TOWARDS DEFENDING MULTIPLE ℓ_p-NORM BOUNDED ADVERSARIAL PERTURBATIONS VIA GATED BATCH NORMALIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

There has been extensive evidence demonstrating that deep neural networks are vulnerable to adversarial examples, which motivates the development of defenses against adversarial attacks. Existing adversarial defenses typically improve model robustness against individual specific perturbation types. However, adversaries are likely to generate multiple perturbations in practice. Some recent methods improve model robustness against adversarial attacks in multiple ℓ_p balls, but their performance against each perturbation type is still far from satisfactory. We observe that different ℓ_p bounded adversarial perturbations induce different statistical properties that can be separated and characterized by the statistics of Batch Normalization (BN). We thus propose Gated BN (GBN) to adversarially train a perturbation-invariant predictor for defending multiple ℓ_p bounded adversarial perturbations. GBN consists of a multi-branch BN layer and a gated sub-network. Each BN branch in GBN is in charge of one perturbation type to ensure that the normalized output is aligned towards learning perturbation-invariant representation. Meanwhile, the gated sub-network is designed to separate inputs added with different perturbation types. We perform an extensive evaluation of our approach on MNIST, CIFAR-10, and Tiny-ImageNet, and demonstrate that GBN outperforms previous defense proposals against multiple perturbation types (*i.e*, ℓ_1, ℓ_2, and ℓ_∞ perturbations) by large margins of 10-20%.[1]

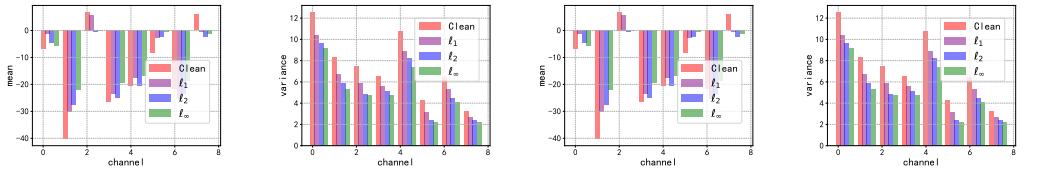
1 INTRODUCTION

Deep neural networks (DNNs) have achieved remarkable performance across a wide areas of applications (Krizhevsky et al., 2012; Bahdanau et al., 2014; Hinton et al., 2012), but they are susceptible to *adversarial examples* (Szegedy et al., 2013). These perturbations are imperceptible to humans but can easily lead DNNs to wrong predictions (Kurakin et al., 2016; Liu et al., 2019).

To improve model robustness against adversarial perturbations, a number of *adversarial defense* methods have been proposed (Papernot et al., 2015; Engstrom et al., 2018; Goodfellow et al., 2014; Zhang et al., 2020). Most adversarial defenses are designed to counteract a single type of perturbation (*e.g*, small ℓ_∞-noise) (Madry et al., 2018; Kurakin et al., 2017; Dong et al., 2018). These defenses offer no guarantees for other ℓ_p bounded adversarial perturbations (*e.g*, ℓ_1, ℓ_2), and sometimes even increase model vulnerability (Kang et al., 2019; Tramèr & Boneh, 2019). However, adversaries are not designated to generate individual specific perturbation types, and are likely to create multiple perturbations in practice. To address this problem, other adversarial defense strategies have been proposed, with the goal of simultaneously achieving robustness against multiple ℓ_p bounded attacks, *i.e*, ℓ_∞, ℓ_1, and ℓ_2 attacks (Tramèr & Boneh, 2019; Maini et al., 2020b). Although these methods improve overall model robustness against adversarial attacks in multiple ℓ_p balls, the performance for each individual perturbation type is still far from satisfactory.

In this work, we focus on improving model robustness against multiple ℓ_p bounded adversarial perturbations. Our primary observation is that different types of ℓ_p-norm adversarial perturbations induce different statistical properties (Figure 1) and have separable characteristics, which we refer to

[1]Our code can be found at `https://anonymous.4open.science/r/GBN-625C/`.



(a) means at $conv2\_1$ (b) variances at $conv2\_1$ (c) means at $conv3\_1$ (d) variances at $conv3\_1$

Figure 1: Running means and variances of multiple BN branches on 8 randomly sampled channels in a VGG-16's $conv2\_1$ and $conv3\_1$ layer. The results show that different ℓ_p bounded adversarial perturbations induce different normalization statistics, which may arise in different domains (See running statistics results at more layers for more models in Appendix E.8).

it as multi-domain hypothesis. Through theoretical and empirical studies, we found that by simply adversarial training using a mixture of perturbation types on a single BN, there inevitably exists a domain gap between the training and testing data, which would cause the performance degeneration. We thus propose to defend multiple perturbation types by designing a multiple-branch BN layer, where each BN branch is in charge of one corresponding perturbation type (*i.e*, domain) to ensure the normalized output aligned towards learning perturbation-invariant representation.

One remaining problem is that the model does not know which type of perturbation is added during inference. Therefore, we simultaneously train a gated sub-network, which aims to separate perturbation types on-the-fly. We combine the multi-branch BN layer and the gated sub-network as a building block for DNNs, referred to as *Gated Batch Normalization* (GBN). GBN improves model robustness by separating perturbation-specific information for different perturbation types, and using BN layer statistics to better align data from the mixture distribution towards perturbation-invariant representations. Extensive experiments on MNIST, CIFAR-10, and Tiny-ImageNet demonstrate that our method outperforms previous defense strategies by large margins, *i.e*, 10-20%.

2 Background and Related Work

In this section, we provide a brief overview of existing work on adversarial attacks and defenses, as well as batch normalization techniques.

2.1 Adversarial Attacks and Defenses

Adversarial examples are inputs intentionally designed to mislead DNNs (Szegedy et al., 2013; Goodfellow et al., 2014). Given a DNN f_Θ and an input image $\mathbf{x} \in \mathbb{X}$ with the ground truth label $\mathbf{y} \in \mathbb{Y}$, an adversarial example $\mathbf{x}_{adv}$ satisfies

$$f_\Theta(\mathbf{x}_{adv}) \neq \mathbf{y} \quad s.t. \quad \|\mathbf{x} - \mathbf{x}_{adv}\| \leq \epsilon,$$

where $\|\cdot\|$ is a distance metric. Commonly, $\|\cdot\|$ is measured by the ℓ_p-norm ($p \in \{1,2,\infty\}$).

Various defense approaches have been proposed to improve model robustness against adversarial examples (Papernot et al., 2015; Xie et al., 2018; Madry et al., 2018; Liao et al., 2018; Cisse et al., 2017), among which adversarial training has been widely studied and demonstrated to be the most effective (Goodfellow et al., 2014; Madry et al., 2018). However, these defenses only improve model robustness for specific ℓ_p-norm perturbation (*e.g*, ℓ_∞) and typically offer no robustness guarantees against other ℓ_p-norm attacks (Kang et al., 2019; Tramèr & Boneh, 2019; Schott et al., 2019).

To address this problem, recent works have attempted to improve the robustness against several types of ℓ_p-norm perturbation. Schott et al. (2019) proposed Analysis by Synthesis (ABS), which used multiple variational autoencoders to defend ℓ_0, ℓ_2, and ℓ_∞ adversaries. However, ABS only works on the MNIST dataset. Croce & Hein (2020a) proposed a provable adversarial defense against all ℓ_p norms for $p \geq 1$ using a regularization term. However, it is not applicable to the empirical setting, since it only guarantees robustness for very small perturbations (*e.g*, 0.1 and $2/255$ for ℓ_2 and ℓ_∞ on CIFAR-10). Tramèr & Boneh (2019) tried to defend against multiple perturbation types (ℓ_1, ℓ_2, and ℓ_∞) by combining different types of adversarial examples for adversarial training. Specifically, they introduced two training strategies, "MAX" and "AVG", where for each input image, the model is either trained on its strongest adversarial example or all types of perturbations. More recently, Maini et al. (2020b) proposed multi steepest descent (MSD), and showed that a simple modification to standard PGD adversarial training improves robustness to ℓ_1, ℓ_2, and ℓ_∞ adversaries. Another

concurrent work (Maini et al., 2020a) proposed a two-stage pipeline with multiple models by categorizing the input and then sending them to corresponding constitute model.

There also exist another line of work, which focuses on improving model robustness against multiple non-ℓ_p adversarial attacks (*e.g*, adversarial patches) and unseen perturbations (*e.g*, corruptions) (Laidlaw et al., 2021; Lin et al., 2020; Kang et al., 2019).

In this work, we follow (Tramèr & Boneh, 2019; Maini et al., 2020b) to focus on defense against ℓ_1, ℓ_2, and ℓ_∞ adversarial perturbations, which are the most representative and commonly used perturbations. However, we propose a completely different perspective and solution to the problem, and we outperform them by large margins, *i.e*, 10-20%.

2.2 Batch Normalization

BN (Ioffe & Szegedy, 2015) is typically used to stabilize and accelerate DNN training. A number of normalization techniques have been proposed to improve BN for style-transfer (Huang & Belongie, 2017) and domain adaption (Li et al., 2017; Chang et al., 2019; Deecke et al., 2019; Huang et al., 2020). Let $\mathbf{x} \in \mathbb{R}^d$ denote the input to a neural network layer. During training, BN normalizes each neuron/channel within m mini-batch data by

$$\hat{\mathbf{x}}_j = BN(\mathbf{x}_j) = \gamma_j \frac{\mathbf{x}_j - \mu_j}{\sqrt{\sigma_j^2 + \xi}} + \beta_j, \;\; j = 1, 2, ..., d, \tag{1}$$

where $\mu_j = \frac{1}{m}\sum_{i=1}^{m} \mathbf{x}_j^{(i)}$ and $\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{x}_j^{(i)} - \mu_j)^2$ are the mini-batch mean and variance for each neuron, respectively, and ξ is a small number to prevent numerical instability. The learnable parameters γ and β are used to recover the representation capacity. During inference, the population statistics of mean $\hat{\mu}$ and variance $\hat{\sigma}^2$ are used in Eqn. 1, which are usually calculated as the running average over different training iterations t with update factor α:

$$\begin{cases} \hat{\mu}^t = (1-\alpha)\hat{\mu}^{t-1} + \alpha\mu^{t-1}, \\ (\hat{\sigma}^t)^2 = (1-\alpha)(\hat{\sigma}^{t-1})^2 + \alpha(\sigma^{t-1})^2. \end{cases} \tag{2}$$

We will provide comparisons of our GBN and other normalization methods in Section 3.4 and 4.2.

3 Gated Batch Normalization

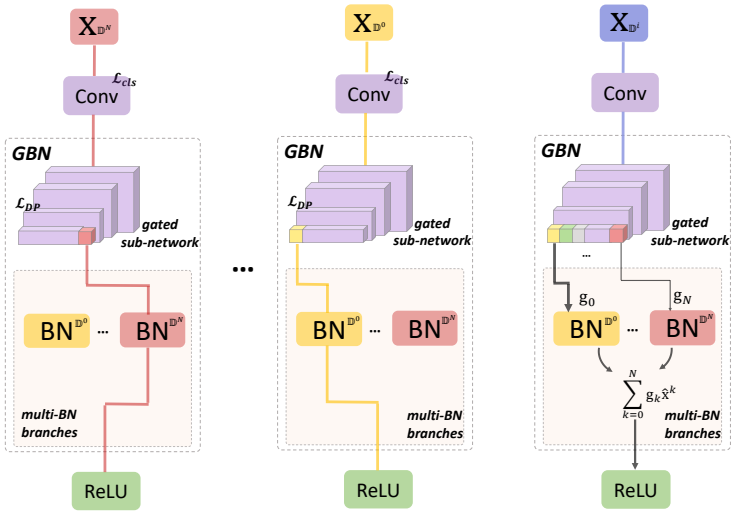
In this section, we first illustrate our multi-domain hypothesis, which states that different ℓ_p bounded adversarial examples are drawn from different domains. Motivated by that, we propose Gated Batch Normalization (GBN), which improves model robustness against multiple adversarial perturbations.

3.1 Motivation: Multi-domain Hypothesis

We assume N *adversarial perturbation types*[2], each characterized by a set $\mathbb{S}^k$ of perturbations for an input $\mathbf{x}$. Let $\mathbb{D}^0$ denote the set of clean examples, and $\mathbb{D}^k (k = 1, ..., N)$ denote the set of adversarial examples generated by the k-th adversarial perturbation type. An adversarial example of the k-th type $\mathbf{x}_{adv}^k$ is generated by the pixel-wise addition of the perturbation δ^k, *i.e*, $\mathbf{x}_{adv}^k = \mathbf{x} + \delta^k$. Our hypothesis states that different ℓ_p norm bounded adversarial perturbations $\mathbb{D}^k$ (for all $k \geq 0$, including the clean examples) are drawn from different domains.

We empirically verify the hypothesis by training a DNN with each BN layer composed of 4 BN branches. During training, we construct different mini-batch of different $\mathbb{D}^k$, *i.e*, clean, ℓ_1, ℓ_2, and ℓ_∞ adversarial images, to estimate the statistics of each BN branch, and optimize the model parameters using the sum of the 4 losses (*c.f.* Appendix A.1). Figure 1 show that different $\mathbb{D}^k$ induce significantly different running statistics, according to the different running means and variances (See Appendix E.1 for fourier studies). During inference, adversaries are likely to generate different perturbation types with different ratios. According to our Theorem 1 (*c.f.* proof in Appendix B), by training with a mixture of perturbation types on a single BN, there inevitably exists a domain gap between training and testing data, which causes the performance degeneration (Benz et al., 2021).

