Simpler Certified Radius Maximization by Propagating Covariances

Xingjian Zhen†, Rudrasis Chakraborty‡, Vikas Singh†
†University of Wisconsin-Madison ‡University of California, Berkeley
xzhen3@wisc.edu rudrasischa@gmail.com vsingh@biostat.wisc.edu

Abstract

One strategy for adversarially training a robust model is to maximize its certified radius – the neighborhood around a given training sample for which the model’s prediction remains unchanged. The scheme typically involves analyzing a “smoothed” classifier where one estimates the prediction corresponding to Gaussian samples in the neighborhood of each sample in the mini-batch, accomplished in practice by Monte Carlo sampling. In this paper, we investigate the hypothesis that this sampling bottleneck can potentially be mitigated by identifying ways to directly propagate the covariance matrix of the smoothed distribution through the network. To this end, we find that other than certain adjustments to the network, propagating the covariances must also be accompanied by additional accounting that keeps track of how the distributional moments transform and interact at each stage in the network. We show how satisfying these criteria yields an algorithm for maximizing the certified radius on datasets including Cifar-10, ImageNet, and Places365 while offering runtime savings on networks with moderate depth, with a small compromise in overall accuracy. We describe the details of the key modifications that enable practical use. Via various experiments, we evaluate when our simplifications are sensible, and what the key benefits and limitations are.

1. Introduction

The prevailing approach for evaluating the performance of a deep learning model involved assessing its overall accuracy profile on one or more benchmarks of interest. But the realization that many models were not robust to even negligible adversarially-chosen perturbations of the input data [44, 8, 23, 11], and may exhibit highly unstable behavior [9, 28, 33] has led to the emergence of robust training methods (or robust models) that offer, to varying degrees, immunity to such adversarial perturbations. Adversarial training has emerged as a popular mechanism to train a given deep model robustly [36, 45]. Each mini-batch of training examples shown to the model is supplemented with adversarial samples. It makes sense that if the model parameter updates are based on seeing enough adversarial samples which cover the perturbation space well, the model is more robust to such adversarial examples at test time [16, 22, 31]. The approach is effective although it often involves paying a premium in terms of training time due to multiple gradient calculations [42]. However, many empirical defenses can fail when the attack is stronger [10, 46, 2].

While ideas to improve the efficiency of adversarial training continue to evolve in the literature, a complementary line of work seeks to avoid adversarial sample generation entirely. One instead derives a certifiable robustness guarantee for a given model [48, 50, 55, 34, 54, 43, 5]. The overall goal is to provide guarantees that no perturbation within a certain range will change the prediction of the network. An earlier proposal, interval bound propagation (IBP) [17], used convex relaxations at different layers of the network to derive the guarantees. Unfortunately, the bounds tend to get very loose as the network depth increases, see Fig. 1 (a). Thus, the applicability to large high resolution datasets remains under-explored at this time.

Recently, following the idea in [30, 28] at a high level, Cohen et al. [12] introduced an interesting randomized smoothing technique, which can be used to certify the robust radius $C^*$. Assume that we have a base network $f_θ(·)$ for classification. On a training image $x \in \mathbb{R}^d$, the output $f_θ(x) \in \mathcal{Y}$ is the predicted label of the image $x$. Using $f_θ(·)$, we can build a “smoothed” neural network $g_θ(·)$.

$$g_θ(x) = \arg\max_{c \in \mathcal{Y}} \mathbb{P}(f_θ(x + ε) = c), \text{ where } ε \sim \mathcal{N}(0, \sigma^2I)$$

Here, $σ$ can be thought of as a trade-off between the robustness and the accuracy of the smoothed classifier $g_θ(·)$. One can obtain a theoretical certified radius $C^*_{IBP}$ which states that when $||δ||_2 \leq C^*_{IBP}$, the classifier $g_θ(x + δ)$ will have same label $y$ as $g_θ(x)$. MACER [52] nicely extended these ideas and also presented a differentiable form of randomized smoothing showing how it enables maximizing the ra-
Main intuition: MACER [52] showed that by sampling from a Gaussian distribution and softening the estimation of the distribution in the last layer, maximizing the certified radius is feasible. It is interesting to ask if tracking the “maximally perturbed” distribution directly—in the style of IBP—is possible without sampling. Results in [51] showed that the pre-activation vectors are i.i.d. Gaussian when the channel size goes to infinity. While unrealistic, it provides us a starting point. Since the Gaussian distribution can be fully characterized by the mean and the covariance, we can track these two quantities as it passes through the network until the final layer, where the radius is calculated. If implemented directly, this scheme must involve keeping track of how pixel correlations influence the entries of the covariances from one layer to the next, and the bookkeeping needs grow rapidly. Alternatively, [51] uses the fixed point of the covariance matrix to characterize it while it passes through the network, but this idea is not adaptable for maximizing the radius task in [52]. We will use other convenient approximations of the covariance to make directly tracking of the distribution of the perturbation feasible.

Other applications of certified radius maximization: Training a robust network is also useful when training in the presence of noisy labels [1, 15, 37]. Normally, both crowdsourcing from non-experts and web annotations, common strategies for curating large datasets introduce noisy labels. It can be difficult to train the model directly with the noisy labels without additional care [53]. Current methods either try to model the noise transition matrix [15, 37], or filter “correct” labels from the noisy dataset by collecting a consensus over different neural networks [18, 25, 32, 39]. This leads us to consider whether we can train the network from noisy labels without training any auxiliary network? A key observation here is that the margin of clean labels should be smoother than the noisy labels (as shown in Fig. 2).

