Jurnal Ilmu Komputer dan Informasi (Journal of Computer Science and Information)
14/1 (2021), 27-37. DOI: http://dx.doi.org/10.21609/jiki.v14i1.907

Reducing Adversarial Vulnerability through Adaptive Training Batch Size

Ken Sasongko, Adila Alfa Krisnadhi, Mohamad Ivan Fanany

Faculty of Computer Science, Universitas Indonesia
Email: ken.sasongko@ui.ac.id, adila@cs.ui.ac.id

Abstract

Neural networks possess an ability to generalize well to data distribution, to an extent that they are capable of fitting to a randomly labeled data. But they are also known to be extremely sensitive to adversarial examples. Batch Normalization (BatchNorm), very commonly part of deep learning architecture, has been found to increase adversarial vulnerability. Fixup Initialization (Fixup Init) has been shown as an alternative to BatchNorm, which can considerably strengthen the networks against adversarial examples. This robustness can be improved further by employing smaller batch size in training. The latter, however, comes with a tradeoff in the form of a significant increase of training time (up to ten times longer when reducing batch size from the default 128 to 8 for ResNet-56). In this paper, we propose a workaround to this problem by starting the training with a small batch size and gradually increase it to larger ones during training. We empirically show that our proposal can still improve adversarial robustness (up to 5.73%) of ResNet-56 with Fixup Init and default batch size of 128. At the same time, our proposal keeps the training time considerably shorter (only 4 times longer, instead of 10 times).

Keywords: *adversarial examples, batch normalization, fixup initialization, batch size variation*

1. Introduction

Deep learning has progressed rapidly for the last couple of years, capable of achieving super human performance. In the field of computer vision, Convolutional neural network (ConvNet) has gained momentum after AlexNet [1] won the ImageNet Challenge in 2012 [2], surpassing the performance of traditional computer vision. Despite its lack of interpretability, it has made its way to security- or safety-critical systems such as medical analysis [3, 4], face recognition [5, 6] and autonomous cars [7].

In 2013, Szegedy et al. found surprising properties of neural network [8]. One of those is that because how ConvNet processes an image, we could craft an adversarial image which has imperceptible change to human eye, but enough to make ConvNet misclassify an image. In 2014, Goodfellow et al. proposed a simple yet efficient technique, called Fast Gradient Sign Method, to craft an adversarial examples [9]. Several stronger methods have been proposed (e.g. Projected Gradient Descent Attack [10], Carlini & Wagner Attack [11], Momentum Iterative Fast Gradient Sign Method Attack [12]). These stronger attacks with larger perturbation value can reduce the accuracy of undefended ConvNet models to zero. One of the extreme examples is one-pixel attack that only modified one pixel using differential evolution algorithm [13]. One-pixel attack does not require the gradient of the model; only the probability of each label is needed to compute the perturbation and it has 31.40% success rate when performed against VGG [13]. Transferrability of adversarial perturbations between ConvNet architectures also has been observed [14].

In 2019, Galloway et al. studied the effect of Batch Normalization (BatchNorm) [16] on adversarial robustness [15] and found that by avoiding the usage of BatchNorm, adversarial robustness can be increased. They proposed to use Fixup Initialization (Fixup Init) [17] as an alternative to increase adversarial robustness.

The use of a larger batch size has been observed to reduce accuracy [18, 19]. Similarly, our experiments show that using smaller batch size reduces adversarial vulnerability, i.e., lessens the accuracy reduction due to adversarial examples. However, it makes training longer (up to 10 times from the default batch size of 128 to 8). Consequently, reduc-

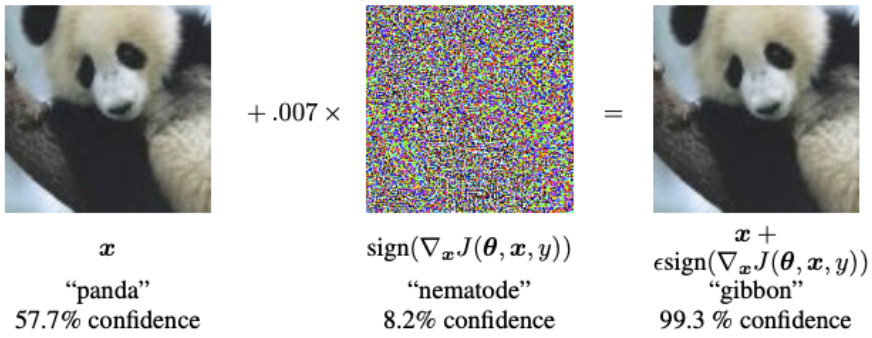


Figure 1. Example of Adversarial Attack: Fast Gradient Sign Method (FGSM) by [9]

ing batch size may not feasible in some scenario, especially when a large dataset such as ImageNet is being used. Increasing batch size has a similar effect to decaying learning rate [20]. Based on these observations we raise the following question: **Can we increase adversarial robustness by reducing the starting batch size and gradually increase it during training?** In this paper, we show that adversarial robustness of Fixup Init can be further increased by modifying the learning schedule while keeping training time reasonably low.