[2]In this work, we consider $N = 3$ adversarial perturbation types: ℓ_1, ℓ_2, and ℓ_∞.



(a) Training of GBN (b) Inference of GBN

Figure 2: The training and inference procedures of GBN. During training, samples from different domains (denoted by different colors) are fed into corresponding BN branches to update the running statistics. During inference, given mini-batch data (from unknown domain), the gated sub-network first predicts the domain of input $\mathbf{x}$ (the confidences are denoted by lines in different thickness), and then jointly normalizes it using multiple BN branches.

Theorem 1. *For a specific batch normalization layer, assuming the corresponding input feature set of the dataset with N adversarial perturbation types can be expressed as $\mathbb{F} = [\mathbb{F}^1, ..., \mathbb{F}^N]$, the k-th type data $\mathbb{F}^k$ follows the Gaussian distribution $\mathcal{N}(\mu_k, \sigma_k)$. When the sampling probability of $\mathbb{F}$ is $\mathbf{w}; \|\mathbf{w}\|_{\ell 1} = 1$ and $\mathbf{w}' = \mathbf{w} + \mathbf{e}_w; \|\mathbf{w}'\|_{\ell 1} = 1$ in two different sets $\mathbb{S}_1$ and $\mathbb{S}_2$, respectively, the difference of the mixture distribution statistics between $\mathbb{S}_1$ and $\mathbb{S}_2$ can be expressed as $\Delta_\mu = \mathbf{e}_w \boldsymbol{\mu}^T$ and $\Delta_\sigma = \mathbf{e}_w \mathbf{t}^T - 2(\mathbf{w}\boldsymbol{\mu}^T)(\mathbf{e}_w\boldsymbol{\mu}^T) - (\mathbf{e}_w\boldsymbol{\mu}^T)^2$, where $\boldsymbol{\mu} = [\mu_1, ..., \mu_k]$ and $\mathbf{t} = [\mu_1^2 + \sigma_1^2, ..., \mu_k^2 + \sigma_k^2]$, $\Delta_\mu = 0$ and $\Delta_\sigma = 0 \iff e_w = \mathbf{0}$.*

We also verify this by training several individual models with different ratios of perturbation types for the estimation of the BN layer (see Appendix A.2). Thus, instead of learning a mixture distributions with a single BN, we utilize a multi-branch BN layer, in which each BN branch is in charge of one corresponding perturbation type (*i.e*, domain). Besides, the output of each perturbation is normalized by corresponding BN towards learning domain-invariant representation.

One remaining problem is that the model does not know the input domain during inference. Existing work suggested that adversarial examples are separable from clean examples (Metzen et al., 2018), and clean/adversarial examples are drawn from different domains (Xie & Yuille, 2020). Therefore, we further introduce a gated sub-network to separate perturbations from different domains during inference, *i.e*, separates domain-specific (perturbation-specific) information.

To sum up, the multi-domain hypothesis motivates our design of GBN, which consists of a gated sub-network and a multi-branch BN layer.

3.2 Gated Batch Normalization Architecture

To train a perturbation-invariant predictor and obtain domain-invariant representations for multiple ℓ_p bounded adversarial perturbations, we use the BN layer statistics to align data from the mixture distribution. Specifically, GBN uses a BN layer with multiple branches $\Psi = \{BN^k(\cdot), k = 0, ..., N\}$ during training, where each branch $BN^k(\cdot)$ is exclusively in charge of the domain $\mathbb{D}^k$. The aligned data are then aggregated as the input to subsequent layers to train a classifier, which achieves the minimum risk across different domains (*i.e*, the best robustness to different perturbations with a high accuracy on clean data).

To separate the input domain during inference and calculate the normalized output, GBN utilizes a gated sub-network $\Phi_\theta(\mathbf{x})$ to predict the domain for each layer input $\mathbf{x}$, and we will illustrate how to train $\Phi_\theta(\mathbf{x})$ in Section 3.3. Given the output of sub-network $\mathbf{g} = \Phi_\theta(\mathbf{x}) \in \mathbb{R}^{N+1}$, we calculate the

normalized output in a soft-gated way:

$$\hat{\mathbf{x}} = GBN(\mathbf{x}) = \sum_{k=0}^{N} \mathbf{g}_k \hat{\mathbf{x}}^k, \tag{3}$$

where $\mathbf{g}_k$ represents the confidence of $\mathbf{x}$ belonging to the k-th domain; $\hat{\mathbf{x}}^k$ is the normalized output of $BN^k(\cdot)$, which uses the population statistics of the domain $\mathbb{D}^k$ to perform the normalization.

We provide an overview of the inference procedure in Figure 2(b) and Algorithm 1. In the Appendix E.5, we discussed an alternative approach that takes the top-1 prediction of $\mathbf{g}$ (the hard label), and show that our soft-label version achieves slightly better results than the hard-label one.

Our GBN aims to disentangle domain-invariant features and domain-specific features: (1) the distributions of different domains are aligned by their normalized outputs (all are standardized distributions), which ensures that the following linear layers learn domain-invariant representations; and (2) the domain-specific features for each domain $\mathbb{D}^k$ are obtained by the population statistics $\{\hat{\mu}^k, \hat{\sigma}^k\}$ of its corresponding BN branch $BN^k(\cdot)$. As shown in Figure 1, $\{\hat{\mu}^k, \hat{\sigma}^k\}$ of $BN^k(\cdot)$ captures the domain-specific statistics.

Algorithm 1 Inference of Gated Batch Normalization (GBN)

Input: Layer input $\mathbf{x}$
Output: Normalized output by GBN

1: Compute the output of gated sub-network $\mathbf{g} = \Phi_\theta(\mathbf{x})$
2: **for** k in N+1 domains **do**
3: Let $\mathbf{x}$ go through $BN^k(\cdot)$ and obtain the normalized output $\hat{\mathbf{x}}^k$ based on Eqn. 1.
4: **end for**
5: Compute GBN output based on Eqn. 3.

3.3 Training

We provide an overview of the training procedure in Figure 2(a), and the details are in Algorithm 2. Specifically, for each mini-batch $\mathcal{B}^0$ consisting of clean samples, we use PGD (Madry et al., 2018) to generate batches of adversarial examples $\mathcal{B}^k$ ($k \in \{1, ..., N\}$) for each domain $\mathbb{D}^k$. To capture the domain-specific statistics of different perturbation types, given the mini-batch data $\mathcal{B}^k$ from domain $\mathbb{D}^k$, we ensure that $\mathcal{B}^k$ goes through its corresponding BN branch $BN^k(\cdot)$, and we use Eqn. 1 to compute the normalized output $\widehat{\mathcal{B}}^k$. The population statistics $\{\hat{\mu}^k, \hat{\sigma}^k\}$ of $BN^k(\cdot)$ are updated based on Eqn. 2. In other words, we disentangle the mixture distribution for normalization, and apply separate BN branches to different perturbation types for statistics estimation.

To train the gated sub-network $\mathbf{g}$, we provide the supervision of the input domain for each training sample. Specifically, we introduce the domain prediction loss $\mathcal{L}_{DP}$:

$$\mathcal{L}_{DP} = \sum_{k=0}^{N} \sum_{(\mathbf{x},k)\in\mathbb{D}^k} \ell(\Phi_\theta(\mathbf{x}), k). \tag{4}$$

Finally, we optimize the parameters Θ of the entire neural network (*e.g*, the weight matrices of the convolutional layers, except for the gated sub-network) using the classification loss $\mathcal{L}_{cls}$:

$$\mathcal{L}_{cls} = \sum_{k=0}^{N} \sum_{(\mathbf{x},\mathbf{y})\in\mathbb{D}^k} \ell(f_\Theta(\mathbf{x}; BN^k(\cdot)), \mathbf{y}). \tag{5}$$

3.4 Analysis and Discussions

Computational cost. As for the parameters, we have N+1 pairs of $\{\hat{\mu}, \hat{\sigma}\}$ to estimate compared to one in standard BN. Regarding the time consumption, in the adversarial learning literature (Madry et al., 2018), the most time-consuming procedure in adversarial training is generating adversarial examples. Although we introduce more parameters, the computational cost of our approach is comparable to existing adversarial defenses, and we defer more discussion to the Appendix E.4).

Algorithm 2 Training of Gated Batch Normalization (GBN) for Each Iteration.

Input: Network f with GBN
Output: Model parameters Θ and θ

1: Given the mini-batch data $\mathcal{B}^0$, use PGD algorithm to generate batches of adversarial examples $\mathcal{B}^k$ ($k \in \{1, ..., N\}$) of different perturbation types
2: **for** k in N+1 domains **do**
3: Let $\mathcal{B}^k$ go through $BN^k(\cdot)$ at each layer to obtain normalized outputs $\widehat{\mathcal{B}}^k$, using Eqn. 1.
4: Update the population statistics $\{\hat{\mu}^k, \hat{\sigma}^k\}$ of $BN^k(\cdot)$ at each layer, using Eqn. 2.
5: **end for**
6: Update θ (parameters of the gated sub-network at each layer) based on Eqn. 4.
7: Update Θ (parameters of the whole network) based on Eqn. 5.

Comparison over other normalization techniques. As for Mode Normalization (MN) (Deecke et al., 2019), it aims to learn a mixture distribution for multi-task learning (several domains), while our GBN separates domain-specific statistics for different domains and then aligns data to build domain-invariant representations. As for (Xie & Yuille, 2020; Xie et al., 2020), they apply separate BNs to clean and adversarial examples for statistics estimation, which try to improve image recognition or analyze the properties of BN to adversarial training. They manually selected the BN branch for each input image, which requires prior knowledge of the input domain (the same as domain adaptation). By contrast, we aim to defend against multiple ℓ_p bounded adversarial perturbations, and our GBN does not require the knowledge of which domain the input belongs to during inference.

4 EXPERIMENTS

We evaluate the effectiveness of our GBN block to simultaneously defend against ℓ_1, ℓ_2, and ℓ_∞ perturbations, which are the most representative and commonly used adversarial perturbation types. We conduct experiments on image classification benchmarks, including MNIST (LeCun, 1998), CIFAR-10 (Krizhevsky & Hinton, 2009), and Tiny-ImageNet (Wu et al., 2017).

4.1 EXPERIMENTAL SETUP

Architectures and hyperparameters. We use LeNet (LeCun et al., 1998) for MNIST; ResNet-20 (He et al., 2016), VGG-16 (Simonyan & Zisserman, 2015), and WRN-28-10 (Zagoruyko & Komodakis, 2016) for CIFAR-10; and ResNet-34 (He et al., 2016) for Tiny-ImageNet. For fair comparisons, we keep the architecture and main hyper-parameters the same for GBN and other baselines. Unless otherwise specified, we add GBN in all layers. See Appendix C and D for details.

Adversarial attacks. We follow existing guidelines (Schott et al., 2019; Tramèr & Boneh, 2019; Maini et al., 2020b) and incorporate multiple adversarial attacks for different perturbation types. For MNIST, the magnitude of perturbation for ℓ_1, ℓ_2, and ℓ_∞ is $\epsilon = 10$, 2, 0.3. For CIFAR-10 and Tiny-ImageNet, the perturbation magnitude for ℓ_1, ℓ_2, and ℓ_∞ is $\epsilon = 12$, 0.5, 0.03. For ℓ_1 attacks, we adopt PGD (Madry et al., 2018), and BBA (Brendel et al., 2019); for ℓ_2 attacks, we use PGD, C&W (Carlini & Wagner, 2017), Gaussian noise (Rauber et al., 2017), and boundary attack (BA) (Brendel et al., 2018); For ℓ_∞ attacks, we use PGD, FGSM (Goodfellow et al., 2014), SPSA (Uesato et al., 2018), Nattack (Li et al., 2019), Momentum Iterative Method (MI-FGSM) (Dong et al., 2018), C&W, and AutoAttack (Croce & Hein, 2020b). We adopt FoolBox (Rauber et al., 2017) for implementation. To demonstrate that our GBN does not introduce obfuscated gradients, we use white-box and black-box adversarial examples, generated with both gradient-based and gradient-free algorithms. See Appendix D.1 for more implementation details.

Adversarial defenses. We compare with existing defenses against the union of ℓ_1, ℓ_2, ℓ_∞ adversaries.[3] We compare with ABS[4] (Schott et al., 2019), MAX, AVG (Tramèr & Boneh, 2019), and MSD (Maini et al., 2020b). For completeness, we also compare with TRADES (Zhang et al., 2019),

[3] Schott et al. (2019) consider the ℓ_0 perturbations, which is subsumed within the ℓ_1 ball of the same radius.
[4] ABS is designed for MNIST. Due to the limited code, we use the results reported in (Schott et al., 2019).

Table 1: Model accuracies on different datasets (%). We also provide the Standard Deviation for *All attacks* of each method on each dataset.

