Large Batch Size Training of Neural Networks with Adversarial Training and Second-Order Information

ZHEWEI YAO[1]*, AMIR GHOLAMI[1]*, KURT KEUTZER[1] , AND MICHAEL W. MAHONEY[1]
[1] University of California at Berkeley, {zheweiy, amirgh, keutzer, and mahoneymw}@berkeley.edu

Abstract.
The most straightforward method to accelerate Stochastic Gradient Descent (SGD) is to distribute the randomly selected batch of inputs over multiple processors. To keep the distributed processors fully utilized requires commensurately growing the batch size; however, large batch training usually leads to poor generalization. Existing solutions for large batch training either significantly degrade accuracy or require massive hyper-parameter tuning. To address this issue, we propose a novel large batch training method which combines recent results in adversarial training and second order information. We extensively evaluate our method on Cifar-10/100, SVHN, TinyImageNet, and ImageNet datasets, using multiple NNs, including residual networks as well as smaller networks such as SqueezeNext. Our new approach exceeds the performance of the existing solutions in terms of both accuracy and the number of SGD iterations (up to 1% and 5×, respectively). We emphasize that this is achieved without any additional hyper-parameter tuning to tailor our proposed method in any of these experiments. With slight hyper-parameter tuning, our method can reduce the number of SGD iterations of ResNet18 on Cifar-10/ImageNet to 44.8× and 28.8×, respectively. We have open sourced the method including tools for computing Hessian spectrum [2].

1. Introduction.

Finding the right NN architecture for a particular application requires extensive hyper-parameter tuning and architecture search, often on a very large dataset. The delays associated with training NNs are often the main bottleneck in the design process. One of the ways to address this issue is to use large distributed processor clusters; however, to efficiently utilize each processor, the portion of the batch associated with each processor (sometimes called the mini-batch) must grow correspondingly. In the ideal case, the goal is to decrease the computational time proportional to the increase in batch size, without any drop in generalization quality. However, large batch training has a number of well known drawbacks [16, 27, 36]. These include degradation of accuracy, poor generalization, and even poor robustness to adversarial perturbations [22, 46].

In order to address these drawbacks, many solutions have been proposed [11, 18, 21, 39, 48]. However, these methods either work only for particular models on a particular dataset, or they require massive hyper-parameter tuning. The latter point is often not discussed in the presentation of results. Note that while extensive hyper-parameter turning may result in good result tables, it is antithetical to the original motivation of using large batch sizes to reduce the total time to train the model.

One solution to reduce the brittleness of SGD to hyper-parameter tuning is to use second-order methods. Full Newton method with line search is parameter-free, and it does not require a learning rate. This is achieved by using a second-order Taylor series approximation to the loss function, instead of a first-order one as in SGD, to obtain curvature information. Schaul, Zhang, and LeCun [33], Xu, Roosta-Khorasan, and Mahoney [44], and Xu, Roosta-Khorasani, and Mahoney [45] show that Newton/quasi-Newton methods outperform SGD for training NNs. However, their results only consider simple fully connected NNs and auto-encoders. A problem with naive second-order methods is that they can exacerbate the large batch problem, as by construction they have a higher tendency to get attracted to local minima as compared to SGD. For these reasons, early attempts at using second-order methods for training convolutional NNs have so far not been successful.

Ideally, if we could find a regularization scheme to avoid local/bad minima during training, this could resolve many of these issues. In the seminal works of El Ghaoui and Lebret [12] and Xu, Caramanis, and Mannor [43], a very interesting connection was made between robust optimization and regularization. It was shown that the solution to a robust optimization problem for least squares is the same as the solution of a Tikhonov regularized problem [12]. This was also extended to the Lasso problem in Xu, Caramanis, and Mannor [43]. Adversarial learning/training methods, which are a special case of robust optimization methods, are usually described as a min-max optimization procedure to make the model more robust. Recent studies with NNs have empirically found that

*Both authors contributed equally

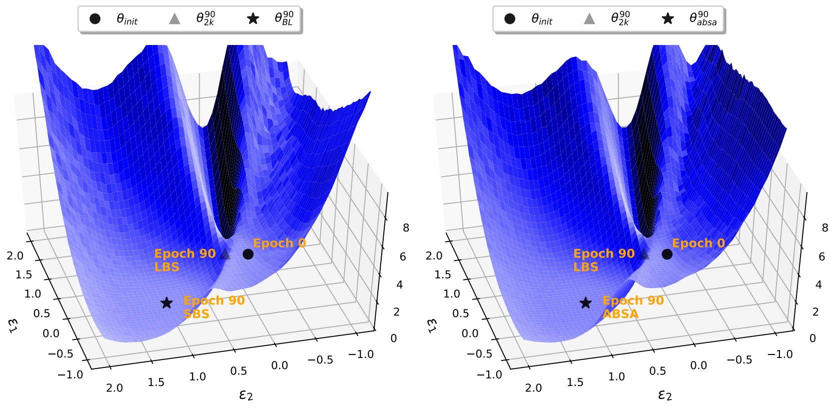

Fig. 1: *(left) 3D parametric plot for C1 model on Cifar-10. Points are labeled with the number of epochs (e.g. 90) and the technique that was used to arrive at that point (e.g. large batch size, LBS). The ϵ_1 direction shows how the loss changes across the path between initial model at epoch 0, and the final model achieved with Large Batch Size (LBS) of $B = 2K$. Similarly, ϵ_2 direction computes loss when the model parameters are interpolated between epoch 0 and final model at epoch 90 with Small Batch Size(SBS). Notice the sharp curvature that Large Batch Size gets attracted to. On the right we show a similar plot except that we use ABSA algorithm with final batch of $16K$ rather than SGD with a small batch size for interpolating the ϵ_2 direction. Notice the visual similarity between the point that ABSA converges to after 90 epochs (ABSA, 84.24% accuracy) and the point that small batch SGD (SBS, 83.05% accuracy) converges to after 90 epochs. Also note, that both avoid the sharp landscape that large batch gets attracted to (LBS, 76.82%). Generalization errors are shown in Table 5.*

robust optimization usually converges to points in the optimization landscape that are flatter and are more robust to adversarial perturbation [46].

Inspired by these results, we explore whether second order information regularized by robust optimization can be used to do large batch size training of NNs. We show that both classes of methods have properties that can be exploited in the context of large batch training to help reduce the brittleness of SGD with large batch size training, thereby leading to significantly improved results.

Main Contributions. In more detail, we propose an adaptive batch size method based on curvature information extracted from the Hessian, combined with a robust optimization method. The latter helps regularize against sharp landscape, especially during early stages of training. We show that this combination leads to superior testing performance, as compared to the proposed methods for large batch size training. Furthermore, in addition to achieving better testing performance, we show that the total number of SGD updates of our method is significantly lower than state-of-the-art methods for large batch size training. We emphasize that we achieve these results without any additional hyper-parameter tuning of our algorithm (which would, of course, have helped us to tailor our solution to these experiments). Here is a more detailed itemization of the main contributions of this work:

- We propose an Adaptive Batch Size (ABS) method for SGD training that is based on second order information, computed by backpropagating second-derivative (i.e. Hessian). Our method automatically changes the batch size and learning rate based on Hessian information. We show a basic result that this method is convergent for a convex problem. More importantly, we empirically test the algorithm for important non-convex problems in deep learning and show that it achieves equal or better test performance, as compared to small batch SGD.
- We propose a regularization method using robust training by solving a min-max optimization problem. We combine the second order adaptive batch size method with recent results of Yao et al. [46], which show that robust training can be used to regularize against sharp minima. We show that

this combination of Hessian-based adaptive batch size and robust optimization achieves significantly better test performance with little computational overhead (we refer to this Adaptive Batch Size Adversarial method as ABSA).

- We test the proposed strategies extensively on a wide range of datasets (Cifar-10/100, SVHN, TinyImageNet, and ImageNet), using different NNs, including residual networks. Importantly, we use the *same hyper-parameters* for all of the experiments, and we do not perform any kind of tuning to tailor our results. The empirical results show the clear benefit of our proposed method, as compared to the state-of-the-art. The proposed algorithm achieves equal or better test accuracy (up to 1%) and requires significantly fewer SGD updates (up to 5×). For just an example, we show that the number of SGD updates can be reduced 44.8× on Cifar-10 and 28.8× on ImageNet, by slight hyper-parameter tuning of the warmup schedule.
- We empirically show that we can use a block approximation of the Hessian operator (i.e., the Hessian of the last fewer layers) to reduce the computational overhead of backpropagating the second order information. This approximation is especially effective in reducing the overhead of ABSA algorithm for deep NNs.

While a number of recent works have discussed adaptive batch size or increasing batch size during training [3, 11, 13, 39], to the best of our knowledge this is the first paper to introduce Hessian information and adversarial training in adaptive batch size training, with extensive testing on many datasets.

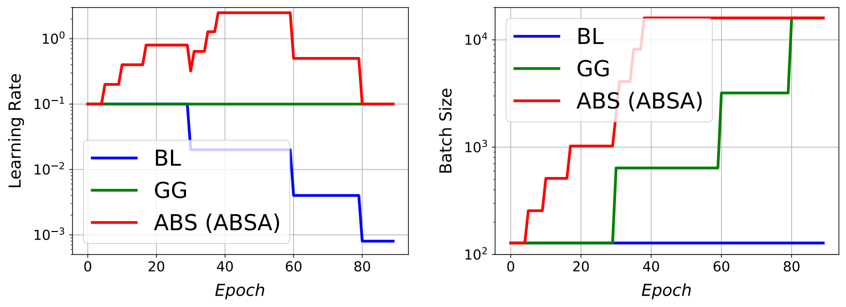

Fig. 2: *Illustration of learning rate (left) and batch size (right) schedules of adaptive batch size as a function of training epochs based on C2 model on Cifar-10.*

Limitations. We believe that it is important for every work to state its limitations (in general, but in particular in this area). We were particularly careful to perform extensive experiments and repeated all the reported tests multiple times. We test the algorithm on models ranging from a few layers to hundreds of layers, including residual networks as well as smaller networks such as SqueezeNext. We have also open sourced our code to allow reproducibility [2].

