Adversarial Attack and Defense Based on Data Mixup

Ruihang Lai
ACM Class, Zhiyuan College
Shanghai Jiao Tong University
masterjh5574@sjtu.edu.cn

Hongyi Jin
ACM Class, Zhiyuan College
Shanghai Jiao Tong University
jinhongyi@sjtu.edu.cn

Jiaxin Lu
ACM Class, Zhiyuan College
Shanghai Jiao Tong University
lujiaxin@sjtu.edu.cn

1 Introduction

Deep neural networks have shown their powerful applications in various fields including computer vision and natural language processing. However, despite their promising performance, neural networks can be easily attacked by small perturbations on the input. Many approaches [11, 1, 5, 14] are proposed to conduct such attacks, the most powerful one among them can reduce the model accuracy to zero.

One "universal" first-order adversary is proposed in [11] called projected gradient descent (PGD). It is derived from prior method Fast Gradient Sign Method (FGSM) [1] by taking the one-step scheme to the multi-step variant, which is essentially projected gradient descent on the negative loss function,

$$x^{t+1} = \prod_{x+\mathcal{S}} (x^t + \alpha \cdot \text{sgn}(\nabla_x L(\theta, x, y)))$$

where $\mathcal{S}$ represents a set of allowed perturbations and α be the maximum step size.

To tackle these adversaries, some techniques have been proposed to increase the robustness of these models. Zhang et al. [16] introduced a simple and data-agnostic data augmentation routine called *mixup* which constructs virtual training samples

$$\tilde{x} = \lambda x_i + (1-\lambda)x_j, \text{where } x_i, x_j \text{ are raw input vectors}$$
$$\tilde{y} = \lambda y_i + (1-\lambda)y_j, \text{where } y_i, y_j \text{ are one-hot label encodings}$$

where (x_i, y_i) and (x_j, y_j) are two samples randomly drawn from original training data and $\lambda \in (0, 1)$. This sampling method helps model to extend its training data distribution, allowing linear interpolations of different targets, which increase the model understanding of corrupted inputs. Based on this idea, Verma et al. [15] introduced *manifold mixup*, which basically move the "mixup" sampling scheme to the model's hidden layers.

Another technique is to use soft labels. That is, instead of directly assigning the class label, we use a score to represent the probability or likelihood of the object belonging to one class. This implies that one object can be a member of multiple classes. Previous work [10, 12] have addressed the potential of this method on tolerating noise.

From a machine learning perspective, since we have had a strong attack model, the idea of adversarial training is introduced to defense the attacks. That is, one first feed samples into adversary to perturb the input and then the model learn perturbed samples. Madry et al. [11] gives a saddle point problem view of this process, which is the central object of adversarial training:

$$\min_\theta \rho(\theta), \text{ where } \rho(\theta) = \mathbb{E}_{(x,y)\sim D}\left[\max_{\delta\in\mathcal{S}} L(\theta, x', y)\right]$$

Actually, the previous attack model PGD can be viewed as specific attempts to solve the inner maximization problem.

Based on methods above, in this report, we have done the following work. [1]

- We first train *mixup* and *manifold mixup* model separately and then test their performance when facing PGD attacks. Meanwhile, we visualize the PGD attacked data to observe how PGD affect the model accuracy.
- We then conduct adversarial training with *mixup* and *manifold mixup* and compare their performance to the vanilla model on *clean* and attacked input.
- We further implement soft label technique on adversarial training with *mixup*. We also test how different mixup policy influence the performance of the model.

2 Proposed Model

We choose MNIST [8] and CIFAR-10 [4] as our dataset, with pixels of images in CIFAR-10 being normalized from 0-255 to 0-1. We simply use LeNet5 [7] for MNIST dataset and use PreAct-ResNet18 [2, 3] for CIFAR-10 dataset.

