Defense Through Diverse Directions

Christopher M. Bender [1] **Yang Li** [1] **Yifeng Shi** [1] **Michael K. Reiter** [1] **Junier B. Oliva** [1]

Abstract

In this work we develop a novel Bayesian neural network methodology to achieve strong adversarial robustness without the need for online adversarial training. Unlike previous efforts in this direction, we do not rely solely on the stochasticity of network weights by minimizing the divergence between the learned parameter distribution and a prior. Instead, we additionally require that the model maintain some expected uncertainty with respect to all input covariates. We demonstrate that by encouraging the network to distribute evenly across inputs, the network becomes less susceptible to localized, brittle features which imparts a natural robustness to targeted perturbations. We show empirical robustness on several benchmark datasets.

1. Introduction

Neural networks currently achieve greater-than-human performance in a variety of tasks such as object recognition (He et al., 2016), language understanding (Vaswani et al., 2017; Devlin et al., 2018), and game playing (Silver et al., 2016; 2017). Despite their incredible successes, these same networks are easily fooled by seemingly-trivial perturbations that humans overcome with minimal difficulty (Goodfellow et al., 2014). This weakness poses considerable concern in a world that is increasingly reliant on machines from the perspectives of both security (e.g., face recognition) and safety (driverless cars). Despite considerable effort to overcome these difficulties, the problem persists (Elsayed et al., 2018; Athalye et al., 2018).

The most successful methods for improving adversarial robustness utilize online adversarial training (Madry et al., 2017). Online adversarial training requires an iterative training procedure where adversarial examples are produced based on a particular attack scheme with respect to the current network state and the model is updated to resist the particular attack. Unfortunately, this method is computationally expensive, as attacks must be generated and the model updated multiple times. Zhang et al. and Sharma & Chen have demonstrated that while this process makes the model robust to the particular type of attack used in the training process, the model can be susceptible to attacks from alternate schemes.

To scale adversarial training, researchers have tried to transfer adversarial examples from another model. Tramèr et al. has found that this offline adversarial training scheme can perform equitably well in practice but can be much more efficient since it decouples the adversarial examples generation from the training process.

Alternative lines of research introduce randomness into the model. Early attempts include adding Gaussian noise to the inputs (Zantedeschi et al., 2017) and randomly pruning the network (Wang et al., 2018; Dhillon et al., 2018). Liu et al. propose to add Gaussian noise to all the intermediate activations. Wang et al. train multiple copies for each block of the network and randomly select one during inference. Along these lines, we utilize Bayesian neural networks (BNNs) as a principled way to inject noise into the model.

Recent work (Liu et al., 2018a;b) has incorporated stochasticity by utilizing BNNs. Similar to our method, they demonstrate that randomness of BNNs alone is not sufficient for robust classification. They turn to an online adversarial training scheme to implicitly boost the randomness.

We instead choose to explicitly penalize the model so that it evenly distributes the sensitivity of the output w.r.t. the input elements. We estimate the output sensitivity for each input through a first order Taylor approximation and exploit the inherent ensembling of BNNs to evaluate statistics for each input example. These statistics become the basis for a defense-promoting regularization scheme by diversifying the directions of the output.

Our contributions are as follows:

- We propose several general penalties that can be added to the loss function of any Bayesian neural network to diversify the output variation with respect to the input covariates.

[1]The University of North Carolina, North Carolina, USA. Correspondence to: Christopher M. Bender <bender@cs.unc.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

- We demonstrate that increased output diversity leads to natural adversarial robustness, without requiring online adversarial training.
- We show that models trained with our diversity inducing penalties generalize to a variety of attack schemes.

This work begins with a review of Bayesian neural networks and the adversarial problem. We then discuss our motivations and methods for improving model robustness. Finally, we illustrate our methods on several datasets and discuss the implications. Particularly, we show that our method improves robustness over state-of-the-art BNN methods (Liu et al., 2018b), all without online adversarial training.

2. Background & Related Work

In this section we provide an overview of background material and related work.

2.1. Bayesian Neural Networks

In the context of supervised deep learning, a conventional neural network seeks to perform some variant of a classical functional estimation task, to learn a *point estimate* of the optimal function in the chosen functional space that maps each input from the input space to its corresponding output in the output space. However, such an estimate does not consider, and thus cannot effectively adapt to, the inherent uncertainty throughout the training procedure (e.g., data collection, random initialization of network weights). Such deficiency leads to problems including over-fitting and overly confident predictions.

To remedy this deficiency, a *Bayesian neural network* (BNN), introduced in the same vein as continuous stochastic processes such as the Gaussian process, seeks to directly model the distribution, whose density we denote as p, over random functions

$$f \sim p(f), \quad f : \mathbb{X} \mapsto \mathbb{O}$$

where $\mathbb{X}$ and $\mathbb{O}$ denote the input and the output spaces, respectively. However, directly learning such a distribution can be arduous as functional spaces are usually infinite-dimensional. Utilizing the fact that neural networks can be regarded as universal approximators for functions (Hornik et al., 1989), a distribution over random functions can be thought of as a choice over of neural networks. We materialize such connection through learning a distribution over the network weights. More specifically, in assuming that the network weights are random and distributed according to a prior distribution $P(\mathbf{w})$, a BNN seeks to learn the posterior distribution of the network weights, $\mathbf{w}$, given the available data, i.e. $P(\mathbf{w}|D)$.