(a) LeNet on MNIST

	Vanilla	PAT	TRADES	P_1	P_2	P_∞	AVG	MAX	ABS	MSD	MN	MBN	**GBN(ours)**
ℓ_1 attacks	5.2	50.6	10.1	0.01	41.9	11.1	30.9	31.8	/	41.9	24.2	60.9	**73.2**
ℓ_2 attacks	1.4	54.8	12.0	0.0	49.3	15.1	60.2	59.0	83.0	71.4	18.9	65.1	**93.2**
ℓ_∞ attacks	0.0	1.3	**77.0**	0.0	0.0	50.1	45.8	58.8	17.0	38.9	0.0	19.2	69.1
All attacks	0.0 ±0.06	1.2 ±0.06	9.8 ±0.12	0.0 ±0.15	0.0 ±0.18	10.4 ±0.09	29.7 ±0.22	31.2 ±0.21	17.0 ±0.00	38.9 ±0.24	0.0 ±0.04	19.2 ±0.36	**68.9** ±0.30
Clean	**99.3**	91.2	99.1	98.9	99.1	98.4	98.7	98.2	99.0	97.3	98.1	98.7	98.7

(b) ResNet-20 on CIFAR-10

	Vanilla	PAT	TRADES	P_1	P_2	P_∞	AVG	MAX	MSD	MN	MBN	**GBN(ours)**
ℓ_1 attacks	0.0	34.3	16.1	22.3	34.2	17.9	44.9	33.3	43.7	39.8	44.9	**57.7**
ℓ_2 attacks	0.0	48.1	59.5	1.1	51.0	50.3	59.1	56.0	58.9	30.0	20.8	**68.9**
ℓ_∞ attacks	0.0	25.5	45.0	0.0	19.3	42.1	29.2	25.1	38.0	13.2	40.1	**49.9**
All attacks	0.0 ±0.03	25.1 ±0.38	16.0 ±0.32	0.0 ±0.04	19.0 ±0.42	17.8 ±0.38	28.2 ±0.51	24.9 ±0.45	37.9 ±0.58	13.0 ±0.52	20.7 ±0.61	**48.7** ±0.66
Clean	**89.7**	62.8	86.9	84.3	87.9	84.2	80.6	77.0	79.1	82.3	79.4	80.7

(c) ResNet-34 on Tiny-ImageNet

	Vanilla	PAT	TRADES	P_1	P_2	P_∞	AVG	MAX	MSD	MN	MBN	**GBN(ours)**
ℓ_1 attacks	5.8	13.4	25.6	32.1	25.8	24.8	27.3	17.0	7.6	8.6	36.6	**44.2**
ℓ_2 attacks	10.3	13.2	31.6	30.7	32.3	31.2	29.1	24.2	11.0	17.9	31.1	**42.1**
ℓ_∞ attacks	0.0	2.2	9.4	0.44	0.76	5.2	4.5	2.7	5.1	6.8	19.4	**38.7**
All attacks	0.0 ±0.06	2.1 ±0.34	9.2 ±0.46	0.41 ±0.21	0.74 ±0.36	5.1 ±0.53	4.3 ±0.55	2.2 ±0.58	5.0 ±0.69	6.7 ±0.63	19.0 ±0.52	**38.5** ±0.62
Clean	**54.2**	20.5	43.1	52.8	54.3	44.4	41.6	36.3	28.8	46.7	45.7	43.4

PAT (Laidlaw et al., 2021), and PGD adversarial training (Madry et al., 2018) with a single perturbation type (*i.e*, the model is trained on ℓ_1, ℓ_2, or ℓ_∞, denoted as P_1, P_2, and P_∞ respectively).

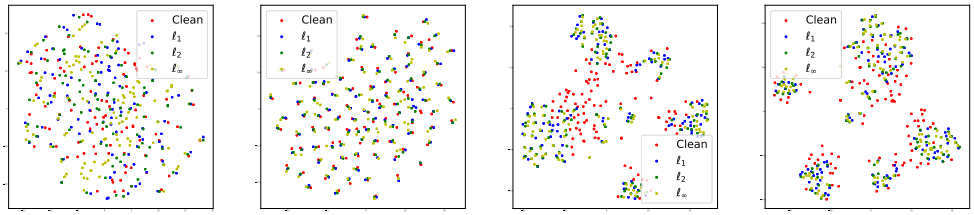
Normalization techniques. We further compare our methods to existing normalization techniques, including MN and MBN. MN extends BN to more than a single mean and variance, detects the modes of the data, and then normalizes samples that share common features. As discussed in Section 2.2, MBN includes two BN branches for clean and adversarial examples respectively. Since manually selecting the BN branch is infeasible in the adversarial setting, we add a 2-way gated sub-network to each MBN block. We defer more implementation details in Appendix D.3 .

4.2 RESULTS

Here, we evaluate model robustness against multiple perturbation types (*i.e*, ℓ_1, ℓ_2, and ℓ_∞) in both the white-box and black-box settings. For each perturbation type, we measure the *worst-case* top-1 classification accuracy (the higher the better), where we report the worst result among various attacks. We also report *all attacks*, which represents the worst-case performance over all attacks of different perturbations for each input (Maini et al., 2020b). Specifically, given an image, if any attack generates an adversarial example that leads to the wrong prediction, we mark it as a failure. For all attacks except FGSM and APGD$_{CE}$ (we use the default setting in AutoAttack), we ran 5 random restarts for each input. For GBN, we trained each model 3 times, and the results are similar.

The results on MNIST using LeNet, CIFAR-10 using ResNet-20, and Tiny-ImageNet using ResNet-34 are shown in Table 1. In the Appendix E.2, we present the detailed breakdown results for individual attacks, and more experimental results on CIFAR-10 using VGG-16 and WRN-28-10. We observe that: (1) on multiple perturbation types (*i.e*, ℓ_1, ℓ_2, ℓ_∞), our GBN consistently outperforms the baselines by large margins of over 10% on all three datasets; (2) due to the trade-off between adversarial robustness and standard accuracy (Tsipras et al., 2019), our clean accuracy is lower than the vanilla model. However, GBN maintains a comparatively high clean accuracy compared to other adversarial defenses; and (3) GBN is easy to train on different datasets, while other defenses for multiple perturbation types (*e.g*, MSD) are hard to converge on large datasets such as Tiny-ImageNet.

We also evaluate GBN on PGD-ℓ_∞ attacks using different iteration steps (*i.e*, 50, 100, 200, and 1000). GBN still outperforms others and remains a high level of robustness (*c.f.* Appendix E.3).



(a) before GBN block (b) after GBN block (c) before BN block (d) after BN block

Figure 3: We use the BN block at layer $conv3\_1$ of a VGG-16 trained using AVG, and the GBN block at layer $conv3\_1$ of a VGG-16. We visualize the features before and after BN/GBN blocks.

Comparison with other normalization methods. As shown in Table 1, GBN outperforms MN and MBN by large margins (up to 60%). MN and MBN do not align the feature representations for different data distributions. Specifically, MN aims to preserve the diversity among different distributions, by learning the mixture of Gaussian; as for the modified MBN, the data from different perturbation types are mixed in the same BN. For our GBN, it first aligns the distribution among different perturbations by using one BN branch to handle one perturbation type (learn domain-specific features), and then the aligned distribution contributes to the subsequent modules or layers (*e.g*, convolutional layer) for learning domain-invariant representations. GBN ensures that the normalized output for each perturbation type has a Gaussian distribution, and these Gaussian distributions among different perturbations are well aligned. As the visualization using t-SNE (Laurens & Hinton, 2008) shown in Figure 3, the features of different domains are aligned to domain-invariant representations after GBN, *i.e*, one clean example (the red dot) and its corresponding adversarial examples (the blue, green, and yellow dots) tend to be close in the space of normalized output. However, after the BN block, distances between the clean example and its corresponding adversarial examples are still far.

Different variants of white-box attacks. We further examine the performance of GBN against white-box attacks specially designed for GBN, aiming to provide a more rigorous analysis of our defense. We conduct the following two experiments: (1) attacking the gated sub-networks, *i.e*, generating adversarial examples to fool the classification of the gated sub-networks; (2) manually selecting the BN branch to generate adversarial examples. All experiments are conducted on CIFAR-10 using ResNet-20. For the first type of attack, given clean, ℓ_1, ℓ_2, and ℓ_∞ adversarial examples, we then perturb these samples to fool the gated sub-networks at all layers using PGD attacks. Our whole model achieves 73.2%, 71.4%, 72.4%, and 71.8% classification accuracy for fooling the gated sub-networks to wrong predictions towards clean, ℓ_1, ℓ_2, and ℓ_∞ domains. For the second attack, GBN remains the same level of robustness (see Table 2).

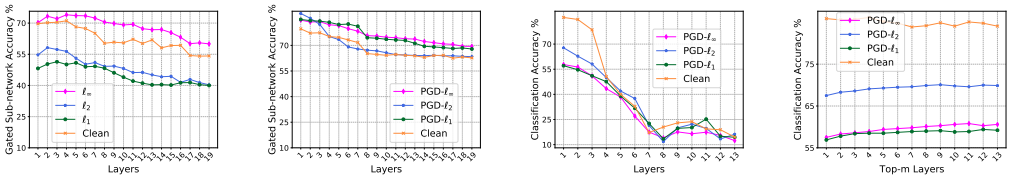
Compared to attacking the full model, as done in Table 1, the model accuracies of these variants are higher, indicating that our main white-box experiments demonstrate the worst-case results for our defense. We conjecture that: (1) our GBN does not rely on obfuscated gradients (Athalye et al., 2018); and (2) different GBN layers use different features to predict the domains, thus it is difficult to simultaneously fool all GBN layers under the magnitude constraint (see Figure 4(a) for gated sub-networks prediction accuracy using the first type of attacks). See more results in Appendix E.6.

Table 2: White-box attacks on MNIST and CIFAR-10 by manually selecting BN branches to generate adversarial examples (BN^0, BN^1, BN^2, and BN^3 denotes the BN branch for clean, ℓ_1, ℓ_2, and ℓ_∞ adversarial examples in GBN, respectively). Results are shown in model accuracy (%).

	MNIST				CIFAR-10			
	BN^0	BN^1	BN^2	BN^3	BN^0	BN^1	BN^2	BN^3
PGD-ℓ_1	87.2	86.9	87.1	97.0	58.9	58.6	59.4	60.3
PGD-ℓ_2	98.1	98.4	97.8	98.5	69.8	69.9	69.7	69.7
PGD-ℓ_∞	97.1	98.3	96.6	96.4	60.6	61.3	61.1	60.1
Clean accuracy	98.7	98.7	98.7	98.7	80.7	80.7	80.7	80.7

4.3 Ablation Study

Predictions of the gated sub-network. We provide the gated sub-networks classification accuracy for the domain of different input samples. On CIFAR-10 (Figure 4(b)), the prediction accuracy of the gated sub-network drops with the increasing of the layer depth. The reason might be that low-level statistics learned in shallow layers are easier to train (Asano et al., 2020), while the features



(a) attack gated sub-net (b) attack whole network (c) single layer study (d) top-m layers study

Figure 4: Gated sub-network prediction accuracy in different scenarios (ResNet-20 on CIFAR-10): (a) shows the results of fooling gated sub-networks in all GBN layers (specially designed for attacking GBN); (b) shows the results of attacking whole network (common white-box attacks for the model). Adding GBN to different layers (VGG-16 on CIFAR-10): (c) shows the results of adding GBN to single layers; (d) shows the results of adding GBN to top-m layers.

Table 3: Classification accuracy (%) of ResNet-20 on CIFAR-10, where models need to generalize to attack types not included for training (the higher the better).

	Vanilla	AVG	MAX	MSD	GBN	AVG$_{\ell_{1+\infty}}$	MSD$_{\ell_{1+\infty}}$	GBN$_{\ell_{1+\infty}}$	AVG$_{\ell_{2+\infty}}$	MSD$_{\ell_{2+\infty}}$	GBN$_{\ell_{2+\infty}}$
PGD-ℓ_1	31.7	47.4	38.1	49.9	**58.5**	50.6	50.0	**57.4**	30.0	23.9	**40.9**
PGD-ℓ_2	32.1	59.2	58.0	62.8	**69.5**	59.2	60.1	**64.7**	58.1	56.7	**65.4**
PGD-ℓ_∞	3.2	42.1	39.7	44.5	**60.1**	35.6	44.4	**57.2**	34.5	46.4	**56.8**
Clean accuracy	89.7	80.6	77.0	79.1	**80.7**	74.4	71.7	**81.3**	77.0	73.4	**81.1**

from different domains are more and more entangled as the layer depth increases, making it harder for models to separate them (*c.f.* Appendix E.8). We suggest to improve the gated sub-network prediction accuracy to further improve model robustness.

Robustness against unseen ℓ_p perturbations. We examine the generalization of GBN to unseen ℓ_p bounded adversarial perturbations by training a variant of GBN that only includes 3 BN branches. These GBN models are trained on two ℓ_p perturbations only, but are evaluated on all the three ℓ_p perturbations, including the unseen perturbation. Unsurprisingly, compared to the full GBN model with 4 BN branches, the robustness of the held-out perturbation type decreases. However, the robustness is still significantly better than other baselines with the same setting, and sometimes even outperforms others that are trained on all perturbation types (*c.f.* Table 3).

Adding GBN to different layers. First, we add GBN to different single layers. As shown in Figure 4(c), the standard performance and adversarial robustness decrease as we only add GBN into the deeper layers. The reasons might be: (1) shallow layers are more critical to model robustness (Liu et al., 2021) and (2) the features from different domains are highly entangled in the deeper layers, which are harder to separate (*c.f.* Appendix E.8). We then add GBN to layer groups (*i.e*, top-m layers), and present the results in Figure 4(d). The robustness improves as more layers are involved, while the clean accuracy remains comparatively stable. Therefore, we add GBN in all layers by default (See more studies in Appendix E.7 including using gated subnetworks only in the first GBN).