2. Preliminaries

2.1. Adversarial Examples

It has been argued that deep learning works by encoding a non-local generalization prior over the input space [21]. It assumes that the target function is smooth or can be approximated with a smooth function. Given a training example (x_i, y_i), a function $f(x_j)$ will outputs y_i if x_j is within a small radius of x_i. This helps the generalization because x_j might represent x_i from different point of view or different scale. By exploiting the smoothness prior, we can make a perturbation in the input space, which is small enough for human eyes not to notice but large enough to make the output jump to a region of the input space with non significant probabilities that contains no training examples in their vicinity [8]. The perturbation is usually constrained by ϵ to ensure that the perturbation is small enough. Szegedy et al. demonstrated this intriguing property by proposing a method to compute the perturbation using L-BFGS algorithm [22]. This method is able to fool ConvNet although rather weak and, considering the result, is expensive to compute by today's standard.

One of the simplest yet very efficient adversarial attack algorithms is Fast Gradient Sign Method (FGSM) [9]. This method is a pure one-shot optimization problem. It computes adversarial examples by adding one-pixel-wide perturbation in the opposite direction of the gradient of the cost function for the input. The magnitude of the perturbation is scaled by a constraint which makes the magnitude of the gradient is not important and only the direction (sign) is used. The formula to compute FGSM is defined as follows:

$$\boldsymbol{Z}^{adv} = \boldsymbol{Z} + \epsilon \cdot \text{sign}(\nabla_x J(\boldsymbol{\theta}, \boldsymbol{Z}, y)) \tag{1}$$

where $\boldsymbol{Z}^{adv}$ is the perturbed image, $\boldsymbol{Z}$ is the original image, ϵ is the constraint, J is the cost function for the input, $\boldsymbol{\theta}$ is the weight of the model, and y is the original label. This is an untargeted attack because the goal is only to maximize the error that would change the prediction.

Kurakin et al. modified FGSM to be an iterative method [10]. This method is often called Basic Iterative Method (BIM) or Projected Gradients Descent (PGD) without random start, which is formulated as follows.

$$\begin{aligned} \boldsymbol{Z}_0^{adv} &= \boldsymbol{Z} \\ \boldsymbol{Z}' &= \boldsymbol{Z}_N^{adv} + \alpha \cdot \text{sign}(\nabla_x J(\boldsymbol{\theta}, \boldsymbol{Z}_N^{adv}, y)) \\ \boldsymbol{Z}_{N+1}^{adv} &= \begin{cases} \boldsymbol{Z} - \epsilon & \text{if } \boldsymbol{Z}' < \boldsymbol{Z} - \epsilon \\ \boldsymbol{Z}' & \text{if } \boldsymbol{Z} - \epsilon \leq \boldsymbol{Z}' \leq \boldsymbol{Z} + \epsilon \\ \boldsymbol{Z} + \epsilon & \text{if } \boldsymbol{Z}' > \boldsymbol{Z} + \epsilon \end{cases} \end{aligned} \tag{2}$$

where α is the magnitude of the perturbation for each iteration and N is the number of iteration.

2.2. Batch Normalization

BatchNorm can be considered one of best discoveries for the progress of deep learning [16].

Nevertheless despite its indisputable success, there is no consensus where the benefit of using BatchNorm comes from [23, 24]. The authors of the original paper hypothesized that BatchNorm reduces the internal covariate shift problem, the distribution of activation tends to drift during training which affects subsequent layers. BatchNorm attempts to stabilize the distributions of layer inputs by controlling its mean and variance. BatchNorm can be formulated as:

$$\boldsymbol{X}' \leftarrow \frac{\boldsymbol{X} - \mu}{\sqrt{\sigma^2 + \delta}} \tag{3}$$

where μ is the mini-batch mean, σ^2 is the mini-batch variance, and δ is a small constant added for numerical stability. After normalizing, affine transformation is applied.

$$\hat{\boldsymbol{X}} \leftarrow \gamma \boldsymbol{X}' + \beta \tag{4}$$

where γ and β are learnable parameters, used to scale and shift the values respectively.

Santurkar et al. argues that BatchNorm might not even reduce internal covariate shift [23]. The success of BatchNorm comes from how BatchNorm regularize the optimization problem itself that makes the gradient smoother, thus more predictive. That is how BatchNorm allows the use of larger learning rate and faster convergence.

2.3. Fixup Initialization

Zhang et al. proposed Fixup Initialization (Fixup Init) to solve the exploding gradient problem of skip connection by scaling down the initialization [17].