While a clever idea, learning $P(\mathbf{w}|D)$ is prohibitively expensive even for moderately sized networks because of the high-dimensional integral that needs to be evaluated. Borrowing ideas from variational inference and the recent success of unsupervised methods like the *variational auto-encoder* (Kingma & Welling, 2014), Blundell et al. propose to learn a *variational posterior distribution*, $q(\mathbf{w}|\theta)$, to approximate the true posterior by optimizing the following objective

$$\max_{\theta} \ \mathbb{E}_{q(\mathbf{w}|\theta)} \left(\log P((D|\mathbf{w})) - \mathrm{KL} \left(q(\mathbf{w}|\theta) || p(\mathbf{w}) \right) \right) \quad (1)$$

which is the *evidence lower bound* (ELBO) for the data likelihood. The expectation term ensures the learnt variational posterior distribution is informed by the data, and the KL divergence acts as a regularizer over the weights. Although technically any choice of the variational posterior and prior distributions pair is possible, the convention, which we adopt in this work, is to choose both to be independent Gaussians where we learn the mean and variances of the variational posterior distribution. One benefit of deploying a BNN, which we exploit in our proposed framework, is that for one input one can *draw* multiple functions f, which in practice is actualized by drawing a set of different weights from the posterior distribution $P(\mathbf{w}|D)$, to form an ensemble of inferences for various purposes, such as assessing the uncertainties in predictions, gradient evaluations, etc. In this work, we exploit the network's variation to control how much sensitivity we expect from each input element.

2.2. Adversarial Attack

Adversarial examples are constructed by making small perturbations to the input that induce a dramatic change in the output. Attacks are typically broken down into two categories: white and black box attacks. The exact definition of both methods vary, but we will use the following definition in this work. In the white box setting, the attack has access to the training data set, the fully specified underlying model, and the loss functions. Attacks are found by performing gradient ascent with respect to the input. In the black box setting, the attacker has access to the training data and the loss functions but does not have access to the underlying model parameters. A black box attack can then be constructed by using a stand-in network trained on the same data with the same loss. Previous works have shown that these examples are still effective against a variety of other models (Liu et al., 2016; Tramèr et al., 2017b).

The attacker's goal is:

$$\max_{\|\varepsilon\|_p < \varepsilon_{\max}} \mathbb{E} \left[\mathcal{L} \left(f(\mathbf{x} + \varepsilon; \theta), \mathbf{y} \right) \right] \quad (2)$$

where ε is the attack perturbation, p is the norm (typically taken to be ∞), $\varepsilon_{\max}$ is the attack budget (the maximum perturbation), $\mathcal{L}$ is the loss, f is the network, $\mathbf{x}$ is the input, θ is the network parameters, and $\mathbf{y}$ is the truth.

Typically the attack methods generate the adversarial examples by leveraging the gradient of the loss function with respect to the inputs. The Fast Gradient Sign Method (FGSM) (Goodfellow et al., 2014), for instance, takes one step along the gradient direction to perturb an input by the amount ε:

$$x_{adv} = x + \varepsilon \cdot \text{sign}(\nabla_x \mathcal{L}(f(\mathbf{x}; \theta), \mathbf{y})). \tag{3}$$

Projected Gradient Descent (PGD) (Madry et al., 2017) generalizes FGSM by taking multiple gradient updates:

$$x_k = x_{k-1} + \alpha \cdot \text{sign}(\nabla_x \mathcal{L}(f(\mathbf{x}; \theta), \mathbf{y})), \tag{4}$$

where α is the step size. After each update, PGD projects the perturbed inputs back into the ε-ball of the normal inputs.

There are also other types of attacks, such as C&W attack (Carlini & Wagner, 2017), Jacobian Saliency Map Attack (JSMA) (Papernot et al., 2016) and DeepFool (Moosavi-Dezfooli et al., 2016). Among all the attack methods, PGD is regarded as the strongest attack in terms of the L_∞ norm.

2.3. Adversarial Defense

The goal of adversarial defense is to render all (bounded) perturbations ineffective against a model. It is necessary to bound the perturbation or the attack could simply replace an input with an example from a different distribution or class. A simple intuition to achieve this would be to require that the model be Lipschitz smooth such that

$$\|f(\mathbf{x} + \varepsilon; \theta) - f(\mathbf{x}; \theta)\| < C \|\varepsilon\|. \tag{5}$$

Unfortunately, estimating the Lipschitz coefficient C for an arbitrary network can be extremely difficult, making optimizing over it nontrivial. Cissé et al. has attempted to control the Lipschitz coefficient of each layer in the network and, therefore, the network as a whole.

Recent works have introduced a number of defense methods, such as distillation (Papernot et al., 2016), label smoothing (Hazan et al., 2016), input denoising (Song et al., 2017), feature denoising (Xie et al., 2019), gradient regularization (Ross & Doshi-Velez, 2018), and preprocessing based approaches (Das et al., 2017; Guo et al., 2017; Buckman et al., 2018). Most of these defenses have unfortunately been defeated by subsequent attacks.

The most popular adversarial defense technique incorporates adversarial examples in the training process (Madry et al., 2017). Online adversarial training requires generating new examples throughout the training process to map out the local region around known examples and require that the region map to the expected output. Unfortunately, while this approach does produce a robust defense, it is computationally expensive.

Pang et al. utilize a diversity promotion scheme across several, independently-trained, non-Bayesian networks. They promote diversity by encouraging the distribution of the probabilities across all classes, excluding the true class, should be different for each independent model. Our method could be considered a generalization that uses a diversity promotion penalty that is not unique to the classification problem and that uses a stochastic ensembling scheme instead of a deterministic scheme. The stochastic scheme allows access to a potentially unlimited number of models instead of the fixed number chosen at the outset of the training process. Several other methods utilize different forms of ensembling to improve robustness, e.g., (Sharif et al., 2019).

