsymbol{\mu}_Q\|_2^2}{\sigma_P^2} - k + k\log\left(\frac{\sigma_P^2}{\sigma_Q^2}\right)\right] \tag{6}$$

Given a posterior standard deviation σ_Q, one could choose a prior standard deviation σ_P to minimize the above KL divergence and hence the generalization bound by taking the derivative[7] of the above KL with respect to σ_P and setting it to zero. We would then have ${\sigma_P^*}^2 = \sigma_Q^2 + \|\boldsymbol{\mu}_P - \boldsymbol{\mu}_Q\|_2^2/k$. However, since σ_P should be chosen before observing the training data $\mathcal{S}$ and $\boldsymbol{\mu}_Q$,σ_Q could depend on $\mathcal{S}$, we are not allowed to optimize σ_P in this way. Instead, one can have a set of predefined values for σ_P and pick the best one in that set. See Langford and Caruana (2002) for the discussion around this technique. Given fixed $a, b > 0$, let $T = \{c\exp((1-j)/k) | j \in \mathbb{N}\}$ be that predefined set of values for σ_P^2. If for any $j \in \mathbb{N}$, the above PAC-Bayesian bound holds for $\sigma_P^2 = c\exp((1-j)/k)$ with probability $1-\delta_j$ with $\delta_j = \frac{6\delta}{\pi^2 j^2}$, then by the union bound, all above bounds hold simultaneously with probability at least $1 - \sum_{j=1}^{\infty}\frac{6\delta}{\pi^2 j^2} = 1-\delta$.

Let $\sigma_Q = \rho$, $\boldsymbol{\mu}_Q = \boldsymbol{w}$ and $\boldsymbol{\mu}_P = \boldsymbol{0}$. Therefore, we have:

$$\sigma_Q^2 + \|\boldsymbol{\mu}_P - \boldsymbol{\mu}_Q\|_2^2/k \leq \rho^2 + \|\boldsymbol{w}\|_2^2/k \leq \rho^2(1+\exp(4n/k)) \tag{7}$$

---

[7] Despite the nonconvexity of the function here in σ_P^2, it has a unique stationary point which happens to be its minimizer.

We now consider the bound that corresponds to $j = \lfloor 1 - k\log((\rho^2 + \|\boldsymbol{w}\|_2^2/k)/c)\rfloor$. We can ensure that $j \in \mathbb{N}$ using inequality equation 7 and by setting $c = \rho^2(1 + \exp(4n/k))$. Furthermore, for $\sigma_P^2 = c\exp((1-j)/k)$, we have:

$$\rho^2 + \|\boldsymbol{w}\|_2^2/k \le \sigma_P^2 \le \exp(1/k)\left(\rho^2 + \|\boldsymbol{w}\|_2^2/k\right) \tag{8}$$

Therefore, using the above value for σ_P, KL divergence can be bounded as follows:

$$\begin{aligned}
KL(\mathscr{P}\|\mathscr{Q}) &= \frac{1}{2}\left[\frac{k\sigma_Q^2 + \|\boldsymbol{\mu}_P - \boldsymbol{\mu}_Q\|_2^2}{\sigma_P^2} - k + k\log\left(\frac{\sigma_P^2}{\sigma_Q^2}\right)\right] && (9)\\
&\le \frac{1}{2}\left[\frac{k(\rho^2 + \|\boldsymbol{w}\|_2^2/k)}{\rho^2 + \|\boldsymbol{w}\|_2^2/k} - k + k\log\left(\frac{\exp(1/k)\left(\rho^2 + \|\boldsymbol{w}\|_2^2/k\right)}{\rho^2}\right)\right] && (10)\\
&= \frac{1}{2}\left[k\log\left(\frac{\exp(1/k)\left(\rho^2 + \|\boldsymbol{w}\|_2^2/k\right)}{\rho^2}\right)\right] && (11)\\
&= \frac{1}{2}\left[1 + k\log\left(1 + \frac{\|\boldsymbol{w}\|_2^2}{k\sigma_Q^2}\right)\right] && (12)
\end{aligned}$$

Given the bound that corresponds to j holds with probability $1 - \delta_j$ for $\delta_j = \frac{6\delta}{\pi^2 j^2}$, the log term in the bound can be written as:

$$\begin{aligned}
\log\frac{n}{\delta_j} &= \log\frac{n}{\delta} + \log\frac{\pi^2 j^2}{6}\\
&\le \log\frac{n}{\delta} + \log\frac{\pi^2 k^2 \log^2(c/(\rho^2 + \|\boldsymbol{w}\|_2^2/k))}{6}\\
&\le \log\frac{n}{\delta} + \log\frac{\pi^2 k^2 \log^2(c/\rho^2)}{6}\\
&\le \log\frac{n}{\delta} + \log\frac{\pi^2 k^2 \log^2(1 + \exp(4n/k))}{6}\\
&\le \log\frac{n}{\delta} + \log\frac{\pi^2 k^2 (2 + 4n/k)^2}{6}\\
&\le \log\frac{n}{\delta} + 2\log\left(6n + 3k\right)
\end{aligned}$$

Therefore, the generalization bound can be written as follows:

$$\mathbb{E}_{\epsilon_i\sim\mathcal{N}(0,\sigma)}[L_{\mathscr{D}}(\boldsymbol{w}+\boldsymbol{\epsilon})] \le \mathbb{E}_{\epsilon_i\sim\mathcal{N}(0,\sigma)}[L_{\mathcal{S}}(\boldsymbol{w}+\boldsymbol{\epsilon})] + \sqrt{\frac{\frac{1}{4}k\log\left(1 + \frac{\|\boldsymbol{w}\|_2^2}{k\sigma^2}\right) + \frac{1}{4} + \log\frac{n}{\delta} + 2\log\left(6n + 3k\right)}{n-1}} \tag{13}$$

In the above bound, we have $\epsilon_i \sim \mathcal{N}(0,\sigma)$. Therefore, $\|\boldsymbol{\epsilon}\|_2^2$ has chi-square distribution and by Lemma 1 in Laurent and Massart (2000), we have that for any positive t:

$$P(\|\boldsymbol{\epsilon}\|_2^2 - k\sigma^2 \ge 2\sigma^2\sqrt{kt} + 2t\sigma^2) \le \exp(-t) \tag{14}$$

Therefore, with probability $1 - 1/\sqrt{n}$ we have that:

$$\|\boldsymbol{\epsilon}\|_2^2 \le \sigma^2(2\ln(\sqrt{n}) + k + 2\sqrt{k\ln(\sqrt{n})}) \le \sigma^2 k\left(1 + \sqrt{\frac{\ln(n)}{k}}\right)^2 \le \rho^2$$

Substituting the above value for σ back to the inequality and using theorem's assumption gives us following inequality:

$$\begin{aligned}
L_{\mathscr{D}}(\boldsymbol{w}) &\leq (1 - 1/\sqrt{n}) \max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} L_{\mathcal{S}}(\boldsymbol{w} + \boldsymbol{\epsilon}) + 1/\sqrt{n} \\
&+ \sqrt{\frac{\frac{1}{4} k \log\left(1 + \frac{\|\boldsymbol{w}\|_2^2}{\rho^2}\left(1 + \sqrt{\frac{\log(n)}{k}}\right)^2\right) + \log \frac{n}{\delta} + 2 \log(6n + 3k)}{n - 1}} \\
&\leq \max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} L_{\mathcal{S}}(\boldsymbol{w} + \boldsymbol{\epsilon}) + \\
&+ \sqrt{\frac{k \log\left(1 + \frac{\|\boldsymbol{w}\|_2^2}{\rho^2}\left(1 + \sqrt{\frac{\log(n)}{k}}\right)^2\right) + 4 \log \frac{n}{\delta} + 8 \log(6n + 3k)}{n - 1}}
\end{aligned}$$

$\square$

B Additional Experimental Results

B.1 SVHN and Fashion-MNIST

We report in table 5 results obtained on SVHN and Fashion-MNIST datasets. On these datasets, SAM allows a simple WideResnet to reach or push state of the art accuracy (0.99% error rate for SVHN, 3.59% for Fashion-MNIST).

For SVHN, we used all the available data (73257 digits for training set + 531131 additional samples). For auto-augment, we use the best policy found on this dataset as described in (Cubuk et al., 2018) plus cutout (Devries and Taylor, 2017). For Fashion-MNIST, the auto-augmentation line correspond to cutout only.

Table 5: Results on SVHN and Fashion-MNIST.

		SVHN		Fashion-MNIST	
Model	Augmentation	SAM	Baseline	SAM	Baseline
Wide-ResNet-28-10	Basic	$1.42_{\pm 0.02}$	$1.58_{\pm 0.03}$	$3.98_{\pm 0.05}$	$4.57_{\pm 0.07}$
Wide-ResNet-28-10	Auto augment	$\mathbf{0.99}_{\pm 0.01}$	$1.14_{\pm 0.04}$	$\mathbf{3.61}_{\pm 0.06}$	$3.86_{\pm 0.14}$
Shake-Shake (26 2x96d)	Basic	$1.44_{\pm 0.02}$	$1.58_{\pm 0.05}$	$3.97_{\pm 0.09}$	$4.37_{\pm 0.06}$
Shake-Shake (26 2x96d)	Auto augment	$1.07_{\pm 0.02}$	$1.03_{\pm 0.02}$	$\mathbf{3.59}_{\pm 0.01}$	$3.76_{\pm 0.07}$

C Experiment Details

C.1 Hyperparameters for Experiments

We report in table 6 the hyper-parameters selected by gridsearch for the CIFAR experiments, and the ones for SVHN and Fashion-MNIST in 7. For CIFAR10, CIFAR100, SVHN and Fashion-MNIST, we use a batch size of 256 and determine the learning rate and weight decay used to train each model via a joint grid search prior to applying SAM; all other model hyperparameter values are identical to those used in prior work.