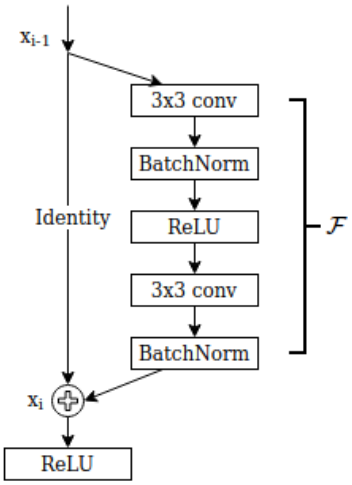


Figure 2. Residual Block from ResNet [25]

As we can see in Figure 2, in residual block, skip connection adds the original value to the final value of the block. Combining ReLU [26] and skip connection solves the vanishing gradient problem where during backpropagation, gradient information will be lost as it passes through many layers. Skip connection can be formulated as:

$$\boldsymbol{X}_i = \boldsymbol{X}_{i-1} + \mathcal{F}(\boldsymbol{X}_{i-1}, \boldsymbol{\theta}_{i-1}). \tag{5}$$

where $\boldsymbol{\theta}$ is the weight of the layers inside the block. However, combined by He Initialization [27], skip connection may double the magnitude of the variance of input to each layer. This causes the gradient to grow exponentially:

$$\begin{aligned} \text{Var}(\boldsymbol{X}_i) &= \text{Var}(\boldsymbol{X}_{i-1}) + \text{Var}(\mathcal{F}(\boldsymbol{X}_{i-1}, \boldsymbol{\theta}_i)) \\ &\approx 2\text{Var}(\boldsymbol{X}_{i-1}) \end{aligned} \tag{6}$$

Balduzzi et al. demonstrated that we can solve the exploding gradient problem by scaling down the above input to each subsequent layer [28]:

$$\boldsymbol{X}_i = \alpha \cdot (\boldsymbol{X}_{i-1} + \mathcal{F}(\boldsymbol{\theta}, \boldsymbol{X}_{i-1})) \tag{7}$$

where $\alpha = \frac{1}{\sqrt{2}}$. On the other hand, in the original ResNet, BatchNorm is employed to solve the above problem [25].

Fixup Init solves the exploding gradient problem at early stage of training by scaling down the initialization of the first convolution layer from residual block by the number of layers ($\sqrt{L}$). Scaling down alone is enough to be able to train deep network with skip connections, but the network will not be able to perform as good as ResNet with normalization layer. To match the performance of ResNet with normalization layer, Fixup Init performs the following further modifications:

- Initialize classification layer and the last convolution layer from residual block to 0.
- Add learnable parameter as multiplier to the last convolution layer from residual block.
- Add learnable parameter as bias to every linear, convolution and activation layer.

The latter two modifications above are similar to the affine transformation performed by BatchNorm.

3. Increasing Adversarial Robustness of Fixup Initialization

Our proposed solution to increase adversarial robustness of ResNet with Fixup Init through two main steps: a reduction of initial batch size, and then gradually increasing the batch size during training.

3.1. Reduce Starting Batch Size

As has been observed before, using larger batch size tends to reduce accuracy [18, 19]. Therefore first we simply reduce the batch size from 128 (FRN-56-BS128) to 32 (FRN-56-BS32) and 8 (FRN-56-BS8). We also use Linear Scaling Rule as proposed by Goyal et al. which states that "When the minibatch size is multiplied by k, multiply the learning rate by k." [19] The following formula is used to calculate the initial learning rate:

$$\eta = \eta_{orig}/bs * bs_{orig} \tag{8}$$

where η is the current learning rate, η_{orig} is the original learning rate (0.1), bs is the current batch size, and bs_{orig} is the original batch size (128). Linear Scaling Rule is also used by the original Fixup Init implementation, although the intention is to increase learning rate if larger batch size is used. Both batch size and learning rate are multiplied by the number of GPUs.

3.2. Increase Batch Size during Training

Reducing batch size to 8 significantly increases training time. Training Fixup ResNet-56 with batch size of 128 takes 36:53, while with batch size of 8 takes 6:00:26. To reduce the training time, we borrow the idea from Smith et al. to increase batch size during training [20], they proposed to increase the batch size when learning rate should be decayed and keep learning rate constant. The difference from what Smith et al. proposed is that we reduce starting batch size, and gradually increase to the original batch size before epoch 100, when the learning rate starts to be decayed. We consider the following learning schedule:

- Batch Size 8 Schedule 1 (FRN56-BS8-S1): Reduce the starting batch size to 8, gradually increase it to 512, multiply by 2 at epoch 20, 40, 60, 80, 100, and 150. The idea is to divide evenly for the first 100 epochs to increase batch size. So we multiply by 2 every 20 epochs.
- Batch Size 32 Schedule 1 (FRN56-BS32-S1): Reduce the starting batch size to 32, gradually increase it to 512, multiply by 2 at epoch 20, 40, 100, and 150. In this scenario, we simply increase batch size as early as possible.