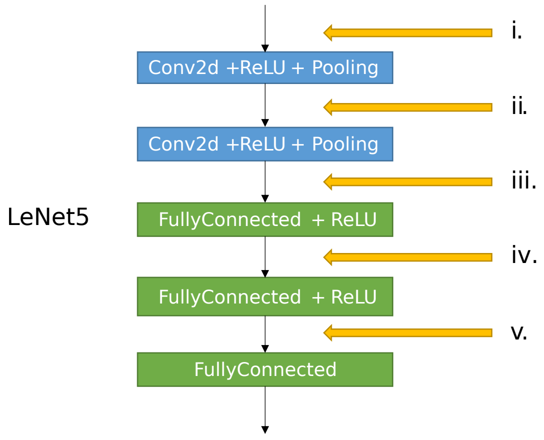


Figure 1: Location of mixup and manifold mixup. Take LeNet5 as an example. For mixup, we perform data mixup at position (i.). For manifold mixup, we first pick a position from (i.) to (v.) uniformly at random, and then perform data mixup at the position we pick. Each time a batch of data is fed to the model, we will pick such a position, which introduces a lot of randomness.

2.1 Mixup and Manifold Mixup

As Fig. 1 shows, every time before a batch of data is fed to the model, we randomly pick a position from a layer set $\mathcal{S}$. If manifold mixup is adopted, $\mathcal{S} = \{\text{all layer}\}$. Otherwise if mixup is adopted, $\mathcal{S} = \{\text{the first layer}\}$.

After selecting a layer for mixup, the remaining problem is to perform the mixup. The mixup process is shown in Fig. 2. Since we cannot perform mixup with only one batch, we make a copy of the original copy and shuffle this copy randomly. Then we multiply the original copy by λ, multiply the shuffled copy by $(1 - \lambda)$, and finally add these two multiplication results. This yields the mixed copy, which is what we desired.

Observe that to mix the data, we should first convert the labels of dataset to one-hot labels. If not, for instance, the $\frac{1}{2}$-$\frac{1}{2}$ mixed result of label 3 and label 6 is $\frac{1}{2} \cdot 3 + \left(1 - \frac{1}{2}\right) \cdot 6 = 4.5$. But one cannot say that in MNIST dataset, the mixed image of 3 and 6 is an image of number 4.5. Therefore, the label conversion is necessary.

[1]Code and results are available at https://github.com/MasterJH5574/MS326-Project-2020.

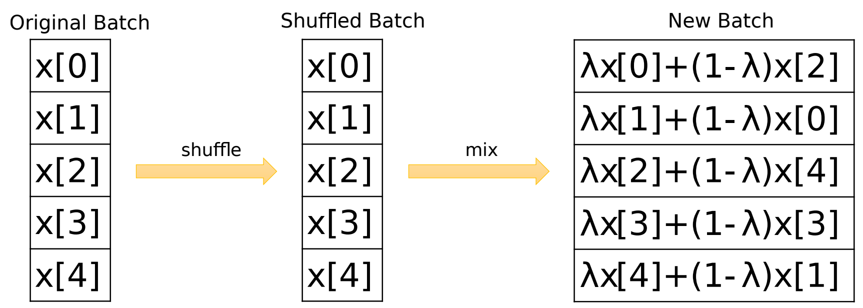


Figure 2: The mixup process. The original batch was mixed with the shuffled batch by the parameter λ.

2.2 Adversarial Training

Recall the adversarial training is a process that optimizes the function

$$\min_{\theta} \rho(\theta), \text{ where } \rho(\theta) = \mathbb{E}_{(x,y)\sim D}\left[\max_{\delta\in\mathcal{S}} L(\theta, x', y)\right].$$

This means that we need to perform PGD (corresponding to the $\max$ operation) and model update (corresponding to the $\min$ operation) alternately to make the process converge.

Algorithm 1 describes how we perform adversarial training in the vanilla models. We first set the number of iterations. Then in each iteration, we randomly sample a batch without replacement. This ensures that each data can be sampled for approximately the same number of times. After that, we perform PGD attack on the whole batch, and update the model by the PGD-attacked batch.

Algorithm 1: Adversarial training

```
1 for $i = 0 : num\_iter$ do
2     sample a batch $(X, y)$ from dataset;
3     apply PGD attack on the batch and get $(X', y)$ ;
4     update model by $(X', y)$.
5 end
```

To combine adversarial training and mixup, we can just replace our model with its mixup or manifold mixup versions. The algorithm flow remains the same.

3 Experiments

3.1 Train and Test with Clean Data

At the very beginning, for each dataset/network, we use the clean training data to train three models: one with no mixup (namely "vanilla model"), one with mixup and one with manifold mixup. After training, we examine the prediction accuracies of each model on the clean test data.