For the Imagenet results (Resnet models), the models are trained for 100, 200 or 400 epochs on Google Cloud TPUv3 32 cores with a batch size of 4096. The initial learning rate is set to 1.0 and

		Cifar10		Cifar100	
Model	Augmentation	$\rho = 0.05$	SGD	rho=0.05	SGD
WRN-28-10 (200 epochs)	Basic	**2.7**	3.5	**16.5**	18.8
WRN-28-10 (200 epochs)	Cutout	**2.3**	2.6	**14.9**	16.9
WRN-28-10 (200 epochs)	AA	**2.1**	2.3	**13.6**	15.8
WRN-28-10 (1800 epochs)	Basic	**2.4**	3.5	**16.3**	19.1
WRN-28-10 (1800 epochs)	Cutout	**2.1**	2.7	**14.0**	17.4
WRN-28-10 (1800 epochs)	AA	**1.6**	2.2	**12.8**	16.1
WRN 26-2x6 ss	Basic	**2.4**	2.7	**15.1**	17.0
WRN 26-2x6 ss	Cutout	**2.0**	2.3	**14.2**	15.7
WRN 26-2x6 ss	AA	**1.7**	1.9	**12.8**	14.1
PyramidNet	Basic	**2.1**	4.0	**15.4**	19.7
PyramidNet	Cutout	**1.6**	2.5	**13.1**	16.4
PyramidNet	AA	**1.4**	1.9	**12.1**	14.6
PyramidNet+ShakeDrop	Basic	**2.1**	2.5	**13.3**	14.5
PyramidNet+ShakeDrop	Cutout	**1.6**	1.9	**11.3**	11.8
PyramidNet+ShakeDrop	AA	**1.4**	1.6	**10.3**	10.6

Table 10: Results for the Cifar10/Cifar100 experiments, using $\rho = 0.05$ for all models/datasets/augmentations

Dataset	Efficientnet-b7 + SAM (optimal)	Efficientnet-b7 + SAM ($\rho = 0.05$)	Efficientnet-b7
FGVC_Aircraft	6.80	7.06	8.15
Flowers	0.63	0.81	1.16
Oxford_IIIT_Pets	3.97	4.15	4.24
Stanford_Cars	5.18	5.57	5.94
cifar10	0.88	0.88	0.95
cifar100	7.44	7.56	7.68
Birdsnap	13.64	13.64	14.30
Food101	7.02	7.06	7.17

Table 11: Results for the the finetuning experiments, using $\rho = 0.05$ for all datasets.

reaches a lower test error, showing that the most efficient method is also the one providing the best generalization on this example. The reason for this is quite unclear and should be analyzed in follow up work.

C.5 Choice of p-norm

Our theorem is derived for $p = 2$, although generalizations can be considered for $p \in [1, +\infty]$ (the expression of the bound becoming way more involved). Empirically, we validate that the choice $p = 2$ is optimal by training a wide resnet on cifar10 with SAM for $p = \infty$ (in which case we have $\hat{\boldsymbol{\epsilon}}(\boldsymbol{w}) = \rho \,\text{sign}\left(\nabla_{\boldsymbol{w}} L_{\mathcal{S}}(\boldsymbol{w})\right)$) and $p = 2$ (giving $\hat{\boldsymbol{\epsilon}}(\boldsymbol{w}) = \frac{\rho}{||\nabla_{\boldsymbol{w}} L_{\mathcal{S}}(\boldsymbol{w})||_2^2}(\nabla_{\boldsymbol{w}} L_{\mathcal{S}}(\boldsymbol{w}))$). We do not consider the case $p = 1$ which would give us a perturbation on a single weight. As an additional ablation study, we also use random weight perturbations of a fixed Euclidean norm: $\hat{\boldsymbol{\epsilon}}(\boldsymbol{w}) = \frac{\rho}{||\boldsymbol{z}||_2^2}\boldsymbol{z}$ with $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_d)$. We report the test accuracy of the model in figure 6.

We observe that adversarial perturbations outperform random perturbations, and that using $p = 2$ yield superior accuracy on this example.

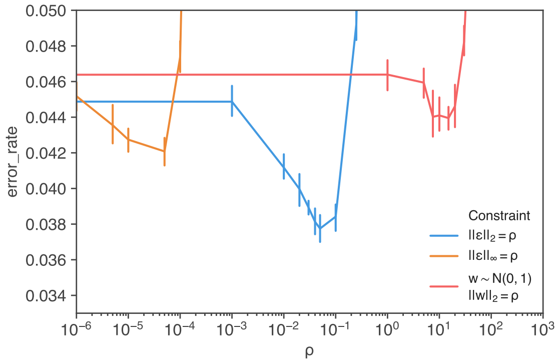


Figure 6: Test accuracy for a wide resnet trained on CIFAR10 with SAM, for different perturbation norms.

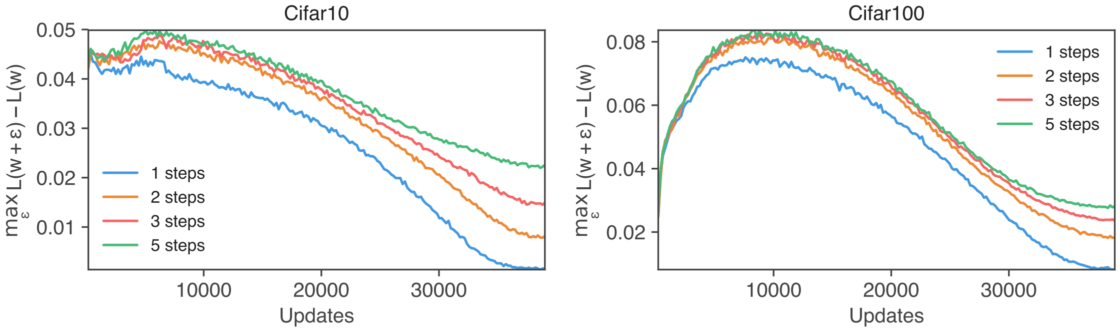


Figure 7: Evolution of $\max_\epsilon L(w+\epsilon) - L(w)$ vs. training step, for different numbers of inner projected gradient steps.

C.6 Several Iterations in the Inner Maximization

To empirically verify that the linearization of the inner problem is sensible, we trained a WideResnet on the CIFAR datasets using a variant of SAM that performs several iterations of projected gradient ascent to estimate $\max_\epsilon L(w+\epsilon)$. We report the evolution of $\max_\epsilon L(w+\epsilon) - L(w)$ during training (where L stands for the training error rate computed on the current batch) in Figure 7, along with the test accuracy and the estimated sharpness ($\max_\epsilon L(w+\epsilon) - L(w)$) at the end of training in Table 12; we report means and standard deviations across 20 runs.

For most of the training, one projected gradient step (as used in standard SAM) is sufficient to obtain a good approximation of the ϵ found with multiple inner maximization steps. We however observe that this approximation becomes weaker near convergence, where doing several iterations of projected gradient ascent yields a better ϵ (for example, on CIFAR-10, the maximum loss found on each batch is about 3% more when doing 5 steps of inner maximization, compared to when doing a single step). That said, as seen in Table 12, the test accuracy is not strongly affected by the number of inner maximization iterations, though on CIFAR-100 it does seem that several steps outperform a single step in a statistically significant way.

When Vision Transformers Outperform ResNets without Pretraining or Strong Data Augmentations

Xiangning Chen[1,2]* **Cho-Jui Hsieh**[2] **Boqing Gong**[1]

[1]Google Research [2]UCLA

Abstract

Vision Transformers (ViTs) and MLPs signal further efforts on replacing hand-wired features or inductive biases with general-purpose neural architectures. Existing works empower the models by massive data, such as large-scale pretraining and/or repeated strong data augmentations, and still report optimization-related problems (e.g., sensitivity to initialization and learning rate). Hence, this paper investigates ViTs and MLP-Mixers from the lens of loss geometry, intending to improve the models' data efficiency at training and generalization at inference. Visualization and Hessian reveal extremely sharp local minima of converged models. By promoting smoothness with a recently proposed sharpness-aware optimizer, we substantially improve the accuracy and robustness of ViTs and MLP-Mixers on various tasks spanning supervised, adversarial, contrastive, and transfer learning (e.g., +5.3% and +11.0% top-1 accuracy on ImageNet for ViT-B/16 and Mixer-B/16, respectively, with the simple Inception-style preprocessing). We show that the improved smoothness attributes to sparser active neurons in the first few layers. The resultant ViTs outperform ResNets of similar size and throughput when trained from scratch on ImageNet without large-scale pretraining or strong data augmentations. They also possess more perceptive attention maps.