As we can see from Figure 4a, test accuracy is still increasing at epoch 20, which is a sign that the model has not converged using the current learning rate and batch size. Thus, we also propose to modify the schedule by delaying the starting batch size multiplier from epoch 20-40-60-80-100-150 to 60-70-80-90-100-150 (Schedule 2 or FRN56-BS8-S2). Similar to FRN56-BS8-S2, we also delay the starting batch size multiplier for Fixup ResNet-56 with a batch size of 32 from 20-40-100-150 to 60-80-100-150 (FRN56-BS32-S2). By looking at Figure 4b, we can see that test accuracy is still improving at epoch 60. Thus, we further delay the starting batch size multiplier from 60-80-100-150 to 80-90-100-150 (Schedule 3 or FRN56-BS32-S3).

Because at epoch 100 and 150, batch size is multiplied by 2, learning rate is decayed by a factor of 5 as opposed to 10 from the original implementation of ResNet and Fixup ResNet [17, 25]. This will help to speed up the training while still adhering to the Linear Scaling Rule.

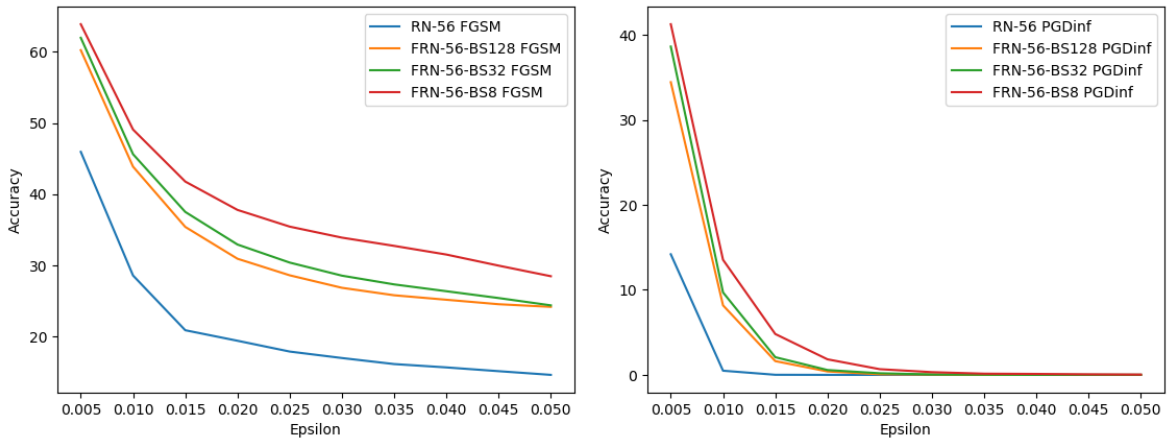
4. Experiment & Result

4.1. Experiment Configuration

The code of the experiment is written in Python 3.5.2, using PyTorch 1.3.1 and we run the experiments on a machine with Intel Core i7 8700 CPU and 2 NVIDIA RTX 2080 TI GPUs. Unless stated otherwise, we use the same configuration as the original paper for ResNet-56 [25] and Fixup ResNet-56 [17]. Fixup Init experiment does not use mixup, a data augmentation technique which can be combined with Fixup Init to improve accuracy as proposed in [17]. We set Python's random seed, NumPy seed, and PyTorch seed to 0. We use CIFAR-10 dataset with the commonly used data augmentation technique for the dataset, random horizontal flip and a 32x32 sample is randomly cropped from the 4-pixel padded image.

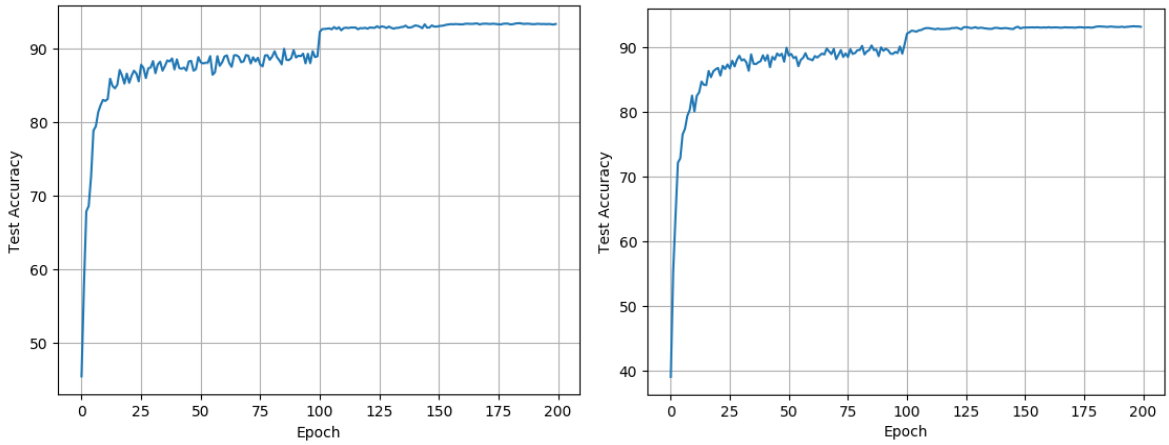
We have observed that network without BatchNorm is more sensitive to input normalization, that is if we scale the input to [0,1] range compared to normalize the input using z-score with input mean and input standard deviation, the latter has higher accuracy. But normalizing each channel with input mean and input standard deviation makes it more difficult to correctly compute the perturbation within the constraint. To simplify the computation, we normalize the input using mean of 0.45 and standard deviation of 0.25 which affects the magnitude of the ϵ. So, we multiply the ϵ by 4 to have the same magnitude of perturbation as what most of other papers reported. For example if we say $\epsilon = 0.05$, the actual ϵ is 0.2.