An important limitation is that second order methods have additional overhead for backpropagating the Hessian. Currently, most of the existing frameworks do not support (memory) efficient backpropagation of the Hessian (thus providing a structural bias against these powerful methods). However, the complexity of each Hessian matvec is the same as a gradient computation [26]. Our method requires Hessian spectrum, which typically needs ten Hessian matvecs (for power method iterations to reach a relative tolerance of 1e-2). Thus, the benefits that we show in terms of testing accuracy and reduced number of updates do come at a cost (see Table 3 for details). We measure this additional overhead and report it in terms of wall clock time. Furthermore, we (empirically) show that this power iteration needs to be done only at the end of every epoch, thus significantly reducing the additional overhead.

Another point is that adaptive batch size prevents one from utilizing all of the processes, as compared to using large batch throughout the training. However, a large data center can very efficiently handle and accommodate a growing number of requests for processor resources [37].

2. Related Work.

Optimization methods based on SGD are currently the most effective techniques for training NNs, and this is commonly attributed to SGD's ability to escape saddle-points and "bad" local minima [9].

The sequential nature of weight updates in synchronous SGD limits possibilities for parallel computing. In recent years, there has been considerable effort on breaking this sequential nature, through asynchronous methods [50] or symbolic execution techniques [25]. A main problem with asynchronous methods is reproducibility, which, in this case, depends on the number of processes used [1, 51]. Due to this issue, recently there have been attempts to increase parallelization opportunities in synchronous SGD by using large batch size training. With large batches, it is possible to distribute more efficiently the computations to parallel compute nodes [14], thus reducing the total training time. However, large batch training often leads to sub-optimal test performance [22, 46]. This has been attributed to the observation that large batch size training tends to get attracted to local minima or sharp curvature directions, which are not robust to (possible) mismatch between training and testing curves [22]. A full understanding of this, however, remains elusive.

Algorithm 1: Algorithm needs to be fixed: Adaptive Batch Size (ABS) and Adaptive Batch Size Adversarial (ABSA)

```
Input: Learning rate $\eta$, decay epoch $E_{decay}$, decay ratio $\rho_{lr,decay}$; initial batch $B_{init}$,
       maximum batch $B_{max}$; Eigenvalue decay ratio $\alpha$, batch/lr increasing ratio $\beta$, duration
       factor $\kappa$; adversarial input ratio $\gamma$, adversarial magnitude $\epsilon_{adv}$, adversarial input ratio
       decay factor $\omega$, terminate epoch $\tau$; input data $x$.
Result: Final model parameters $\theta$
Initialize old (new) eigenvalue as $\lambda_{old}$ ($\lambda_{new}$), batch size $B_{itr}$, learning rate $\eta_{itr}$, duration time
 $\kappa_{itr} = 0$.
for $Epoch = 1, 2, 3 \dots$ do
    for $t = 1, 2, 3 \dots$ do // mini-batch iterations
        generate adversarial data $x_{adv}$ by the ratio of $\gamma$;
        SGD update;
    compute $\lambda_{new}$; // Hessian power-iteration
    $\kappa_{itr}$ += 1;
    $Flag = \{(\lambda_{new} < \lambda_{old}/\alpha)\ OR\ (\kappa_{itr} = \kappa)\}$;
    if $Flag$ then // adaptively change batch/lr
        $B_{itr}$ *= $\beta$;   $\eta_{itr}$ *= $\beta$; $\gamma$ /= $\omega$;  $\kappa_{itr} = 0$;
    if $\lambda_{new} < \lambda_{old}/\alpha$ then
        $\lambda_{old} = \lambda_{new}$;
    if $Epoch = \tau$ then
        $\gamma = 0$;
    if $Epoch\ in\ E_{decay}$ then // lr decay schedule
        $\eta_{itr}$ /= $\rho_{lr,decay}$ ;
```

There have been several solutions proposed for alleviating the problem with large batch size training. The first notable work here is Goyal et al. [18], where it was shown that by scaling the learning rate, it is possible to achieve the same testing accuracy for large batches. In particular, ResNet-50 model was tested on ImageNet dataset, and it was shown that the baseline accuracy could be recovered up to batch size of 8192. However, this approach does not generalize to other networks/tasks [16]. In You, Gitman, and Ginsburg [48], an adaptive learning rate method (called LARS) was proposed which allowed scaling training to a much larger batch size of 32K with more hyper-parameter tuning. Another notable work is Smith, Kindermans, and Le [39] (and also [11]), which proposed a hybrid increase of batch size and learning rate to accelerate training. In this approach, one would select a strategy to "anneal" the batch size during the training. This is based on the idea that large batches contain less "noise," and that could be used much the same way as reducing learning rate during training. More recent work Jia et al. [21] and Puri et al. [32] proposed mix-precision method to further explore the limit of large batch training. There is also a set of seminal works on dynamic batch size selection to accelerate optimization, where the batch size is selected to ensure a search direction that would decrease the loss function with high probability [4, 5, 7]. Another notable work is the use of distributed K-Fac method to perform large batch training [31].

A recent study has shown that anisotropic noise injection could also help in escaping sharp landscape [52]. The authors showed that the noise from SGD could be viewed as anisotropic, with the Hessian as its covariance matrix. Injecting random noise using the Hessian as covariance was proposed as a method to avoid sharp landscape.

Another recent work by Yao et al. [46] has shown that adversarial training (or robust optimization) could be used to "regularize" against these sharp minima, with preliminary results showing superior testing performance as compared to other methods. Shaham, Yamada, and Negahban [35] and Shrivastava et al. [38] used adversarial training and showed that the model trained using robust optimization is often more robust to perturbations, as compared to normal SGD training. Similar observations have been made by others [17, 41].

We note that our results are in agreement with recent studies showing that there is a limit beyond which increasing batch size will lead to diminishing results [16, 36]. These studies mainly focus on constant batch size. However, the work of [16] showed that warmup with smaller batches can noticeably delay this limit to larger batches. An adaptive batch size strategy would involve several different batches throughout the training, and this adaptive schedule would be more efficient in the regime of large batches [27]. Meaning that increasing the batch size in each segment of this adaptive schedule would be more efficient in terms of minimum number of optimization steps [27]. Our goal is to achieve this using ABSA algorithm to fully exploit larger batches during training, and delay the limit beyond which using large batches would result in diminishing returns.

3. Method.

We consider a supervised learning framework where the goal is to minimize a loss function $L(\theta)$:

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N} l(z_i, \theta), \tag{3.1}$$

where θ are the model weight parameters, $Z = X \times Y$ is the training dataset, and $l(z, \theta)$ is the loss for a datum $z \in Z$. Here, X is the input, Y is the corresponding label, and $N = |Z|$ is the cardinality of the training set. SGD is typically used to optimize Eqn. (3.1) by taking steps of the form:

$$\theta_{t+1} = \theta_t - \eta_t \frac{1}{|B|}\sum_{z \in B} \nabla_\theta l(z, \theta_t), \tag{3.2}$$

where B is a mini-batch of examples drawn randomly from Z, and η_t is the step size (learning rate) at iteration t. In the case of large batch size training, the batch size is increased to large values.

As mentioned above, large batch size training often results in converging to a point with poor generalization. This has been attributed to an observation that large batches get attracted to sharper landscapes [22, 46]. The idea here is that converging to a sharp landscape for the training is less stable to possible noise/shift between training and testing losses. To clearly illustrate this, we show a 3D parametric plot of the loss landscape when interpolating between the model parameters in the beginning of training (Epoch 0) and at the end of training (Epoch 90) with large batch size for the first direction, and interpolate between the same initialization and weights of the model trained with small batch at Epoch 90. We compute this plot for multiple different models and networks (please see Figure 1, 7 and 8). One can clearly see that the large batch experiments get attracted to a sharper landscape, whereas the small batch training ends in a significantly flatter one. One popular argument is that small batch training has sufficient noise to escape such sharp landscape, and does not gets trapped or slowed there. The latter is the main motivation behind anisotropic noise injection [52], or training longer recipes [20] that are proposed to address this problem.

Smith and Le [40] views the learning rate and batch size as noise injected during optimization. Both a large learning rate as well as a small batch size can be considered equivalent to high noise injection. This is explained by modeling the behavior of NNs as a stochastic differential equation (SDE) of the following form:

$$\frac{d\theta}{dt} = \frac{dL}{d\theta} + \epsilon(t), \tag{3.3}$$

where $\epsilon(t)$ is the noise injected by SGD (see Smith and Le [40] for details). The authors then argue that the noise magnitude is proportional to $g = \eta_t(\frac{|Z|}{|B|} - 1)$. For mini-batch $|B| \ll |Z|$, the noise magnitude can be estimated as $g \approx \eta_t \frac{|Z|}{|B|}$.

Hence, in order to achieve the benefits from small batch size training, i.e., the noise generated by small batch training, the learning rate η_t should increase proportionally to the batch size, and vice versa. That is, the same annealing behavior could be achieved by increasing the batch size, which is the method used by Smith, Kindermans, and Le [39]. The empirical study of [16], shows that large batches have higher probability to get attracted to such landscape earlier in the training. An interesting observation is that this problem gets alleviated through warmups with small batch size, to escape more rugged landscape in the beginning of training. Thus using small batch SGD for this initial regime allows one to utilize larger batches for the rest of the iterations. Ideally, we would like to have an automatic method that could provide us with such information and regularize against such sharp landscape. As we show next, this is possible through the use of second order information combined with robust optimization.

Table 1: *Accuracy and the number of parameter updates of C3 on Cifar-10.*

	BL		FB		GG		ABS		ABSA	
BS	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters
128	92.02	78125	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
256	91.88	39062	91.75	39062	91.84	50700	91.7	40792	**92.11**	43352
512	91.68	19531	91.67	19531	91.19	37050	**92.15**	32428	91.61	25388
1024	89.44	9766	91.23	9766	91.12	31980	91.61	17046	**91.66**	23446
2048	83.17	4882	90.44	4882	89.19	30030	91.57	21579	**91.61**	14027
4096	73.74	2441	86.12	2441	91.83	29191	91.91	18293	**92.07**	21909
8192	63.71	1220	64.91	1220	91.51	28947	91.77	22802	**91.81**	16778
16384	47.84	610	32.57	610	90.19	28828	**92.12**	17485	91.97	24361

3.1. Adaptive Batch Size (ABS) based on Hessian Information. In this section, we propose a method for utilizing second order information to adaptively change the batch size. We refer to this as the Adaptive Batch Size (ABS) method; see Algorithm 1. The main intuition is to keep the SGD noise high to escape the sharp landscape shown in Figure 1. To this end, a smaller batch size is used in regions with a "sharper" loss landscape to help avoid attraction to landscapes with poor generalization. We then switch to a larger batch size only in regions where the loss has a "flatter" landscape.