1 Introduction

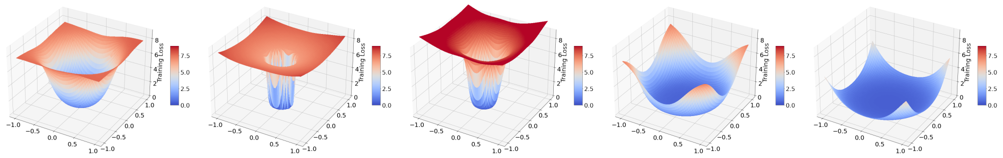
Transformers [56] have become the de-facto model of choice in natural language processing (NLP) [19, 44]. In computer vision, there has recently been a surge of interest in end-to-end Transformers [1, 2, 5, 20, 22, 38, 55] and MLPs [37, 40, 53, 54], prompting the efforts to replace hand-wired features or inductive biases with general-purpose neural architectures powered by data-driven training. We envision these efforts may lead to a unified knowledge base that produces versatile representations for different data modalities, simplifying the inference and deployment of deep learning models in various application scenarios.

Despite the appealing potential of moving toward general-purpose neural architectures, the lack of convolution-like inductive bias also challenges the training of vision Transformers (ViTs) and MLPs. When trained on ImageNet [18] with the conventional Inception-style data preprocessing [51], Transformers *"yield modest accuracies of a few percentage points below ResNets of comparable size"* [20]. To boost the performance, existing works resort to large-scale pre-training [1, 2, 20] and repeated strong data augmentations [55], resulting in excessive demands of data, computing, and sophisticated tuning of many hyper-parameters. For instance, Dosovitskiy et al. [20] pre-train ViTs using 304M labeled images, and Touvron et al. [55] repeatedly stack four strong image augmentations.

---

[*] Work done as a student researcher at Google.

Preprint. Under review.



(a) ResNet (b) ViT (c) Mixer (d) ViT-SAM (e) Mixer-SAM

Figure 1: Cross-entropy loss landscapes of ResNet-152, ViT-B/16, and Mixer-B/16. ViT and MLP-Mixer converge to sharper regions than ResNet when trained on ImageNet with the basic Inception-style preprocessing. SAM, a sharpness-aware optimizer, significantly smooths the landscapes.

Table 1: Number of parameters, NTK condition number κ, Hessian dominate eigenvalue λ_{max}, accuracy on ImageNet, and accuracy/robustness on ImageNet-C. ViT and MLP-Mixer suffer divergent κ and converge to sharp regions of big λ_{max}; SAM rescues that and leads to better generalization.

	ResNet-50	ResNet-152	ViT-B/16	ViT-B/16-SAM	Mixer-B/16	Mixer-B/16-SAM
#Params	25M	60M	87M		59M	
NTK κ	2801.6	2801.6	4205.3		14468.0	
Hessian λ_{max}	122.9	179.8	738.8	**20.9**	1644.4	**22.5**
ImageNet (%)	76.0	78.5	74.6	**79.9**	66.4	**77.4**
ImageNet-C (%)	44.6	50.0	46.6	**56.5**	33.8	**48.8**

We focus on ViTs and MLP-Mixers in this paper. We denote by “S” and “B” the small and base model sizes, respectively, and by an integer the image patch resolution. For instance, ViT-B/16 is the base ViT model taking as input a sequence of 16×16 patches. Appendices contain more details.

3 ViTs and MLP-Mixers Converge to Sharp Local Minima

The current training recipe of ViTs, MLP-Mixers, and related convolution-free architectures relies heavily on massive pretraining [1, 2, 20] or a bag of strong data augmentations [16, 17, 53, 55, 63, 65]. It highly demands data and computing, and leads to many hyper-parameters to tune. Existing works report that ViTs yield inferior accuracy to the ConvNets of similar size and throughput when trained from scratch on ImageNet without the combination of those advanced data augmentations, despite using various regularization techniques (e.g., large weight decay, Dropout [49], etc.). For instance, ViT-B/16 [20] gives rise to 74.6% top-1 accuracy on the ImageNet validation set (224 image resolution), compared with 78.5% of ResNet-152 [24]. Mixer-B/16 [53] performs even worse (66.4%). There also exists a large gap between ViTs and ResNets in robustness tests (e.g., against 19 corruptions in ImageNet-C [26]).

Moreover, Chen et al. [15] find that the gradients can spike and cause a sudden accuracy dip when training ViTs, and Touvron et al. [55] report the training is sensitive to initialization and hyperparameters. These all point to optimization problems. In this paper, we investigate the loss landscapes of ViTs and MLP-Mixers to understand them from the optimization perspective, intending to reduce their dependency on the large-scale pretraining or strong data augmentations.

ViTs and MLP-Mixers converge to extremely sharp local minima. It has been extensively studied that the convergence to a flat region whose curvature is small benefits the generalization of neural networks [10, 13, 29, 30, 33, 48, 64]. Following [36], we plot the loss landscapes at convergence when ResNets, ViTs, and MLP-Mixers are trained from scratch on ImageNet with the basic Inception-style preprocessing [51] (see Appendices for details). As shown in Figures 1(a) to 1(c), ViTs and MLP-Mixers converge to much sharper regions than ResNets. In Table 1, we further validate the results by computing the dominate Hessian eigenvalue λ_{max}, which is a mathematical evaluation of the landscape curvature. The λ_{max} values of ViT and MLP-Mixer are orders of magnitude larger than that of ResNet, and MLP-Mixer suffers the largest curvature among the three species (see Section 4.4 for a detailed analysis).

Small training errors. This convergence to sharp regions coincides with the training dynamics shown in Figure 2 (left). Although Mixer-B/16 has fewer parameters than ViT-B/16 (59M vs. 87M), it has a smaller training error but much worse test accuracy, implying that using the cross-token MLP to learn the interplay across image patches is more prone to overfitting than ViTs’ self-attention

Table 3: Dominant eigenvalue λ_{max} of the sub-diagonal Hessians for different network components, and norm of the model parameter w and the post-activation a_k of block k. Each ViT block consists of a MSA and a MLP, and MLP-Mixer alternates between a token MLP a channel MLP. Shallower layers have larger λ_{max}. SAM smooths every component.

Model	λ_{max} **of diagonal blocks of Hessian**							$\|w\|_2$	$\|a_1\|_2$	$\|a_6\|_2$	$\|a_{12}\|_2$
	Embedding	MSA/Token MLP	MLP/Channel MLP	Block1	Block6	Block12	Whole				
ViT-B/16	300.4	179.8	281.4	44.4	32.4	26.9	738.8	269.3	104.9	104.3	138.1
ViT-B/16-SAM	3.8	8.5	9.6	1.7	1.7	1.5	20.9	353.8	117.0	120.3	97.2
Mixer-B/16	1042.3	95.8	417.9	239.3	41.2	5.1	1644.4	197.6	96.7	135.1	74.9
Mixer-B/16-SAM	18.2	1.4	9.5	4.0	1.1	0.3	22.5	389.9	110.9	176.0	216.1

C [26]). ViT-B/16 achieves 79.9%, 26.4%, and 56.6% top-1 accuracy on ImageNet, ImageNet-R, and ImageNet-C, while the counterpart numbers for ResNet-152 are 79.3%, 25.7%, and 52.2%, respectively (see Table 2). The gaps between ViTs and ResNets are even wider for small architectures. ViT-S/16 outperforms a similarly sized ResNet-50 by 1.4% on ImageNet, and 6.5% on ImageNet-C. SAM also significantly improves MLP-Mixers' results.

4.4 Intrinsic changes after SAM

We take a deeper look into the models to understand how they intrinsically change to reduce the Hessian' eigenvalue λ_{max} and what the changes imply in addition to the enhanced generalization.

Smoother loss landscapes for every network component. In Table 3, we break down the Hessian of the whole architecture into small diagonal blocks of Hessians concerning each set of parameters, attempting to analyze what specific components cause the blowing up of λ_{max} in the models trained without SAM. We observe that shallower layers have larger Hessian eigenvalues λ_{max}, and the first linear embedding layer incurs the sharpest geometry. This agrees with the finding in [15] that spiking gradients happen early in the embedding layer. Additionally, the multi-head self-attention (MSA) in ViTs and the Token MLPs in MLP-Mixers, both of which mix information across spatial locations, have comparably lower λ_{max} than the other network components. SAM consistently reduces the λ_{max} of all network blocks.

We can gain insights into the above findings by the recursive formulation of Hessian matrices for MLPs [7]. Let h_k and a_k be the pre-activation and post-activation values for layer k, respectively. They satisfy $h_k = W_k a_{k-1}$ and $a_k = f_k(h_k)$, where W_k is the weight matrix and f_k is the activation function (GELU [27] in MLP-Mixers). Here we omit the bias term for simplicity. The diagonal block of Hessian matrix H_k with respect to W_k can be recursively calculated as:

$$H_k = (a_{k-1}a_{k-1}^T) \otimes \mathcal{H}_k, \quad \mathcal{H}_k = B_k W_{k+1}^T \mathcal{H}_{k+1} W_{k+1} B_k + D_k, \tag{3}$$

$$B_k = \text{diag}(f_k'(h_k)), \qquad D_k = \text{diag}(f_k''(h_k)\frac{\partial L}{\partial a_k}), \tag{4}$$

where $\otimes$ is the Kronecker product, $\mathcal{H}_k$ is the pre-activation Hessian for layer k, and L is the objective function. Therefore, the Hessian norm accumulates as the recursive formulation backpropagates to shallow layers, explaining why the first block has much larger λ_{max} than the last block in Table 3.