5 Conclusions

Most adversarial defenses are typically tailored to a single perturbation type (*e.g*, small ℓ_∞-noise), but offer no guarantees against multiple ℓ_p bounded adversarial attacks. To better understand this phenomenon, we explored the *multi-domain* hypothesis that different ℓ_p bounded adversarial perturbations are drawn from different domains. Thus, we propose a novel building block for DNNs, *Gated Batch Normalization* (GBN), which consists of a gated network and a multi-branches BN layer. The gated sub-network separates different perturbation types, and each BN branch is in charge of a single perturbation type and learns the domain-specific statistics for input transformation. Then, features from different branches are aligned as domain-invariant representations for the subsequent layers. Extensive experiments on MNIST, CIFAR-10, and Tiny-ImageNet demonstrate that our method outperforms previous proposals against ℓ_1, ℓ_2, and ℓ_∞ adversarial attacks by large margins of 10-20%.

REFERENCES

Maksym Andriushchenko, Francesco Croce, Nicolas Flammarion, and Matthias Hein. Square attack: a query-efficient black-box adversarial attack via random search. In *European Conference on Computer Vision*, 2020.

Yuki M Asano, Christian Rupprecht, and Andrea Vedaldi. A critical analysis of self-supervision, or what we can learn from a single image. In *International Conference on Learning Representations*, 2020.

Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *International Conference on Machine Learning*, 2018.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Philipp Benz, Chaoning Zhang, Adil Karjauv, and In So Kweon. Revisiting batch normalization for improving corruption robustness. In *WACV*, 2021.

Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *International Conference on Learning Representations*, 2018.

Wieland Brendel, Jonas Rauber, Matthias Kümmerer, Ivan Ustyuzhaninov, and Matthias Bethge. Accurate, reliable and fast robustness evaluation. In *Advances in Neural Information Processing Systems*, 2019.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy*, 2017.

Woong-Gi Chang, Tackgeun You, Seonguk Seo, Suha Kwak, and Bohyung Han. Domain-specific batch normalization for unsupervised domain adaptation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In *International Conference on Machine Learning*, 2017.

Francesco Croce and Matthias Hein. Provable robustness against all adversarial l_p-perturbations for $p \geq 1$. In *International Conference on Learning Representations*, 2020a.

Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *International Conference on Machine Learning*, 2020b.

Francesco Croce and Matthias Hein. Minimally distorted adversarial examples with a fast adaptive boundary attack. In *International Conference on Machine Learning*, 2020c.

Lucas Deecke, Iain Murray, and Hakan Bilen. Mode normalization. In *International Conference on Learning Representations*, 2019.

Yinpeng Dong, Fangzhou Liao, Tianyu Pang, and Hang Su. Boosting adversarial attacks with momentum. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Logan Engstrom, Andrew Ilyas, and Anish Athalye. Evaluating and understanding the robustness of adversarial logit pairing. *arXiv preprint arXiv:1807.10272*, 2018.

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

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdelrahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, and Tara N. Sainath. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 2012.

Lei Huang, Jie Qin, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. Normalization techniques in training dnns: Methodology, analysis and application. *arXiv preprint arXiv:2009.12836*, 2020.

Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *IEEE International Conference on Computer Vision*, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015.

Daniel Kang, Yi Sun, Dan Hendrycks, Tom Brown, and Jacob Steinhardt. Testing robustness against unforeseen adversaries. *arXiv preprint arXiv:1908.08016*, 2019.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *International Conference on Neural Information Processing Systems*, 2012.

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. In *International Conference on Learning Representations*, 2017.

Cassidy Laidlaw, Sahil Singla, and Soheil Feizi. Perceptual adversarial robustness: Defense against unseen threat models. In *International Conference on Learning Representations*, 2021.

Van Der Maaten Laurens and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 2008.

Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

Yandong Li, Lijun Li, Liqiang Wang, Tong Zhang, and Boqing Gong. Nattack: Learning the distributions of adversarial examples for an improved black-box attack on deep neural networks. In *International Conference on Machine Learning*, 2019.

Yanghao Li, Naiyan Wang, Jianping Shi, Jiaying Liu, and Xiaodi Hou. Revisiting batch normalization for practical domain adaptation. In *International Conference on Learning Representations*, 2017.

Fangzhou Liao, Ming Liang, Yinpeng Dong, Tianyu Pang, Xiaolin Hu, and Jun Zhu. Defense against adversarial attacks using high-level representation guided denoiser. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Wei-An Lin, Chun Pong Lau, Alexander Levine, Rama Chellappa, and Soheil Feizi. Dual manifold adversarial robustness: Defense against lp and non-lp adversarial attacks. In *Advances in Neural Information Processing Systems*, 2020.

Aishan Liu, Xianglong Liu, Jiaxin Fan, Yuqing Ma, Anlan Zhang, Huiyuan Xie, and Dacheng Tao. Perceptual-sensitive gan for generating adversarial patches. In *33rd AAAI Conference on Artificial Intelligence*, 2019.

Aishan Liu, Xianglong Liu, Chongzhi Zhang, Hang Yu, Qiang Liu, and Dacheng Tao. Training robust deep neural networks via adversarial noise propagation. *IEEE Transactions on Image Processing*, 2021.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

Pratyush Maini, Xinyun Chen, Bo Li, and Dawn Song. Perturbation type categorization for multiple ℓ_p bounded adversarial robustness. 2020a. URL `https://openreview.net/pdf?id=Oe2XI-Aft-k`.

Pratyush Maini, Eric Wong, and Zico J. Kolter. adversarial robustness against the union of multiple perturbation model. In *International Conference on Machine Learning*, 2020b.

Jan Hendrik Metzen, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. In *International Conference on Learning Representations*, 2018.

Nicolas Papernot, Patrick Mcdaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. *arXiv preprint arXiv:1511.04508*, 2015.

Jonas Rauber, Wieland Brendel, and Matthias Bethge. Foolbox: A python toolbox to benchmark the robustness of machine learning models. 2017.

Lukas Schott, Jonas Rauber, Matthias Bethge, and Wieland Brendel. Towards the first adversarially robust neural network model on mnist. In *International Conference on Learning Representations*, 2019.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations*, 2015.

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

Florian Tramèr and Dan Boneh. Adversarial training and robustness for multiple perturbations. In *Advances in Neural Information Processing Systems*, 2019.

Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. In *International Conference on Learning Representations*, 2019.

Jonathan Uesato, Brendan O'Donoghue, Aäron van den Oord, and Pushmeet Kohli. Adversarial risk and the dangers of evaluating against weak attacks. In *International Conference on Machine Learning*, 2018.

Jiayu Wu, Qixiang Zhang, and Guoxi Xu. Tiny imagenet challenge. 2017.

Cihang Xie and Alan Yuille. Intriguing properties of adversarial training at scale. In *International Conference on Learning Representations*, 2020.

Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization. In *International Conference on Learning Representations*, 2018.

Cihang Xie, Mingxing Tan, Boqing Gong, Jiang Wang, Alan L Yuille, and Quoc V Le. Adversarial examples improve image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

Dong Yin, Gontijo Raphael Lopes, Jonathon Shlens, D Ekin Cubuk, and Justin Gilmer. A fourier perspective on model robustness in computer vision. In *Advances in Neural Information Processing Systems*, 2019.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *The British Machine Vision Conference*, 2016.

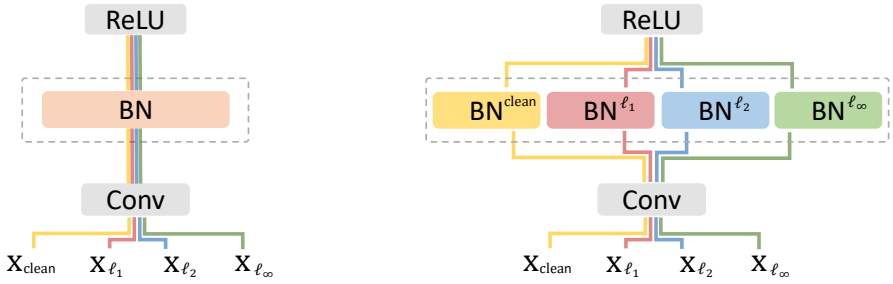
Chongzhi Zhang, Aishan Liu, Xianglong Liu, Yitao Xu, Hang Yu, Yuqing Ma, and Tianlin Li. Interpreting and improving adversarial robustness with neuron sensitivity. *IEEE Transactions on Image Processing*, 2020.

Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P Xing, Laurent El Ghaoui, and Michael I Jordan. Theoretically principled trade-off between robustness and accuracy. 2019.

A EXPERIMENTS FOR MULTI-DOMAIN HYPOTHESIS

A.1 SEPARATING BNS

We provide the experimental setting for separating BNs study in Section 3.1 of the main body. Specifically, we train a DNN on CIFAR-10 with each BN layer composed of 4 branches (*i.e*, we replace each BN with 4 BN branches, the structure of standard BN and multiple BN branches are shown in Figure 5). During training, we construct different mini-batch for clean, ℓ_1, ℓ_2, and ℓ_∞ adversarial images to estimate the normalization statistics of each branch of BN; during inference, given a mini-batch of data from a specific domain, we manually activate the corresponding BN branch and deactivate the others. We update the running statistics of each BN branch separately but optimize the model parameters using the sum of the four losses.



(a) standard BN (b) multiple BN branches

Figure 5: (a): the standard BN structure. (b): the structure with 4 BN branches, which are fed with different mini-batches from different $\mathbb{D}^k$ to estimate the normalization statistics.

A.2 DIFFERENT RATIOS OF PERTURBATION TYPES

We provide the experimental settings and results for training several individual models with different ratios of perturbation types for the estimation of the BN layer. For each individual DNN with a single BN, we use different ratios of perturbation types (*e.g*, clean, ℓ_1, ℓ_2, and ℓ_∞ adversarial examples) to estimate the running statistics of BN, but only use ℓ_1 adversarial examples to optimize the model parameters. Specifically, for the first model M_1, we use ℓ_1 adversarial examples to estimate the running statistics of the BN layer; for the second model M_2, we use ℓ_1 and ℓ_2 adversarial examples to estimate the running statistics of the BN layer; for the third model M_3, we use clean example, ℓ_1, and ℓ_∞ adversarial examples to estimate the running statistics of the BN layer; for the fourth model M_4, we use clean example, ℓ_1, and ℓ_∞ adversarial examples to estimate the running statistics of the BN layer; for the fifth model M_5, we use clean example, ℓ_1, ℓ_2, and ℓ_∞ adversarial examples to estimate the running statistics of the BN layer. For all these models, we use ℓ_1 adversarial examples to optimize the model weights, and we keep the same amount of training data. Thus, for all these models, they use different ratios of perturbations to estimate the BN layer, and use the same data to optimize the model weight.

On CIFAR-10, the predication accuracy for PGD-ℓ_1 attacks (the same settings for our main experiment) from M_1 to M_5 is shown as: **92.5%**, 84.0%, 89.1%, 85.5%, and 83.8%, respectively. M_1 achieves the strongest robustness on ℓ_1 attacks, which empirically verify our Theorem 1 that the model achieves the best performance when the proportion of the different perturbations in the training set is most close to that in the test set.

B PROOF OF THEOREM 1

Theorem 1. *Assuming for a specific batch normalization layer, given the corresponding input feature set of the dataset with N adversarial perturbation types can be expressed as $\mathbb{F} = [\mathbb{F}^1, ..., \mathbb{F}^N]$, the k-th type data $\mathbb{F}^k$ follows the Gaussian distribution $\mathcal{N}(\mu_k, \sigma_k)$. When the sampling probability*

of $\mathbb{F}$ *is* $\mathbf{w}$; $\|\mathbf{w}\|_{\ell 1} = 1$ *and* $\mathbf{w}' = \mathbf{w} + \mathbf{e}_w$; $\|\mathbf{w}'\|_{\ell 1} = 1$ *in training and testing sets, respectively, the difference of the mixture distribution statistics between the training and test sets can be expressed as* $\Delta_\mu = \mathbf{e}_w \boldsymbol{\mu}^T$ *and* $\Delta_\sigma = \mathbf{e}_w \mathbf{t}^T - 2(\mathbf{w}\boldsymbol{\mu}^T)(\mathbf{e}_w \boldsymbol{\mu}^T) - (\mathbf{e}_w \boldsymbol{\mu}^T)^2$, *where* $\boldsymbol{\mu} = [\mu_1, ..., \mu_k]$ *and* $\mathbf{t} = [\mu_1^2 + \sigma_1^2, ..., \mu_k^2 + \sigma_k^2]$, $\Delta_\mu = 0$ *and* $\Delta_\sigma = 0 \iff e_w = \mathbf{0}$.