We can measure magnitude of the loss curvature through the Hessian (second order) information. We compute the dominant Hessian curvature through power iteration [1]. We have developed an efficient implementation to compute curvature information by using the same automatic differentiation pipeline that is used for backpropogating the gradient [2]. The power iteration algorithm starts with a vector drawn randomly from a Gaussian distribution, followed by successive application of the Hessian operator to this vector (so called 'matvec'). Thus instead of computing the full Hessian matrix we simply backpropogate this matvec.

[1]One could also use Lanczos method; however, due to numerical errors the Hessian will not be exactly symmetric. Meanwhile, it only takes within ten power iterations to get 10^{-2} accuracy, which is accurate enough for ABS/ABSA.

Once we compute this curvature, we then adaptively change the batch size based on the relative magnitude of the Hessian eigenvalue as compared to previous iterations.[2] The batch size is increased only if the relative curvature reduces or stays stable for several epochs (fixed to be ten in all of the experiments) Algorithm 1.

The second component of our framework is robust optimization. In the seminal works of [12, 43], a connection between robust optimization and regularization was proved in the context of ridge and lasso regression. In Yao et al. [46], the authors empirically showed that adversarial training leads to more robust models with respect to adversarial perturbation. An interesting corollary was that, after adversarial training, the model converges to regions that are considerably flatter, as compared to the baseline. Thus, we can combine our ABS algorithm with adversarial training as a form of regularization against "sharp" landscape. We refer to this as the Adaptive Batch Size Adversarial (ABSA) method; see Algorithm 1.

In practice, we find that ABSA often achieves better error rates than ABS. In ABSA we solve a min-max problem instead of a normal empirical risk minimization problem [22, 46]. Solving this min-max problem for NNs is an intractable problem, and thus we approximately solve the maximization part through the Fast Gradient Sign Method (FGSM) proposed by Goodfellow, Shlens, and Szegedy [17]. This basically corresponds to generating adversarial inputs using one gradient ascent step (i.e., the perturbation is computed by $\Delta x = \epsilon \, \text{sign}(\nabla_x l(z, \theta))$). Other possible choices are proposed by [8, 29, 42].[3]

Figure 2 illustrates our ABS schedule as compared to a normal training strategy and the adaptive batch size method of Devarakonda, Naumov, and Garland [11] and Smith, Kindermans, and Le [39]. Note that our learning rate adaptively changes based on the Hessian eigenvalue in order to keep the same noise level as in the baseline SGD training (by adaptively changing learning rate proportional to batch size). As we show in section 4, our combined approach (second order and robust optimization) not only achieves better accuracy, but it also requires significantly fewer SGD updates, as compared to Devarakonda, Naumov, and Garland [11] and Smith, Kindermans, and Le [39].

Table 2: *Accuracy and the number of parameter updates of I1 on TinyImageNet.*

	BL		FB		GG		ABSA	
BS	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters
128	60.41	93750	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
256	58.24	46875	59.82	46875	60.31	70290	**61.28**	60684
512	57.48	23437	59.28	23437	59.94	58575	**60.55**	51078
1024	54.14	11718	59.62	11718	59.72	52717	**60.72**	19011
2048	50.89	5859	59.18	5859	59.82	50667	**60.43**	17313
4096	40.97	2929	58.26	2929	60.09	49935	**61.14**	22704
8192	25.01	1464	16.48	1464	60.00	49569	**60.71**	22334
16384	10.21	732	0.50	732	60.37	48995	**60.71**	20348

4. Results.

We evaluate the performance of our ABS and ABSA methods on different datasets (ranging from O(1E4) to O(1E7) training examples) and multiple NN models. We compare the baseline SGD performance, along with other state-of-the-art methods proposed for large batch training [18, 39]. Notice that GG [39] and ABS/ABSA have different batch sizes during training. Hence the batch size reported in our results represents the maximum batch size during training. To allow for a direct comparison we also report the number of weight updates in our results (lower is better). It should be mentioned that the number of SGD updates is not necessarily the same as the wall-clock time.

[2]Note that sharpness is a relative measure since the absolute value of curvature could be different for each model/dataset.

[3]In Yao et al. [46], similar behavior was observed with other methods for solving the robust optimization problem.

The two main metrics for comparison are (1) the final accuracy and (2) the total number of updates. Preferably we would want a higher testing accuracy along with fewer SGD updates. We emphasize that for all of the datasets and models we tested, we do *not* change any of the hyper-parameters in our algorithm. We use the exact same parameters used in the baseline model, and we do not tailor any parameters to suit our algorithm. A detailed explanation of the different NN models, and the datasets is given in Appendix B.

Section 4.1 shows the result of ABS (ABSA) compared to BaseLine (BL), FB [18] and GG [39]. Section 4.2 presents the results on more challenging datasets of TinyImageNet and **ImageNet**. Before discussing the reults in detail, we note that the higher performance of ABS/ABSA does come at the cost of backpropagating the Hessian. Thus, in section 4.4, we discuss how approximate Hessian information could be used to alleviate the computational costs.

4.1. ABS and ABSA for SVHN and Cifar. Tables 1 and 4-8 (see Appendix D for Tables 4-8) report the test accuracy and the number of parameter updates for different datasets and models. First, note the drop in BL accuracy for large batch confirming the accuracy degradation problem. Moreover, note that the FB strategy only works well for moderate batch sizes (it diverges for large batch). However, the GG method has a very consistent performance, but its number of parameter updates are usually greater than our method. Looking at the last two major columns of Tables 1 and 4-8, the test performances ABS achieves are similar accuracy as BL. Overall, the number of updates of ABS is 3-10 times smaller than BL with batch size 128. For most cases, ABSA achieves superior accuracy. This confirms the effectiveness of adversarial training combined with the second order information. Furthermore, we show a similar 3D parametric plot for ABSA algorithm in Figures 1, 7 and 8. Note that ABSA avoids the sharp landscape of large batch and is able to converge to a landscape with similar curvature as that of small batch training.

Furthermore, we show the loss landscape of model obtained throughout the training process along the dominant eigenvalue [46] shown in Figure 9., and compare the models of small/large batch training with baseline (SGD with constant batch) and ABSA (which uses SGD with adaptive batch). It can be clearly seen that throughout training larger batches tend to get stuck/attracted to areas with larger curvature, while small batch SGD and ABSA are able to avoid them.

4.2. ABSA for TinyImageNet and ImageNet. SVHN is a very simple dataset, and Cifar-10/100 are relatively small datasets, and one might wonder whether the improvements we reported in Section 4.1 hold for more complex problems. Here, we report the ABSA method on more challenging datasets, specifically, TinyImageNet and ImageNet. We use the exact same hyper-parameters in our algorithm, even though tuning them could potentially be preferable for us.

TinyImageNet is an image classification problem, with 200 classes and only 500 images per class, and it is very easy to overfit its training data. The results for the I1 model (see Section B in the Appendix for descriptions of these models) is reported in Table 2. Note that with fewer SGD iterations, ABSA can achieve better test accuracy than other methods. The performance of ABSA is actually about 1% higher (the training loss and test performance of I1 on TinyImagenet is shown in Figure 5 in appendix).

ImageNet classification task is perhaps among the most challenging open-source classification problems. Due to the limited computational resources, we only test ABSA and BL (baseline with small batch), and report results in Figure 6 (see Appendix D). We first start with I2 model (AlexNet) whose baseline requires $450K$ parameter updates (i.e. SGD iterations), reaching 56.32% validation accuracy. For ABSA, the final validation accuracy is 56.40%, with only $76K$ parameter updates. The maximum batch size reached by ABSA is $16,384$, with initial batch size 256.

We also test a residual model (I3) on ImageNet as shown in Figure 3. The BL with batch of 256 reaches 70.46% validation accuracy with $450K$ parameter updates. The final validation accuracy of ABS and ABSA are 70.15% and 70.24%, respectively, both with $66,393$ parameter updates. The maximum batch size reached by ABS and ABSA is $16,384$ with initial batch size 256. If GG schedule is implemented, the total number of parameter updates would have been $166K$. (Due to the limitation of resource, we do not run GG for I3 on ImageNet.)

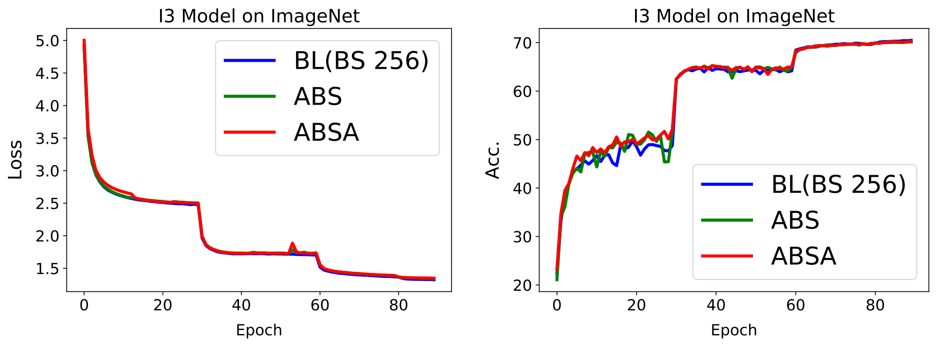

Fig. 3: *I3 model on ImageNet. Training set loss (left), and testing set accuracy (right), evaluated as a function of epochs.*

4.3. ABSA with Warm-up Tuning. To *effectively* speed up training, large batch should not require extensive hyper-parameter tuning. However, a recent trend (mainly motivated by industry labs) is focused on showing how fast training on a benchmark task could be finished on a supercomputer [18, 28, 40, 47, 48]. The approaches used include novel systems level contributions to reduce communication overhead [14, 34], combined with exhaustive hyper-parameter search to increase batch size. This is mostly not possible with academic resources. However, to test how ABSA's performance could be boosted by tuning, we solely focus on the warmup schedule (and keep all the other hyper-parameters such as α, β the same).