Greater weight norms. After applying SAM, we find that the norm of the post-activation value a_{k-1} and the weight W_{k+1} become even bigger (see Table 3), indicating that the commonly used weight decay may not effectively regularize ViTs and MLP-Mixers.

Sparser active neurons in MLP-Mixers. Given the recursive formulation (3) to (4), we identify another intrinsic measure of MLP-Mixers that contribute to the Hessian: the number of activated neurons. Indeed, B_k is determined by the activated neurons whose values are greater than zero, since the first-order derivative of GELU becomes much smaller when the input is negative. As a result, the number of active GELU neurons is directly connected to the Hessian norm. Figure 2 (right) shows the proportion of activated neurons for each block, counted using 10% of the ImageNet training set. We can see that SAM greatly reduces the proportion of activated neurons for the first few layers, pushing them to much sparser states. This result also suggests the potential redundancy of image patches.

ViTs' active neurons are highly sparse. Although Equations (3) and (4) only involve MLPs, we still observe a decrease of activated neurons in the first layer of ViTs (but not as significant as in MLP-Mixers). More interestingly, we find that the proportion of activated neurons in ViT is much

Table 4: Data augmentation, SAM, and their combination applied to different model architectures trained on ImageNet and its subsets.

Training Set	**#Images**	**ResNet-152**		**ViT-B/16**				**Mixer-B/16**			
		Vanilla	SAM	Vanilla	SAM	AUG	SAM + AUG	Vanilla	SAM	AUG	SAM + AUG
ImageNet	1,281,167	78.5	79.3	74.6	79.9	79.6	81.5	66.4	77.4	76.5	78.1
i1k (1/2)	640,583	74.2	75.6	64.9	75.4	73.1	75.8	53.9	71.0	70.4	73.1
i1k (1/4)	320,291	68.0	70.3	52.4	66.8	63.2	65.6	37.2	62.8	61.0	65.8
i1k (1/10)	128,116	54.6	57.1	32.8	46.1	38.5	45.7	21.0	43.5	43.0	51.0

training is proportional to the number of training images. When trained on i1k (1/4), it boosts ViT-B/16 and Mixer-B/16 by 14.4% and 25.6%, escalating their results to 66.8% and 62.8%, respectively. It also tells that ViT-B/16-SAM matches the performance of ResNet-152-SAM even with only 1/2 ImageNet training data.

5.2 SAM complements strong augmentation and is more robust to different training settings

Following the training pipeline in [53], we also study how SAM interplays with the strong data augmentations of mixup [65] (with probability 0.5) and RandAugment [17] (with two layers and magnitude 15). Table 4 shows the effects of the augmentations, SAM, and their combination on ImageNet and three subsets of training images. SAM benefits ViT-B/16 and Mixer-B/16 more than the strong data augmentations, especially when the training set is small. When the training set contains only 1/10 of ImageNet training images, SAM outperforms data augmentations by 7.6% for ViT-B/16. Besides, SAM and the strong data augmentations are complementary for most test cases.

The results demonstrate SAM is more robust to the change of training settings than the combination of strong augmentation methods. Figure 2 (middle) plots the training loss when using strong augmentations. It is very noisy, implying the difficulty of tuning their hyperparameters. In comparison, SAM is a principled optimizer that introduces only one additional hyperparameter ρ. Appendices report the experiments of tuning ρ, which does not complicate other hyper-parameters.

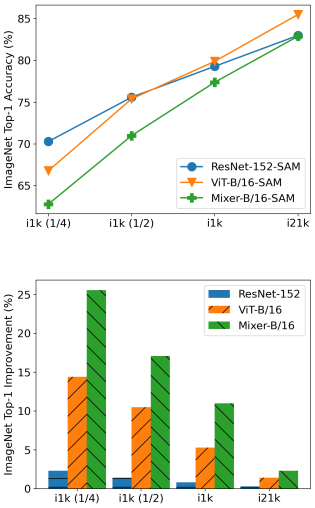


Figure 4: ImageNet validation accuracy (**Top**) and improvement (**Bottom**) brought by SAM on different training sets.

5.3 SAM complements contrastive learning

In addition to data augmentations and large-scale pretraining, another notable way of improving a neural model's generalization is (supervised) contrastive learning [9, 11, 25, 31]. We couple SAM with the supervised contrastive learning [31] for 350 epochs, followed by fine-tuning the classification head by 90 epochs for both ViT-S/16 and ViT-B/16. Please see the Appendices for more implementation details. Compared to the training procedure without SAM, we find considerable performance gain thanks to SAM's smoothing of the contrastive loss geometry, improving the ImageNet top-1 accuracy of ViT-S/16 from 77.0% to 78.1%, and ViT-B/16 from 77.4% to 80.0%.

5.4 When ViTs and MLP-Mixers meet both SAM and adversarial training

Interestingly, SAM and adversarial training are both minimax problems except that SAM's inner maximization is with respect to the network weights, while the latter concerns about the input for defending contrived attack [39, 57]. Moreover, similar to SAM, Shafahi et al. [46] suggest that adversarial training can flatten and smooth the loss landscape. In light of these connections, we study ViTs and MLP-Mixers under the adversarial training framework. To incorporate SAM, we formulate a three-level objective:

$$\min_{w} \max_{\epsilon \in \mathbb{S}_{sam}} \max_{\delta \in \mathbb{S}_{adv}} L_{train}(w+\epsilon, x+\delta, y), \tag{5}$$

where $\mathbb{S}_{sam}$ and $\mathbb{S}_{adv}$ denote the allowed perturbation norm balls for the model parameter w and input image x, respectively. Note that we can simultaneously obtain the gradients for computing ϵ

Table 5: Comparison under the adversarial training framework on ImageNet (numbers in the parentheses denote the improvement over the standard adversarial training without SAM). With similar model size and throughput, ViTs-SAM can still outperform ResNets-SAM for clean accuracy and adversarial robustness.

Model	#params	Throughput (img/sec/core)	ImageNet	Real	V2	PGD-10	ImageNet-R	ImageNet-C
ResNet								
ResNet-50-SAM	25M	2161	70.1 (-0.7)	77.9 (-0.3)	56.6 (-0.8)	54.1 (+0.9)	27.0 (+0.9)	42.7 (-0.1)
ResNet-101-SAM	44M	1334	73.6 (-0.4)	81.0 (+0.1)	60.4 (-0.6)	58.8 (+1.4)	29.5 (+0.6)	46.9 (+0.3)
ResNet-152-SAM	60M	935	75.1 (-0.4)	82.3 (+0.2)	62.2 (-0.4)	61.0 (+1.8)	30.8 (+1.4)	49.1 (+0.6)
Vision Transformer								
ViT-S/16-SAM	22M	2043	73.2 (+1.2)	80.7 (+1.7)	60.2 (+1.4)	58.0 (+5.2)	28.4 (+2.4)	47.5 (+1.6)
ViT-B/32-SAM	88M	2805	69.9 (+3.0)	76.9 (+3.4)	55.7 (+2.5)	54.0 (+6.4)	26.0 (+3.0)	46.4 (+3.0)
ViT-B/16-SAM	87M	863	76.7 (+3.9)	82.9 (+4.1)	63.6 (+4.3)	62.0 (+7.7)	30.0 (+4.9)	51.4 (+5.0)
MLP-Mixer								
Mixer-S/16-SAM	18M	4005	67.1 (+2.2)	74.5 (+2.3)	52.8 (+2.5)	50.1 (+4.1)	22.9 (+2.6)	37.9 (+2.5)
Mixer-B/32-SAM	60M	4209	69.3 (+9.1)	76.4 (+10.2)	54.7 (+9.4)	54.5 (+13.9)	26.3 (+8.0)	43.7 (+8.8)
Mixer-B/16-SAM	59M	1390	73.9 (+11.1)	80.8 (+11.8)	60.2 (+11.9)	59.8 (+17.3)	29.0 (+10.5)	45.9 (+12.5)

Table 6: Accuracy on downstream tasks of the models pretrained on ImageNet. SAM improves ViTs and MLP-Mixers' transferabilities to the tasks. ViTs transfer better than ResNets of similar sizes.

%	ResNet-50-SAM	ResNet-152-SAM	ViT-S/16	ViT-S/16-SAM	ViT-B/16	ViT-B/16-SAM	Mixer-S/16	Mixer-S/16-SAM	Mixer-B/16	Mixer-B/16-SAM
CIFAR-10	97.4	98.2	97.6	98.2	98.1	98.6	94.1	96.1	95.4	97.8
CIFAR-100	85.2	87.8	85.7	87.6	87.6	89.1	77.9	82.4	80.0	86.4
Flowers	90.0	91.1	86.4	91.5	88.5	91.8	83.3	87.9	82.8	90.0
Pets	91.6	93.3	90.4	92.9	91.9	93.1	86.1	88.7	86.1	92.5
Average	91.1	92.6	90.0	92.6	91.5	93.2	85.4	88.8	86.1	91.7

and δ by backpropagation only once. To lower the training cost, we use fast adversarial training [57] with the l_∞ norm for δ, and the maximum per-pixel change is set as 2/255.

