Supplementary Materials

A Limiting Cycle

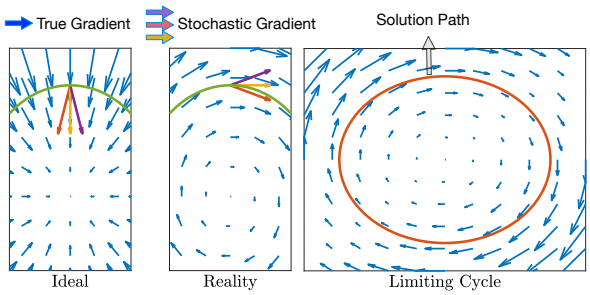

Figure 6: *Illustrative hardness for solving problem* (1). *Wrong directions can lead to a limiting cycle. Then algorithms fail to converge. Details in Appendix A.*

Limiting cycle is a well-known issue for bilevel machine learning problems [4,5]. The reason behind limiting cycle is that different from minimization problems, a bilevel optimization problem is more complicated and could be highly nonconvex-nonconcave, where the inner problem can not be solved exactly. Here we provide a simple bilevel problem example, which is convex-concave, but the iterations still cannot converge due to the inexact solutions. Specifically, we consider the following optimization problem:

$$\min_x \max_y f(x,y) = xy.$$

Then at the t-th iteration, the update direction will be $(-y_t, x_t)$. If we start from $(1,0)$ with a stepsize of 0.0001, this update will result in a limiting circle: $x^2+y^2=1$ and never reach the stable equilibrium $(0,0)$ as shown in Figure 7.

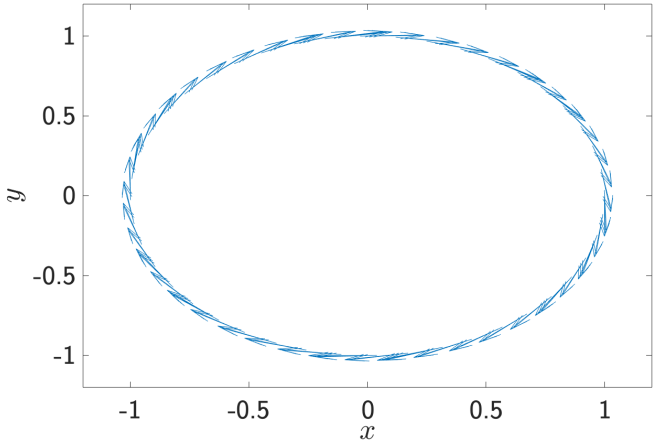

Figure 7: An example of the limiting circle: Arrows denote the update directions

B Attacker Architecture

In the following, we study how the attacker architecture affects the stability of L2L training. Table 5 presents another attacker architecture: *slim attacker*. In this network, the second convolutional layer uses downsampling, while the second last deconvolutional layer uses upsampling. Such a bottleneck design is widely used in deep

neural networks due to computational considerations. For example the running time of per epoch for L2L with slim attacker is 480; whereas L2L with the original architecture is 620. However, it loses some information of input and is significant worse than the original architecture (Table 1). Inspired by residual learning in He et al. (2016), we address the stability issue by using a skip layer connection to ease the training of this network. Specifically, the last layer takes the concatenation of $\mathcal{A}_{f_{\boldsymbol{\theta}}}(\boldsymbol{x}, y)$ and the output of the second last layer as input. Figure 8 presents the architecture of ResBlocks. PReLU is a special type of Leaky ReLU with a learnable slope parameter.

Table 6 shows the results of L2L with the slim attacker shown in Table 5. The performance of GradL2L under PGM attacker on CIFAR10 for slim attacker is comparable to the original attacker. However, under other scenarios, the robust performance is worse than the original attacker. We tried to make the slim attacker deeper or take more L2L steps and observe little improvement. These results suggest a very important design choice of attacker architecture for L2L that the widely used bottleneck design causes the loss of information and can make the training difficult.