For the warmup tuning, our goal is to ramp up to large batches quickly. On Cifar-10 with C1 model, we increase the initial batch size to 1920 (compared to baseline batch size 128) with largest batch size 16384, and gradually increase the learning rate to 1.5 in five epochs as in [18]. The final accuracy of this mixed schedule is 83.24% with 784 SGD updates (compared to 83.04% for baseline with $35K$ SGD updates, which is $48.1\times$ smaller).

On ImageNet, we increase the initial batch size to 4096 (compared to baseline batch size 256) with largest batch size $16K$, and gradually increasing the learning rate to 1.6 in 20 epochs. Also, we add another 10 epochs, i.e. total 100 epochs to train it [20], and decay the learning rate by a factor of 10 at 90-th epoch. The final accuracy is 70.04% with $14.8K$ SGD updates (as compared to 70.4% for baseline with 450k SGD updates, which is $28.8\times$ smaller).

We emphasize that these results are achieved with moderate hyper-parmaeter tuning though only 10 trials. We also did not tune any other hyper-parameters introduced by ABSA. It is quite feasible that with an industrial infrastructure and tuning of learning rate, momentum, and ABSA's parameters, the results could significantly be improved.

4.4. Approximate Hessian. One of the limitations of ABS (ABSA) method is the additional computational cost for computing the top Hessian eigenvalue. If we use the full Hessian operator, the second backpropagation needs to be done all the way to the first layer of NN. For deep networks this could lead to high ciputatinoal cost. Here, we empirically explore whether we could use approximate second order information, and in particular we test a block Hessian approximation as illustrated in Figure 10. The block approximation corresponds to only analyzing the Hessian of the last few layers.

In Figure 10 (see Appendix D), we plot the trace of top eigenvalues of full Hessian and block Hessian for C1 model. Although the top eigenvalue of block Hessian has more variance than that of full Hessian, the overall trends are similar for C1. The test performance of C1 on Cifar-10 with block Hessian is 84.82% with 4600 parameter updates (as compared to 84.42% for full Hessian ABSA). The test performance of C4 on Cifar-100 with block Hessian is 68.01% with 12500 parameter updates (as compared to 68.43% for full Hessian ABSA). These results suggest that using a block Hessian to estimate the trend of the full Hessian might be a good choice to overcome computation cost, but a more detailed analysis is needed.

5. Conclusion. We introduce an adaptive batch-size algorithm based on Hessian information to speed up the training process of NNs, and show how this approach can be combined with adversarial training (which is a form of robust optimization, and which could be viewed as a regularization term for large batch training). We extensively test the methods on multiple datasets (SVHN, Cifar-10/100, TinyImageNet and ImageNet) with multiple NN models (AlexNet, ResNet, Wide ResNet and SqueezeNext). As the goal of using large batches is to reduce training time, we did not perform any hyper-parameter tuning to tailor our method for any of these tests. Our method enables one to increase batch size and learning rate automatically, based on Hessian information. This helps significantly reduce the number of parameter updates, and it achieves superior generalization performance, without the need to tune any of the additional hyper-parameters. We also showed how the results could be significantly improved with slight hyper-parameter tuning (up to 44.8×). Finally, we show that a block Hessian can be used to approximate the trend of the full Hessian to reduce the additional computational overhead of using second-order information. These improvements are useful to reduce NN training time in practice. We also note that our results are in agreement with recent work of [16], where it is shown that there is a saturation limit for how much large batch size could be increased, and how warmup with small batch affects this limit. The goal of ABSA is to reduce warmup with small batch to result in smaller number of SGD updates for the whole training.

Acknowledgments. This would was supported by a gracious fund from Intel corporation. We would like to thank Intel VLAB team for providing us with access to their computing cluster. We also gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research. Furthermore, MWM would like to acknowledge ARO, DARPA, NSF, and ONR for providing partial support of this work.

References.

[1] Alekh Agarwal and John C Duchi. "Distributed delayed stochastic optimization". In: *Advances in Neural Information Processing Systems*. 2011, pp. 873–881.

[2] *anonymized version in supplementary material*. Jan. 2019.

[3] Lukas Balles, Javier Romero, and Philipp Hennig. "Coupling adaptive batch sizes with learning rates". In: *arXiv preprint arXiv:1612.05086* (2016).

[4] Raghu Bollapragada, Richard Byrd, and Jorge Nocedal. "Adaptive sampling strategies for stochastic optimization". In: *SIAM Journal on Optimization* 28.4 (2018), pp. 3312–3343.

[5] Raghu Bollapragada, Dheevatsa Mudigere, Jorge Nocedal, Hao-Jun Michael Shi, and Ping Tak Peter Tang. "A progressive batching L-BFGS method for machine learning". In: *arXiv preprint arXiv:1802.05374* (2018).

[6] Léon Bottou, Frank E Curtis, and Jorge Nocedal. "Optimization methods for large-scale machine learning". In: *SIAM Review* 60.2 (2018), pp. 223–311.

[7] Richard H Byrd, Gillian M Chin, Jorge Nocedal, and Yuchen Wu. "Sample size selection in optimization methods for machine learning". In: *Mathematical programming* 134.1 (2012), pp. 127–155.

[8] Nicholas Carlini and David Wagner. "Towards evaluating the robustness of neural networks". In: *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2017, pp. 39–57.

[9] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization". In: *Advances in neural information processing systems*. 2014, pp. 2933–2941.

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database". In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. Ieee. 2009, pp. 248–255.

[11] Aditya Devarakonda, Maxim Naumov, and Michael Garland. "AdaBatch: Adaptive Batch Sizes for Training Deep Neural Networks". In: *arXiv preprint arXiv:1712.02029* (2017).

[12] Laurent El Ghaoui and Hervé Lebret. “Robust solutions to least-squares problems with uncertain data”. In: *SIAM Journal on matrix analysis and applications* 18.4 (1997), pp. 1035–1064.

[13] Michael P Friedlander and Mark Schmidt. “Hybrid deterministic-stochastic methods for data fitting”. In: *SIAM Journal on Scientific Computing* 34.3 (2012), A1380–A1405.

[14] Amir Gholami, Ariful Azad, Peter Jin, Kurt Keutzer, and Aydin Buluc. “Integrated Model, Batch and Domain Parallelism in Training Neural Networks”. In: *ACM Symposium on Parallelism in Algorithms and Architectures(SPAA'18)* (2018).

[15] Amir Gholami, Kiseok Kwon, Bichen Wu, Zizheng Tai, Xiangyu Yue, Peter Jin, Sicheng Zhao, and Kurt Keutzer. “SqueezeNext: Hardware-Aware Neural Network Design”. In: *arXiv preprint arXiv:1803.10615* (2018).

[16] Noah Golmant, Nikita Vemuri, Zhewei Yao, Vladimir Feinberg, Amir Gholami, Kai Rothauge, Michael W Mahoney, and Joseph Gonzalez. “On the Computational Inefficiency of Large Batch Sizes for Stochastic Gradient Descent”. In: *arXiv preprint arXiv:1811.12941* (2018).

[17] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and harnessing adversarial examples (2014)”. In: *arXiv preprint arXiv:1412.6572* (2014).

[18] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. “Accurate, large minibatch SGD: training imagenet in 1 hour”. In: *arXiv preprint arXiv:1706.02677* (2017).

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2016, pp. 770–778.

[20] Elad Hoffer, Itay Hubara, and Daniel Soudry. “Train longer, generalize better: closing the generalization gap in large batch training of neural networks”. In: *Advances in Neural Information Processing Systems.* 2017, pp. 1731–1741.

[21] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, et al. “Highly Scalable Deep Learning Training System with Mixed-Precision: Training ImageNet in Four Minutes”. In: *arXiv preprint arXiv:1807.11205* (2018).

[22] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. “On large-batch training for deep learning: Generalization gap and sharp minima”. In: *arXiv preprint arXiv:1609.04836* (2016).

[23] Alex Krizhevsky and Geoffrey Hinton. *Learning multiple layers of features from tiny images.* Tech. rep. Citeseer, 2009.

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems.* 2012, pp. 1097–1105.

[25] Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. “Parallel Stochastic Gradient Descent with Sound Combiners”. In: *arXiv preprint arXiv:1705.08030* (2017).

[26] James Martens. “Deep learning via Hessian-free optimization.” In: *ICML*. Vol. 27. 2010, pp. 735–742.

[27] Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. “An Empirical Model of Large-Batch Training”. In: *arXiv preprint arXiv:1812.06162* (2018).

[28] Hiroaki Mikami, Hisahiro Suganuma, Yoshiki Tanaka, Yuichi Kageyama, et al. “ImageNet/ResNet-50 Training in 224 Seconds”. In: *arXiv preprint arXiv:1811.05233* (2018).

[29] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. “Deepfool: a simple and accurate method to fool deep neural networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2016, pp. 2574–2582.

[30] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. “Reading digits in natural images with unsupervised feature learning”. In: *NIPS workshop on deep learning and unsupervised feature learning.* Vol. 2011. 2011, p. 5.

[31] Kazuki Osawa, Yohei Tsuji, Yuichiro Ueno, Akira Naruse, Rio Yokota, and Satoshi Matsuoka. "Second-order Optimization Method for Large Mini-batch: Training ResNet-50 on ImageNet in 35 Epochs". In: *arXiv preprint arXiv:1811.12019* (2018).

[32] Raul Puri, Robert Kirby, Nikolai Yakovenko, and Bryan Catanzaro. "Large Scale Language Modeling: Converging on 40GB of Text in Four Hours". In: *arXiv preprint arXiv:1808.01371* (2018).

[33] Tom Schaul, Sixin Zhang, and Yann LeCun. "No more pesky learning rates". In: *International Conference on Machine Learning*. 2013, pp. 343–351.

[34] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns". In: *Fifteenth Annual Conference of the International Speech Communication Association*. 2014.

[35] Uri Shaham, Yutaro Yamada, and Sahand Negahban. "Understanding adversarial training: Increasing local stability of neural nets through robust optimization". In: *arXiv preprint arXiv:1511.05432* (2015).

[36] Christopher J Shallue, Jaehoon Lee, Joe Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E Dahl. "Measuring the effects of data parallelism on neural network training". In: *arXiv preprint arXiv:1811.03600* (2018).