Table 5 evaluates the models' clean accuracy, real-world robustness, and adversarial robustness (under 10-step PGD attack [39]). It is clear that the landscape smoothing significantly improves the convolution-free architectures for both clean and adversarial accuracy. However, we observe a slight accuracy decrease on clean images for ResNets despite gain for robustness. Similar to our previous observations, *ViTs surpass similar-size ResNets when adversarially trained on ImageNet with the basic Inception-style preprocessing for both clean accuracy and adversarial robustness.*

5.5 ViTs and MLP-Mixers with smoother loss geometry transfer better to downstream tasks

Finally, we study the role of smoothed loss geometry in transfer learning. We select four datasets to test ViTs and MLP-Mixers' transferabilities: CIFAR-10/100 [35], Oxford-IIIT Pets [43], and Oxford Flowers-102 [42]. We fine-tune all the models with image resolution 224 using vanilla SGD. For comparison, we also include ResNet-50-SAM and ResNet-152-SAM in the experiments. Table 6 summarizes the results, which confirm that the enhanced models also perform better after fine-tuning and that MLP-Mixers gain the most from the sharpness-aware optimization.

6 Conclusion and Discussion

This paper presents a detailed analysis of the convolution-free ViTs and MLP-Mixers from the lens of the loss landscape geometry, intending to reduce the models' dependency on massive pretraining and/or strong data augmentations. We arrive at the sharpness-aware minimizer (SAM) after observing sharp local minima of the converged models. By explicitly regularizing the loss geometry through SAM, the models enjoy much flatter loss landscapes and improved generalization regarding accuracy and robustness. The resultant ViT models outperform ResNets of comparable size and throughput when learned with no pretraining or strong augmentations. Further investigation reveals that the smoothed loss landscapes attribute to much sparser activated neurons in the first few layers. Moreover, ViTs after SAM offer perceptive attention maps.

Future work will focus on the following limitations of the work. The update to ϵ is approximated up to the first order by one step only, and it may be improved by considering multiple steps or higher

SURROGATE GAP MINIMIZATION IMPROVES SHARPNESS-AWARE TRAINING

Juntang Zhuang[1] *
`j.zhuang@yale.edu`

Boqing Gong[2], **Liangzhe Yuan**[2], **Yin Cui**[2], **Hartwig Adam**[2]
`{bgong, lzyuan, yincui, hadam}@google.com`

Nicha C. Dvornek[1], **Sekhar Tatikonda**[1], **James S. Duncan**[1]
`{nicha.dvornek, sekhar.tatikonda, james.duncan}@yale.edu`

Ting Liu[2]
`liuti@google.com`
[1] Yale University, [2] Google Research

ABSTRACT

The recently proposed Sharpness-Aware Minimization (SAM) improves generalization by minimizing a *perturbed loss* defined as the maximum loss within a neighborhood in the parameter space. However, we show that both sharp and flat minima can have a low perturbed loss, implying that SAM does not always prefer flat minima. Instead, we define a *surrogate gap*, a measure equivalent to the dominant eigenvalue of Hessian at a local minimum when the radius of neighborhood (to derive the perturbed loss) is small. The surrogate gap is easy to compute and feasible for direct minimization during training. Based on the above observations, we propose Surrogate **G**ap Guided **S**harpness-**A**ware **M**inimization (GSAM), a novel improvement over SAM with negligible computation overhead. Conceptually, GSAM consists of two steps: 1) a gradient descent like SAM to minimize the perturbed loss, and 2) an *ascent* step in the *orthogonal* direction (after gradient decomposition) to minimize the surrogate gap and yet not affect the perturbed loss. GSAM seeks a region with both small loss (by step 1) and low sharpness (by step 2), giving rise to a model with high generalization capabilities. Theoretically, we show the convergence of GSAM and provably better generalization than SAM. Empirically, GSAM consistently improves generalization (e.g., +3.2% over SAM and +5.4% over AdamW on ImageNet top-1 accuracy for ViT-B/32). Code is released at `https://sites.google.com/view/gsam-iclr22/home`.

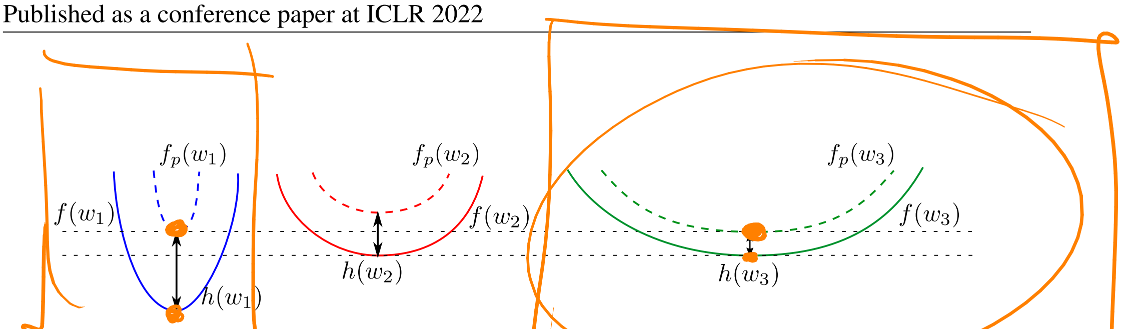
1 INTRODUCTION

Modern neural networks are typically highly over-parameterized and easy to overfit to training data, yet the generalization performances on unseen data (test set) often suffer a gap from the training performance (Zhang et al., 2017a). Many studies try to understand the generalization of machine learning models, including the Bayesian perspective (McAllester, 1999; Neyshabur et al., 2017), the information perspective (Liang et al., 2019), the loss surface geometry perspective (Hochreiter & Schmidhuber, 1995; Jiang et al., 2019) and the kernel perspective (Jacot et al., 2018; Wei et al., 2019). Besides analyzing the properties of a model after training, some works study the influence of training and the optimization process, such as the implicit regularization of stochastic gradient descent (SGD) (Bottou, 2010; Zhou et al., 2020), the learning rate's regularization effect (Li et al., 2019), and the influence of the batch size (Keskar et al., 2016).

These studies have led to various modifications to the training process to improve generalization. Keskar & Socher (2017) proposed to use Adam in early training phases for fast convergence and then switch to SGD in late phases for better generalization. Izmailov et al. (2018) proposed to average weights to achieve a wider local minimum, which is expected to generalize better than sharp minima. A similar idea was later used in Lookahead (Zhang et al., 2019). Entropy-SGD (Chaudhari

---

[*] Work was done during an internship at Google



$sharpness(w_1) > sharpness(w_2) > sharpness(w_3)$, $h(w_1) > h(w_2) > h(w_3)$,
$f_p(w_1) = f_p(w_3) < f_p(w_2)$, hence h agrees with sharpness while f_p might not.

Figure 1: Consider original loss f (solid line), perturbed loss $f_p \triangleq \max_{||\delta|| \leq \rho} f(w+\delta)$ (dashed line), and surrogate gap $h(w) \triangleq f_p(w) - f(w)$. Intuitively, f_p is approximately a max-pooled version of f with a pooling kernel of width 2ρ, and SAM minimizes f_p. From left to right are the local minima centered at w_1, w_2, w_3, and the valleys become flatter. Since $f_p(w_1) = f_p(w_3) < f_p(w_2)$, SAM prefers w_1 and w_3 to w_2. *However, a low f_p could appear in both sharp (w_1) and flat (w_3) minima, so f_p might disagree with sharpness.* On the contrary, a smaller surrogate gap h indicates a flatter loss surface (Lemma 3.3). From w_1 to w_3, the loss surface is flatter, and h is smaller.

- $\nabla f(w_t) = \nabla f_{||}(w_t) + \nabla f_{\perp}(w_t)$: Decompose $\nabla f(w_t)$ into parallel component $\nabla f_{||}(w_t)$ and vertical component $\nabla f_{\perp}(w_t)$ by projection $\nabla f(w_t)$ onto $\nabla f_p(w_t)$.

2.2 SHARPNESS-AWARE MINIMIZATION

Conventional optimization of neural networks typically minimizes the training loss $f(w)$ by gradient descent w.r.t. $\nabla f(w)$ and searches for a single point w with a low loss. However, this vanilla training often falls into a sharp valley of the loss surface, resulting in inferior generalization performance (Chaudhari et al., 2019). Instead of searching for a single point solution, SAM seeks a region with low losses so that small perturbation to the model weights does not cause significant performance degradation. SAM formulates the problem as:

$$\min_w f_p(w) \text{ where } f_p(w) \triangleq \max_{||\delta|| \leq \rho} f(w+\delta) \tag{1}$$

where ρ is a predefined constant controlling the radius of a neighborhood. This perturbed loss f_p induced by $f(w)$ is the maximum loss within the neighborhood. When the perturbed loss is minimized, the neighborhood corresponds to low losses (below the perturbed loss). For a small ρ, using Taylor expansion around w, the inner maximization in Eq. 1 turns into a linear constrained optimization with solution

$$\arg\max_{||\delta|| \leq \rho} f(w+\delta) = \arg\max_{||\delta|| \leq \rho} f(w) + \delta^\top \nabla f(w) + O(\rho^2) = \rho \frac{\nabla f(w)}{||\nabla f(w)||} \tag{2}$$

As a result, the optimization problem of SAM reduces to

$$\min_w f_p(w) \approx \min_w f(w^{adv}) \text{ where } w^{adv} \triangleq w + \rho \frac{\nabla f(w)}{||\nabla f(w)|| + \epsilon} \tag{3}$$

where ϵ is a scalar (default: 1e-12) to avoid division by 0, and w^{adv} is the "perturbed weight" with the highest loss within the neighborhood. Equivalently, SAM seeks a solution on the surface of the perturbed loss $f_p(w)$ rather than the original loss $f(w)$ (Foret et al., 2020).