Most similar to our work, ADV-BNN (Liu et al., 2018b) attempts to use BNN to combat adversarial attack. Their proposed method is dependent on online adversarial training and aims to incorporate it into the standard ELBO objective, Eq. (1), as a min-max problem. In contrast, our proposed methodology does not require online adversarial training, and achieves better performance on CIFAR-10 compared to ADV-BNN.

2.3.1. Obfuscated Gradients

Athalye et al. warn of a failure mode in defense methods that they term "obfuscated gradients," where seemingly high adversarial accuracy is only superficial. Networks that achieve apparent improvements in white box attacks through obfuscated gradients do so by making it difficult for an attacker to find ε. However, successful ε still exist, which indicates that the network has not increased adversarial accuracy despite its improved adversarial test accuracy. Distinguishing between whether a defense mechanism has increased adversarial accuracy or merely increased attack difficulty is nontrivial. Typical methods rely on comparing white-box and black-box attacks. A strong indication that a defense is obfuscating gradients is when black-box attacks are successful while white-box attacks fail (low black-box accuracy and high white-box accuracy). A defense with high-white box accuracy and fair black-box accuracy is indicative of a mixture of obfuscated gradients and substantive adversarial accuracy improvement.

3. Motivation

Our primary motivation comes from the observation that the fewer inputs an attack needs to perturb, the less robust a model is. Therefore, we wish to diversify the importance of each input to the output. Alternatively, we wish to reduce the sensitivity of the output to variations in each input, possibly weighted by some foreknowledge of input uncertainty.

Since adversarial examples are best known for image recognition due to the dramatic difference in human robustness

versus machine robustness, we attempt to provide some intuition in this setting. For the general image setting, the object of interest may exist anywhere in image, and there is no way to know ahead of time which pixels are more reliable. Therefore, we assume that, on average, the sensitivity due to any single pixel should be roughly equal. So, we estimate the sensitivity per pixel, normalize across pixels, and penalize any divergence from our expectation.

We can estimate the sensitivity of the mth output, δy_m, with respect to the expected sensitivity of the dth input, δx_d, through a truncated Taylor series

$$\delta y_{m,d}(\mathbf{x}) \equiv y_m(\mathbf{x} - \delta \mathbf{x}_d) - y_m(\mathbf{x}) \tag{6}$$

$$\approx \delta \mathbf{x}_d^T \frac{\partial y_m(\mathbf{x})}{\partial \mathbf{x}} = \delta x_d \frac{\partial y_m(\mathbf{x})}{\partial x_d} \tag{7}$$

and collecting across inputs

$$\delta \mathbf{y}_m(\mathbf{x}) \approx \delta \mathbf{x} \odot \nabla y_m(\mathbf{x}) \tag{8}$$

where $\delta \mathbf{y}_m$ is a length D vector corresponding to the sensitivity of the mth output with respect to each input. For simplicity, we assume there is only one output and drop the dependence on m.

There are several ways to normalize the result across inputs. We choose to normalize by the L_2 norm. For datasets where each input is equitably important/reliable, $\delta x_d = \delta x$ and the normalized result becomes

$$\mathbf{u}_y(\mathbf{x}) = \delta \mathbf{y} / ||\delta \mathbf{y}||_2 = \nabla y / \left\|\nabla y\right\|_2 \tag{9}$$

where $\mathbf{u}_y(\mathbf{x})$ is the direction of the gradient of the the output with respect to the input, $\mathbf{x}$.

This result means that we expect the direction of the output gradient to be uniformly distributed over the unit hypersphere. In terms of the loss surface, all directions become equally likely.

$$\mathbf{u}_y \sim U_{\text{sph}}(\mathbf{u}) \tag{10}$$

In the event where the input uncertainty varies, the distribution would be uniformly distributed over a hyper-ellipsoid.

4. Method

Typically, neural networks are supervised to map an input to an output. We use the approximation from Sec. 3 to further supervise the uncertainty of the output. However, all networks have some inherent uncertainty (e.g., from random initializations) that changes the expected sensitivity. We execute K draws of the BNN and include additional penalties that attempt to maintain the expected distribution of the output sensitivity. We experiment with a variety of penalties based on this premise. For the sake of brevity, we drop the dependence of $\mathbf{u}_y$ on $\mathbf{x}$.

4.1. Entropy and Variances

As motivated previously, in order to increase the network's robustness against adversarially perturbed input, we encourage the network to maintain, and hence to evenly distribute, some expected sensitivity with respect to all input covariates. We materialize this idea by encouraging the normalized gradient in Eq. (9) to be uniformly distributed over the unit hypersphere, which in turn is equivalent to maximizing the entropy of $\mathbf{u}_y$, denoted as $H(\mathbf{u}_y)$, because $\mathbf{u}_y$ is bounded within the unit hypersphere. However, as a function of the network weights $\mathbf{w}$, the density of $\mathbf{u}_y$ is intractable even for moderate-sized network, making maximizing $H(\mathbf{u}_y)$ directly prohibitively expensive and impractical in practice.

In a simplified setting where the elements are independent, maximizing the sum of the variances of each element is equivalent to maximizing the entropy of the random vector.

Proposition 1. *Given a random vector* $\boldsymbol{X} = (X_1, X_2, \cdots, X_D)^T$ *where its elements are independent and* $X_i \in [a_i, b_i]$ *for all* i*, there exists a monotonically increasing relationship between the entropy of* $\boldsymbol{X}$*,* $H(\boldsymbol{X})$*, and the sum of the variances of each element of* $\boldsymbol{X}$*,* $\sum_i \mathbb{V}ar(X_i)$*.*

Proposition 1 indicates that maximizing the entropy of $\mathbf{X}$ is equivalent to maximizing $\sum_i \text{Var}(X_i)$. See Appendix A for the proof. While $\mathbf{u}_y$ is not independent in our case, we use Prop. 1 as an analogy and adopt the sum of the variances of the elements of $\mathbf{u}_y$ as a surrogate for $H(\mathbf{u}_y)$.