Table 5: *Slim Attacker Network Architecture.*

Conv:	$[k = 3 \times 3, c = 128, s = 1, p = 1]$, BN+ReLU
ResBlocks:	[channel = 256]
ResBlocks:	[channel = 128], BN
DeConv:	$[k = 4 \times 4, c = 16, s = 2, p = 1]$, BN+ReLU
Conv:	$[k = 3 \times 3, c = 3, s = 1, p = 1]$, tanh

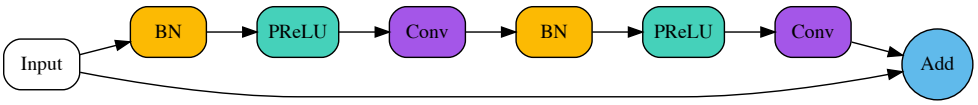

Figure 8: *An illustration example for the architecture of ResBlocks.*

Table 6: *Results of L2L with Slim Attacker under White-box Setting over CIFAR.*

Defense Method	Attack Method	Accuracy	
		Clean	Robust
Dataset: CIFAR10			
Grad L2L	PGM-20	85.31%	53.02%
2-Step L2L	PGM-20	75.36%	46.12%
Grad L2L	CW	85.31%	42.72%
2-Step L2L	CW	75.36%	40.82%
Dataset: CIFAR100			
Grad L2L	PGM-20	60.60%	27.37%
2-Step L2L	PGM-20	60.23%	20.23%
Grad L2L	CW	60.60%	22.14%
2-Step L2L	CW	60.23%	22.70%

C Black-box Attack

Under the black-box setting, we first train a surrogate model with the same architecture of the target model but a different random seed, and then attackers generate adversarial examples to attack the target model by querying gradients from the surrogate model.

The black-box attack highly relies on the transferability, which is the property that the adversarial examples of one model are likely to fool others. However, the transferred attack is very unstable, and often has a large

variation in its effectiveness. Therefore, results of the black-box setting might not be reliable and effective. Thus we only present one result here to demonstrate the robustness of different models.

Table 7: *Results of the Black-box Setting over CIFAR-10. We Evaluate L2L Methods with Slim Attacker Networks.*

Surrogate	Plain Net		FGSM Net		PGM Net	
	FGSM	PGM10	FGSM	PGM10	FGSM	PGM10
Plain Net	40.03	5.60	74.42	75.25	67.37	65.92
FGSM Net	79.20	85.02	**89.90**	80.40	64.28	63.89
PGM Net	83.80	84.73	84.33	85.29	67.05	65.54
Naive L2L	45.52	25.95	83.99	77.94	68.14	67.13
Grad L2L	**86.10**	86.87	87.93	**88.01**	**71.15**	**69.95**
2-Step L2L	85.83	**87.10**	86.51	87.60	70.58	69.38

Table 8: *Experiments under the Black-box Setting over CIFAR-100. Note that here We only Evaluate L2L Methods Using the Slim Attacker Network.*

Surrogate	Plain Net		FGSM Net		PGM Net	
	FGSM	PGM10	FGSM	PGM10	FGSM	PGM10
Plain Net	21.04	9.04	50.57	54.06	40.06	41.30
FGSM Net	42.87	50.73	**61.68**	44.70	39.34	40.08
PGM Net	56.63	58.34	56.99	57.97	40.19	39.87
Naive L2L	20.97	10.47	50.36	54.07	38.63	39.91
Grad L2L	57.63	59.62	59.18	**61.26**	41.71	41.15
2-Step L2L	**58.66**	**59.31**	58.92	59.46	**45.80**	**45.31**

D Extension

As we mentioned earlier that our proposed L2L framework is quite general, and applicable to a *broad* class of minimax optimization problems, here we present an extension of our proposed L2L framework to generative adversarial imitation learning (GAIL, Ho and Ermon (2016)) and conduct some numerical experiments for comparing the original GAIL and GAIL with L2L on two environments: CartPole and Mountain Car Brockman et al. (2016).

D.1 L2L for Generative Adversarial Imitation Learning

Imitation learning aims to learn to perform a task from expert demonstrations, in which the learner is given only samples of trajectories from the expert. To solve this problem, GAIL tries to recover the expert's cost function and extract such a a policy from the recovered cost function, which can be formulated as the following bilevel optimization problem:

$$\begin{aligned} &\min_{\theta_\pi} L(\theta_\pi, \theta_{\mathrm{D}}^*) - \lambda H(\pi), \\ \text{s. t.} \quad &\theta_{\mathrm{D}}^* \in \operatorname*{argmax}_{\theta_{\mathrm{D}}} L(\theta_\pi, \theta_{\mathrm{D}}) + L_{\mathrm{E}}(\theta_{\mathrm{D}}) - \lambda H(\pi), \end{aligned} \tag{9}$$