3 THE SURROGATE GAP MEASURES THE SHARPNESS AT A LOCAL MINIMUM

3.1 THE PERTURBED LOSS IS NOT ALWAYS SHARPNESS-AWARE

Despite that SAM searches for a region of low losses, we show that a solution by SAM is not guaranteed to be flat. Throughout this paper we measure the sharpness at a local minimum of loss $f(w)$ by the dominant eigenvalue σ_{max} (eigenvalue with the largest absolute value) of Hessian. For simplicity, we do not consider the influence of reparameterization on the geometry of loss surfaces, which is thoroughly discussed in (Laurent & Massart, 2000; Kwon et al., 2021).

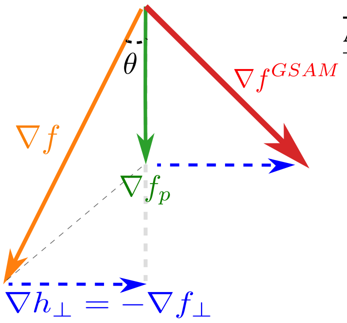


Figure 2: ∇f is decomposed into parallel and vertical ($\nabla f_\perp$) components by projection onto ∇f_p. $\nabla f^{GSAM} = \nabla f_p - \alpha \nabla f_\perp$

Algorithm 1 GSAM Algorithm

```
For $t = 1$ to $T$
  0) $\rho_t$ schedule: $\rho_t = \rho_{min} + \frac{(\rho_{max}-\rho_{min})(lr-lr_{min})}{lr_{max}-lr_{min}}$
  1a) $\Delta w_t = \rho_t \frac{\nabla f^{(t)}}{||\nabla f^{(t)}||+\epsilon}$
  1b) $w_t^{adv} = w_t + \Delta w_t$
  2) Get $\nabla f_p^{(t)}$ by back-propagation at $w_t^{adv}$.
  3) $\nabla f^{(t)} = \nabla f_{||}^{(t)} + \nabla f_\perp^{(t)}$ Decompose $\nabla f^{(t)}$ into components that are parallel and orthogonal to $\nabla f_p^{(t)}$.
  4) Update weights:
      Vanilla  $w_{t+1} = w_t - \eta_t \nabla f^{(t)}$
      SAM      $w_{t+1} = w_t - \eta_t \nabla f_p^{(t)}$
      GSAM     $w_{t+1} = w_t - \eta_t (\nabla f_p^{(t)} - \alpha \nabla f_\perp^{(t)})$
```

Lemma 3.1. *For some fixed ρ, consider two local minima w_1 and w_2, $f_p(w_1) \leq f_p(w_2) \not\Rightarrow \sigma_{max}(w_1) \leq \sigma_{max}(w_2)$, where σ_{max} is the dominant eigenvalue of the Hessian.*

We leave the proof to Appendix. Fig. 1 illustrates Lemma 3.1 with an example. Consider three local minima denoted as w_1 to w_3, and suppose the corresponding loss surfaces are flatter from w_1 to w_3. For some fixed ρ, we plot the perturbed loss f_p and surrogate gap $h \triangleq f_p - f$ around each solution. Comparing w_2 with w_3: Suppose their vanilla losses are equal, $f(w_2) = f(w_3)$, then $f_p(w_2) > f_p(w_3)$ because the loss surface is flatter around w_3, implying that SAM will prefer w_3 to w_2. Comparing w_1 and w_2: $f_p(w_1) < f_p(w_2)$, and SAM will favor w_1 over w_2 because it only cares about the perturbed loss f_p, even though the loss surface is sharper around w_1 than w_2.

3.2 THE SURROGATE GAP AGREES WITH SHARPNESS

We introduce the surrogate gap that agrees with sharpness, defined as:

$$h(w) \triangleq \max_{||\delta|| \leq \rho} f(w+\delta) - f(w) \approx f(w^{adv}) - f(w) \tag{4}$$

Intuitively, the surrogate gap represents the difference between the maximum loss within the neighborhood and the loss at the center point. The surrogate gap has the following properties.

Lemma 3.2. *Suppose the perturbation amplitude ρ is sufficiently small, then the approximation to the surrogate gap in Eq. 4 is always non-negative, $h(w) \approx f(w^{adv}) - f(w) \geq 0, \forall w$.*

Lemma 3.3. *For a local minimum w^*, consider the dominate eigenvalue σ_{max} of the Hessian of loss f as a measure of sharpness. Considering the neighborhood centered at w^* with a small radius ρ, the surrogate gap $h(w^*)$ is an equivalent measure of the sharpness: $\sigma_{max} \approx 2h(w^*)/\rho^2$.*

The proof is in Appendix. Lemma 3.2 tells that the surrogate gap is non-negative, and Lemma 3.3 shows that the loss surface is flatter as h gets closer to 0. The two lemmas together indicate that we can find a region with a flat loss surface by minimizing the surrogate gap $h(w)$.

4 SURROGATE GAP GUIDED SHARPNESS-AWARE MINIMIZATION

4.1 GENERAL IDEA: SIMULTANEOUSLY MINIMIZE THE PERTURBED LOSS AND SURROGATE GAP

Inspired by the analysis in Section 3, we propose Surrogate **G**ap Guided **S**harpness-**A**ware **M**inimzation (GSAM) to simultaneously minimize two objectives, the perturbed loss f_p and the surrogate gap h:

$$\min_w \left(f_p(w), h(w) \right) \tag{5}$$

Intuitively, by minimizng f_p we search for a region with a low perturbed loss similar to SAM, and by minimizing h we search for a local minimum with a flat surface. A low perturbed loss implies low training losses within the neighborhood, and a flat loss surface reduces the generalization gap between training and test performances (Chaudhari et al., 2019). When both are minimized, the solution gives rise to high accuracy and good generalization.

Potential caveat in optimization It is tempting and yet sub-optimal to combine the objectives in Eq. 5 to arrive at $\min_w f_p(w)+\lambda h(w)$, where λ is some positive scalar. One caveat when solving this weighted combination is the potential conflict between the gradients of the two terms, i.e., $\nabla f_p(w)$ and $\nabla h(w)$. We illustrate this conflict by Fig. 2, where $\nabla h(w) = \nabla f_p(w) - \nabla f(w)$ (the grey dashed arrow) has a negative inner product with $\nabla f_p(w)$ and $\nabla f(w)$. Hence, the gradient descent for the surrogate gap could potentially increase the loss f_p, harming the model's performance. We empirically validate this argument in Sec. 6.4.

4.2 GRADIENT DECOMPOSITION AND ASCENT FOR THE MULTI-OBJECTIVE OPTIMIZATION

Our primary goal is to minimize f_p because otherwise a flat solution of high loss is meaningless, and the minimization of h should not increase f_p. We propose to decompose $\nabla f(w_t)$ and ∇h into components that are parallel and orthogonal to $\nabla f_p(w_t)$, respectively (see Fig. 2):

$$\begin{aligned}\nabla f(w_t) &= \nabla f_{\parallel}(w_t) + \nabla f_{\perp}(w_t)\\ \nabla h(w_t) &= \nabla h_{\parallel}(w_t) + \nabla h_{\perp}(w_t)\\ \nabla h_{\perp}(w_t) &= -\nabla f_{\perp}(w_t)\end{aligned} \tag{6}$$

The key is that updating in the direction of $\nabla h_{\perp}(w_t)$ does *not* change the value of the perturbed loss $f_p(w_t)$ because $\nabla h_{\perp} \perp \nabla f_p$ by construction. Therefore, we propose to perform a **descent step in the $\nabla h_{\perp}(w_t)$ direction**, which is equivalent to an **ascent step in the $\nabla f_{\perp}(w_t)$ direction** (because $\nabla h_{\perp} = -\nabla f_{\perp}$ by the definition of h), and achieve two goals simultaneously — it keeps the value of $f_p(w_t)$ intact and meanwhile decreases the surrogate gap $h(w_t) = f_p(w_t) - f(w_t)$ (by increasing $f(w_t)$ and not affect $f_p(w_t)$).

The full GSAM Algorithm is shown in Algo. 1 and Fig. 2, where $g^{(t)}, g_p^{(t)}$ are noisy observations of $\nabla f(w_t)$ and $\nabla f_p(w_t)$, respectively, and $g_{\parallel}^{(t)}, g_{\perp}^{(t)}$ are noisy observations of $\nabla f_{\parallel}(w_t)$ and $\nabla f_{\perp}(w_t)$, respectively, by projecting $g^{(t)}$ onto $g_p^{(t)}$. We introduce a constant α to scale the stepsize of the ascent step. Steps 1) to 2) are the same as SAM: At current point w_t, step 1) takes a gradient ascent to w_t^{adv} followed by step 2) evaluating the gradient $g_p^{(t)}$ at w_t^{adv}. Step 3) projects $g^{(t)}$ onto $g_p^{(t)}$, which requires negligible computation compared to the forward and backward passes. In step 4), $-\eta_t g_p^{(t)}$ is the same as in SAM and minimizes the perturbed loss $f_p(w_t)$ with gradient descent, and $\alpha \eta_t g_{\perp}^{(t)}$ performs an *ascent* step in the orthogonal direction of $g_p^{(t)}$ to minimize the surrogate gap $h(w_t)$ (equivalently increase $f(w_t)$ and keep $f_p(w_t)$ intact). In coding, GSAM feeds the "surrogate gradient" $\nabla f_t^{GSAM} \triangleq g_p^{(t)} - \alpha g_{\perp}^{(t)}$ to first-order gradient optimizers such as SGD and Adam.