4.2. Direct Loss

A simple method might be to maximize the sum over inputs of the variance of $\mathbf{u}_y$ across the K draws

$$\sum_{d=1}^{D} \mathbb{V}\text{ar}\left[u_{y,d}\right]. \tag{11}$$

However, since $\mathbf{u}_y$ is a unit vector, this loss degenerates and only serves to minimize the average value of the output sensitivity

$$\sum_{d=1}^{D} \mathbb{V}\text{ar}\left[u_{y,d}\right] = \mathbb{E}\left[\sum_{d=1}^{D} u_{y,d}^2\right] - \sum_{d=1}^{D} \mathbb{E}\left[u_{y,d}\right]^2 \tag{12}$$

$$= 1 - \sum_{d=1}^{D} \mathbb{E}\left[u_{y,d}\right]^2. \tag{13}$$

Since we wish to maximize the variances w.r.t. $\mathbf{u}_y(\mathbf{x})$ we consider the mean penalty $\Omega_M(x)$:

$$\Omega_M(\mathbf{x}) = \sum_{d=1}^{D} \mathbb{E}\left[u_{y,d}\right]^2. \tag{14}$$

4.3. MinVar

While Equation (14) increases the total variance across inputs, it does not necessarily encourage diversity. We consider several additional penalties to include with Eq. (14) that do increase diversity.

A simple penalty to increase the minimum variance over dimensions is

$$\Omega_{V,1}(\mathbf{x}) = -\min_d \mathbb{V}\text{ar}\left[u_{y,d}\right]. \tag{15}$$

Equation (15) exploits the fact that the sum (over dimensions) of the variance is fixed. Therefore, increasing the minimum necessarily decreases the other values. In theory, as the network trains, the minimum element changes and eventually all the elements converge to the same variance.

Unfortunately, since the loss only supervises one pixel at a time, the minimum shifts across a few elements and never influences the pixels with the largest variance. We consider two simple methods to correct this difficulty. One is to supervise all the pixels simultaneously by replacing the $\min$ operation with a soft-min weighted sum so that Eq. (15) becomes

$$\Omega_{V,2}(\mathbf{x}) = -\sum_{d=1}^{D} \text{softmin}(\alpha u_{y,d}) \cdot \mathbb{V}\text{ar}\left[u_{y,d}\right] \tag{16}$$

where α corresponds to the temperature. However, since the sum of the variances is fixed and this loss attempts to increase the variance of all pixels simultaneously, we find that it can result in a counterproductive competition across pixels.

A more direct method to supervise the variance is to minimize the distance between the observed variance and the expected variance of $1/N$. Using the Euclidean distance, Eq. (15) becomes

$$\Omega_{V,3}(\mathbf{x}) = \sum_{d=1}^{D} \left(\mathbb{V}\text{ar}\left[u_{y,d}\right] - 1/D\right)^2. \tag{17}$$

We find that this final representation yields the best results amongst the variance encouraging losses and choose it as the variance penalty $\Omega_V(\mathbf{x})$.

4.4. Non-Sparse Promoting Losses

Directly encouraging the model to match a specific distribution's second order moment may be too strict a requirement. As an alternative to matching the second moment of the uniform hypersphere, we consider penalizing *small* L_1 norms of $\mathbf{u}_y$. By requiring that the L_1 norm be large, we bias the network away from over reliance on a few features and encourage greater diversity across the input covariates. Unlike in the variance-based losses, the network does not have to match each input direction to the same value. This allows for increased flexibility while still maintaining the same intuitive effect on the diversity of dependence. The added penalty becomes

$$\Omega_S(x) = -\mathbb{E}\left[\left\|\mathbf{u}_y\right\|_1\right]. \tag{18}$$

4.5. Batch Loss

We summarize the full possible loss of our model with respect to posterior parameters, $\mathcal{L}(\theta)$, with all the above penalties, a supervised loss l and a batch of data $\{(x_n, y_n)\}_{n=1}^N$:

$$\begin{aligned}\mathcal{L}(\theta) = \frac{1}{N}\sum_{n=1}^{N}\Big[&l(x_n, y_n) + \lambda_M \Omega_M(x_n) \\ &+\lambda_V \Omega_V(x_n) + \lambda_S \Omega_S(x_n)\Big].\end{aligned} \tag{19}$$

We vary Eq. (19) below, by setting various penalty weights λ to zero.

4.6. Further Benefits of Adversarial Training

Finally, we test the benefits of offline adversarial training (Tramèr et al., 2017a) in addition to the diversity inducing penalties. Adversarial examples are pre-computed from a trained, non-Bayesian neural network of a matching architecture and then added statically to the training set. Thus, this form of defense is more efficient than online adversarial training, which requires on-the-fly computation of adversarial examples.

5. Experiments

In this section we explore the effect of inducing diversity on a variety of open-source datasets.

5.1. Penalty Shorthand

We present our results by adding different combinations of the diversity-encouraging penalties in Sec. 4. To declutter the results, we use the following shorthand to refer to the different penalties we apply to the model. The unit gradient mean penalty, Eq. (14), is given as "M;" the variance penalty, Eq. (17), by "V;" the non-sparse penalty, Eq. (18), by "S;" and any offline training by "Off." We additionally denote when a network is Bayesian by prepending the network name with a "B." For example, the case where we use a Bayesian VGG16 with the mean and variance penalty is shorthanded as "BVGG16-M-V."