where $L(\theta_\pi, \theta_D) = \mathbb{E}_{s,a\sim\pi(s;\theta_\pi)}[\log(D(s,a;\theta_{\mathrm{D}}))]$, $L_{\mathrm{E}}(\theta_{\mathrm{D}}) = \mathbb{E}_{\widetilde{s},\widetilde{a}\sim\pi_{\mathrm{E}}}[\log(1 - D(\widetilde{s},\widetilde{a};\theta_{\mathrm{D}}))]$, $\pi(\cdot;\theta_\pi)$ is the trained policy parameterized by θ_π, π_{E} denotes the expert policy, $D(\cdot,\cdot;\theta_{\mathrm{D}})$ is the discriminator parameterized by θ_{D}, $\lambda H(\pi)$ denotes a entropy regularizer with tuning parameter λ, (s,a) and $(\widetilde{s},\widetilde{a})$ denote the state-action for the trained policy and expert policy, respectively. By optimizing 9, the discriminator D distinguishes the state-action (s,a) generated from the learned policy π with the sampled trajectories $(\widetilde{s},\widetilde{a})$ generated from some expert policy π_{E}. In the original GAIL training, for each iteration, we update the parameter of D, θ_{D}, by stochastic gradient ascend and then update θ_π by the trust region policy optimization (TRPO, Schulman et al. (2015)).

Similar to the adversarial training with L2L, we apply our L2L framework to GAIL by parameterizing the inner optimizer as a neural network $U(;\theta_{\mathrm{U}})$ with parameter θ_{U}. Its input contains two parts: parameter θ_{D} and the gradient of loss function with respect to θ_{D}:

$$g_{\mathrm{D}}(\theta_{\mathrm{D}},\theta_{\pi}) = \mathop{\mathbb{E}}_{s,a\sim\pi(s;\theta_{\pi})}[\nabla_{\theta_{\mathrm{D}}}\log\left(D(s,a;\theta_{\mathrm{D}})\right)] + \mathop{\mathbb{E}}_{\widetilde{s},\widetilde{a}\sim\pi_E}[\nabla_{\theta_{\mathrm{D}}}\log(1-D(\widetilde{s},\widetilde{a};\theta_{\mathrm{D}}))].$$

In practice, we use a minibatch (several sample trajectories) to estimate $g_{\mathrm{D}}(\theta_{\mathrm{D}},\theta_{\pi})$, denoted as $\widehat{g}_{\mathrm{D}}(\theta_{\mathrm{D}},\theta_{\pi})$. Specifically, at the t-th iteration, we first calculate $\widehat{g}_{\mathrm{D}}^{t} = \widehat{g}_{\mathrm{D}}(\theta_{\mathrm{D}}^{t},\theta_{\pi}^{t})$ and then update $\theta_{\mathrm{D}}^{t+1} = U(\theta_{\mathrm{D}}^{t},\widehat{g}_{\mathrm{D}}^{t};\theta_{\mathrm{U}}^{t})$. Next, we update θ_{U} by gradient ascend based on the sample estimate of

$$\mathop{\mathbb{E}}_{s,a\sim\pi(s;\theta_{\pi}^{t})}[\nabla_{\theta_{\mathrm{U}}}\log\left(D(s,a;\theta_{\mathrm{D}}^{t+1})\right)] + \mathop{\mathbb{E}}_{\widetilde{s},\widetilde{a}\sim\pi_{\mathrm{E}}}[\nabla_{\theta_{\mathrm{U}}}\log(1-D(\widetilde{s},\widetilde{a};\theta_{\mathrm{D}}^{t+1}))].$$

The detailed algorithm is presented in Algorithm 5.