To check adversarial robustness, we use the implementation of FGSM and PGD-ℓ_∞ by Ding et al.



(a) FGSM **(b)** PGD-ℓ_∞

Figure 3. Result of FGSM and PGD$_\infty$ on ResNet-56, Fixup ResNet-56 with batch size of 128, 32, and 8



(a) Batch Size of 8 **(b)** Batch Size of 32

Figure 4. Test Accuracy of Fixup Init with a batch size of 32 and 8

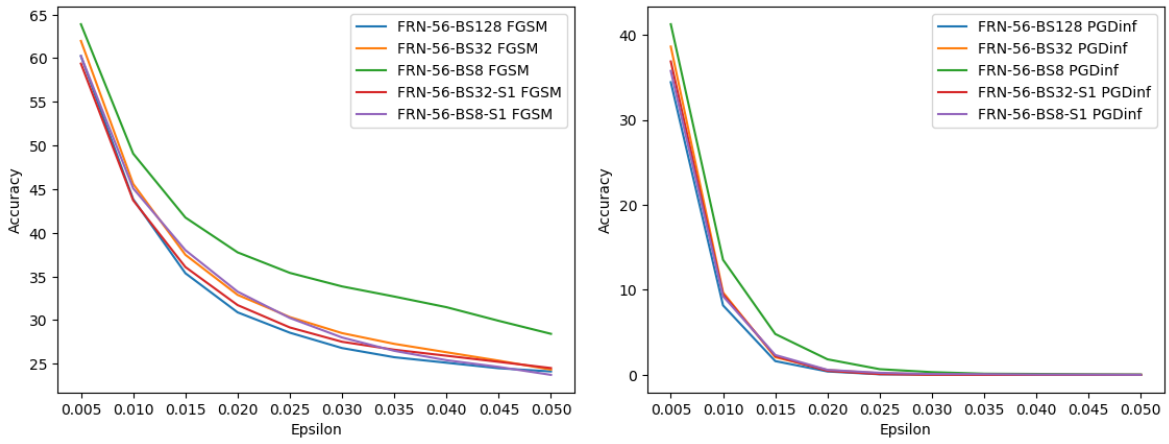
[1] [29] with ϵ from 0.005 to 0.05 with step of 0.005. For PGD-ℓ_∞, we use step size of $\epsilon/10$, iteration of 20, and disable random initialization. Using both one-shot attack and iterative attack with the same epsilons will show that the proposed solution does not exhibit the obfuscated gradient problem [30]. The code snippets to generate the attack are listed in Appendix B.

4.2. Result

First we see the adversarial robustness of the original ResNet-56 (RN56), Fixup ResNet-56 with a batch size of 128 (FRN-56-BS128), 32 (FRN-56-BS32), and 8 (FRN-56-BS8). As we can see from Figure 3a and Figure 3b, FRN-56-BS8 has the highest adversarial robustness. Attacked with FGSM, the difference is up to over 20% compared to ResNet-56 and up to over 6% compared to FRN-56-BS128, but at the cost of increased training time.

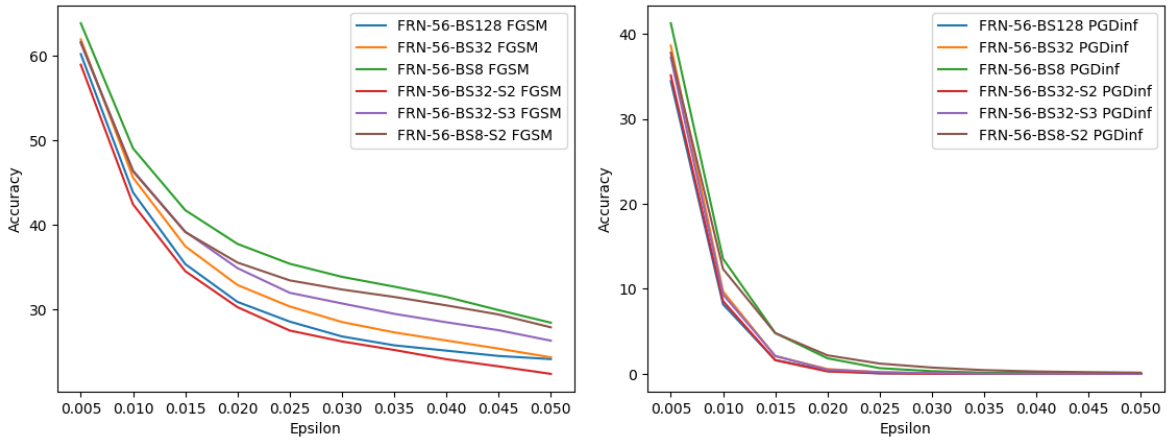
Naively modifying the learning schedule significantly reduces robustness. As seen on Figure 5a and Figure 5b, while FRN56-BS8-S1 performs slightly better than Fixup ResNet-56 with a batch size of 128, they fare far below Fixup ResNet-56 with a batch size of 8. Similarly, increasing batch size starting from epoch 20 for Fixup ResNet-56 with a batch size of 32 also reduce robustness, from