As mentioned previously, all adversarial examples appended to the training set for offline training were constructed using conventional networks with matching architectures. Aside from the models trained with offline adversaries, we do not include any form of data augmentation.

5.2. Practical Considerations

In this section we discuss several practical steps taken to implement various networks and diversity-promoting penalties. We utilize TensorFlow (Abadi et al., 2015) and Tensorflow Probability (Dillon et al., 2017) to implement the general and probabilistic components of our models, respectively. During training, the classification loss is assessed per network draw and then averaged. During inference and attack, ensembling is performed by averaging the logits over draws.

5.2.1. Drawing from the Bayesian Network

Since we optimize over statistics of the probabilistic network, we require multiple network samples for the same input data. A simple method to exploit extant parallelisms in most neural network frameworks would be to duplicate each element in the batch K (the number of draws) times. Unfortunately, Bayesian networks implement the Bayesian layers using either FlipOut (Wen et al., 2018) or the reparameterization trick (Kingma & Welling, 2014) to efficiently draw parameters that are *shared* across each element in the batch. While this shortcut is sufficient to decorrelate training gradients, it is not sufficient to obtain the level of independence required to inform our methods. As a result, we resort to executing the network K times for *each batch*.

This becomes prohibitively expensive for networks of sufficient depth. To offset some of this cost, we break our networks into two parts. In the first part, the layers are constructed in the conventional fashion without Bayesian components. We will refer to this component of our models as the deterministic network. After the deterministic network, we construct a Bayesian network. The deterministic component is executed once per batch and the Bayesian component, K times. The gradient that informs diversity promotion is still taken with respect to the input of the full network. This means that while the deterministic component does not directly add variation it is still trained to encourage overall network diversity. The specifics of where the transition between determinism and Bayesian can be found in each experiment's section. In general, we found that it was sufficient to set the last quarter of the network as Bayesian and use $K = 10$ draws.

5.2.2. Attack Schemes

We test our defense against two types of common attacks: the Fast-Gradient Sign Method (FGSM) and Projected Gradient Descent (PGD). We deploy both L_∞ and L_2 attacks. Black box attacks are performed using examples from external sources when available. All attacks are performed using the Adversarial Robustness Toolbox (Nicolae et al., 2018) and use the same number of network draws as are used in training.

Table 1. Adversarial accuracy (%) on MNIST with various combinations of diversity promotion against L_∞ attacks.

Method	FGSM	PGD	Black-Box
CNN	36.8	2.6	57.8
BNN	55.2	0.9	56.4
BNN-Off	40.0	2.1	**93.6**
M	93.0	55.2	60.1
M-V	94.0	70.0	61.8
M-S	96.3	94.0	54.3
M-V-S	97.2	95.8	59.8
M-S-Off	**97.6**	**95.8**	89.9
M-V-S-Off	97.4	94.5	90.6

5.3. MNIST

We test our diversity induced models on MNIST (LeCun & Cortes, 2010). For these tests we use a simple CNN with three convolutional layers followed by two fully connected layers. Since this network is fairly shallow, we compose the deterministic part of the network using the first two convolutional layers and use Bayesian layers for the final convolutional and both fully connected layers. When the corresponding penalty is used, the loss hyperparameters are: $\lambda_M = 20$, $\lambda_V = 40$, and $\lambda_S = 40$.

Table 1 shows the accuracy of the model when trained using different combinations of diversity inducing penalties and adversarial training against L_∞ attacks with a maximum attack budget of 0.3. All models obtain better than 99% standard accuracy. Black box attacks were obtained using examples from an open repository.[1]

When only the mean penalty (Eq. (14)) is imposed, the model shows modest improvements in both forms of attack; indicating that the defense has succeeded in improving the robustness of the model against attack. While this defense has not completely succeeded in overcoming attacks, it is a positive step. As we include additional penalties, we observe that the white-box attack improves, most notably with the inclusion of the non-sparse penalty (Sec. 4.4). However, these models generally show a slight *decrease* in black-box performance. We infer that these penalties tend to cause the model to over fit the defense by obfuscating gradients. Fortunately, offline adversarial training appears to sufficiently augment the training set so that the defenses can generalize.

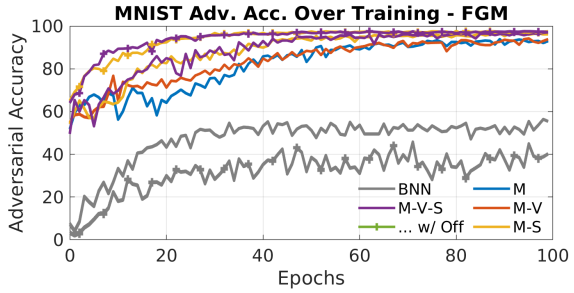
5.3.1. MNIST Accuracy Evolution

Figure 1 illustrates how the adversarial accuracy evolves over the training process. We chose to train for 100 epochs to give the defenses adequate opportunity to influence the model. Figure 1a shows attack accuracy against the FGSM

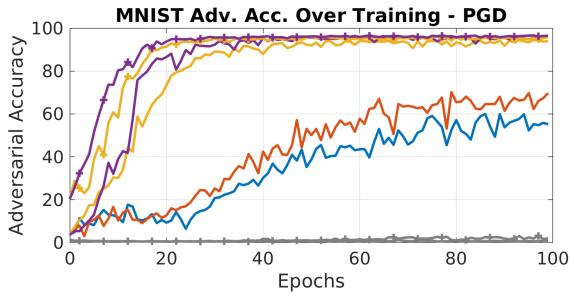
[1] https://github.com/MadryLab/mnist_challenge

and Fig. 1b shows accuracy against PGD. Models trained with diversity promoting penalties are given in color and models without are given in gray-scale. Models supplemented with offline adversarially training have "+" symbols in addition to the matched colors to identify their training penalties. In both cases, attacks are generated using the *test* set against the current model state. These figures provide some insight into how the defense is instilled in the model as it is trained. The mean and variances penalties appear to slowly (but consistently) influence the model throughout the training process. These penalties seem to still be improving the adversarial accuracy against PGD attacks even after 100 epochs have elapsed, well after the standard accuracy has converged.