For MNIST and LeNet5, we train the model for 500 epochs. Learning rate is set to 0.1 at the beginning, and will decay to 0.01 and 0.001 after 300 and 400 epochs respectively. For CIFAR-10 and PreAct-Resnet, we train the model for 2000 epochs. The initial learning rate is 0.1, and will decrease to 0.01 and 0.001 after 500 and 1000 epochs. Both models use the SGD optimizer with momentum 0.9.

Table 1, Fig. 3, Fig. 4 show the results of LeNet5. And Table 2, Fig. 5, Fig. 6 show the results of PreAct-ResNet18.

By referring to the above tables and figures, we can immediately conclude that the manifold mixup model performs better than the mixup model, and the mixup model performs better than the vanilla

Model	Accuracy on Test Data (%)	Cross-Entropy Loss
Vanilla	99.30	0.022
Mixup	99.31	0.048
Manifold Mixup	**99.44**	0.076

Table 1: Accuracies and losses of vanilla model, mixup model and manifold mixup model on clean MNIST test data.

Model	Accuracy on Test Data (%)	Cross-Entropy Loss
Vanilla	95.58	0.172
Mixup	96.89	0.186
Manifold Mixup	**97.19**	0.161

Table 2: Accuracies and losses of vanilla model, mixup model and manifold mixup model on clean CIFAR-10 test data.

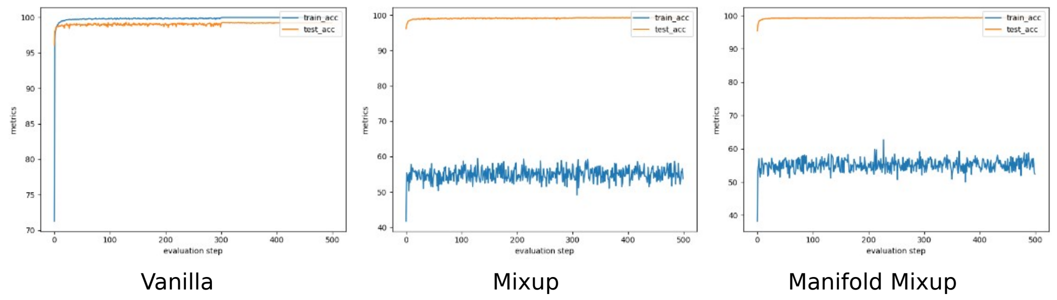


Figure 3: Accuracy curves of three models on clean MNIST test data.

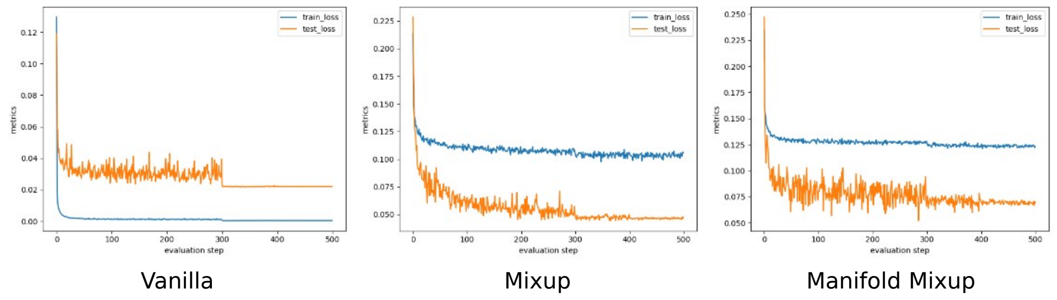


Figure 4: Loss curves of three models on clean MNIST test data.

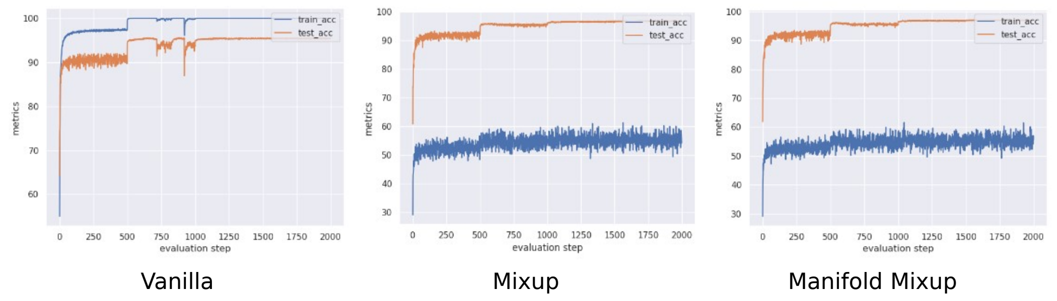


Figure 5: Accuracy curves of three models on clean CIFAR-10 test data.