The ascent step along $g_{\perp}^{(t)}$ does not harm convergence SAM demonstrates that minimizing f_p makes the network generalize better than minimizing f. Even though our ascent step along $g_{\perp}^{(t)}$ increases $f(w)$, it does not affect $f_p(w)$, so GSAM still decreases the perturbed loss f_p in a way similar to SAM. In Thm. 5.1, we formally prove the convergence of GSAM. In Sec. 6 and Appendix C, we empirically validate that the loss decreases and accuracy increases with training.

Illustration with a toy example We demonstrate different algorithms by a numerical toy example shown in Fig. 3. The trajectory of GSAM is closer to the ridge and tends to find a flat minimum. Intuitively, since the loss surface is smoother along the ridge than in sharp local minima, the surrogate gap $h(w)$ is small near the ridge, and the ascent step in GSAM minimizes h to pushes the trajectory closer to the ridge. More concretely, $\nabla f(w_t)$ points to a sharp local solution and deviates from the ridge; in contrast, w_t^{adv} is closer to the ridge and $\nabla f(w_t^{adv})$ is closer to the ridge descent direction than $\nabla f(w_t)$. Note that ∇f_t^{GSAM} and $\nabla f(w_t)$ always lie at different sides of $\nabla f_p(w_t)$ by construction (see Fig. 2), hence ∇f_t^{GSAM} pushes the trajectory closer to the ridge than $\nabla f_p(w_t)$ does. The trajectory of GSAM is like descent along the ridge and tends to find flat minima.

Table 1: Top-1 Accuracy (%) on ImageNet datasets for ResNets, ViTs and MLP-Mixers trained with Vanilla SGD or AdamW, SAM, and GSAM optimizers.

Model	Training	ImageNet-v1	ImageNet-Real	ImageNet-V2	ImageNet-R	ImageNet-C
ResNet						
ResNet50	Vanilla (SGD)	76.0	82.4	63.6	22.2	44.6
	SAM	76.9	83.3	64.4	**23.8**	46.5
	GSAM	**77.2**	**83.9**	**64.6**	23.6	**47.6**
ResNet101	Vanilla (SGD)	77.8	83.9	65.3	24.4	48.5
	SAM	78.6	84.8	66.7	25.9	51.3
	GSAM	**78.9**	**85.2**	**67.3**	**26.3**	**51.8**
ResNet152	Vanilla (SGD)	78.5	84.2	66.3	25.3	50.0
	SAM	79.3	84.9	67.3	25.7	52.2
	GSAM	**80.0**	**85.9**	**68.6**	**27.3**	**54.1**
Vision Transformer						
ViT-S/32	Vanilla (AdamW)	68.4	75.2	54.3	19.0	43.3
	SAM	70.5	77.5	56.9	21.4	46.2
	GSAM	**73.8**	**80.4**	**60.4**	**22.5**	**48.2**
ViT-S/16	Vanilla (AdamW)	74.4	80.4	61.7	20.0	46.5
	SAM	78.1	84.1	65.6	24.7	53.0
	GSAM	**79.5**	**85.3**	**67.3**	**25.3**	**53.3**
ViT-B/32	Vanilla (AdamW)	71.4	77.5	57.5	23.4	44.0
	SAM	73.6	80.3	60.0	24.0	50.7
	GSAM	**76.8**	**82.7**	**63.0**	**25.1**	**51.7**
ViT-B/16	Vanilla (AdamW)	74.6	79.8	61.3	20.1	46.6
	SAM	79.9	85.2	67.5	26.4	**56.5**
	GSAM	**81.0**	**86.5**	**69.2**	**27.1**	55.7
MLP-Mixer						
Mixer-S/32	Vanilla (AdamW)	63.9	70.3	49.5	16.9	35.2
	SAM	66.7	73.8	52.4	18.6	39.3
	GSAM	**68.6**	**75.8**	**55.0**	**22.6**	**44.6**
Mixer-S/16	Vanilla (AdamW)	68.8	75.1	54.8	15.9	35.6
	SAM	72.9	79.8	58.9	20.1	42.0
	GSAM	**75.0**	**81.7**	**61.9**	**23.7**	**48.5**
Mixer-S/8	Vanilla (AdamW)	70.2	76.2	56.1	15.4	34.6
	SAM	75.9	82.5	62.3	20.5	42.4
	GSAM	**76.8**	**83.4**	**64.0**	**24.6**	**47.8**
Mixer-B/32	Vanilla (AdamW)	62.5	68.1	47.6	14.6	33.8
	SAM	72.4	79.0	58.0	22.8	46.2
	GSAM	**73.6**	**80.2**	**59.9**	**27.9**	**52.1**
Mixer-B/16	Vanilla (AdamW)	66.4	72.1	50.8	14.5	33.8
	SAM	77.4	83.5	63.9	24.7	48.8
	GSAM	**77.8**	**84.0**	**64.9**	**28.3**	**54.4**

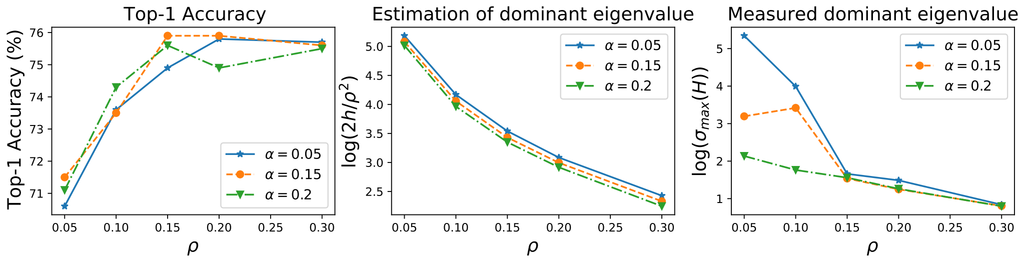


Figure 4: Influence of ρ (set as constant for ease of comparison, other experiments use decayed ρ_t schedule) and α on the training of ViT-B/32. **Left**: Top-1 accuracy on ImageNet. **Middle**: Estimation of the dominant eigenvalues from the surrogate gap, $\sigma_{max} \approx 2h/\rho^2$. **Right**: Dominant eigenvalues of the Hessian calculated via the power iteration. Middle and right figures match in the trend of curves, validating that the surrogate gap can be viewed as a proxy of the dominant eigenvalue of Hessian.

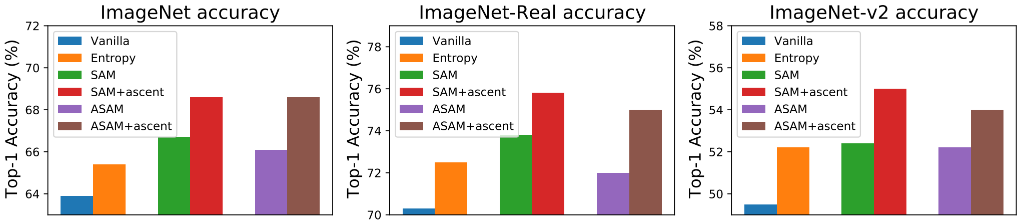


Figure 5: Top-1 accuracy of Mixer-S/32 trained with different methods. "+ascent" represents applying the ascent step in Algo. 1 to an optimizer. Note that our GSAM is described as SAM+ascent(**=GSAM**) for consistency.

Table 2: Results (%) of GSAM and $\min(f_p + \lambda h)$ on ViT-B/32

Dataset	$\min(f_p + \lambda h)$	GSAM
ImageNet	75.4	**76.8**
ImageNet-Real	81.1	**82.7**
ImageNet-v2	60.9	**63.0**
ImageNet-R	23.9	**25.1**

Table 3: Transfer learning results (top-1 accuracy, %)

	ViT-B/16			ViT-S/16		
	Vanilla	SAM	GSAM	Vanilla	SAM	GSAM
Cifar10	98.1	98.6	**98.8**	97.6	98.2	**98.4**
Cifar100	87.6	89.1	**89.7**	85.7	87.6	**88.1**
Flowers	88.5	**91.8**	91.2	86.4	**91.5**	90.3
Pets	91.9	93.1	**94.4**	90.4	92.9	**93.5**
mean	91.5	93.2	**93.5**	90.0	**92.6**	**92.6**

eigenvalues estimated by the surrogate gap, $\sigma_{max} \approx 2h/\rho^2$ (Lemma 3.3). In the right subfigure, we directly calculate the dominant eigenvalues using the power-iteration (Mises & Pollaczek-Geiringer, 1929). The estimated dominant eigenvalues (middle) match the real eigenvalues σ_{max} (right) in terms of the trend that σ_{max} decreases with α and ρ. Note that the surrogate gap h is derived over the whole training set, while the measured eigenvalues are over a subset to save computation. These results show that the ascent step in GSAM minimizes the dominant eigenvalue by minimizing the surrogate loss, validating Thm 5.3.