As speculated in Sec. 4.4, the non-sparse penalty appears to give the model greater flexibility and is easier to learn. This quick increase and the disparity between white and black box performance in Table 1 may indicate that the penalty is prone to causing the defense to over fit by obfuscating gradients. However, this is assuaged with the use of offline adversarial training. All models converged to a standard accuracy of 99% within the first two epochs.



(a) FGSM L_∞ $\varepsilon = 0.3$



(b) PGD L_∞ $\varepsilon = 0.3$

Figure 1. Evolving white box attack accuracy after each epoch.

5.4. CIFAR-10

In this section, we evaluate our proposed diversity inducing penalties on the CIFAR-10 dataset (Krizhevsky et al.). The backbone network is VGG16. A Bayesian version of VGG16 is built by replacing the last convolutional block and the fully connected layers with variational alternatives. When training with our proposed penalties, we use the hyperparameters: $\lambda_M = 10$ and $\lambda_S = 20$. We report the L_∞ attack with a budget of $\varepsilon = 0.03$ here. PGD attack perform 40 gradient updates with a step size 0.001. Refer to Sec. 6 for ablation experiments on the attack budget. We report the black box attack accuracy using examples from an open repository.[2]

Table 2. Adversarial accuracy (%) under L_∞ white and black box attacks on CIFAR-10 with various methods of diversity promotion.

Loss	Std.	FGSM	PGD	Black-Box
VGG16	**90.4**	14.4	5.9	58.0
BVGG16	89.1	12.9	5.6	24.5
BVGG16-Off	85.8	58.8	57.0	84.3
M	90.1	14.2	6.1	64.1
M-V	87.1	19.6	10.2	55.5
M-S	88.4	50.2	47.7	55.2
M-V-S	88.7	56.6	60.2	55.1
M-S-Off	86.3	73.1	**72.3**	**84.8**
M-V-S-Off	85.7	**73.8**	63.8	83.4

Table 3. Comparison of accuracy (%) under PGD attack with different budget. Results for "Adv-CNN" and "Adv-BNN," representing adversarially trained CNN and BNN, are from (Liu et al., 2018b).

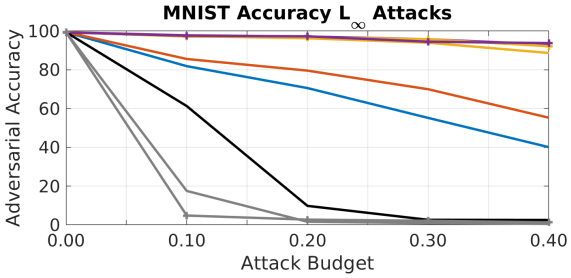
Method/Budget	0.0	0.015	0.035	0.055	0.07
Adv-CNN	80.3	58.3	31.1	15.5	10.3
Adv-BNN	79.7	68.7	45.4	26.9	18.6
M-S	**88.4**	61.3	47.8	39.8	34.3
BVGG16-Off	85.8	58.2	57.8	57.3	57.3
M-S-Off	86.3	**73.2**	**73.1**	**73.0**	**73.0**

Table 2 demonstrates the accuracy of the defended models on CIFAR-10. The standard loss is included in the table since it varies across models. Given the marginal improvements in adversarial accuracy from the variance-based penalties, we forego their use in this case.

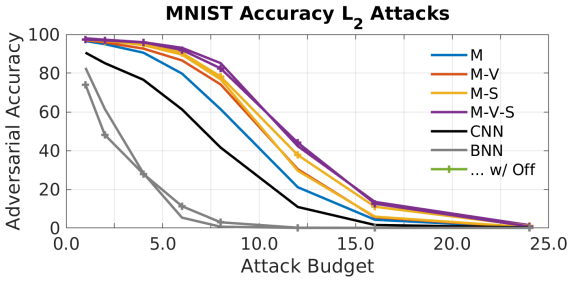
Unlike in the MNIST experiments, the mean penalty is insufficient to overcome white box attacks; however, it does offer improvements in the black box setting. Also unlike MNIST, these results do not indicate that our methods suffer from the obfuscated gradients problem as white box attacks are consistently more effective than black box attacks. The additional data augmentation from offline adversarial training further improves defense.

Table 3 juxtaposes several of our proposed defenses against results reported in Liu et al. for several attack budgets. We observe that our method without any adversarial training maintains higher standard accuracy and shows improved robustness to higher attacks budgets. Surprisingly, we observe

[2]https://github.com/MadryLab/cifar10_challenge

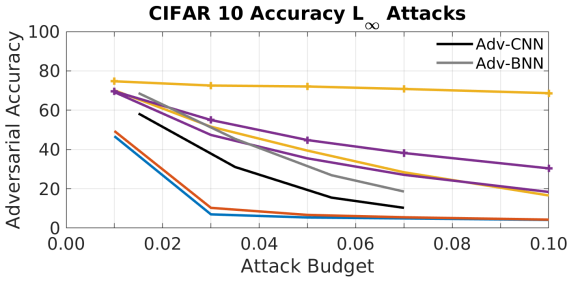


(a) L_∞ attack budgets.