Algorithm 5 *L2L-based GAIL.*

Input: $\pi_E(\widetilde{s})$: Expert; θ_{π}: Policy parameter; θ_{D}: Discriminator parameter; θ_{U}: Updater parameter.
for $t \leftarrow 1$ *to* N **do**
- $(s,a\sim\pi(a;\theta_{\pi}))$ $(\widetilde{s},\widetilde{a}\sim\pi_E(\widetilde{s}))$
- // Sample trajectories and expert trajectories.
- $g_{\mathrm{D}}^{t} \leftarrow \frac{1}{|(s,a)|}\sum_{(s,a)}[\nabla_{\theta_{\mathrm{D}}}\log\left(D(s,a;\theta_{\mathrm{D}}^{t})\right)] + \frac{1}{|(\widetilde{s},\widetilde{a})|}\sum_{(\widetilde{s},\widetilde{a})}[\nabla_{\theta_{\mathrm{D}}}\log(1-D(\widetilde{s},\widetilde{a};\theta_{\mathrm{D}}^{t}))]$
- //Compute gradient.
- $\theta_{\mathrm{D}}^{t+1} = U(\theta_{\mathrm{D}}^{t},g_{\mathrm{D}}^{t};\theta_{\mathrm{U}}^{t})$
- //Update the discriminator parameters.
- $\theta_{\mathrm{U}}^{t+1} \leftarrow \mathop{\mathrm{argmin}}_{\theta_{\mathrm{U}}}\frac{1}{|(s,a)|}\sum_{(s,a)}[\log\left(D(s,a;\theta_{\mathrm{D}}^{t+1})\right)] + \frac{1}{|(\widetilde{s},\widetilde{a})|}\sum_{(\widetilde{s},\widetilde{a})}[\log(1-D(\widetilde{s},\widetilde{a};\theta_{\mathrm{D}}^{t+1}))]$
- //Update θ_{U} of updater.
- Update θ_{π} by a policy step using the TRPO rule Ho and Ermon (2016)
- //Update policy parameter θ_{π}.

D.2 Numerical Experiments

Updater Architecture. We use a simple 3-layer perceptron with a skip layer as our updater. The number hidden units are $(2m \to 8m \to 4m \to m)$, where m is the dimension of θ_{D} that depends on the original task. For the first and second layers, we use Parametric ReLU (PReLU, He et al. (2015)) as the activation function, while the last layer has no activation function. Finally we add the output to θ_{D} in the original input as the updated parameter for the discriminator network.

Hyperparameter Settings. For all baselines we exactly follows the setting in Ho and Ermon (2016), except that we use a 2-layer discriminator with number of hidden units $((s,a) \to 64 \to 32 \to 1)$ using tanh as the activation function. We use the same neural network architecture for π and the same optimizer configuration. The expert trajectories are obtained by an expert trained using TRPO. For L2L based GAIL, we also use Adam optimizer to update the θ_{U} with the same configuration as updating θ_{D} in the original GAIL.

Numerical Results. As can be seen in Figure 9, GAIL has a sudden performance drop after training for a long time. We conjecture that this is because the discriminator overfits the expert trajectories and converges to a bad optimum, which is not generalizable. On the other hand, GAIL with L2L is much more stable. It is very important to real applications of GAIL: since the reward in real-world environment is usually unaccessible, we cannot know whether there is a sudden performance drop or not. With L2L, we can stabilize the training and obtain a much more reliable algorithm for real-world applications.

E Robustness Evaluation Checklist

Recently, there are many works on robustness defense that have been proven ineffective Athalye et al. (2018); Carlini et al. (2019). Our work follows the most reliable and widely used robust model approach adversarial training, which finds a set parameters to make the model robust. We do not make any modification to final

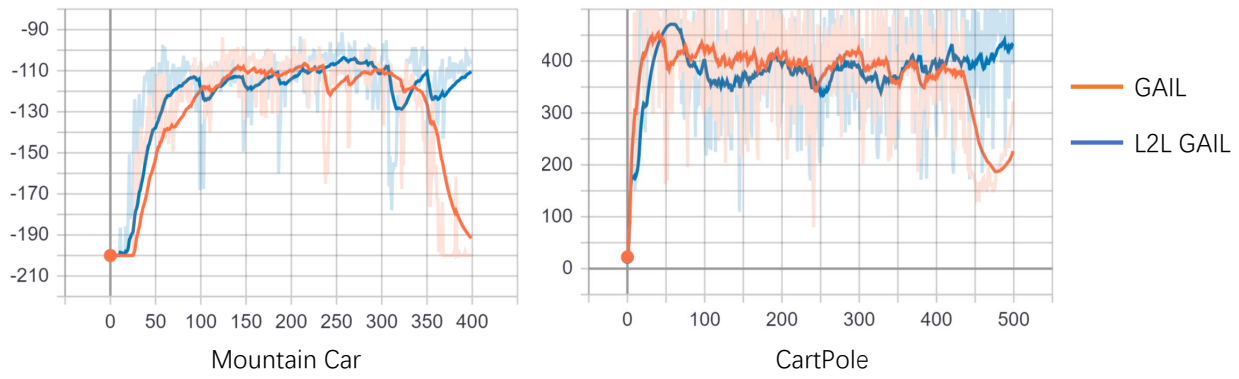

Figure 9: *Reward vs. iteration of the trained policy using original GAIL and L2L GAIL under two environments: Mountain Car and CartPole.*

classifier model. Unlike previous works (e.g., Defense-GAN, Samangouei et al. (2018)), our model does not take the attacker as a part of the final model and does not use shattered/obfuscated/masked gradient as a defense mechanism. We also demonstrate that the evaluation of the robustness of our proposed L2L method is trustworthy by verifying all items listed in Carlini et al. (2019).