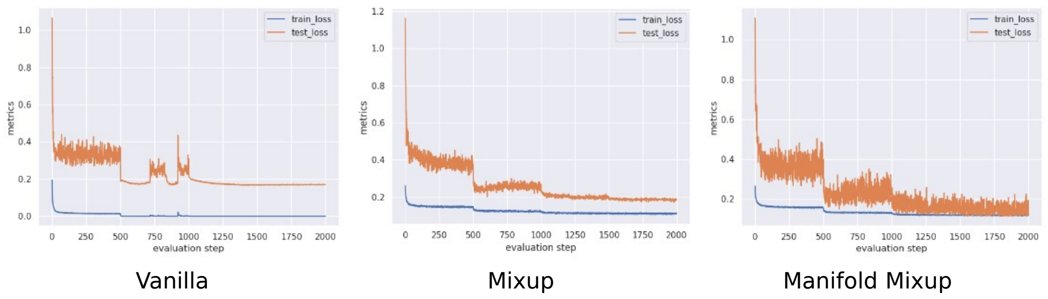


Figure 6: Loss curves of three models on clean CIFAR-10 test data.

Model	Accuracy (%) MNIST	Accuracy (%) CIFAR-10
Vanilla	0.00	5.01
Mixup	0.00	5.02
Manifold Mixup	**0.01**	**5.06**

Table 3: The accuracy of the three different models on attacked MNIST and CIFAR-10 test data. All models are attacked by PGD attack. The accuracy of CIFAR-10 is higher than MNIST due to the fact that the CIFAR-10 dataset is more complex than the MNIST dataset.

model, when both trained and tested with clean data, namely,

$$\text{acc}_{\text{manifold mixup}} > \text{acc}_{\text{mixup}} > \text{acc}_{\text{vanilla}}.$$

A fun phenomenon is that no matter what the dataset is, the mixup and manifold mixup model have much lower training accuracies (and higher training losses) than the vanilla model. Moreover, the manifold mixup model has a much more fluctuating training loss than the mixup model, and the loss of the mixup model fluctuates more greatly than the vanilla model. This is because in the mixup model, each time a batch of data will be randomly shuffled. This makes the data fed to the model are always different, new, and mixed. And models may not predict correctly when given a bunch of totally different data. Manifold mixup model will pick a random position to perform mixup, which introduces huge randomness again. The randomness is the reason why mixup and manifold mixup models have fluctuating training accuracies and losses.

3.2 PGD Attack

MNIST We set the parameters $\epsilon = 0.3, \alpha = 0.1$, and the number of iterations is 40. Fig. 7 shows the original images and the attacked images.

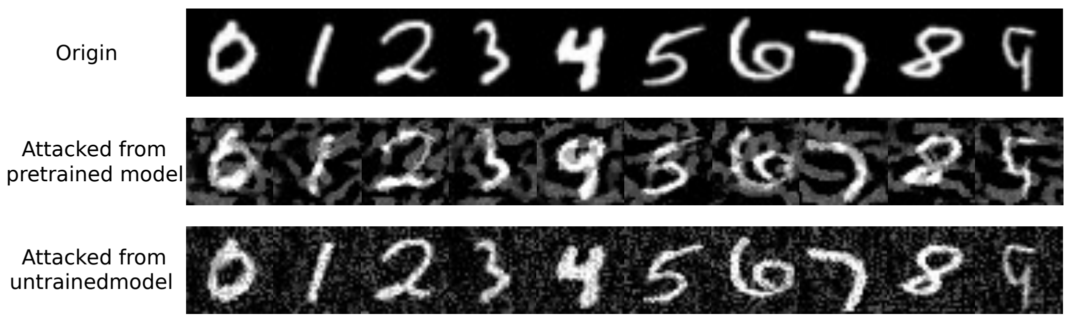


Figure 7: Attacked MNIST images. The first row shows the original images in MNIST. The second row shows the images generated from PGD by attacking the vanilla model trained in section 3.1. The third row shows the images generated from PGD by attacking a untrained model.