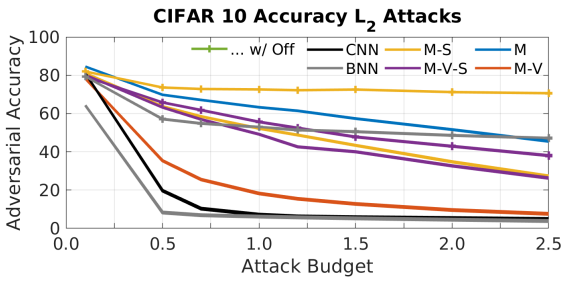


(b) L_2 attack budgets.

Figure 2. Varying attack budgets with 40 step PGD attacks against MNIST defenses.



(a) L_∞ attack budgets.



(b) L_2 attack budgets.

Figure 3. Varying attack budgets with 40 step PGD attacks against CIFAR-10 defenses.

that a Bayesian VGG16 with offline adversarial training obtains consistent accuracy above the other methods. We suspect this is because the adversarial examples used in offline training were constructed using PGD with 40 steps whereas the examples in ADV-BNN were defended using online training with 10 steps. Including our penalties consistently increases the performance of the offline model by approximately 15%.

6. Ablation

To test our defenses' efficacy under more diverse attack cases, we vary the PGD attack budget against models trained on MNIST and CIFAR-10. Figure 2 illustrates results from tests on MNIST and Fig. 3 from tests on CIFAR-10. Colors and symbols follow the same conventions as in Fig. 1.

Figures 2a and 3a demonstrates how the various models respond to increases in the attack budget of PGD L_∞ attacks. For L_∞ attacks against CIFAR-10, we compare against results reported in (Liu et al., 2018b) instead of a generic CNN or BNN as in other cases. Unsurprisingly, all the models show decreases in performance as the attack budget is increased. The best model consistently utilizes the mean and non-sparse penalized models with offline adversarial training. The inclusion of the variance penalty shows a slight improvement on MNIST but significantly worse performance on CIFAR-10. Against MNIST, the diversity penalties dramatically increase adversarial accuracy over offline training and likewise improve CIFAR-10 accuracy by approximately 15%. The online adversarially trained models outperform diversity-based models that do not include the non-sparse penalty, but are consistently out-performed by models that do include the penalty.

Figures 2b and 3b uses L_2 attacks but otherwise demonstrates the same variation as in Figs. 2b and 3a (changes in the attack budget of PGD). The most interesting feature is how well the mean penalty performs without any additional augmentations. It begins with the highest accuracy and is consistently higher than the model trained with the mean and non-sparse penalties and is primarily outperformed only by the full mean and non-sparse penalized models with offline training. Otherwise, the results are consistent with those found in the L_∞ study.

7. Discussion

We have demonstrated the efficacy of explicitly encouraging diversity of the output with respect to the input. On MNIST, we show that we can obtain strong adversarial robustness without the need for any form of adversarial training. In this case, our black box accuracy falls short of the white box accuracy, indicating we may be obfuscating gradients. Fortunately, the black box performance is still fair, which may mean that our method improves model robustness and obfuscates gradients. Including offline adversarial training in these models improves the black box accuracy so that it is on par with the white box accuracy. We suspect that the original MNIST training set is not diverse enough itself

and that additional data or augmentations may be sufficient to prevent the defense from over fitting and obfuscating gradients.

Our results on CIFAR-10 are quite encouraging. We demonstrate that our method is capable of improving adversarial accuracy with only a small reduction in standard accuracy. These models do not appear to suffer from the obfuscated gradients problem: black box accuracy is consistently higher than white box performance. Further, the ablation studies show a consistent reduction in accuracy as attack strength is increased. Finally, our model compares favorably with other Bayesian defense mechanisms, achieving superior performance in most cases without any adversarial training. The added use of offline adversarial training improves our models' performance so that they are superior in all cases.

We speculate that it may be possible to further improve our defense results by including additional forms of standard augmentation, e.g. the addition of Gaussian noise, shifting, etc. Similarly, we performed only a small search for the hyperparameters of the diversity penalties, λ, and used them for all penalty combinations and regardless of the use of offline training. Additional gains may be possible by performing a finer-grained search over these parameters.

8. Conclusions

In this work, we built upon the Bayesian neural network framework and introduced a novel technique, namely Defense through Diverse Directions, to achieve strong adversarial robustness. This is a daunting task, where models that utilize the concept of randomness to combat adversarial attack were previously limited to adding random noise layers to the network (Liu et al., 2018a) or simply applying a Bayesian neural network to take advantage of the stochasticity of the weights (Liu et al., 2018b). To the best of our knowledge, this is the first attempt to attain adversarial robustness through explicitly requiring the network to maintain and evenly distribute expected and sensible uncertainty with respect to input covariates, achieved by the various penalty terms we introduce. Without the need for online adversarial training, we demonstrate the effectiveness and robustness of our approach by achieving the strong adversarial (after-attack) accuracies on various datasets against different adversarial attacks.

Acknowledgements

This work was supported in part by NIH 1R01AA026879-01A1, NSF 1801494 grants and by the National Security Agency under Award No. H9823018D0008. We would like to thank Patrick Emmanuel and Stephen Ashurst for helpful discussions and computational assistance.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL http://tensorflow.org/. Software available from tensorflow.org.

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

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

Buckman, J., Roy, A., Raffel, C., and Goodfellow, I. Thermometer encoding: One hot way to resist adversarial examples. 2018.

Carlini, N. and Wagner, D. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pp. 39–57. IEEE, 2017.