E.1 Shattered/Obfuscated/Masked Gradient

In this section we verify that our proposed L2L method does not fall into the pitfall of shattered/obfuscated/masked gradient, which have proven ineffective. To see this, we checked every item recommended in Section 3.1 of Athalye et al. (2018):

- One-step attacks perform better than iterative attacks: Figure 3 shows that the PGM attack is stronger with larger number of iterations.
- Black-box attacks are better than white-box attacks: Appendix C shows that the black-box transfer attack is much weaker than white white-box attacks.
- Unbounded attacks do not reach 100% success: We evaluate the model robustness against attack with extremely large perturbation to show that unbounded attacks do reach 100% success. Specifically, we use the PGM-10 attack with various perturbation magnitudes $\epsilon \in [0, 1]$ and stepsize $\frac{\epsilon}{10}$. Figure 10 shows that the PGM attack eventually reach 100% success as the perturbation magnitude increases.

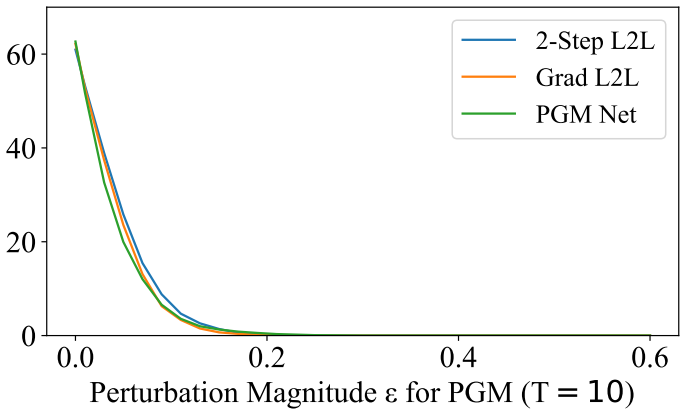

Figure 10: *Robust accuracy against perturbation magnitudes of PGM over CIFAR-100.*

- Random sampling finds adversarial examples: In Table 2, we show that random search is not better than gradient-based method and is rather weak against our model.

- Increasing distortion bound does not increase success: Figure 3 shows that the PGM attack becomes stronger as the perturbation magnitude increases.

E.2 Robustness Evaluation Checklist

Carlini et al. (2019) also provide an evaluation checklist, and we now check each of common severe flaws and common pitfalls as follows:

- State a precise threat model: We do not have any adversary detector; We do not use shattered/Obfuscated/Masked gradient. We do not have a denoiser. Our model has no aware of the attack mechanism, including PGM and CW attacks.
- Adaptive attacks: We used CW, PGM, and L2L attacker attack.
- Report clean model accuracy: We reported.
- Do not use Fast Gradient Sign Method. We use PGM-20 and PGM-100 and CW.
- Do not only use attacks during testing that were used during training. We use different evaluation criteria to evaluate all models.
- Perform basic sanity tests: It is provided in Figure 3.
- Generate an attack success rate vs. perturbation budget: Figure 3.
- Verify adaptive attacks perform better than any other (e.g., blackbox, and brute-force search): The above table and Appendix C in the paper.
- Describe the attacks applied: In Section 4.
- Apply a diverse set of attacks: We tried PGM attack (with different perturbation magnitude and iterations), blackbox attack (transfer attack), CW attack (adaptive attack), L2L attack (adaptive and designed for this particular model), Bruteforce random search (gradient-free attack)
- Suggestions for randomized defenses: We are not.
- Suggestions for non-differentiable components (e.g., by performing quantization or adding extra randomness): We have no additional non-differentiable component.
- Verify that the attacks have converged: Figure 3 shows that the PGM attack eventually converges.
- Carefully investigate attack hyperparameters: Figure 3.
- Compare against prior work: We compared our algorithm to PDM net. L2L is more computationally efficient and the L2L model is more robust due to the fact that L2L attack is strong enough. Unlike Defense-GAN, we do not use the generator (attacker in L2L) as the denoising module and do not change the final prediction model.