[1] https://github.com/BorealisAI/advertorch



(a) FGSM **(b)** PGD-ℓ_∞

Figure 5. Result of FGSM and PGD-ℓ_∞ on Fixup ResNet-56 with batch size of 128, 32, and 8 and Fixup ResNet-56 Schedule 1 with a batch size of 32 and 8 attacked



(a) FGSM **(b)** PGD-ℓ_∞

Figure 6. Result of FGSM and PGD-ℓ_∞ on Fixup ResNet-56 with batch size of 128, 32, and 8 and Fixup ResNet-56 Schedule 2 & 3 with a batch size of 32 and 8 attacked with PGD-ℓ_∞

the same figures we can see that FRN56-BS32-S1's performance is similar to FRN56-BS128.

As we can see from Figure 6a and Figure 6b, delaying the starting batch size multiplier to the point where test accuracy has stabilized can improve robustness. FRN56-BS8-S2's performance is the closest to FRN-56-BS8's performance, albeit slightly lower. As seen on Figure 6b, for some ϵ, FRN56-BS8-S2 even has better performance than FRN-56-BS8. And FRN56-BS8-S2 only takes 2:19:07, which is significantly faster than the time it takes to train FRN56-BS8. FRN56-BS32-S2 is another example of blindly modifying the learning schedule, where the starting increment is taken from FRN56-BS8-S2, the result shows that FRN56-BS32-S2 fails spectacularly, performs worse than even FRN56-BS128. But FRN56-BS32-S3 which is chosen by looking at Figure 4b surprisingly has better robustness than FRN56-BS32 when attacked with FGSM, and slightly lower compared to FRN56-BS32 when attacked with PGD-ℓ_∞. The detailed result of the attacks can be seen in Appendix A. Choosing the correct schedule also affects accuracy. The accuracy is shown in 1, where the accuracy of FRN56-BS32-S1 drops 1.16% from FRN56-BS32, but FRN56-BS32-S3 and FRN56-BS32 have similar accuracy

(+0.1%).

5. Related Work

A large number of research works have been conducted to devise a defense method against adversarial examples. Several techniques has been studied, such as adversarial training [9, 31], modifying network architecture [32], and modifying input [33, 34]. Defensive quantization has been proposed to improve both robustness and efficiency [35]. More recently, a number of certified defence researches have been proposed [36–38]. Certified defence is a type of defense with a proof that prediction at any point inside a small norm-bounded ball around point x will be constant.

Stutz et al. has disentangled the relationship between robustness and generalization [39]. But they also showed that on-manifold adversarial examples are a result of generalization errors, if we train with an intent to reduce on-manifold adversarial examples, it would also increase test accuracy.

In 2019, Galloway et al. showed that by using Fixup Initialization [17] instead of BatchNorm, Wide ResNet [40] is more robust against adversarial examples. Similarly, adding BatchNorm to VGG-like network improves performance at the cost of increased adversarial vulnerability [15].

6. Conclusion & Future Works

Szegedy et al. found a property of neural network that can be exploited to fool them and methods to defend against adversarial examples keep getting circumvented. For example, Athalye et al. immediately published techniques to circumvent six of adversarial defence methods after those defence papers were accepted to ICRL 2018 [30]. We do not try to improve robustness by devising a specific defence technique, but by simply naturally increase it. The increment is not large, but combined with an adversarial defence technique, it might be able to retain a degree of accuracy when the defence technique fails.

The differences between each proposed schedules are as follow.

- Schedule 1 (S1) with Batch Size of 8 divides evenly for the first 100 epochs. S1 with Batch Size of 32 simply follows the schedule but with larger starting batch size.
- Schedule 2 (S2) with Batch Size of 8 delay the starting batch size multiplier due to test accuracy is still increasing at epoch 40. Again, S2 with Batch Size of 8 simply follows the schedule.
- Schedule 3 (S3) with Batch Size of 8 further delay the starting batch size multiplier again due to test accuracy is still increasing at epoch 60.