Cissé, M., Bojanowski, P., Grave, E., Dauphin, Y., and Usunier, N. Parseval networks: Improving robustness to adversarial examples. In *ICML*, 2017.

Das, N., Shanbhogue, M., Chen, S.-T., Hohman, F., Chen, L., Kounavis, M. E., and Chau, D. H. Keeping the bad guys out: Protecting and vaccinating deep learning with jpeg compression. *arXiv preprint arXiv:1705.02900*, 2017.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Dhillon, G. S., Azizzadenesheli, K., Lipton, Z. C., Bernstein, J., Kossaifi, J., Khanna, A., and Anandkumar, A. Stochastic activation pruning for robust adversarial defense. *arXiv preprint arXiv:1803.01442*, 2018.

Dillon, J. V., Langmore, I., Tran, D., Brevdo, E., Vasudevan, S., Moore, D., Patton, B., Alemi, A., Hoffman, M. D., and Saurous, R. A. Tensorflow distributions. *CoRR*, abs/1711.10604, 2017. URL http://arxiv.org/abs/1711.10604.

Elsayed, G., Shankar, S., Cheung, B., Papernot, N., Kurakin, A., Goodfellow, I., and Sohl-Dickstein, J. Adversarial examples that fool both computer vision and time-limited humans. In *Advances in Neural Information Processing Systems*, pp. 3910–3920, 2018.

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

Guo, C., Rana, M., Cisse, M., and Van Der Maaten, L. Countering adversarial images using input transformations. *arXiv preprint arXiv:1711.00117*, 2017.

Hazan, T., Papandreou, G., and Tarlow, D. *Perturbations, Optimization, and Statistics*. MIT Press, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hornik, K., Stinchcombe, M. B., and White, H. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. URL http://dblp.uni-trier.de/db/journals/nn/nn2.html#HornikSW89.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In Bengio, Y. and LeCun, Y. (eds.), *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL http://arxiv.org/abs/1312.6114.

Krizhevsky, A., Nair, V., and Hinton, G. Cifar-10 (canadian institute for advanced research). URL http://www.cs.toronto.edu/~kriz/cifar.html.

LeCun, Y. and Cortes, C. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

Liu, X., Cheng, M., Zhang, H., and Hsieh, C.-J. Towards robust neural networks via random self-ensemble. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 369–385, 2018a.

Liu, X., Li, Y., Wu, C., and Hsieh, C.-J. Adv-bnn: Improved adversarial defense through robust bayesian neural network. *arXiv preprint arXiv:1810.01279*, 2018b.

Liu, Y., Chen, X., Liu, C., and Song, D. Delving into transferable adversarial examples and black-box attacks. *arXiv preprint arXiv:1611.02770*, 2016.

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

Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2574–2582, 2016.

Nicolae, M.-I., Sinn, M., Tran, M. N., Buesser, B., Rawat, A., Wistuba, M., Zantedeschi, V., Baracaldo, N., Chen, B., Ludwig, H., Molloy, I., and Edwards, B. Adversarial robustness toolbox v1.1.0. *CoRR*, 1807.01069, 2018. URL https://arxiv.org/pdf/1807.01069.

Pang, T., Xu, K., Du, C., Chen, N., and Zhu, J. Improving adversarial robustness via promoting ensemble diversity. *CoRR*, abs/1901.08846, 2019. URL http://arxiv.org/abs/1901.08846.

Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., and Swami, A. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*, pp. 372–387. IEEE, 2016.

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

Sharif, M., Bauer, L., and Reiter, M. K. n-ML: Mitigating adversarial examples via ensembles of topologically manipulated classifiers. arXiv preprint 1912.09059, December 2019. URL https://arxiv.org/abs/1912.09059.

Sharma, Y. and Chen, P.-Y. Attacking the madry defense model with l_1-based adversarial examples. *arXiv preprint arXiv:1710.10733*, 2017.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

Song, Y., Kim, T., Nowozin, S., Ermon, S., and Kushman, N. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. *arXiv preprint arXiv:1710.10766*, 2017.

Tramèr, F., Kurakin, A., Papernot, N., Goodfellow, I., Boneh, D., and McDaniel, P. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017a.

Tramèr, F., Papernot, N., Goodfellow, I., Boneh, D., and McDaniel, P. The space of transferable adversarial examples. *arXiv preprint arXiv:1704.03453*, 2017b.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Wang, S., Wang, X., Zhao, P., Wen, W., Kaeli, D., Chin, P., and Lin, X. Defensive dropout for hardening deep neural networks under adversarial attacks. In *Proceedings of the International Conference on Computer-Aided Design*, pp. 1–8, 2018.

Wang, X., Wang, S., Chen, P.-Y., Wang, Y., Kulis, B., Lin, X., and Chin, P. Protecting neural networks with hierarchical random switching: Towards better robustness-accuracy trade-off for stochastic defenses. *arXiv preprint arXiv:1908.07116*, 2019.

Wen, Y., Vicol, P., Ba, J., Tran, D., and Grosse, R. B. Flipout: Efficient pseudo-independent weight perturbations on mini-batches. *CoRR*, abs/1803.04386, 2018. URL http://arxiv.org/abs/1803.04386.

Xie, C., Wu, Y., Maaten, L. v. d., Yuille, A. L., and He, K. Feature denoising for improving adversarial robustness. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 501–509, 2019.

Zantedeschi, V., Nicolae, M.-I., and Rawat, A. Efficient defenses against adversarial attacks. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pp. 39–49, 2017.

Zhang, H., Chen, H., Song, Z., Boning, D., Dhillon, I. S., and Hsieh, C.-J. The limitations of adversarial training and the blind-spot attack. *arXiv preprint arXiv:1901.04684*, 2019.