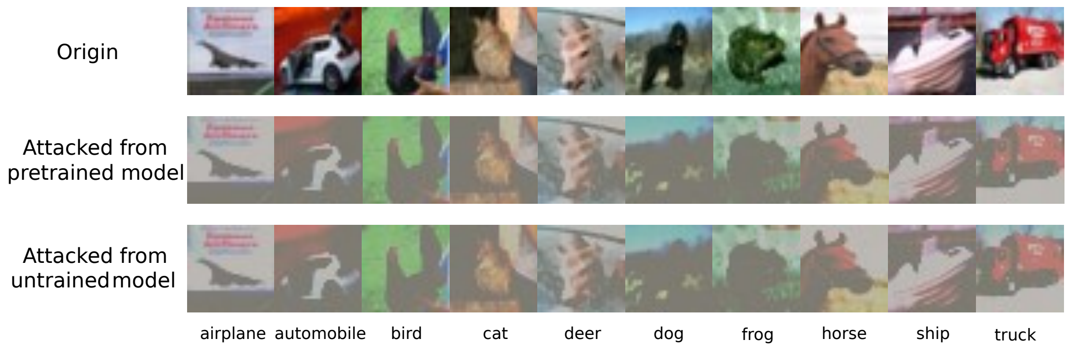


Figure 8: Attacked CIFAR-10 images. The first row shows the original images in CIFAR-10. The second row shows the images generated from PGD by attacking the vanilla model trained in subsection 3.1. The third row shows the images generated from PGD by attacking a untrained model. Since ϵ here is much smaller than the ϵ in PGD attack to MNIST, the second row and the third row have little visual difference.

From the second row, one can easily conclude that PGD attack works well. To observe this, note that the attacked 3 is somewhat like a 5, the attacked 4 is somewhat a bit like a 9, the attacked 5 is like a 3, the attacked 7 is like a 9, and the attacked 9 is like a 5.

The third row is used for comparison. Since the attacked model is untrained, that is, the model being attacked is newly initialized, its parameters are all uniformly random. It follows that the attack against the model should also be a random attack. The "noise" in the third row can just confirm this idea.

We can feed the vanilla model, mixup model and manifold mixup model with the attacked test data which is generated by the attack against the vanilla model trained in Subsection 3.1 to see the accuracies. The results is shown in Table 3, demonstrating that PGD attack works really well.

CIFAR-10 For CIFAR-10 dataset, we choose the parameters $\epsilon = 8/255, \alpha = 2/255$, and the number of iterations is chosen to be 7. The reason why both ϵ and α are divided by 255 is that at the very beginning we have normalized the RGB channels of a CIFAR-10 image from 0-255 to 0-1. Fig. 8 shows the original images and the attacked images.

Unlike attacked MNIST images, which can be easily told the difference between the original images and the attacked images (for example, say an original 5 is attacked to become more like a 3), the attacked CIFAR-10 images just turned grayer. For the CIFAR-10 dataset which contains numerous images and classes, the average channel value is close to $(127, 127, 127)$, which is the color gray. Thus turning an image grayer is making the image closer to the decision boundary of the model, which is consistent with the goal of PGD.

Similar to MNIST, we can test the prediction accuracies of the three pretrained models on the attacked test data. Table 3 shows the results. The accuracies, which are about 5%, are a little higher than the accuracies of the previous MNIST results. We can also conclude that PGD attack is effective since 5% is actually very small.

Table 3 show that the model trained with manifold mixup has slightly higher robustness than the the other two models, and the mixup model has slightly higher robustness than the vanilla model. Actually, this minor gap can be ignored. Finally we can conlude that PGD works very well on attacking a model trained with clean data.

3.3 Adversarial Training

We use the same PGD hyperparameters as described in section PGD Attack. Since calculating the adversaries costs much time, we decrease the number of epochs to 500 for CIFAR-10. The learning rate is set to 0.1. For MNIST, we find that using the previous setting of hyperparameters is not enough for the adversarial training to converge. After several attempts, we set the number of epochs

Model	Accuracy on Attacked Data (%) From Pretrained	Accuracy on Attacked Data (%) From Untrained	Accuracy on Clean Data (%) From Pretrained	Accuracy on Clean Data (%) From Untrained
Vanilla	85.81	84.26	96.87	96.49
Mixup	90.09	72.83	98.13	95.73
Manifold mixup	87.88	83.28	97.97	96.69

Table 4: MNIST: Performance of the adversarially trained networks of different training starts on attacked and clean test data.