Proof. Given the corresponding input feature set of the dataset with N adversarial perturbation types can be expressed as $\mathbb{F} = [\mathbb{F}^1, ..., \mathbb{F}^N]$, where the k-th type data $\mathbb{F}^k$ following the Gaussian distribution $\mathcal{N}(\mu_k, \sigma_k)$. When the sampling probability of $\mathbb{F}$ is $\mathbf{w}$; $\|\mathbf{w}\|_{\ell 1} = 1$ in training data $\mathbf{F}_{\text{train}}$, the probability density function of its distribution can be expressed as

$$p(x) = \sum_{k=1}^{n} w_k p_k(x), \tag{6}$$

where $p_k(x)$ is the probability density function of $\mathbb{F}^k$ and is expressed as

$$p_k(x) = \frac{1}{\sigma_k \sqrt{2\pi}} \exp\left(-\frac{(x-\mu_k)^2}{2\sigma_k^2}\right). \tag{7}$$

Let $X_1, X_2, \cdots, X_N$ denote random variables with N component distributions of training data sampled from $\mathbb{F}$, and X denote random variables with mixture distributions. Therefore, for any function $H(\cdot)$, if $\mathbb{E}[H(X_k)]$ exists, and assuming that the component distribution $p_k(x)$ exists, we have

$$\mathbb{E}[H(X)] = \int_{-\infty}^{\infty} H(x) \sum_{k=1}^{N} w_k p_k(x) dx \tag{8}$$

$$= \sum_{k=1}^{N} w_k \int_{-\infty}^{\infty} p_k(x) H(x) \tag{9}$$

$$= \sum_{k=1}^{N} w_k \mathbb{E}\left[H\left(X_k\right)\right] \tag{10}$$

and when $H(x) = (x-\mu)^i$, where μ is the mean of X, we have

$$\mathbb{E}\left[(X-\mu)^j\right] = \sum_{k=1}^{N} w_k \mathbb{E}\left[\left(X_k - \mu_k + \mu_i - \mu\right)^i\right] \tag{11}$$

$$= \sum_{k=1}^{N} w_k \sum_{k=0}^{i} \binom{i}{j} \left(\mu_k - \mu\right)^{i-k} \mathbb{E}\left[\left(X_k - \mu_k\right)^j\right]. \tag{12}$$

Therefore, the mean and variance of X can be expressed as

$$\mathbb{E}[X] = \mu = \sum_{i=1}^{n} w_i \mu_i, \tag{13}$$

$$E\left[(X-\mu)^2\right] = \sigma^2 = \sum_{i=1}^{n} w_i \left[\left(\mu_i - \mu\right)^2 + \sigma_i^2\right] = \sum_{i=1}^{n} w_i \left(\mu_i^2 + \sigma_i^2\right) - \mu^2 \tag{14}$$

$$= \sum_{i=1}^{n} w_i \left(\mu_i^2 + \sigma_i^2\right) - \left(\sum_{i=1}^{n} w_i \mu_i\right)^2. \tag{15}$$

when we let $\mathbf{w} = [w_1, ..., w_N]$, $\boldsymbol{\mu} = [\mu_1, ..., \mu_N]$, $\boldsymbol{\sigma} = [\sigma_1, ..., \sigma_N]$, $\mathbf{t} = [t_1, ..., t_N]$, where $t_k = \mu_k^2 + \sigma_k^2$, the the mean and variance of X can be expressed as

$$\mathbb{E}[X] = \mathbf{w}\boldsymbol{\mu}^\top, \tag{16}$$

$$E\left[(X-\mu)^2\right] = \mathbf{w}\boldsymbol{t}^\top - (\mathbf{w}\boldsymbol{\mu}^\top)^2. \tag{17}$$

And for the test data X', when the sampling probability is expressed as $\mathbf{w}' = \mathbf{w} + \mathbf{e}_w; \|\mathbf{w}'\|_{\ell 1} = 1$, we have

$$\mathbb{E}[X'] = \mathbf{w}'\boldsymbol{\mu}^\top = \mathbf{w}\boldsymbol{\mu}^\top + \mathbf{e}_w\boldsymbol{\mu}^\top, \tag{18}$$

$$E\left[(X'-\mu')^2\right] = \mathbf{w}'\boldsymbol{t}^\top - (\mathbf{w}'\boldsymbol{\mu}^\top)^2 \tag{19}$$

$$= \mathbf{w}\boldsymbol{t}^\top + \mathbf{e}_w\boldsymbol{t}^\top - \left(\mathbf{w}\boldsymbol{\mu}^\top + \mathbf{e}_w\boldsymbol{\mu}^\top\right)^2 \tag{20}$$

$$= \mathbf{w}\boldsymbol{t}^\top + \mathbf{e}_w\boldsymbol{t}^\top - \left(\mathbf{w}\boldsymbol{\mu}^\top\right)^2 - 2\left(\mathbf{w}\boldsymbol{\mu}^\top\right)\left(\mathbf{e}_w\boldsymbol{\mu}^\top\right) - \left(\mathbf{e}_w\boldsymbol{\mu}^\top\right)^2. \tag{21}$$

The we get

$$\Delta_\mu = \mathbb{E}[X'] - \mathbb{E}[X] = \mathbf{e}_w\boldsymbol{\mu}^\top, \tag{22}$$

$$\Delta_\sigma = E\left[(X'-\mu')^2\right] - E\left[(X-\mu)^2\right] \tag{23}$$

$$= \mathbf{e}_w\boldsymbol{t}^\top - 2\left(\mathbf{w}\boldsymbol{\mu}^\top\right)\left(\mathbf{e}_w\boldsymbol{\mu}^\top\right) - \left(\mathbf{e}_w\boldsymbol{\mu}^\top\right)^2. \tag{24}$$

Since $\mathbf{t} > 0$, we can trivially get that $\Delta_\mu = 0 \; and \; \Delta_\sigma = 0 \iff \mathrm{e}_w = \mathbf{0}$.

□

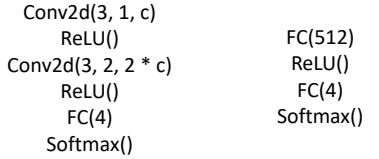
C GBN IMPLEMENATION DETAILS

Network architecture. We first illustrate the architecture of the gated sub-network $\mathbf{g}$ in GBN. As shown in Figure 6, we devise two types of gated sub-network, namely *Conv gate* and *FC gate*.

The notation Conv2d(d, s, k) refers to a convolutional layer with k filters whose size are $d \times d$ convolved with stride s. The notation ReLU$(\cdot)$ denotes the rectified linear unit used as the activation function in the network. FC(m) denotes a fully connected layer with output size m.

Conv gate is a convolutional neural network consisting of two convolutional layers, two ReLU layers and one fully-connected layer. c denotes the input channel of this gate. In the default setting, we use stride 1 for the first convolutional layer and 2 for the second. For the filter sizes in both layers, padding is set to 3 and 1, respectively. We set the fully-connected layer output size m=4 to match the four data distributions used in our experiment.

FC gate is a simple fully-connected neural network that includes two fully-connected layers and one ReLU layer. We use m=512 and m=4 for the two fully-connected layers.



(a) Conv gate (b) FC gate

Figure 6: The architecture of the gated sub-network used in this paper.

To train models containing GBN, we add the GBN block into all layers within a model. Specifically, we set the $\mathbf{g}$ in the first GBN as the *Conv gate* and use the *FC gates* for other GBN blocks to further capture domain-specific information and improve model robustness. We set ξ as 0.00001 and α as 0.1.

Discussion. To empirically prove the effectiveness of the above strategy, we conduct additional experiments using the *Conv gate* and *FC gate* for all GBN blocks. In other words, we train a VGG-16 model with all GBN blocks using the *Conv gate* denoted "Conv$_{all}$", and train another model with all GBN blocks using the *FC gate* denoted "FC$_{all}$". As shown in Table 4, our strategy (denoted "Conv+FC") achieves the greatest robustness. Thus, we use *Conv gate* for all the GBN blocks in the single layer study, and use *Conv gate* for the first GBN and *FC gates* for the other GBN blocks in the layer group study (Section 4.3 in the main body).

We conjecture that there are two reasons for this: (1) the running statistics between different domains in the first layer are almost indistinguishable (*c.f.* Section E.8 in the supplementary material). Thus, solely using the FC layer (*FC gate*) fails to extract sufficient features from the first layer to perform correct classification; (2) using conv layers for all GBN blocks may suffer from over-fitting problem, since only adding GBN into the first layer achieves considerable robustness (as shown in Section 4.3 in the main body). We will further address it in the future studies.

Table 4: Model robustness of VGG-16 on CIFAR-10 (the higher the better).

	Conv$_{all}$	FC$_{all}$	Conv+FC
PGD-ℓ_1	30.9%	35.5%	58.5%
PGD-ℓ_2	29.9%	34.6%	69.5%
PGD-ℓ_∞	20.6%	32.6%	60.1%
Clean accuracy	41.6%	39.6%	80.7%

D Experimental Settings

Dataset details. MNIST is a dataset containing 10 classes of handwritten digits of size 28×28 with 60,000 training examples and 10,000 test instances. CIFAR-10 consists of 60,000 natural scene color images in 10 classes of size 32×32. Tiny-Imagenet has 200 classes of size 64×64, and each class contains 500 training images, 50 validation images, and 50 test images.

Implementation details. Here, we provide the details of the experimental settings for GBN and other baselines on different datasets. All the models for each method on MNIST, CIFAR-10, and Tiny-ImageNet are trained for 40, 40, and 20 epochs, respectively. We set the mini-batch size=64, use the SGD optimizer with weight decay 0.0005 for Tiny-ImageNet and use no weight decay for MNIST and CIFAR-10. We set the learning rate as 0.1 for MNIST and 0.01 for CIFAR-10 and Tiny-ImageNet. For these baselines, we use the published implementations for ABS[5], MAX/AVG[6], MSD[7], TRADES[8], and PAT[9].

D.1 Adversarial Attacks

In this paper, we adopt both the white-box and black-box adversarial attacks. In the white-box setting, adversaries have the complete knowledge of the target model and can fully access the model; while in the black-box setting, adversaries have limited knowledge of the target classifier (*e.g*, its architecture) but can not access the model weights.

[5] https://github.com/bethgelab/AnalysisBySynthesis
[6] https://github.com/ftramer/MultiRobustness
[7] https://github:com/locuslab/robust_union
[8] https://github.com/yaodongyu/TRADES
[9] https://github.com/cassidylaidlaw/perceptual-advex

For ℓ_1 attacks, we use PGD attack (Madry et al., 2018), and Brendel & Bethge attack (BBA) (Brendel et al., 2019).

For ℓ_2 attacks, we use PGD attack (Madry et al., 2018), C&W attack (Carlini & Wagner, 2017), Gaussian noise attack (Rauber et al., 2017), and boundary attack (BA) (Brendel et al., 2018).

For ℓ_∞ attacks, though ℓ_∞ PGD adversary is quite effective, for completeness, we additionally use attacks including, Fast Gradient Sign Method (FGSM) (Goodfellow et al., 2014), SPSA (Uesato et al., 2018), Nattack (Li et al., 2019), Momentum Iterative Method (MI-FGSM) (Dong et al., 2018), and AutoAttack (Croce & Hein, 2020b) (a stronger ensemble of parameter-free attacks, including APGD, FAB (Croce & Hein, 2020c), and Square Attack (Andriushchenko et al., 2020)).

Among the attacks mentioned above, BA, SPSA, Nattack, and Square Attack are black-box attacks. We then provide the hyper-parameters of each attack.

PGD-ℓ_1. On MNIST, we set the perturbation magnitude ϵ=10, iteration number k=50, and step size α=ϵ/10. On CIFAR-10 and Tiny-ImageNet, we set the perturbation magnitude ϵ=12, iteration number k=50, and step size α=0.05.

PGD-ℓ_2. On MNIST, we set the perturbation magnitude ϵ=2.0, iteration number k=100, and step size α=0.1. On CIFAR-10 and Tiny-ImageNet, we set the perturbation magnitude ϵ=0.5, iteration number k=50, and step size α=ϵ/10.

PGD-ℓ_∞. On MNIST, we set the perturbation magnitude ϵ=0.3, iteration number k=50, and step size α=0.01. On CIFAR-10 and Tiny-ImageNet, we set the perturbation magnitude ϵ=0.03, iteration number k=40, and step size α=ϵ/10.

PGD-1000-ℓ_∞. On CIFAR-10, we set the perturbation magnitude ϵ=0.03, iteration number k=1000, and step size α=ϵ/10.

BBA. On MNIST, CIFAR-10 and Tiny-ImageNet, we set the perturbation magnitude ϵ=10, 12, 12 in terms of ℓ_1 norm, respectively. For all datasets, we set the number of optimization steps as 1000, learning rate as 0.001, momentum as 0.8, and the binary search steps as 10.

C&W-ℓ_2. On MNIST, CIFAR-10 and Tiny-ImageNet, we set the perturbation magnitude ϵ=2.0, 0.5, 0.5 in terms of ℓ_2 norm, respectively. For all datasets, we set the number of optimization steps as 10000, each step size as 0.01, and the confidence required for an example to be marked as adversarial as 0.

C&W-ℓ_∞. On CIFAR-10, we set the perturbation magnitude ϵ=0.03 in terms of ℓ_∞ norm. We set the number of optimization steps as 10000, each step size as 0.01, and the confidence required for an example to be marked as adversarial as 0.

Gaussian noise. On MNIST, CIFAR-10 and Tiny-ImageNet, we set the perturbation magnitude ϵ=2.0, 0.5, 0.5 in terms of ℓ_2 norm, respectively.

BA. On MNIST, CIFAR-10 and Tiny-ImageNet, we set the perturbation magnitude ϵ=2.0, 0.5, 0.5 in terms of ℓ_2 norm, respectively. For all datasets, we set the maximum number of steps as 25000, initial step size for the orthogonal step as 0.01, and initial step size for the step towards the target as 0.01.