[37] Vaishaal Shankar, Karl Krauth, Qifan Pu, Eric Jonas, Shivaram Venkataraman, Ion Stoica, Benjamin Recht, and Jonathan Ragan-Kelley. "numpywren: serverless linear algebra". In: *arXiv preprint arXiv:1810.09679* (2018).

[38] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Joshua Susskind, Wenda Wang, and Russell Webb. "Learning from Simulated and Unsupervised Images through Adversarial Training." In: *CVPR*. Vol. 2. 2017, p. 5.

[39] Samuel L Smith, Pieter-Jan Kindermans, and Quoc V Le. "Don't Decay the Learning Rate, Increase the Batch Size". In: *arXiv preprint arXiv:1711.00489* (2017).

[40] Samuel L Smith and Quoc V Le. "A Bayesian perspective on generalization and Stochastic Gradient Descent". In: *arXiv preprint arXiv:1710.06451* (2018).

[41] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. "Intriguing properties of neural networks". In: *arXiv preprint arXiv:1312.6199* (2013).

[42] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. "Optimization of collective communication operations in MPICH". In: *The International Journal of High Performance Computing Applications* 19.1 (2005), pp. 49–66.

[43] Huan Xu, Constantine Caramanis, and Shie Mannor. "Robust regression and Lasso". In: *Advances in Neural Information Processing Systems*. 2009, pp. 1801–1808.

[44] Peng Xu, Farbod Roosta-Khorasan, and Michael W Mahoney. "Second-order optimization for non-convex machine learning: An empirical study". In: *arXiv preprint arXiv:1708.07827* (2017).

[45] Peng Xu, Farbod Roosta-Khorasani, and Michael W Mahoney. "Newton-type methods for non-convex optimization under inexact hessian information". In: *arXiv preprint arXiv:1708.07164* (2017).

[46] Zhewei Yao, Amir Gholami, Qi Lei, Kurt Keutzer, and Michael W Mahoney. "Hessian-based Analysis of Large Batch Training and Robustness to Adversaries". In: *arXiv preprint arXiv:1802.08241* (2018).

[47] Chris Ying, Sameer Kumar, Dehao Chen, Tao Wang, and Youlong Cheng. "Image Classification at Supercomputer Scale". In: *arXiv preprint arXiv:1811.06992* (2018).

[48] Yang You, Igor Gitman, and Boris Ginsburg. "Scaling sgd batch size to 32k for imagenet training". In: *arXiv preprint arXiv:1708.03888* (2017).

[49] Sergey Zagoruyko and Nikos Komodakis. "Wide residual networks". In: *arXiv preprint arXiv:1605.07146* (2016).

[50] Sixin Zhang, Anna E Choromanska, and Yann LeCun. "Deep learning with elastic averaging SGD". In: *Advances in Neural Information Processing Systems*. 2015, pp. 685–693.

[51] Shuxin Zheng, Qi Meng, Taifeng Wang, Wei Chen, Nenghai Yu, Zhi-Ming Ma, and Tie-Yan Liu. “Asynchronous stochastic gradient descent with delay compensation”. In: *arXiv preprint arXiv:1609.08326* (2016).

[52] Zhanxing Zhu, Jingfeng Wu, Bing Yu, Lei Wu, and Jinwen Ma. “The Anisotropic Noise in Stochastic Gradient Descent: Its Behavior of Escaping from Minima and Regularization Effects”. In: *arXiv preprint arXiv:1803.00195* (2018).

Appendix A. Convergence Rate of ABS. Here, we provide a straightforward proof that ABS algorithm does converge for strongly convex problems. This proof is a basic modification of SGD's convergence proof with constant batch size. Based on an assumption about the loss (Assumption 2 in Appendix A.1), it is not hard to prove the following theorem.

Theorem 1. *Under Assumption 2, assume at step t, the batch size used for parameter update is b_t, the step size is $b_t\eta_0$, where η_0 is fixed and satisfies,*

$$0 < \eta_0 \le \frac{1}{L_g(M_v + B_{\max})}, \tag{A.1}$$

where $B_{\max}$ is the maximum batch size during training. Then, with θ_0 as the initialization, the expected optimality gap satisfies the following inequality,

$$\mathbb{E}[L(\theta_{t+1})] - L_* \le \prod_{k=1}^{t}(1 - b_k\eta_0 c_s)(L(\theta_0) - L_* - \frac{\eta_0 L_g M}{2c_s}) + \frac{\eta_0 L_g M}{2c_s}. \tag{A.2}$$

From Theorem 1, if $b_t \equiv 1$, the convergence rate for t steps, based on equation A.2, is $(1 - \eta_0 c_s)$. However, the convergence rate of Algorithm 1 becomes $\prod_{k=1}^{t}(1 - b_k\eta_0 c_s)$, where $1 \le b_k \le B_{max}$. With an adaptive b_t Algorithm 1 can converge faster than basic SGD. We show empirical results for a logistic regression problem in the Appendix A.1, which is a simple convex problem.

A.1. Proof of Theorem. For a finite sum objective function $L(\theta)$, i.e., equation 3.1, we assume that:

Assumption 2. *The objective function $L(\theta)$ satisfies:*

- *$L(\theta)$ is continuously differentiable and the gradient function of L is Lipschitz continuous with Lipschitz constant L_g, i.e.*

$$\|\nabla L(\theta_1) - \nabla L(\theta_2)\| \le L_g\|\theta_1 - \theta_2\|, \qquad \text{for all } \theta_1 \text{ and } \theta_2. \tag{A.3}$$

- *$L(\theta)$ is strongly convex, i.e., there exists a constant $c_s > 0$ s.t.*

$$L(\theta_2) \ge L(\theta_1) + \nabla L(\theta_1)^T(\theta_2 - \theta_1) + \frac{1}{2}c_s\|\theta_1 - \theta_2\|^2, \qquad \text{for all } \theta_1 \text{ and } \theta_2. \tag{A.4}$$

  Also, the global minima of $L(\theta)$ is achieved at θ_* and $L(\theta_*) = L_*$.
- *Each gradient of each individual $l_i(z_i)$ is an unbiased estimation of the true gradient, i.e.*

$$\mathbb{E}[\nabla l_i(z_i, \theta)] = \nabla L(\theta), \qquad \text{for all } i. \tag{A.5}$$

- *There exist scalars $M \ge 0$ and $M_v \ge 0$ s.t.*

$$\mathbb{V}(\nabla l_i(z_i, \theta)) \le M + M_v\|\nabla L(\theta)\|, \qquad \text{for all } i, \tag{A.6}$$

  where $\mathbb{V}(\cdot)$ is the variance operator, i.e.

$$\mathbb{V}(\nabla l_i(z_i, \theta)) = \mathbb{E}[\|\nabla l_i(z_i, \theta)\|^2] - \|\mathbb{E}[\nabla l_i(z_i, \theta)]\|^2.$$

From the Assumption 2, it is not hard to get,

$$\mathbb{E}[\|\nabla l_i(z_i, \theta)\|^2] \le M + M_g\|\nabla L(\theta)\|^2, \tag{A.7}$$

with $M_g = M_v + 1$.

With Assumption 2, the following two lemmas could be found in any optimization reference, e.g. [6]. We give the proofs here for completeness.

Lemma 3. *Under Assumption 2, after one iteration of stochastic gradient update with step size η_t at θ_t, we have*

$$\mathbb{E}[L(\theta_{t+1})] - L(\theta_t) \le -(1 - \frac{1}{2}\eta_t L_g M_g)\eta_t\|\nabla L(\theta_t)\|^2 + \frac{1}{2}\eta_t^2 L_g M, \tag{A.8}$$

where $\theta_{t+1} = \theta_t - \eta_t\nabla l_i(\theta, z_i)$ for some i.

Proof. With the L_g smooth of $L(\theta)$, we have

$$\begin{aligned}\mathbb{E}[L(\theta_{t+1})] - L(\theta_t) &\le -\eta_t \nabla L(\theta_t)\mathbb{E}[\nabla l_i(\theta, z_i)] + \frac{1}{2}\eta_t^2 L_g \mathbb{E}[\|\nabla l_i(\theta, z_i)\|^2] \\ &\le -\eta_t \|\nabla L(\theta_t)\|^2 + \frac{1}{2}\eta_t^2 L_g (M + M_g \|\nabla L(\theta_t)\|^2).\end{aligned}$$

From above, the result follows. □

Lemma 4. *Under Assumption 2, for any θ, we have*

$$2c_s(L(\theta) - L_*) \le \|\nabla L(\theta)\|^2. \tag{A.9}$$

Proof. Let

$$h(\bar{\theta}) = L(\theta) + \nabla L(\theta)^T(\bar{\theta} - \theta) + \frac{1}{2}c_s\|\bar{\theta} - \theta\|^2.$$

Then $h(\bar{\theta})$ has a unique global minima at $\bar{\theta}_* = \theta - \frac{1}{c_s}\nabla L(\theta)$ with $h(\bar{\theta}_*) = L(\theta) - \frac{1}{2c_s}\|\nabla L(\theta)\|^2$. Using the strong convexity of $L(\theta)$, it follows

$$L(\theta_*) \ge L(\theta) + \nabla L(\theta)^T(\theta_* - \theta) + \frac{1}{2}c_s\|\theta - \theta_*\|_2^2 = h(\bar{\theta}_*) = L(\theta) - \frac{1}{2c_s}\|\nabla L(\theta)\|^2.$$

□

The following lemma is trivial, we omit the proof here.

Lemma 5. *Let $L_B(\theta) = \frac{1}{|B|}\sum_{z_i \in B} l_i(\theta, z_i)$. Then the variance of $\nabla L_B(\theta)$ is bounded by*

$$\mathbb{V}(\nabla L_B(\theta)) \le M/|B| + M_v\|\nabla L(\theta)\|/|B|, \qquad \textit{for all } B. \tag{A.10}$$

Proof of Theorem 1. Given these lemmas, we now proceed with the proof of Theorem 1.