Model	Accuracy on Attacked Data (%) Standard Training	Accuracy on Attacked Data (%) PGD Training	Accuracy on Clean Data (%) Standard Training	Accuracy on Clean Data (%) PGD Training
Vanilla	0	85.81	99.30	96.87
Mixup	0	90.09	99.31	98.13
Manifold mixup	0.01	87.88	99.44	97.97

Table 5: MNIST: Performance of networks of different training methods on attacked and clean data.

Model	Accuracy on Attacked Data (%) Standard Training	Accuracy on Attacked Data (%) PGD Training	Accuracy on Clean Data (%) Standard Training	Accuracy on Clean Data (%) PGD Training
Vanilla	5.01	22.89	95.58	78.13
Mixup	5.02	55.20	96.89	82.49
Manifold mixup	5.06	48.57	97.19	81.92

Table 6: CIFAR-10: Performance of networks of different training methods on attacked and clean data.

to 800. The learning rate is set to 0.002 at the beginning, and will decay to 0.001 and 0.0005 after 400 and 600 epochs respectively. The batch size for both model is 100.

Since an untrained model has a random gradient, PGD attack against such a model is meaningless. Therefore, we expect that adversarial training from a pretrained model can converge faster and achieve a better performance. We first evaluate whether starting adversarial training from a pretrained model increases the performance on MNIST. The results appear in Table 4.

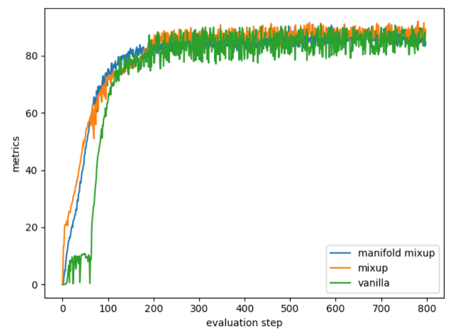
We observe that choosing a pretrained model as the adversarial training start could achieve higher accuracy both against PGD adversaries and on clean test data, as expected. Then we compare the performance of adversarially training models with different training methods. The method is the combination of choices of:

- whether to train models on clean data (standard training) or PGD-made adversarials (adversarial training), and
- whether to use mixup/manifold mixup.

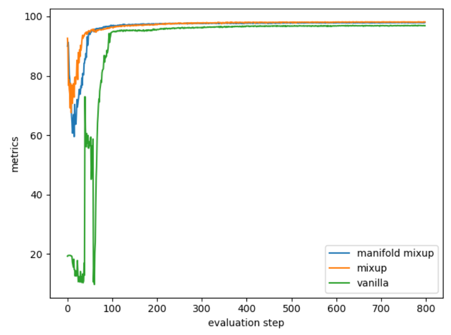
For MNIST, the training curves are shown in Fig. 9 and the results are shown in Table 5. For CIFAR-10, the training curves are shown in Fig. 10 and the results are shown in Table 6.

It is obvious that the PGD-trained models all significantly outperform the standard-trained ones against adversaries. So here we can conclude that adversarial training indeed increases robustness. Also, an adversarially trained mixup model has the best robustness in both datasets. This meets our expectation that the mixup and adversarial training, two methods that can both improve robustness independently, will have a greater effect when combined together.

However, we can observe that adversarial training will decrease the accuracy on clean test data. For CIFAR-10, adversarially trained mixup model has 14.4% less accuracy on clean test data. This can not be neglected compared with the high accuracy of the standard-trained mixup model. In most applications, the reduction in accuracy is unacceptable.