FGSM. On MNIST, we set the perturbation magnitude ϵ=0.3 in terms of ℓ_∞ norm. On CIFAR-10 and Tiny-ImageNet, we set the perturbation magnitude ϵ=0.03 in terms of ℓ_∞ norm.

MI-FGSM. On MNIST, we set the perturbation magnitude ϵ=0.3, the decay factor μ=1 in terms of ℓ_∞ norm, and set the step number k=10. On CIFAR-10 and Tiny-ImageNet, we set the perturbation magnitude ϵ=0.03, the decay factor μ=1 in terms of ℓ_∞ norm, and set the step number k=10. The step size α=ϵ/k.

SPSA. We set the maximum iteration as 100, the batch size as 8192, and the learning rate as 0.01. The magnitude of perturbation is 0.3 in terms of ℓ_∞ norm on MNIST, and 0.03 in terms of ℓ_∞ norm on CIFAR-10 and Tiny-ImageNet.

NATTACK. We set T=600 as the maximum number of optimization iterations, b=300 for the sample size, variance of the isotropic Gaussian σ^2=0.01, and learning rate as 0.008. The magnitude of

perturbation is 0.3 in terms of ℓ_∞ norm on MNIST, and 0.03 in terms of ℓ_∞ norm on CIFAR-10 and Tiny-ImageNet.

AutoAttack. AutoAttack selects the following variants of adversarial attacks: APGD$_{CE}$ without random restarts, APGD$_{DLR}$, the targeted version of FAB as FAB^T, and Square Attack with one run of 5000 queries. We use 100 iterations for each run of the white-box attacks. For APGD, we set the momentum coefficient α=0.75, ρ=0.75, initial step size $\eta^{(0)}$=2ϵ, where ϵ is the perturbation magnitude in terms of the ℓ_∞ norm. For FAB, we keep the standard hyper-parameters based on the implementation of AdverTorch. For Square Attack, we set the initial value for the size of the squares p = 0.8.

D.2 Adversarial Defenses

ABS. ABS uses multiple variational autoencoders to construct a complex generative architecture to defend against adversarial examples in the MNIST dataset. As for the limited code released by the authors, we directly use the results reported in (Schott et al., 2019). They set the perturbation magnitude ϵ=12, 1.5, 0.3 for ℓ_0, ℓ_2, and ℓ_∞ attack, respectively. ABS used an ℓ_0 perturbation model of a higher radius and evaluated against ℓ_0 attacks. So the reported number is a near estimate of the ℓ_1 adversarial accuracy.

AVG. For each batch of clean data (size=64), we generate corresponding adversarial examples (ℓ_1, ℓ_2, and ℓ_∞) using PGD attack. We train the model using a combination of clean, ℓ_1, ℓ_2, and ℓ_∞ adversarial examples simultaneously. The hyper-parameters of PGD adversaries can be found in Appendix D.1 (PGD-ℓ_1, PGD-ℓ_2, and PGD-ℓ_∞).

MAX. For each batch of clean data (size=64), we generate the strongest adversarial examples (one of the ℓ_1, ℓ_2, and ℓ_∞ attack) using PGD attack. We train the model using a combination of clean examples and the strongest attack. The hyper-parameters of PGD adversaries can be found in Appendix D.1 (PGD-ℓ_1, PGD-ℓ_2, and PGD-ℓ_∞).

MSD. MSD creates a single adversarial perturbation by simultaneously maximizing the worst-case loss over all perturbation models at each projected steepest descent step. For MNIST, we set the perturbation magnitude ϵ=10, 2.0, 0.3 for ℓ_1, ℓ_2, and ℓ_∞ attack, respectively, and iteration number k=100. For CIFAR-10 and Tiny-ImageNet, we set the perturbation magnitude ϵ=12, 0.5, 0.03 for ℓ_1, ℓ_2, and ℓ_∞ attack respectively, and iteration number k=50.

TRADES. TRADES is an adversarial defense method trading adversarial robustness off against accuracy which won 1st place in the NeurIPS 2018 Adversarial Vision Challenge. We set $1/\lambda$=3.0 and set other hyper-parameters as the default values following the original paper.

PAT. PAT is an adversarial training method that can generalize to unforeseen perturbation types without training on them. It generates adversarial examples with a bounded neural perceptual distance to natural images and then uses the generated images to train models. Specifically, we use the proposed Fast Lagrange Perceptual Attack to conduct adversarial training, with setting attack iter=10 and perturbation bound=0.5 for MNIST and CIFAR-10 and 0.25 for Tiny-ImageNet.

P_1**.** We adversarially train the model with 50% clean examples and 50% adversarial examples generated by ℓ_1 PGD attack for each mini-batch data. For PGD attack, on CIFAR-10, we set the perturbation magnitude ϵ=12, iteration number k=50, and step size α=0.05.

P_2**.** We adversarially train the model with 50% clean examples and 50% adversarial examples generated by ℓ_2 PGD attack for each mini-batch data. For PGD attack, on CIFAR-10, we set the perturbation magnitude ϵ=0.5, iteration number k=50, and step size α=ϵ/10.

P_∞**.** We adversarially train the model with 50% clean examples and 50% adversarial examples generated by ℓ_∞ PGD attack for each mini-batch data. For PGD attack, on CIFAR-10, we set the perturbation magnitude ϵ=0.03, iteration number k=40, and step size α=ϵ/10.

D.3 Normalization Techniques

MN. We set the number of modes in MN to 2, which achieves the best performance according to the original paper (Deecke et al., 2019). During training, we feed the model with a mixture of clean, ℓ_1, ℓ_2, and ℓ_∞ adversarial examples using the same setting as AVG.

MBN. The original MBN (Xie et al., 2020; Xie & Yuille, 2020) manually selects the BN branches for clean and adversarial examples during inference, which is infeasible in adversarial defense setting (the model is unaware of the type of inputs). Thus, we add the 2-way gated sub-network in MBN to predict the input domain label; we then keep the following 2 BN branches the same. During training, we compel the clean examples to go through the first BN branch and the adversarial examples (*i.e*, ℓ_1, ℓ_2, and ℓ_∞) to the second BN branch. The adversarial examples are generated via PGD using the same setting as AVG.

E More Experimental Results

E.1 Multiple Perturbations from the Fourier Perspective

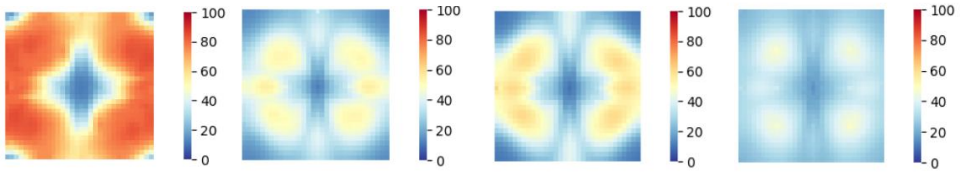


Figure 7: Model sensitivity to additive noise aligned with different Fourier basis vectors on CIFAR-10. From left to right: vanilla, ℓ_1-trained, ℓ_2-trained, and ℓ_∞-trained models. The numbers indicate the model error rates.

We study the differences between different perturbation types from the Fourier perspective (Yin et al., 2019). We evaluate four different ResNet-20 models: a vanilla model and 3 adversarially-trained models using only ℓ_1, ℓ_2, and ℓ_∞ adversarial examples, respectively. We visualize the Fourier heatmap following (Yin et al., 2019) and shift the low frequency components to the center of the spectrum. Error rates are averaged over 1000 randomly sampled images from the test set on CIFAR-10. The redder the zone, the higher the error rate of the model to perturbations of specific frequency. As shown in Figure 7, models trained using different ℓ_p perturbations demonstrate different model weaknesses in the Fourier heatmap (*i.e*, different hot zones). For example, ℓ_2-PAT is more susceptible to high-frequency perturbations while less susceptible to some low-frequency perturbations (the smaller blue zone in the center), while ℓ_∞-PAT is comparatively more robust to middle-frequency perturbations (light yellow zones). Therefore, different perturbation types have different frequency properties, and models trained on different perturbations are sensitive to noise from different frequencies. This observation further demonstrates that different perturbation types may arise from different domains (our multi-domain hypothesis).

E.2 Adversarial Robustness against multiple perturbations

In this part, we provide the breakdown for each individual attack on MNIST, CIFAR-10, and Tiny-ImageNet in Tabel 5, 6, and 7. We also report the results for individual attacks of AutoAttack in 10. Further, we provide the breadkdown for each individual attack on CIFAR-10 using VGG-16 and WideResNet-28-10 in Table 8 and Table 9.

According to the results, our GBN outperforms other methods for almost all attacks by large margins. However, it is reasonable to notice that our GBN shows slightly weaker or comparable performance on some individual attacks compared to defenses trained for the specific perturbation types. For example, P_1 outperforms GBN for PGD-ℓ_1 on CIFAR-10 and TRADES shows better performance for some ℓ_∞ attacks on MNIST.

In summary, our proposed GBN trains robust models in terms of multiple perturbation types (*i.e*, ℓ_1, ℓ_2, ℓ_∞) and outperforms other methods by large margins.

Table 5: Model robustness of LeNet on MNIST over each individual attack (the higher the better).

		Vanilla	PAT	TRADES	P_1	P_2	P_∞	AVG	MAX	ABS	MSD	MN	MBN	**GBN**
ℓ_1 attacks	PGD-ℓ_1	74.4%	69.9%	80.8%	94.3%	90.5%	83.2%	77.9%	77.3%	/	77.0%	80.4%	79.6%	**86.8%**
	BBA	6.5%	53.6%	10.2%	0.0%	41.9%	11.1%	31.2%	33.1%	/	42.6%	25.2%	64.5%	**79.6%**
ℓ_2 attacks	PGD-ℓ_2	24.7%	61.6%	90.1%	0.0%	54.6%	68.5%	73.5%	74.6%	/	69.5%	77.2%	78.5%	**97.8%**
	C&W-ℓ_2	2.0%	56.5%	42.5%	0.0%	49.3%	46.0%	68.1%	67.6%	/	72.4%	20.0%	67.3%	**98.1%**
	Gaussian Noise	99.2%	89.9%	98.5%	81.3%	99.1%	98.4%	98.7%	98.2%	98.0%	97.3%	97.6%	98.6%	**99.2%**
	BA	10.5%	64.7%	12.1%	0.0%	61.2%	15.1%	60.9%	59.2%	83.0%	78.7%	24.2%	90.2%	**97.7%**
ℓ_∞ attacks	PGD-ℓ_∞	64.2%	19.4%	96.1%	0.9%	86.5%	87.3%	72.6%	74.7%	/	51.6%	78.9%	80.3%	**96.3%**
	FGSM	48.7%	54.4%	96.9%	27.5%	90.7%	92.7%	87.5%	86.3%	34.0%	69.7%	**97.5%**	75.4%	89.5%
	MI-FGSM	36.2%	35.9%	**96.8%**	1.1%	88.2%	90.6%	82.1%	81.5%	17.0%	60.9%	90.7%	40.7%	85.2%
	SPSA	37.6%	7.0%	93.3%	0.6%	71.6%	81.6%	64.2%	60.5%	/	71.4%	77.8%	77.8%	**98.2%**
	NATTACK	40.4%	61.7%	91.7%	83.1%	85.3%	83.7%	79.5%	78.9%	/	85.2%	90.2%	89.3%	**97.7%**
	AutoAttack	0.0%	1.5%	**77.1%**	0.0%	0.0%	50.2%	46.1%	60.1%	/	39.0%	0.0%	20.0%	71.7%
All attacks	-	0.0%	1.2%	9.8%	0.0%	0.0%	10.4%	29.7%	31.2%	17.0%	38.9%	0.0%	19.2%	**68.9%**
Clean accuracy	-	**99.3%**	91.2%	99.1%	98.9%	99.1%	98.4%	98.7%	98.2%	99.0%	97.3%	98.1%	98.7%	98.7%

Table 6: Model robustness of ResNet-20 on CIFAR-10 over each individual attack (the higher the better).