Proof. Assume the batch used at step t is b_t, according to Lemma 3 and 5,

$$\begin{aligned}\mathbb{E}[L(\theta_{t+1})] - L(\theta_t) &\le -(1 - \frac{1}{2}b_t\eta_0 L_g(\frac{M_v}{b_t} + 1))b_t\eta_0\|\nabla L(\theta_t)\|^2 + \frac{1}{2}(b_t\eta_0)^2 L_g \frac{M}{b_t} \\ &\le -(1 - \frac{1}{2}\eta_0 L_g(M_v + b_t))b_t\eta_0\|\nabla L(\theta_t)\|^2 + \frac{1}{2}b_t\eta_0^2 L_g M \\ &\le -(1 - \frac{1}{2}\eta_0 L_g(M_v + B_{\max}))b_t\eta_0\|\nabla L(\theta_t)\|^2 + \frac{1}{2}b_t\eta_0^2 L_g M \\ &\le -\frac{1}{2}b_t\eta_0\|\nabla L(\theta_t)\|^2 + \frac{1}{2}b_t\eta_0^2 L_g M \\ &\le -b_t\eta_0 c_s(L(\theta_t) - L_*) + \frac{1}{2}b_t\eta_0^2 L_g M,\end{aligned}$$

where the last inequality is from Lemma 4. This yields

$$\begin{aligned}\mathbb{E}[L(\theta_{t+1})] - L_* &\le L(\theta_t) - b_t\eta_0 c_s(L(\theta_t) - L_*) + \frac{1}{2}b_t\eta_0^2 L_g M - L_* \\ &= (1 - b_t\eta_0 c_s)(L(\theta_t) - L_*) + \frac{1}{2}b_t\eta_0^2 L_g M.\end{aligned}$$

It is not hard to see,

$$\mathbb{E}[L(\theta_{t+1})] - L_* - \frac{\eta_0 L_g M}{2c_s} \le (1 - b_t\eta_0 c_s)(L(\theta_t) - L_* - \frac{\eta_0 L_g M}{2c_s}),$$

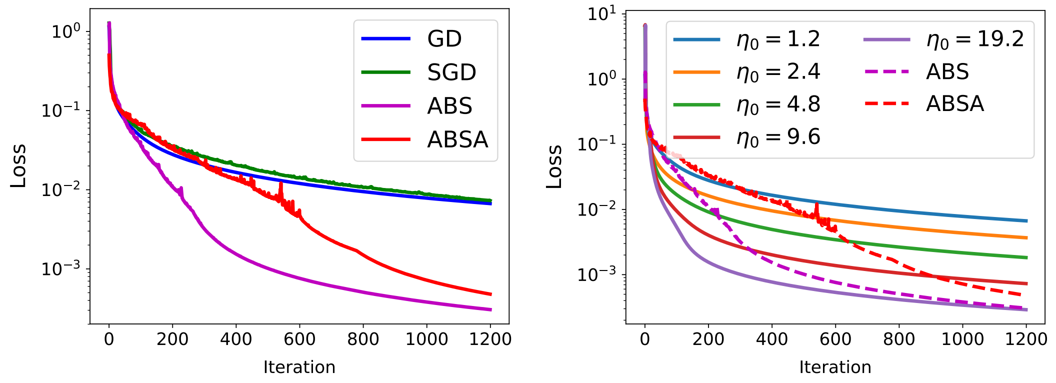

Fig. 4: *Logistic regression model result. The left figure shows the training loss as a function of iterations for full gradient, SGD, ABS and ABSA. The right figure shows the result of ABS/ABSA compared to full gradient with different learning rate.*

from which it follows:

$$\mathbb{E}[L(\theta_{t+1})] - L_* - \frac{\eta_0 L_g M}{2c_s} \leq \prod_{k=1}^{t} (1 - b_k \eta_0 c_s)(L(\theta_0) - L_* - \frac{\eta_0 L_g M}{2c_s}).$$

Therefore,

$$\mathbb{E}[L(\theta_{t+1})] - L_* \leq \prod_{k=1}^{t} (1 - b_k \eta_0 c_s)(L(\theta_0) - L_* - \frac{\eta_0 L_g M}{2c_s}) + \frac{\eta_0 L_g M}{2c_s}.$$

$\square$

We show a toy example of binary logistic regression on mushroom classification dataset[4]. We split the whole dataset to 6905 for training and 1819 for validation. $\eta_0 = 1.2$ for SGD with batch size 100 and full gradient descent. We set $100 \leq b_t \leq 3200$ for our algorithm, i.e. ABS. Here we mainly focus on the training losses of different optimization algorithms. The results are shown in Figure 4. In order to see if η_0 is not an optimal step size of full gradient descent, we vary η_0 for full gradient descent; see results in Figure 4.

Appendix B. Training Details. In this section, we give the detailed outline of our training datasets, models, strategy as well as hyper-parameter used in Alg 1.

Dataset. We consider the following datasets.

- **SVHN.** The original SVHN [30] dataset is small. However, in this paper, we choose the additional dataset, which contains more than 500k samples, as our training dataset.
- **Cifar.** The two Cifar (i.e., Cifar-10 and Cifar-100) datasets [23] have same number of images but different number of classes.
- **TinyImageNet.** TinyImageNet consists of a subset of ImangeNet images [10], which contains 200 classes. Each of the class has 500 training and 50 validation images.[5] The size of each image is 64×64.
- **ImageNet.** The ILSVRC 2012 classification dataset [10] consists of 1000 images classes, with a total of 1.2 million training images and 50,000 validation images. During training, we crop the image to 224×224.

[4]https://www.kaggle.com/uciml/mushroom-classification

[5]In some papers, this validation set is sometimes referred to as a test set.

Model Architecture. We implement the following convolution NNs. When we use data augmentation, it is exactly same the standard data augmentation scheme as in the corresponding model.

[

- **S1.** AlexNet like model on SVHN as same as Yao et al. [46][C1]. We train it for 20 epochs with initial learning rate 0.01, and decay a factor of 5 at epoch 5, 10 and 15. There is no data augmentation. The batch size to compute Hessian information is 128.
- **C1.** ResNet18 on Cifar-10 dataset [19]. We train it for 90 epochs with initial learning rate 0.1, and decay a factor of 5 at epoch 30, 60, 80. There is no data augmentation. The batch size to compute Hessian information is 128.
- **C2.** WResNet 16-4 on Cifar-10 dataset [49]. We train it for 90 epochs with initial learning rate 0.1, and decay a factor of 5 at epoch 30, 60, 80. There is no data augmentation. The batch size to compute Hessian information is 128.
- **C3.** SqueezeNext on Cifar-10 dataset [15]. We train it for 200 epochs with initial learning rate 0.1, and decay a factor of 5 at epoch 60, 120, 160. Data augmentation is implemented. The batch size to compute Hessian information is 128.
- **C4.** 2.0-SqueezeNext (twice width) on Cifar-10 dataset [15]. We train it for 200 epochs with initial learning rate 0.1, and decay a factor of 5 at epoch 60, 120, 160. Data augmentation is implemented.
- **C5.** ResNet18 on Cifar-100 dataset [19]. We training it for 160 epochs with initial learning rate 0.1, and decay a factor of 10 at epoch 80, 120. Data augmentation is implemented. The batch size to compute Hessian information is 1024.
- **I1.** ResNet50 on TinyImageNet dataset [19]. We training it for 120 epochs with initial learning rate 0.1, and decay a factor of 10 at epoch 60, 90. Data augmentation is implemented. The batch size to compute Hessian information is 2048.
- **I2.** AlexNet on ImageNet dataset [24]. We train it for 90 epochs with initial learning rate 0.01, and decay it to 0.0001 quadratically at epoch 60, then keeps it as 0.0001 for the remaining 30 epochs. Data augmentation is implemented. The batch size to compute Hessian information is 4096.
- **I3.** ResNet18 on ImageNet dataset [19]. We train it for 90 epochs with initial learning rate 0.1, and decay a factor of 10 at epoch 30, 60 and 80. Data augmentation is implemented. The batch size to compute Hessian information is 4096.

B.1. Training Strategy:. We use the following training strategies

- **BL.** Use the standard training procedure.
- **FB.** Use linear scaling rule [18] with warm-up stage.
- **GG.** Use increasing batch size instead of decay learning rate [39].
- **ABS.** Use our adaptive batch size strategy *without* adversarial training.
- **ABSA.** Use our adaptive batch size strategy *with* adversarial training.

For adversarial training, the adversarial data are generated using Fast Gradient Sign Method (FGSM) [17]. The hyper-parameters in Alg. 1 (α and β) are chosen to be 2, $\kappa = 10$, $\epsilon_{adv} = 0.005$, $\gamma = 20\%$, and $\omega = 2$ for all the experiments. The only change is that for SVHN, the frequency to compute Hessian information is 65536 training examples as compared to one epoch, due to the small number of total training epochs (only 20).

Appendix C. Simulated Training Time.

As discussed above, the number of SGD updates does not necessarily correlate with wall-clock time, and this is particularly the case because our method require Hessian backpropagation. Here, we use the method suggested in [14], to approximate the wall-clock time of our algorithm when utilizing p parallel processes. For the ring algorithm [42] the communication time per SGD iteration for p processes is:

$$T_{comm} = 2(\alpha_{latency} \log(p) + \beta_{bandwidth} \frac{p-1}{p} |\theta|), \tag{C.1}$$

where $\alpha_{latency}$ is the network latency, $\beta_{bandwidth}$ is the inverse bandwidth, and $|\theta|$ is the size number of model parameters measured in terms of Bits. Moreover, we manually measure the wall-clock time of computing the Hessian information using our in-house code, as well as the cost of forward/backward calculations on a V100 GPU. The total time will consists of this computation time and the communication one along with Hessian computation overhead (if any). Therefore we have:

$$T_{total} = T_{comp} + T_{comm} + T_{Hess}, \tag{C.2}$$

where T_{comp} is the time to compute forward and backward propagation, T_{comm} is the time to communicate between different machine, and T_{Hess} is the time to compute top eigenvalues.