Table 1. Training time & accuracy

Model-BS Schedule	Training Time	Accuracy
RN56-BS128	46:33 (± 15)	93.38%
FRN56-BS128	37:02 (± 12)	93.24%
FRN56-BS32	1:39:19 (± 21)	93.29%
FRN56-BS8	6:00:43 (± 47)	93.45%
FRN56-BS32-S1	41:16 (± 13)	92.34%
FRN56-BS32-S2	52:49 (± 14)	93.13%
FRN56-BS32-S3	58:53 (± 14)	93.39%
FRN56-BS8-S1	1:25:33 (± 15)	92.48%
FRN56-BS8-S2	2:18:51 (± 17)	93.28%

As our experiment has shown, we can improve the adversarial robustness of Fixup ResNet56 by simply reducing batch size. But reducing batch size from the default 128 to 8 increases training time by a magnitude of approximately 10 (from 37:02 to 6:00:43). Modifying the learning schedule by increasing batch size during training can greatly reduce training time, while keeping the adversarial robustness closer to FRN56-BS8. However, randomly picking the learning schedule tends to worsen adversarial robustness (i.e., Schedule 1).

Table 1 shows the training time and clean accuracy of each proposed schedule and the baseline model. Training time is an average of three with standard deviation shown next to it. As we can see from Table 1, compared to Fixup ResNet-56 with a batch size of 8, can halve the training time while slightly reducing adversarial robustness and even in some cases improve robustness and accuracy. Compared to RN56, FRN56 has a faster training time due to less layers in the architecture (i.e., BatchNorm layer is removed).

We have empirically proven that this method increases robustness. We use one architecture (ResNet-56) and one dataset (CIFAR-10), this is a standard benchmark dataset for training many deep learning architectures. We need to conduct in-depth analysis to better understand this behaviour. Furthermore, the schedule is handpicked by looking at the progress of test accuracy during normal training, we might be able to automate this by implementing a technique similar to early stopping but increase the batch size when test accuracy has not been improved for a number of epochs.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[3] M. Helmstaedter, K. L. Briggman, S. C. Turaga, V. Jain, H. S. Seung, and W. Denk, "Connectomic reconstruction of the inner plexiform layer in the mouse retina," *Nature*, vol. 500, no. 7461, pp. 168–174, 2013.

[4] H. Y. Xiong, B. Alipanahi, L. J. Lee, H. Bretschneider, D. Merico, R. K. Yuen, Y. Hua, S. Gueroussov, H. S. Najafabadi, T. R. Hughes *et al.*, "The human splicing code reveals new insights into the genetic determinants of disease," *Science*, vol. 347, no. 6218, p. 1254806, 2015.

[5] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.

[6] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, "Arcface: Additive angular margin loss for deep face recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4690–4699.

[7] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, 2020.

[8] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[9] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[10] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," *arXiv preprint arXiv:1611.01236*, 2016.

[11] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 39–57.

[12] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, and J. Li, "Boosting adversarial attacks with momentum," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 9185–9193.

[13] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Transactions on Evolutionary Computation*, 2019.

[14] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1765–1773.

[15] A. Galloway, A. Golubeva, T. Tanay, M. Moussa, and G. W. Taylor, "Batch normalization is a cause of adversarial vulnerability," *arXiv preprint arXiv:1905.02161*, 2019.

[16] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[17] H. Zhang, Y. N. Dauphin, and T. Ma, "Fixup initialization: Residual learning without normalization," *arXiv preprint arXiv:1901.09321*, 2019.

[18] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *arXiv preprint arXiv:1609.04836*, 2016.

[19] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[20] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, "Don't decay the learning rate, increase the batch size," *arXiv preprint arXiv:1711.00489*, 2017.

[21] Y. Bengio *et al.*, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[22] D. C. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical programming*, vol. 45, no. 1-3, pp. 503–528, 1989.

[23] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?" in *Advances in Neural Information Processing Systems*, 2018, pp. 2483–2493.

[24] N. Bjorck, C. P. Gomes, B. Selman, and K. Q. Weinberger, "Understanding batch normalization," in *Advances in Neural Information Processing Systems*, 2018, pp. 7694–7705.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in

Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[26] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[28] D. Balduzzi, M. Frean, L. Leary, J. Lewis, K. W.-D. Ma, and B. McWilliams, "The shattered gradients problem: If resnets are the answer, then what is the question?" in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 342–350.

[29] G. W. Ding, L. Wang, and X. Jin, "Advertorch v0. 1: An adversarial robustness toolbox based on pytorch," *arXiv preprint arXiv:1902.07623*, 2019.

[30] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," *arXiv preprint arXiv:1802.00420*, 2018.

[31] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[32] A. S. Ross and F. Doshi-Velez, "Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients," in *Thirty-second AAAI conference on artificial intelligence*, 2018.

[33] G. K. Dziugaite, Z. Ghahramani, and D. M. Roy, "A study of the effect of jpg compression on adversarial images," *arXiv preprint arXiv:1608.00853*, 2016.