		Vanilla	PAT	TRADES	P_1	P_2	P_∞	AVG	MAX	MSD	MN	MBN	**GBN**
ℓ_1 attacks	PGD-ℓ_1	31.7%	40.0%	25.7%	**70.3%**	39.8%	18.7%	47.4%	38.1%	49.9%	44.1%	47.7%	58.5%
	BBA	0.0%	34.5%	16.2%	22.4%	34.2%	18.0%	45.0%	33.4%	43.8%	40.0%	45.8%	**69.1%**
ℓ_2 attacks	PGD-ℓ_2	32.1%	51.3%	59.8%	19.8%	62.8%	60.4%	59.2%	58.0%	62.8%	49.4%	60.5%	**69.5%**
	C&W-ℓ_2	0.0%	48.4%	59.5%	4.8%	51.2%	50.4%	59.7%	56.0%	62.5%	30.1%	20.8%	**74.9%**
	Gaussian Noise	**81.2%**	62.7%	79.8%	68.9%	84.8%	81.6%	76.4%	62.9%	69.1%	60.2%	49.5%	79.1%
	BA	0.3%	51.6%	60.2%	1.2%	62.3%	58.4%	62.0%	57.8%	59.9%	39.1%	47.8%	**73.2%**
ℓ_∞ attacks	PGD-1000-ℓ_∞	0.0 %	35.4%	48.1%	18.4%	40.0%	44.7%	40.4%	37.6%	42.6%	32.8%	55.4%	**58.1%**
	FGSM	9.2%	43.1%	55.2%	30.4%	46.7%	54.3%	39.9%	42.7%	46.4%	40.1%	**62.4%**	56.0%
	MI-FGSM	0.0%	40.9%	53.9%	0.9%	30.2%	50.8%	37.8%	31.6%	45.3%	31.2%	60.1%	**72.9%**
	C&W-ℓ_∞	2.0%	35.9%	53.7%	0.1%	40.9%	51.3%	47.8%	44.3%	49.3%	42.1%	51.1%	**67.2%**
	SPSA	0.4%	35.6%	50.2%	0.6%	19.4%	45.8%	33.2%	30.4%	44.9%	38.4%	68.2%	**69.8%**
	NATTACK	1.9%	37.0%	50.1%	2.1%	19.7%	43.1%	34.3%	30.9%	45.1%	40.2%	45.4%	**64.2%**
	AutoAttack	0.0%	26.9%	45.1%	0.0%	20.2%	42.3%	29.3%	25.4%	38.1%	13.2%	40.8%	**50.9%**
All attacks	-	0.0%	25.1%	16.0%	0.0%	19.0%	17.8%	28.2%	24.9%	37.9%	13.0%	20.7%	**48.7%**
Clean accuracy	-	**89.7%**	62.8%	86.9%	84.3%	87.9%	84.2%	80.6%	77.0%	79.1%	82.3%	79.4%	80.7%

E.3 ATTACKS WITH DIFFERENT ITERATION STEPS

Here, we further provide the experimental results of PGD adversarial attacks using different iteration step numbers (*i.e*, 50, 100, 200, 1000) in terms of ℓ_∞ norm on CIFAR-10. For PGD attacks, we set the perturbation magnitude ϵ=0.03, step size α=ϵ/10, and different iteration steps k. As shown in Table 11, our GBN outperforms other methods on PGD attacks using different iteration steps.

E.4 COMPUTATIONAL COST

Here, we provide the time consumption of AVG, MAX, MSD, MN, MBN, and our GBN. All the experiments are conducted on a NVIDIA Tesla V100 GPU cluster. We compute the overall training time using ResNet-20 on CIFAR-10 for 40 epochs. We compute the inference time using ResNet-20 on CIFAR-10 for 10000 images. As seen in Table 12, our method achieves comparable results to other baselines, demonstrating its applicability in practice.

Table 7: Model robustness of ResNet-34 on Tiny-ImageNet over each individual attack (the higher the better).

		Vanilla	PAT	TRADES	P_1	P_2	P_∞	AVG	MAX	MSD	MN	MBN	**GBN**
ℓ_1 attacks	PGD-ℓ_1	10.0%	13.8%	27.9%	33.8%	28.0%	26.6%	28.1%	17.1%	9.0%	12.4%	45.2%	**55.7%**
	BBA	5.8%	13.5%	25.6%	32.1%	25.8%	24.8%	27.5%	26.2%	7.6%	8.8%	37.0%	**45.2%**
ℓ_2 attacks	PGD-ℓ_2	12.3%	14.2%	33.7%	31.4%	33.2%	33.6%	30.5%	24.4%	14.2%	18.1%	31.7%	**53.9%**
	C&W-ℓ_2	10.9%	13.4%	31.6%	30.7%	32.3%	32.0%	29.1%	25.2%	11.2%	19.2%	31.4%	**44.8%**
	Gaussian Noise	**54.2%**	20.8%	43.1%	52.7%	54.1%	44.4%	41.6%	36.3%	26.7%	40.2%	46.2%	42.8%
	BA	27.1%	17.1%	36.9%	41.1%	41.9%	38.0%	35.2%	24.9%	13.7%	30.2%	38.4%	**44.1%**
ℓ_∞ attacks	PGD-ℓ_∞	7.1%	2.4%	12.9%	0.7%	1.4%	10.2%	6.9%	4.7%	7.6%	19.8%	40.2%	**50.2%**
	FGSM	3.4%	5.1%	15.9%	6.2%	7.2%	14.5%	11.1%	6.8%	8.9%	22.4%	41.4%	**50.3%**
	MI-FGSM	1.4%	4.0%	14.8%	3.1%	4.5%	12.9%	9.4%	6.0%	8.0%	15.1%	36.3%	**53.7%**
	SPSA	0.3%	3.2%	14.9%	0.9%	2.3%	9.0%	7.1%	6.5%	10.0%	29.8%	39.2%	**52.0%**
	NATTACK	0.8%	3.5%	15.7%	1.6%	4.6%	10.3%	8.2%	7.5%	12.3%	29.9%	40.3%	**48.5%**
	AutoAttack	0.0%	2.3%	9.5%	0.4%	0.7%	5.2%	4.6%	2.9%	5.1%	6.8%	20.1%	**39.8%**
All attacks	-	0.0%	2.1%	9.2%	0.4%	0.7%	5.1%	4.3%	2.2%	5.0%	6.7%	19.0%	**38.5%**
Clean accuracy	-	**54.2%**	20.5%	43.1%	52.8%	54.3%	44.4%	41.6%	36.3%	28.8%	46.7%	45.7%	43.4%

Table 8: Model robustness of VGG-16 on CIFAR-10 over each individual attack (the higher the better).

		Vanilla	PAT	TRADES	P_1	P_2	P_∞	AVG	MAX	MSD	MN	MBN	**GBN**
ℓ_1 attacks	PGD-ℓ_1	32.8%	45.9%	27.9%	**72.2%**	42.1%	19.8%	48.8%	39.6%	51.2%	46.0%	49.5%	59.9%
	BBA	0.0%	40.1%	18.2%	24.8%	36.1%	18.7%	45.9%	35.5%	44.9%	41.1%	46.8%	**69.5%**
ℓ_2 attacks	PGD-ℓ_2	32.9%	63.4%	60.9%	21.7%	64.4%	62.8%	61.4%	60.0%	64.9%	50.4%	62.1%	**70.7%**
	C&W-ℓ_2	0.0%	60.9%	61.1%	6.8%	52.1%	52.5%	61.9%	58.0%	64.2%	31.4%	21.9%	**75.2%**
	Gaussian Noise	**82.0%**	80.5%	79.9%	70.1%	86.0%	82.8%	74.9%	62.9%	69.9%	61.8%	51.2%	79.9%
	BA	0.9%	66.0%	61.8%	2.3%	64.4%	60.2%	64.1%	59.7%	61.4%	40.2%	49.3%	**74.8%**
ℓ_∞ attacks	PGD-1000-ℓ_∞	0.0 %	40.6%	49.9%	20.2%	41.0%	45.9%	42.2%	39.8%	42.5%	33.1%	56.4%	**58.8%**
	FGSM	9.1%	49.9%	56.2%	32.1%	47.7%	55.8%	41.0%	44.6%	47.3%	41.8%	**63.7%**	57.4%
	MI-FGSM	0.3%	47.1%	54.2%	1.2%	31.8%	51.9%	39.3%	33.4%	47.0%	32.7%	60.9%	**73.8%**
	SPSA	0.6%	40.8%	52.2%	0.9%	20.4%	46.8%	34.9%	32.1%	46.2%	39.9%	69.8%	**70.8%**
	NATTACK	2.4%	41.5%	50.4%	2.8%	19.7%	47.1%	34.3%	30.9%	45.1%	40.2%	45.4%	**64.2%**
	AutoAttack	0.0%	35.0%	46.1%	0.0%	20.2%	43.3%	30.4%	27.4%	39.2%	14.3%	40.8%	**51.9%**
All attacks	-	0.0%	34.1%	17.9%	0.0%	19.5%	18.5%	30.2%	27.0%	38.9%	14.1%	21.5%	**50.6%**
Clean accuracy	-	**90.9%**	80.6%	87.1%	84.9%	88.1%	85.0%	82.1%	79.2%	78.4%	80.2%	84.3%	84.1%

E.5 ALTERNATIVE PREDICTION APPROACH OF THE GATED SUB-NETWORK

In the main body of our paper, we calculate the normalized output using the gated sub-network in a soft-gated way (denoted "soft") based on Eqn. (3). In this section, we also try taking the top-1 prediction of g (the hard label) to normalize the output as an alternative approach (denoted "hard"). As shown in Table 13, our soft-label version achieves slightly better results than the hard-label one.

E.6 DIFFERENT VARIANTS OF WHITE-BOX ATTACKS

Here, we examine the performance of GBN against white-box attacks specially designed for GBN. In this case, the adversary not only knows every detail of the GBN architecture, but also can manually select the BN branch to generate adversarial attacks. All experiments are conducted on CIFAR-10 using a ResNet-20.

We first generate adversarial attacks by fooling all the GBN layers in the model. Specifically, we generate adversarial attacks using PGD-ℓ_1, PGD-ℓ_2, and PGD-ℓ_∞ attacks in Section D.1 in the supplementary material by making the gated sub-network to wrong predictions of the input domain. We force the generated perturbations to fool the predictions of all the GBN layers in a ResNet-20 model at the same time. In particular, we first generate adversarial examples for different perturbation types

Table 9: Model robustness of WideResNet-28-10 on CIFAR-10 over each individual attack (the higher the better).

		Vanilla	PAT	TRADES	P_1	P_2	P_∞	AVG	MAX	MSD	MN	MBN	**GBN**
ℓ_1 attacks	PGD-ℓ_1	24.7%	31.8%	27.2%	**71.4%**	40.9%	19.5%	53.3%	40.0%	53.1%	46.0%	48.8%	62.1%
	BBA	0.0%	36.6%	16.8%	23.1%	35.6%	19.1%	49.0%	36.1%	45.2%	42.0%	47.6%	**71.8%**
ℓ_2 attacks	PGD-ℓ_2	27.1%	44.6%	60.8%	21.8%	63.7%	61.2%	65.3%	62.6%	63.1%	51.1%	61.9%	**70.9%**
	C&W-ℓ_2	0.0%	49.7%	60.5%	5.2%	53.0%	51.9%	62.7%	58.8%	64.1%	33.4%	24.8%	**76.4%**
	Gaussian Noise	**91.2%**	61.8%	80.2%	69.9%	85.9%	82.6%	78.8%	70.3%	77.8%	63.9%	54.2%	77.8%
	BA	0.5%	48.3%	60.7%	1.0%	61.4%	58.1%	67.0%	61.4%	65.4%	42.0%	50.1%	**75.6%**
ℓ_∞ attacks	PGD-1000-ℓ_∞	0.4 %	27.0%	50.2%	18.9%	40.1%	46.5%	42.1%	40.6%	47.5%	35.1%	56.9%	**60.4%**
	FGSM	8.7%	41.1%	58.2%	32.4%	47.4%	52.6%	39.9%	44.4%	52.2%	42.0%	**65.2%**	58.1%
	MI-FGSM	0.0%	37.5%	54.2%	0.6%	30.6%	48.3%	37.8%	36.0%	50.3%	35.1%	61.9%	**70.7%**
	SPSA	0.2%	26.5%	50.1%	0.7%	19.6%	42.3%	33.2%	34.8%	47.2%	39.9%	69.0%	**70.8%**
	NATTACK	1.9%	27.9%	50.6%	2.1%	19.7%	47.4%	34.3%	36.1%	46.1%	43.0%	47.9%	**66.0%**
	AutoAttack	0.0%	20.1%	47.2%	0.0%	20.0%	44.8%	31.3%	30.4%	40.2%	15.8%	38.8%	**52.4%**
All attacks	-	0.0%	18.6%	16.5%	0.0%	19.0%	18.8%	31.3%	29.9%	40.0%	15.6%	24.5%	**51.5%**
Clean accuracy	-	**93.2%**	61.6%	87.6%	83.4%	88.5%	85.1%	83.2%	79.1%	80.3%	83.6%	84.1%	83.9%

Table 10: Robustness evaluation by AutoAttack (the higher the better). We report the clean accuracy, the robust accuracy of the individual attacks as well as the combined one of AutoAttack (denoted AA) using LeNet, ResNet-20, and ResNet-34.

	Vanilla	PAT	TRADES	P_1	P_2	P_∞	AVG	MAX	MSD	MN	MBN	**GBN**
MNIST - ℓ_∞												
APGD$_{CE}$	34.0%	2.3%	90.4%	0.0%	67.3%	74.1%	48.4%	60.2%	39.3%	74.7%	20.2%	80.9%
APGD$_{DLR}$	0.0%	1.7%	82.1%	0.0%	0.0%	51.8%	48.2%	62.7%	43.9%	0.0%	52.2%	88.4%
FAB^T	0.2%	2.2%	85.0%	0.0%	0.1%	63.2%	52.1%	62.1%	42.3%	0.0%	80.7%	96.1%
Square	0.0%	3.7%	81.8%	0.0%	0.2%	53.9%	50.2%	61.1%	41.5%	0.0%	59.7%	79.3%
AA	0.0%	1.5%	77.1%	0.0%	0.0%	50.2%	46.1%	60.1%	39.0%	0.0%	20.0%	71.7%
CIFAR-10 - ℓ_∞												
APGD$_{CE}$	0.0%	34.1%	49.1%	0.0%	20.6%	44.9%	30.8%	29.9%	44.3%	20.8%	53.5%	60.8%
APGD$_{DLR}$	0.0%	27.2%	50.2%	0.0%	21.8%	43.2%	30.1%	25.7%	39.1%	29.9%	62.4%	58.9%
FAB^T	0.0%	27.9%	49.8%	0.8%	20.6%	43.2%	29.6%	26.3%	39.3%	30.1%	40.9%	70.2%
Square	0.1%	41.4%	47.4%	4.6%	28.4%	43.0%	40.5%	31.6%	46.1%	17.8%	44.5%	60.3%
AA	0.0%	26.9%	45.1%	0.0%	20.2%	42.3%	29.3%	25.4%	38.1%	13.2%	40.8%	50.9%
Tiny-ImageNet - ℓ_∞												
APGD$_{CE}$	0.0%	2.9%	13.0%	0.7%	0.8%	7.8%	5.7%	3.7%	6.3%	8.7%	39.3%	44.3%
APGD$_{DLR}$	0.0%	2.4%	10.1%	0.5%	0.9%	5.3%	5.1%	3.1%	9.2%	10.4%	41.0%	43.8%
FAB^T	0.2%	2.5%	9.8%	1.0%	0.9%	6.2%	5.2%	3.0%	8.3%	9.1%	37.3%	50.0%
Square	0.7%	3.3%	15.8%	4.5%	11.7%	12.8%	10.1%	4.0%	5.9%	7.4%	29.3%	39.9%
AA	0.0%	2.3%	9.5%	0.4%	0.7%	5.2%	4.6%	2.9%	5.1%	6.8%	20.1%	39.8%

Table 11: Model performance (classification accuracy %) of ResNet-20 on CIFAR-10 on different PGD-k attacks (k denotes the iteration steps).