6.3 COMPARISON WITH METHODS IN THE LITERATURE

Section 6.1 compares GSAM to SAM and vanilla training. In this subsection, we further compare GSAM against Entropy-SGD (Chaudhari et al., 2019) and Adaptive-SAM (ASAM) (Kwon et al., 2021), which are designed to improve generalization. Note that Entropy-SGD uses SGD in the inner Langevin iteration and can be combined with other base optimizers such as AdamW as the outer loop. For Entropy-SGD, we find the hyper-parameter "scope" from 0.0 and 0.9, and search for the inner-loop iteration number between 1 and 14. For ASAM, we search for ρ between 1 and 7 (10× larger than in SAM) as recommended by the ASAM authors. Note that the only difference between ASAM and SAM is the derivation of the perturbation, so both can be combined with the proposed ascent step. As shown in Fig. 5, the proposed ascent step increases test accuracy when combined with both SAM and ASAM and outperforms Entropy-SGD and vanilla training.

6.4 ADDITIONAL STUDIES

GSAM outperforms a weighted combination of the perturbed loss and surrogate gap With an example in Fig. 2, we demonstrate that directly minimizing $f_p(w) + \lambda h(w)$ as discussed in Sec. 4.1 is sub-optimal because $\nabla h(w)$ could conflict with $\nabla f_p(w)$ and $\nabla f(w)$. We empirically validate this argument on ViT-B/32. We search for λ between 0.0 and 0.5 with a step 0.1 and search for ρ in the same grid as SAM and GSAM. We report the best accuracy of each method. Top-1 accuracy in Table 2 show the superior performance of GSAM, validating our analysis.

$\min(f_p, h)$ vs. $\min(f, h)$ GSAM solves $\min(f_p, h)$ by descent in ∇f_p, decomposing ∇f onto ∇f_p, and an ascent step in the orthogonal direction to increase f while keep f_p intact. Alternatively, we can also optimize $\min(f, h)$ by descent in ∇f, decomposing ∇f_p onto ∇f, and a descent step in the orthogonal direction to decrease f_p while keep f intact. The two GSAM variations perform similarly (see Fig. 6, right). We choose $\min(f_p, h)$ mainly to make the minimal change to SAM.

GSAM benefits transfer learning Using weights trained on ImageNet-1k, we finetune models with SGD on downstream tasks including the CIFAR10/CIFAR100 (Krizhevsky et al., 2009), Oxford-

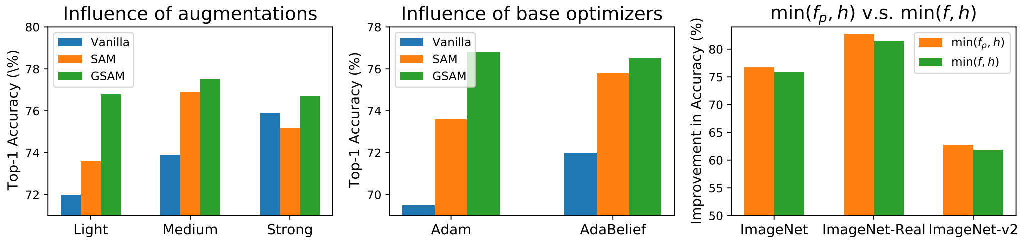


Figure 6: Top-1 accuracy of ViT-B/32 for the additional studies (Section 6.4). **Left**: from left to right are performances under different data augmentations (details in Appendix B.3) , where the vanilla method is trained for $2\times$ the epochs. **Middle**: performance with different base optimizers. **Right**: Comparison between $\min(f_p, h)$ and $\min(f, h)$.

flowers (Nilsback & Zisserman, 2008) and Oxford-IITPets (Parkhi et al., 2012). Results in Table 3 shows that GSAM leads to better transfer performance than vanilla training and SAM.

GSAM remains effective under various data augmentations We plot the top-1 accuracy of a ViT-B/32 model under various Mixup (Zhang et al., 2017b) augmentations in Fig. 6 (left subfigure). Under different augmentations, GSAM consistently outperforms SAM and vanilla training.

GSAM is compatible with different base optimizers GSAM is generic and applicable to various base optimizers. We compare vanilla training, SAM and GSAM using AdamW (Loshchilov & Hutter, 2017) and AdaBelief (Zhuang et al., 2020) with default hyper-parameters. Fig. 6 (middle subfigure) shows that GSAM performs the best, and SAM improves over vanilla training.

7 CONCLUSION

We propose the surrogate gap as an equivalent measure of sharpness which is easy to compute and feasible to optimize. We propose the GSAM method, which improves the generalization over SAM at negligible computation cost. We show the convergence and provably better generalization of GSAM compared to SAM, and validate the superior performance of GSAM on various models.

ACKNOWLEDGEMENT

We would like to thank Xiangning Chen (UCLA) and Hossein Mobahi (Google) for discussions, Yi Tay (Google) for help with datasets, and Yeqing Li, Xianzhi Du, and Shawn Wang (Google) for help with TensorFlow implementation.

ETHICS STATEMENT

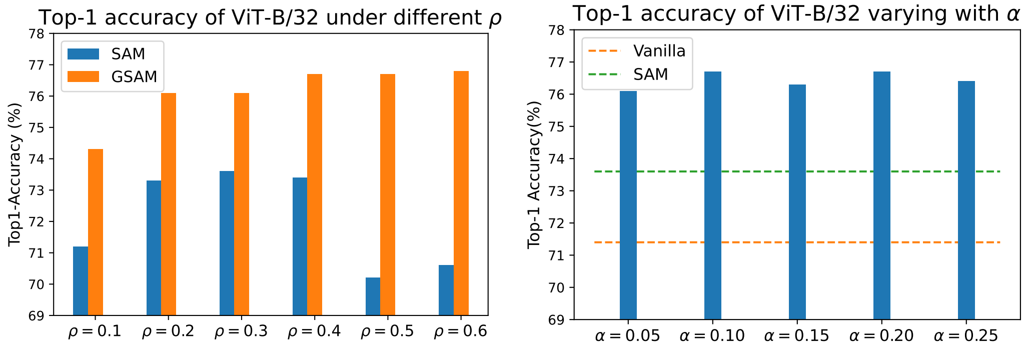
This paper focuses on the development of optimization methodologies and can be applied to the training of different deep neural networks for a wide range of applications. Therefore, the ethical impact of our work would primarily be determined by the specific models that are trained using our new optimization strategy.

REPRODUCIBILITY STATEMENT

We provide the detailed proof of theoretical results in Appendix A and provide the data pre-processing and hyper-parameter settings in Appendix B. Together with the references to existing works and public codebases, we believe the paper contains sufficient details to ensure reproducibility. We plan to release the models trained by using GSAM upon publication.

REFERENCES

Randall Balestriero, Jerome Pesenti, and Yann LeCun. Learning in high dimension always amounts to extrapolation. *arXiv preprint arXiv:2110.09485*, 2021.



(a) Performance of SAM and GSAM under different ρ.

(b) Performance of GSAM under different α

Figure 7: Performance of GSAM varying with ρ and α.

Table 5: Top-1 accuracy of ViT-B/32 on ImageNet with Inception-style data augmentation. For vanilla training we report results for training 300 epochs and 600 epochs, for GSAM we report the results for 300 epochs.

Method	Epochs	ImageNet	ImageNet-Real	ImageNet-v2	ImageNet-R
Vanilla	300	71.4	77.5	57.5	23.4
	600	72.0	78.2	57.9	23.6
GSAM	300	**76.8**	**82.7**	**63.0**	**25.1**

shown in Fig. 7a. Considering that GSAM has one more parameter α, we plot the accuracy varying with α in Fig. 7b, and show that GSAM consistently outperforms SAM and vanilla training.

C.2 CONSTANT ρ V.S. DECAYED ρ_t SCHEDULE

Note that Thm. 5.1 assumes ρ_t to decay with t in order to prove the convergence, while SAM uses a constant ρ during training. To eliminate the influence of ρ_t schedule, we conduct ablation study as in Table. 6. The ascent step in GSAM can be applied to both constant ρ or a decayed ρ_t schedule, and improves accuracy for both cases. Without ascent step, constant ρ and decayed ρ_t achieve similar performance. Results in Table. 6 implies that the ascent step in GSAM is the main reason for improvement of generalization performance.

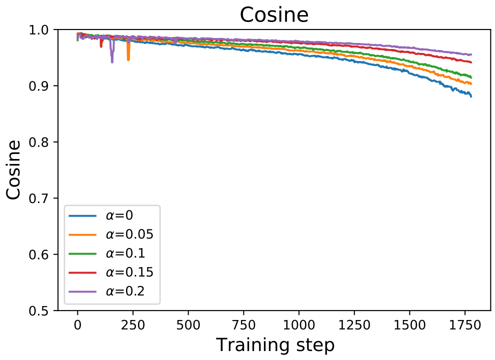


Figure 8: The value of $\cos\theta_t$ varying with training steps, where θ_t is the angle between $\nabla f(w_t)$ and $\nabla f_p(w_t)$ as in Fig. 2.

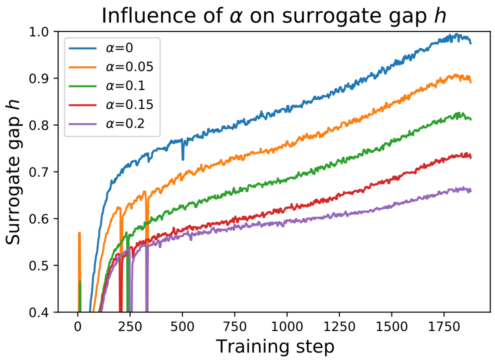


Figure 9: Surrogate gap curve under different α values.