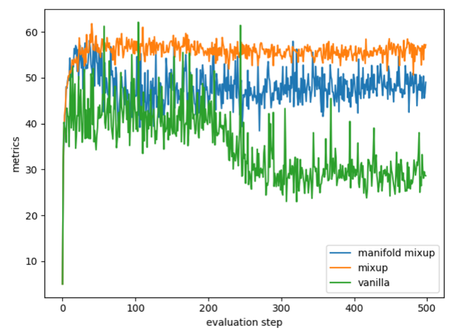


(a) accuracy against PGD adversaries

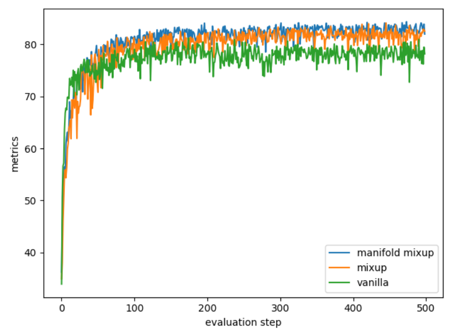


(b) accuracy on clean test data

Figure 9: MNIST: Training curves of differently trained models



(a) accuracy against PGD adversaries



(b) accuracy on clean test data

Figure 10: CIFAR-10: Training curves of differently trained models.

3.4 Some Attempts on Adversarial Training

3.4.1 Soft Labeling

We have used one-hot labels throughout the training process. As stated before, soft labels could be an optimal technique to enhance the model ability of tolerating noises. Here we try a simple labeling policy: based on the one-hot encoded labels, with a given "noise" σ, we replace the zeros with σ and replace the ones with $1-(N-1)\times\sigma$ where N is the number of different classes. With this operation, for each image it becomes that, with probability $1-(N-1)\times\sigma$ it belongs to the right class and with probability σ it belongs to one of the wrong classes. That is, suppose from MNIST we have a image whose right class is 7, with this soft labeling process, the model will get the information that with probability $1-(N-1)\times\sigma$ it belongs to class 7 and with probability σ it belongs to any class other than 7 respectively.

The result of soft labeling is shown in Table 7. We regard the result of the previously trained mixup model without soft labeling as the case $\sigma = 0$. As one can see, on MNIST dataset, the overall performance of model with soft labeling on the accuracy against adversary is weaker than the one without soft labeling. Moreover, both a smaller and larger σ might lead to a significant drop on accuracy. A possible explanation is that *mixup* will enlarge the noise and make the model mislearn the data especially on simple dataset like MNIST. As for why a smaller noise will cost so much accuracy, our explanation is that it may fall into a local optimum, but this requires more evidence.

For both dataset, the soft labeling process shows its potential in increasing the model accuracy on the clean data while remaining its robustness. On CIFAR-10, all the accuracy on the clean data is larger with soft labeling and two gains more accuracy on attacked data. It also worths noting that the different outcomes between MNIST and CIFAR-10 might indicate that soft labeling is more suitable for more complex datasets and more complicated network architectures.

Dataset	noise σ	Accuracy on Attacked Data (%)	Accuracy on Clean Data (%)
MNIST	0	**90.09**	98.13
	0.001	79.25	97.52
	0.003	89.69	**98.41**
	0.005	87.65	97.87
	0.01	74.49	94.78
CIFAR-10	0	55.20	82.49
	0.001	55.60	82.65
	0.005	54.24	**83.47**
	0.01	**57.94**	82.80

Table 7: Performance of soft labeling of different noise on MNIST and CIFAR-10.

Dataset	Mixup Policy	Accuracy on Attacked Data (%)	Accuracy on Clean Data (%)
MNIST	normal mixup	**90.09**	98.13
	partial PGD	74.87	**98.94**
CIFAR-10	normal mixup	**55.20**	**82.49**
	partial PGD	50.74	18.47

Table 8: Performance of model adversarially trained with different mixup policy on MNIST and CIFAR-10.

3.4.2 Mixup with Partial PGD

Here we try a mixup policy called *partial PGD*. Suppose we have a batch (X, y), we first apply PGD on it and get (X', y). Then we perform mixup on (X, y) and (X', y). The algorithm flow is then modified to Algorithm 2. We have mentioned that mixup is intended to extend the training data distribution and we hope that mixing an unattacked data batch and a PGD-attacked data batch will further extend the distribution.