We use the latency and bandwidth values of $\alpha_{latency} = 2\ \mu s$, and $\beta_{bandwidth} = \frac{1}{6\ Gb/s}$ based on NERSC's Cori2 supercomputing platform. Based on above formulas, we give an example of simulated computation time cost of I3 on ImageNet. Note that for large processes and small latency terms, the communication time formula is simplified as,

$$T_{comm} = 2\beta_{bandwidth}|\theta|. \tag{C.3}$$

In Table 3 we report the simulation time of I3 on ImageNet on 512 processes. For GG, we assume it increases batch size by a factor of 10 at epoch 30, 60 and 80. The batch size per GPU core is set to 16 for SGD (and 8 for Hessian computation due to memory limit) and the total batch size used for Hessian computation is set to 4096 images. The T_{comp} and T_{comm} is for one SGD update and T_{Hess} is for one complete Hessian eigenvalue computation (including communication for Hessian computation). Note that the total Hessian computation time for ABS/ABSA is only $1.15 \times 90 = 103.5\ s$ even though the Hessian computation is not efficiently implemented in the existing frameworks. Note that even with the additional Hessian overhead ABS/ABSA is still much faster than BL (and these numbers are with an in-house and not highly optimized code for Hessian computations). We furthermore note that we have added the additional computational overhead of adversarial computations to the ABSA method.

Table 3: *The breakdown of one SGD update training time in terms of forward/backwards computation (T_{comp}), one step communication time (T_{comm}), one total Hessian spectrum computation (if any T_{Hess}), and the total training time. The results correspond to I3 model on ImageNet (for accuracy results please see Figure 3).*

Method	T_{comp}	T_{comm}	T_{Hess}	Total Time
BL	2.2E-2	1.5E-2	0.0	16666
GG	2.2E-2	1.5E-2	0.0	6150 (2.71× faster)
ABS	2.2E-2	1.5E-2	1.15	2666 (6.25× faster)
ABSA	3.6E-2	1.5E-2	1.15	3467 (4.80× faster)

Appendix D. Additional empirical results. In this section, we present additional empirical results that were discussed in Section 4 (i.e., Table 1-2, and Figure 3).

Table 4: *Accuracy and the number of parameter updates of S1 on SVHN.*

	BL		FB		GG		ABS		ABSA	
BS	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters
128	94.90	81986	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
512	94.76	20747	95.24	20747	95.49	51862	95.65	25353	**95.72**	24329
2048	95.17	5186	95.00	5186	95.59	45935	95.51	10562	**95.82**	16578
8192	93.73	1296	19.58	1296	**95.70**	44407	95.56	14400	95.61	7776
32768	91.03	324	10.00	324	95.60	42867	95.60	7996	**95.90**	12616
131072	84.75	81	10.00	81	95.58	42158	95.61	11927	**95.92**	11267

Table 5: *Accuracy and the number of parameter updates of C1 on Cifar-10.*

	BL		FB		GG		ABS		ABSA	
BS	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters
128	83.05	35156	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
640	81.01	7031	**84.59**	7031	83.99	16380	83.30	10578	84.52	9631
3200	74.54	1406	78.70	1406	84.27	14508	83.33	6375	**84.42**	5168
5120	70.64	878	74.65	878	83.47	14449	83.83	6575	**85.01**	6265
10240	68.75	439	30.99	439	83.68	14400	83.56	5709	**84.29**	7491
16000	67.88	281	10.00	281	84.00	14383	83.50	5739	**84.24**	5357

Table 6: *Accuracy and the number of parameter updates of C2 on Cifar-10.*

	BL		FB		GG		ABS		ABSA	
BS	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters
128	87.64	35156	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
640	86.20	7031	87.9	7031	87.84	16380	87.86	10399	**89.05**	10245
3200	82.59	1406	73.2	1406	87.59	14508	88.02	5869	**89.04**	4525
5120	81.40	878	63.27	878	87.85	14449	87.92	7479	**88.64**	5863
10240	79.85	439	10.00	439	87.52	14400	87.84	5619	**89.03**	3929
16000	81.06	281	10.00	281	88.28	14383	87.58	9321	**89.19**	4610

Table 7: *Accuracy and the number of parameter updates of C4 on Cifar-10.*

	BL		FB		GG		ABS		ABSA	
BS	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters
128	93.25	78125	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
256	93.81	39062	89.51	39062	93.53	50700	93.47	43864	**93.76**	45912
512	93.61	19531	89.14	19531	93.87	37058	93.54	25132	**93.57**	28494
1024	91.51	9766	88.69	9766	93.21	31980	**93.40**	19030	93.26	22741
2048	87.90	4882	88.03	4882	93.54	30030	93.40	21387	**93.50**	23755
4096	80.77	2441	86.21	2441	93.32	29191	93.55	14245	**93.82**	20557

Table 8: *Accuracy and the number of parameter updates of C5 on Cifar-100.*

	BL		FB		GG		ABS		ABSA	
BS	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters	Acc.	# Iters
128	67.67	62500	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A
256	67.12	31250	**67.89**	31250	66.79	46800	67.71	33504	67.32	33760
512	66.47	15625	67.83	15625	67.74	39000	67.68	32240	**67.87**	24688
1024	64.7	7812	67.72	7812	67.17	35100	65.31	22712	**68.03**	13688
2048	62.91	3906	67.93	3906	67.76	33735	64.69	25180	**68.43**	12103

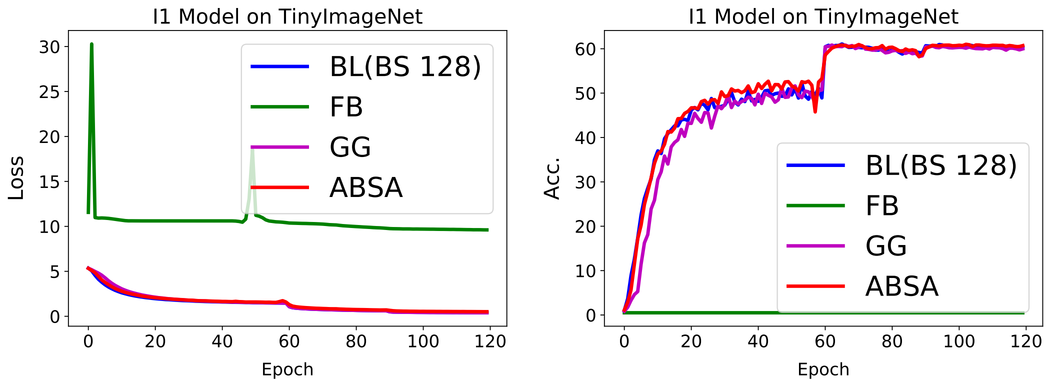

Fig. 5: *I1 model on TinyImageNet. Training set loss (left), and testing set accuracy (right), evaluated as a function of epochs. All results correspond to batch size 16384 (please see Table 2 for details). As one can see, from epoch 60 to 80, the test performance drops due to overfitting. However, ABSA achieves the best performance with apparently less overfitting (it has higher training loss).*

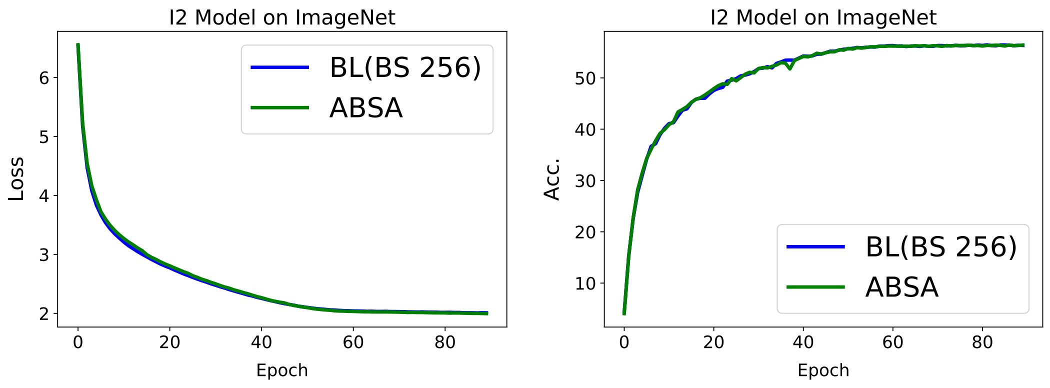

Fig. 6: *I2 model on ImageNet. Training set loss (a), and testing set accuracy (b), evaluated as a function of epochs.*