[34] E. Raff, J. Sylvester, S. Forsyth, and M. McLean, "Barrage of random transforms for adversarially robust defense," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 6528–6537.

[35] J. Lin, C. Gan, and S. Han, "Defensive quantization: When efficiency meets robustness," *arXiv preprint arXiv:1904.08444*, 2019.

[36] A. Raghunathan, J. Steinhardt, and P. Liang, "Certified defenses against adversarial examples," *arXiv preprint arXiv:1801.09344*, 2018.

[37] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, "Certified robustness to adversarial examples with differential privacy," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 656–672.

[38] J. M. Cohen, E. Rosenfeld, and J. Z. Kolter, "Certified adversarial robustness via randomized smoothing," *arXiv preprint arXiv:1902.02918*, 2019.

[39] D. Stutz, M. Hein, and B. Schiele, "Disentangling adversarial robustness and generalization," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2019.

[40] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

Appendix A.
Details of Adversarial Robustness

Table 2. Fixup ResNet-56 - FGSM

	Epsilon (ϵ)									
BS Schedule	0.005	0.01	0.015	0.02	0.025	0.03	0.035	0.04	0.045	0.05
128-Fixed	60.24%	43.85%	35.37%	30.89%	28.56%	26.8%	25.75%	25.12%	24.49%	24.12%
32-Fixed	61.98%	45.60%	37.48%	32.89%	30.35%	28.51%	27.28%	26.32%	25.36%	24.33%
8-Fixed	63.89%	49.07%	41.75%	37.76%	35.41%	33.86%	32.7%	31.48%	29.92%	28.43%
BS32-S1	59.38%	43.73%	36.08%	31.72%	29.16%	27.52%	26.61%	25.92%	25.22%	24.51%
BS32-S2	58.98%	42.44%	34.53%	30.27%	27.5.%	26.21%	25.2%	24.1%	23.26%	22.37%
BS32-S3	61.52%	46.44%	39.21%	34.89%	31.97%	30.71%	29.48%	28.48%	27.54%	26.3%
BS8-S1	60.25%	45.09%	38.01%	33.26%	30.25%	28.01%	26.48%	25.41%	24.64%	23.73%
BS8-S2	61.67%	46.35%	39.16%	35.55%	33.44%	32.36%	31.48%	30.48%	29.4%	27.88%

Table 3. Fixup ResNet-56 - PGD-ℓ_∞

	Epsilon (ϵ)									
BS Schedule	0.005	0.01	0.015	0.02	0.025	0.03	0.035	0.04	0.045	0.05
128-Fixed	34.44%	8.18%	1.62%	0.4%	0.07%	0.04%	0.00%	0.00%	0.00%	0.00%
32-Fixed	38.63%	9.70%	2.08%	0.57%	0.19%	0.05%	0.02%	0.00%	0.00%	0.00%
8-Fixed	41.26%	13.54%	4.81%	1.83%	0.67%	0.32%	0.14%	0.11%	0.06%	0.03%
BS32-S1	36.88%	9.49%	2.20%	0.47%	0.09%	0.01%	0.00%	0.00%	0.00%	0.00%
BS32-S2	35.77%	9.25%	2.34%	0.59%	0.24%	0.10%	0.08%	0.04%	0.03%	0.01%
BS32-S3	37.21%	9.34%	2.11%	0.49%	0.20%	0.08%	0.01%	0.01%	0.01%	0.00%
BS8-S1	39.71%	10.45%	2.57%	0.74%	0.25%	0.06%	0.02%	0.02%	0.01%	0.00%
BS8-S2	37.78%	12.34%	4.79%	2.18%	1.21%	0.74%	0.44%	0.28%	0.20%	0.14%

Appendix B.
Attack Implementation

The code is written in python using PyTorch and to attack we use AdverTorch by Ding et al. [29]. A code snippet to iterate and construct the FGSM attack object is shown below:

Listing 1. FGSM code snippet

```
epsilons = np.arange(0.005, 0.051, 0.005)

for eps in epsilons:
    eps = round(eps, 3)

    adversary = GradientSignAttack(
        model, loss_fn=nn.CrossEntropyLoss(reduction="sum"),
        eps=eps * 4,
        clip_min=-1.8, clip_max=2.2,
        targeted=False)
```

A code snippet to iterate and construct PGD-ℓ_∞ attack object is shown below:

Listing 2. PGD-ℓ_∞ code snippet

```
epsilons = np.arange(0.005, 0.051, 0.005)

for eps in epsilons:
    eps = round(eps, 3)
    adversary = LinfPGDAttack(
        model, loss_fn=nn.CrossEntropyLoss(reduction="sum"),
```

```
eps=eps * 4, nb_iter=20,
eps_iter=(eps * 4) / 10, rand_init=False,
clip_min=-1.8, clip_max=2.2,
targeted=False)
```