Algorithm 2: Adversarial training (mixup with partial PGD)

```
for $i = 0 : num\_iter$ do
    sample a batch $(X, y)$ from dataset;
    apply PGD attack on the batch and get $(X', y)$ ;
    update model by the mixup of $(X', y)$ and $(X, y)$.
end
```

The results are shown in Table 8. We can find that on MNIST dataset, using mixup with partial PGD to train a model will result in a slightly better accuracy on clean dataset, while its robustness is hugely lower than the original mixup policy. On CIFAR-10, the situation becomes worse. Training with partial PGD will lead to an extremely low accuracy on clean dataset. Even when we have tried partial PGD with different hyperparameters on CIFAR-10, the accuracy remains low. A possible explanation is that there exists a gap between the data distribution of original sample and attacked sample. After the mixup of the two samples, the distribution will become unpredictable and thus harm the overall performance. Due to the complicated network architecture of PreAct-Resnet, the gap becomes huge and the reduction in performance on CIFAR-10 becomes more significant than that on MNIST.

4 Related Work

Our work is mainly based on [11, 16, 15]. Madry et al. [11] introduced the PGD method to attack the input samples. It also proposed adversarial training which is proven to be utterly effective to improve the robustness of the machine learning models. Zhang et al. [16] proposed the *mixup* idea of data augmentation and Verma et al. [15] proposed *manifold mixup* as its further application on the hidden layers of the model. Our work is mainly the combination of the adversarial training based on PGD and *mixup* as well as *manifold mixup*.

Similar approaches which also combines *mixup* and adversarial training have been raised in recent works [13, 9, 6]. Pang et al. [13] developed an inference principal MI which mixups the input with other random clean samples (this is similar to our mixup policy "partial-PG" which mixups one attacked image and one clean image) and Lee et al. [9] proposed a new mixup policy AVmixup which considered the adversarial vertex of the input by applying soft labeling techniques. Laugros et al. [6] combines mixup method and learning-based soft labeling with adversarial training. This might be the most related work to one of our method, while the main difference is that it uses a learning model for soft label generation and we use a fixed "noise" in labeling.

5 Contribution of Each Member

The group photo 11 below was taken in the class of Week 15.

分工
• 赖睿航：前期框架，团队核心
• 金弘义：对抗学习算法实现，最强大脑
• 陆嘉馨：查阅文献，无敌的算力支持

Figure 11: Group photo. Left: Jiaxin Lu. Middle: Ruihang Lai. Right: Hongyi Jin.

Here is the contribution of each member.

- Ruihang Lai: Prepare the base code of the whole project (i.e., almost all other code are based on the code he prepared). Train lots of models. Make slides and write the report. *"Core of the Group"*.
- Hongyi Jin: Implement and improve the code of adversarial training (i.e., we attribute the success of adversarial training experiments to him). Train lots of models. Make slides and write the report. *"Strongest Brain"*.
- Jiaxin Lu: Look up the references throughout the whole project (i.e., the materials she collected gave us huge support). Train lots of models. Make slides and write the report. *"Unrivaled Support of Computing Power"*.

References

[1] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014. 1

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 2

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks, 2016. 2

[4] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 2

[5] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016. 1

[6] Alfred Laugros, Alice Caplier, and Matthieu Ospici. Addressing neural network robustness with mixup and targeted labeling adversarial training. *arXiv preprint arXiv:2008.08384*, 2020. 10

[7] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791. 2

[8] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/. 2

[9] Saehyung Lee, Hyungyu Lee, and Sungroh Yoon. Adversarial vertex mixup: Toward better adversarially robust generalization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 272–281, 2020. 10

[10] Tianyu Liu, Kexiang Wang, Baobao Chang, and Zhifang Sui. A soft-label method for noise-tolerant distantly supervised relation extraction. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1790–1795, 2017. 1

[11] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017. 1, 9

[12] Quang Nguyen, Hamed Valizadegan, and Milos Hauskrecht. Learning classification models with soft-label information. *Journal of the American Medical Informatics Association*, 21(3): 501–508, 2014. 1

[13] Tianyu Pang, Kun Xu, and Jun Zhu. Mixup inference: Better exploiting mixup to defend adversarial attacks. *arXiv preprint arXiv:1909.11515*, 2019. 10

[14] Florian Tramèr, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. The space of transferable adversarial examples. *arXiv preprint arXiv:1704.03453*, 2017. 1

[15] Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. Manifold mixup: Better representations by interpolating hidden states. In *International Conference on Machine Learning*, pages 6438–6447. PMLR, 2019. 1, 9, 10

[16] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017. 1, 9