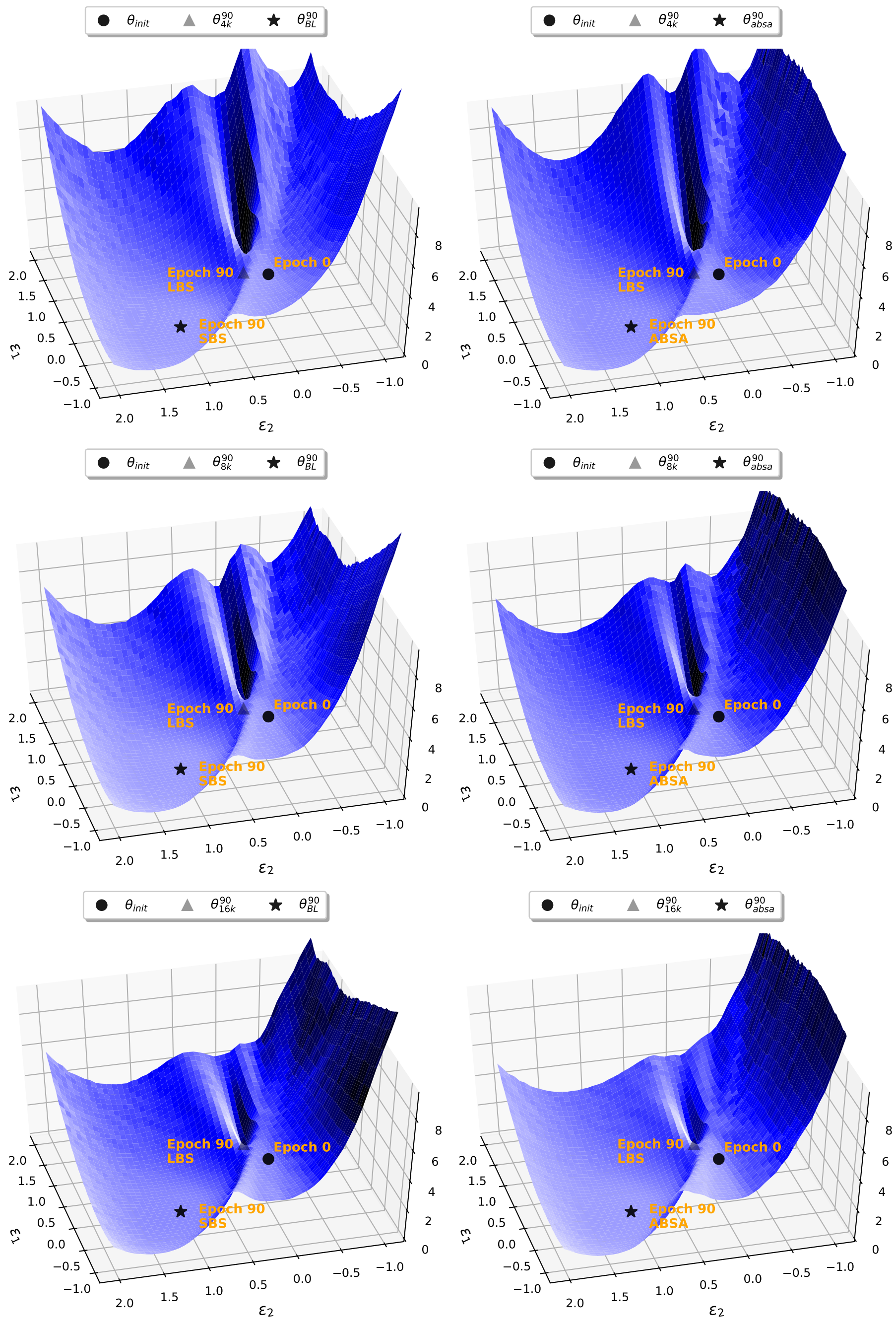

Fig. 7: *(left) 3D parametric plot for C1 model on Cifar-10. Points are labeled with the number of epochs (e.g. 90) and the technique that was used to arrive at that point (e.g. large batch size, LBS). The ϵ_1 direction shows how the loss changes across the path between initial model at epoch 0, and the final model achieved with Large Batch Size (LBS) of $B = 4K,\ 8K,\ 16K$. Similarly, ϵ_2 direction computes loss when the model parameters are interpolated between epoch 0 and final model at epoch 90 with Small Batch Size(SBS). Notice the sharp curvature that Large Batch Size gets attracted to. On the right we show a similar plot except that we use ABSA algorithm with final batch of $16K$ rather than SGD with a small batch size for interpolating the ϵ_2 direction. Notice the visual similarity between the point that ABSA converges to after 90 epochs (ABSA, 84.24% accuracy) and the point that small batch SGD (SBS, 83.05% accuracy) converges to after 90 epochs. Also note, that both avoid the sharp landscape that large batch gets attracted to sharper landscapes.*

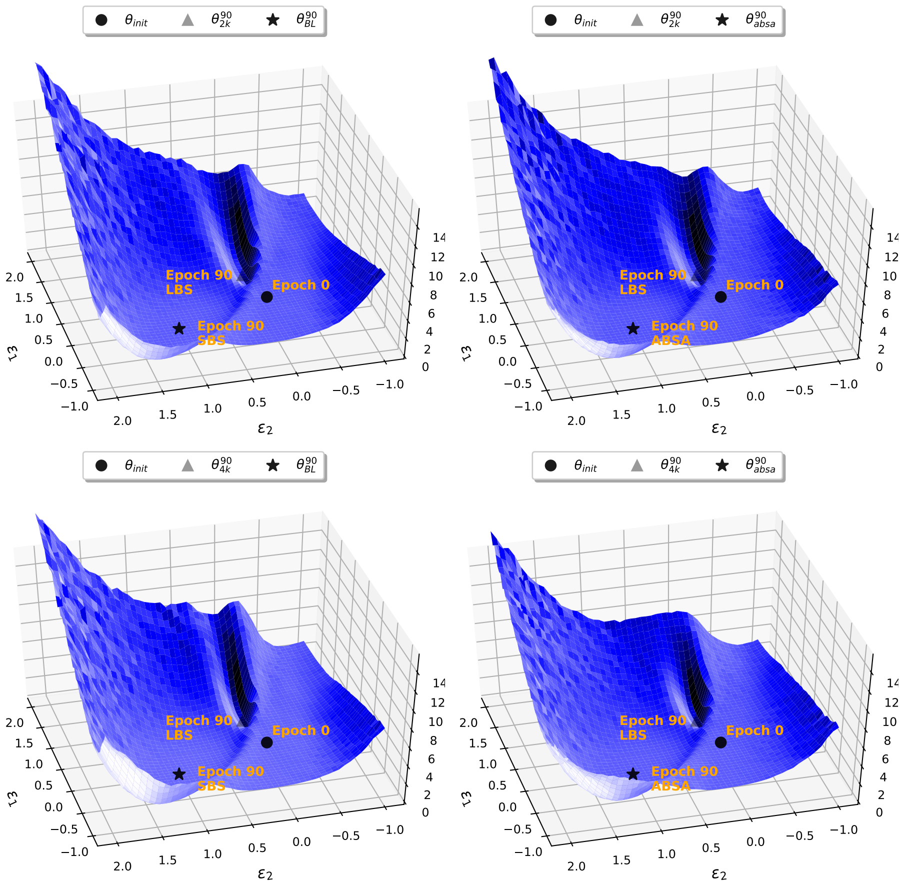

Fig. 8: *(left) 3D parametric plot for C2 model on Cifar-10. Points are labeled with the number of epochs (e.g. 90) and the technique that was used to arrive at that point (e.g. large batch size, LBS). The ϵ_1 direction shows how the loss changes across the path between initial model at epoch 0, and the final model achieved with Large Batch Size (LBS) of $B = 4K,\ 8K,\ 16K$. Similarly, ϵ_2 direction computes loss when the model parameters are interpolated between epoch 0 and final model at epoch 90 with Small Batch Size(SBS). Notice the sharp curvature that Large Batch Size gets attracted to. On the right we show a similar plot except that we use ABSA algorithm with final batch of $16K$ rather than SGD with a small batch size for interpolating the ϵ_2 direction. Notice the visual similarity between the point that ABSA converges to after 90 epochs (ABSA, 89.19% accuracy) and the point that small batch SGD (SBS, 87.64% accuracy) converges to after 90 epochs. Also note, that both avoid the sharp landscape that large batch gets attracted to sharper landscapes.*

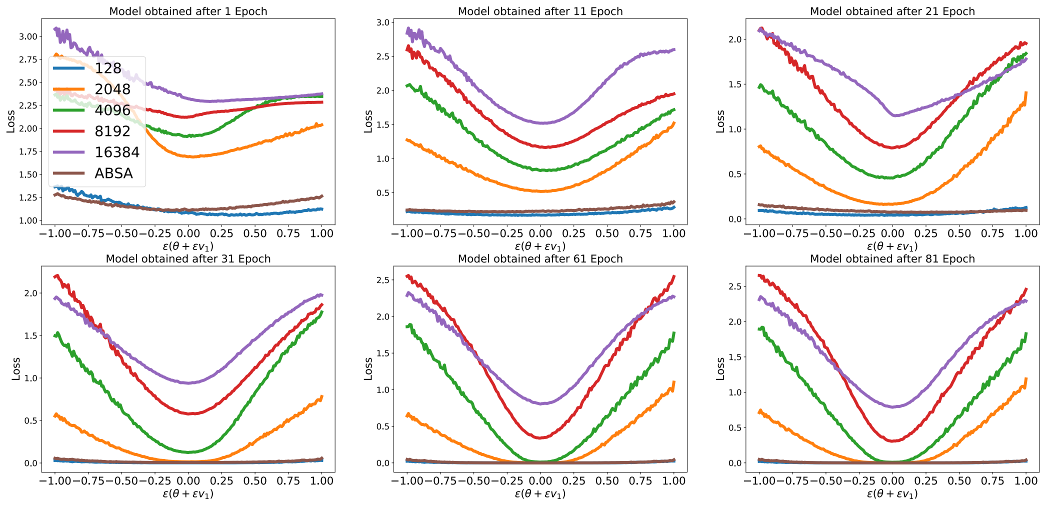

Fig. 9: *The landscape of the loss is shown when the model parameters are perturbed along the dominant Hessian eigenvector, v_1, for C1 on Cifar-10 dataset. Here ϵ is a scalar that perturbs the model parameters along v_1. The top rows show the landscape for epochs 1, 11, and 21 and the bottom row shows epochs 31, 61, and 81. One can clearly see, that larger batches continuously get stuck/attracted in areas with larger curvature whereas small batch SGD or ABSA (with batch of 16K) can escape such landscape. For detailed generalization error please see Table 5.*

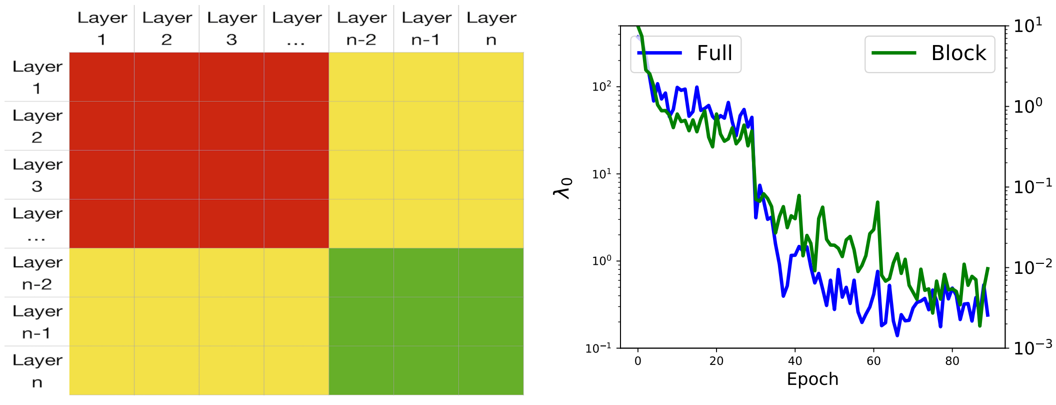

Fig. 10: *Illustration of block Hessian (left). Instead computing the top eigenvalue of whole Hessian, we just compute the eigenvalue of the green block. Top eigenvalues of Block of C1 (right) on Cifar-10. The block Hessian is computed by the last two layers of C1. The maximum batch size of C1 is 16000. The full Hessian is based on BL with batch size of 128.*