	Vanilla	TRADES	AVG	MAX	MSD	MN	MBN	**GBN(ours)**
PGD-50	2.4	49.8	42.1	39.5	44.3	34.6	58.6	**60.0**
PGD-100	0.0	49.5	41.7	39.2	44.1	34.1	58.2	**59.6**
PGD-200	0.0	49.3	41.4	38.7	43.5	33.6	57.7	**59.0**
PGD-1000	0.0	48.1	40.4	37.6	42.6	32.8	55.4	**58.1**

using a ResNet-20 model; based on these examples for each domain, we then perturb them to fool the gated sub-networks in all GBN layers. As shown in Section 4.2 in the main body, our GBN still achieves high robustness against such adaptive attack. To further investigate this phenomena,

Table 12: Runtime analysis of different methods on CIFAR-10 (the lower the better).

	AVG	MAX	MSD	MN	MBN	**GBN(ours)**
Training speed (seconds/ epoch)	1837.9	1113.2	689.8	19329.1	2737.9	2802.6
Training duration (hours)	20.4	12.4	7.7	214.8	30.4	31.1
Inference speed (seconds/ 10000 images)	0.60	0.58	0.61	4.8	1.9	2.0

Table 13: Model robustness of ResNet-20 on CIFAR-10 using different prediction approaches of the gated sub-network (the higher the better). The hyper-parameters for PGD-ℓ_1, PGD-ℓ_2, and PGD-ℓ_∞ attacks are shown in Section D.1 in the supplementary material.

	vanilla	hard	soft
PGD-ℓ_1	31.4%	58.2%	58.5%
PGD-ℓ_2	32.1%	69.0%	69.5%
PGD-ℓ_∞	3.2%	59.9%	60.1%
Clean accuracy	89.7%	80.4%	80.7%

we also provide the prediction accuracy of the gated sub-network in GBN at different layers given adversarial examples above. We can find that it is hard to fool all GBN layers within a model. In other words, though some GBN layers are fooled to misclassify the domains of the samples, some other GBN layers could still keep comparatively high prediction accuracy.

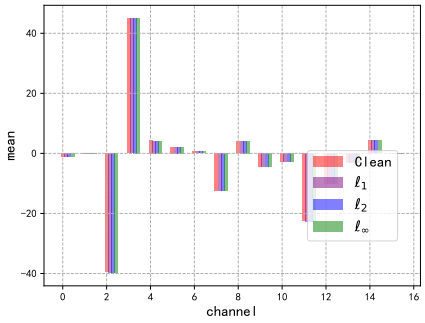
We then generate adversarial examples by manually selecting the BN branches. Specifically, we generate ℓ_1, ℓ_2, and ℓ_∞ adversarial examples through BN^0, BN^1, BN^2, and BN^3 in GBN, respectively. The hyper-parameters for PGD-ℓ_1, PGD-ℓ_2, and PGD-ℓ_∞ attacks in Section D.1 in the Appendix. As presented in Section 4.2 in the main body, the model achieves good robustness on different datasets in this more rigorous white-box attack scenario.

E.7 Using gated sub-network only in the first GBN

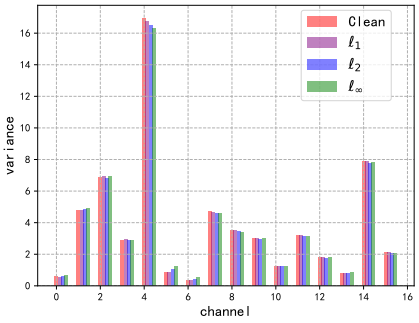
In this paper, we add additional experiments by including gated sub-network in the first GBN layer, and send soft-labels to following layers. In other words, the following GBN layers do not contain gated sub-networks and depend on the prediction results of the first GBN layer to classify the input domains. The accuracy of a ResNet-20 on CIFAR-10 against PGD-ℓ_1, PGD-$\ell - 2$, PGD-$\ell - \infty$ adversarial examples, and clean examples are 57.3%, 67.1%, 57.7%, and 83.5%, which is slightly weaker than our implementation. Thus, we use our original implementations in our main experiments (adding gated sub-networks in each GBN layers). We conjecture that given an input, the weights for each domain in each layer are not the same for the best defense performance. We put it as future work.

E.8 More details of the running statistics of different models

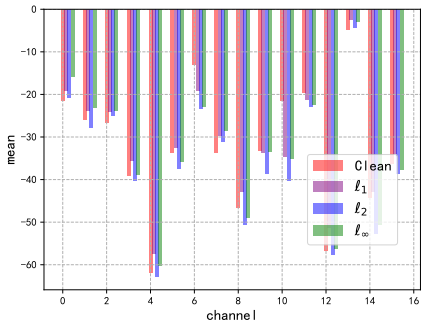
In this part, we provide more running statistics of different deep models to further verify our multi-domain hypothesis. In particular, we train a VGG-16 and WideResNet-28-10 model with the multiple BN branch structure and the running statistics of each BN branch at different layers are shown in Figure 8 and Figure 9.



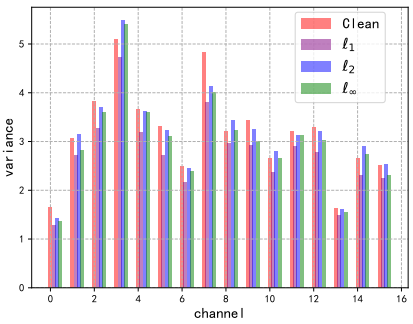
(a) running mean of layer $conv1\_1$



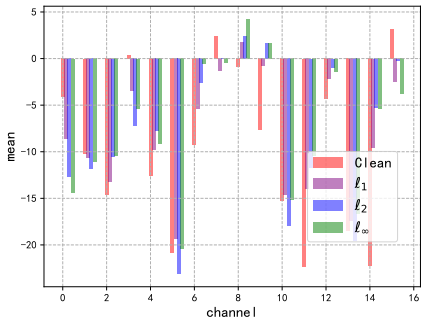
(b) running variance of layer $conv1\_1$



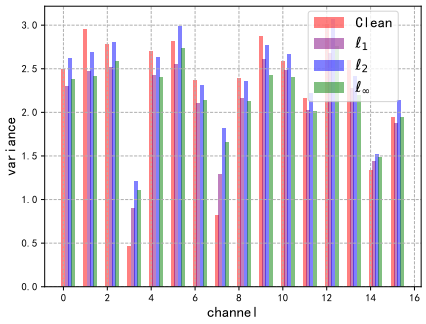
(c) running mean of layer $conv2\_2$



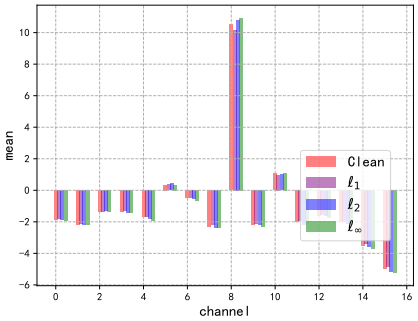
(d) running variance of layer $conv2\_2$



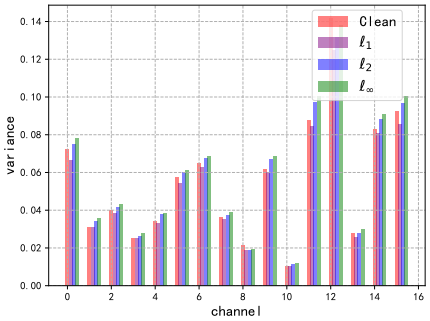
(e) running mean of layer $conv3\_1$



(f) running variance of layer $conv3\_1$

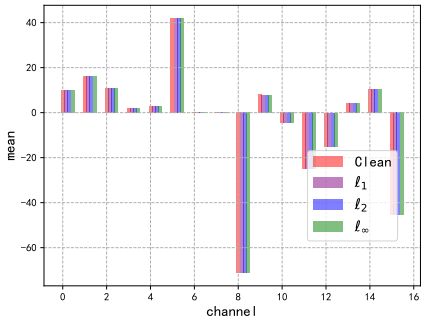


(g) running mean of layer $conv4\_1$

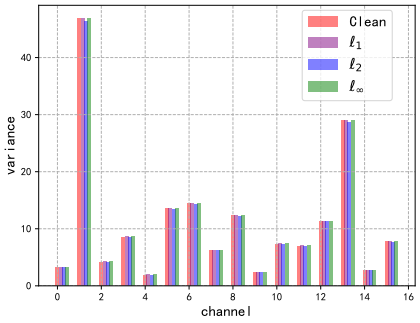


(h) running variance of layer $conv4\_1$

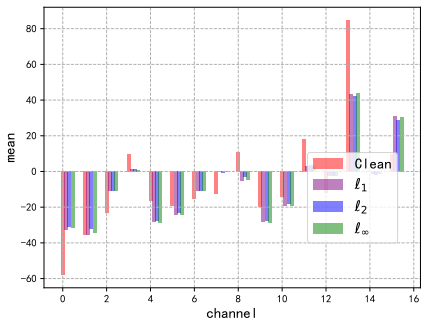
Figure 8: Running statistics (running mean and running variance) of each BN in the multiple BN branches at different layers of a VGG-16 model on CIFAR-10.



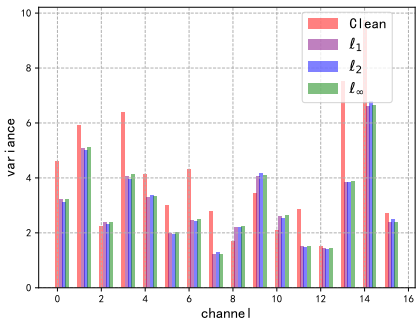
(a) running mean of block1.layer1.bn1



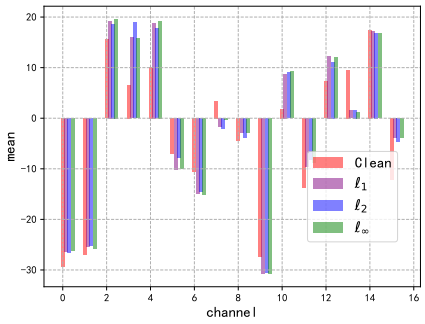
(b) running variance of layer block1.layer1.bn1



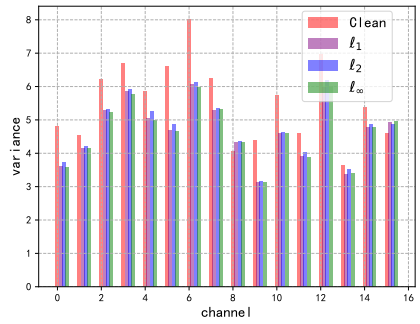
(c) running mean of layer block1.layer1.bn2



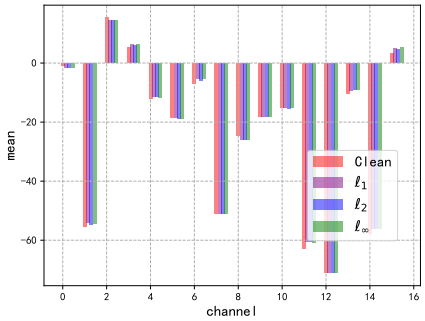
(d) running variance of layer block1.layer1.bn2



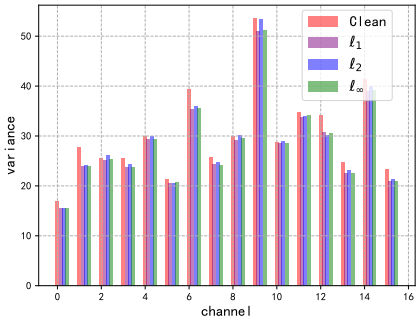
(e) running mean of layer block2.layer1.bn2



(f) running variance of layer block2.layer1.bn2



(g) running mean of layer block3.layer1.bn1



(h) running variance of layer block3.layer1.bn1

Figure 9: Running statistics (running mean and running variance) of each BN in the multiple BN branches at different layers of a WideResNet-28-10 model on CIFAR-10.