Contributions: This paper shows how several known results characterizing the behavior of (and upper bounds on) covariance matrices that arise from interactions between random variables with known covariance structure can be leveraged that can propagate the applied to the training. This leads to a sampling-free method that performs favorably when compared to [52] and other similar approaches when the network depth is moderate. We show that our method is 5× faster on Cifar-10 dataset and 1.5× faster on larger datasets including ImageNet and Places365 relative to the current state-of-the-art without sacrificing much of the performance. Also, we show that the idea is applicable to or training with noisy labels.

2. Robust Radius Via Randomized Smoothing

We will briefly review the relevant background on robust radius calculation using Monte-Carlo (MC) sampling.

What is the robust radius? In order to measure the robustness of a neural network, the robust radius has been shown to be a sensible measure [48, 12]. Given a trained neural network \( f_0 \), the \( \ell_2 \)-robustness at data point \((x, y)\) is defined as the largest radius \( R \) of the ball centered at \( x \) such that all samples within the ball will be classified as \( y \) by the neural network \( f_0 \). Analogously, the \( \ell_2 \)-robustness of \( f_0 \) is defined as the minimum \( \ell_2 \)-robustness at data point \((x, y)\) over the dataset. But calculating the robust radius for the neural network can be hard; [48] provides a hardness result for the \( \ell_1 \)-robust radius. In order to make computing \( \ell_2 \)-robustness tractable, the idea in [12] suggests working with a tight lower bound, called the “Certified Radius”, denoted by \( 0 \leq C_R \leq R \). Let us now briefly review [12] its functions and features for a given base classifier \( f_0(\cdot) \).

Note that we want to certify that there will be no adversarial samples within a radius of \( C_R \). By smoothing out the perturbations \( \varepsilon \) around the input image/data \( x \) for the
base classifier \( f_0(\cdot) \), intuitively it will be harder to find an adversarial sample, since it will actually require finding a “region” of adversarial samples. If we can estimate a lower bound on the probability of the base classifier to correctly classify the perturbed data \( x + \varepsilon \), denoted as \( p_{c_{x_0}} \), as well as an upper bound of the probability of an incorrect classification \( \overline{p}_{c_{x_0}} \), where \( c_{x_0} \) is the true label of \( x \) and \( \overline{c} \) is the “most likely to be confused” incorrect label, a nice result for the smoothed classifier \( g_0(\cdot) \) is available.

**Theorem 1.** [12] Let \( f_0 : \mathbb{R}^d \rightarrow \mathcal{Y} \) be any deterministic or random function, and let \( \varepsilon \sim \mathcal{N}(0, \sigma^2 I) \). Let \( g_0 \) be the randomized smoothing classifier defined as \( g_0(x) = \arg\max_{c \in \mathcal{Y}} P(f_0(x + \varepsilon) = c) \). Suppose \( c_{x_0}, \overline{c} \in \mathcal{Y} \) and \( p_{c_{x_0}} \geq \overline{p}_{c_{x_0}} \). Then \( g_0(x + \delta) = c_{x_0} \) for all \( \|\delta\|_2 < C_R \), where \( C_R = \varepsilon \sigma (\Phi^{-1}(p_{c_{x_0}}) - \Phi^{-1}(\overline{p}_{c_{x_0}})) \).

The symbol \( \Phi \) denotes the CDF of the standard Normal distribution. \( \Phi \) and \( \Phi^{-1} \) are involved because of smoothing the Gaussian perturbation \( \varepsilon \). The proof of this theorem can be found in [12]. We also include it in the appendix.

**How to compute the robust radius?** Using Theorem 1, we will need to compute the lower bound \( p_{c_{x_0}} \), the main ingredient to compute \( C_R \). In [12], the authors introduced a sampling-based method to compute the lower bound of \( p_{c_{x_0}} \) in the test phase. The procedure first samples \( n_0 \) noisy samples around \( x \), and passes it through the base classifier \( f_0 \) to estimate the classified label after smoothing. Then, we sample \( n \) noisy samples, where \( n \gg n_0 \) to estimate the lower bound of \( p_{c_{x_0}} \) for a certain confidence level \( \alpha \).

### 3. Track Distribution Approximately

In the last section, we discussed how to calculate \( p_{c_{x_0}} \) in a sampling (Monte Carlo) based setting. However, this method is based on counting the number of correctly classified samples, which is not differentiable during training. In order to tackle this problem, [52] introduced an alternative – soft randomized smoothing – to calculate the lower bound

\[
p_{c_{x_0}} = \mathbb{E}_{\varepsilon \sim \mathcal{N}(0, \sigma^2 I)} \left[ \frac{e^{\beta u_{0'}^\top (x + \varepsilon)}}{\sum_{c \in \mathcal{Y}} e^{\beta u_{0'}^\top (x + \varepsilon)}} \right]
\]

where \( u_0 \) is the network \( f_0 \) without the last softmax layer, i.e., \( f_0 = \arg\max_{c \in \mathcal{Y}} \text{softmax}(u_0) \), and \( \beta \) is a hyperparameter.

From Fig. 1 (b), observe that if we have enough MC samples, we can reliably estimate \( p_{c_{x_0}} \) effectively by counting the number of correctly classified samples. If we can bypass MC sampling to estimate the final distribution, the gains in runtime can be significant. However, directly computing the joint distribution of the perturbations of all the pixels is infeasible: we need simplifying assumptions.

**Gaussian pre-activation vectors:** The first assumption is to use a Gaussian distribution to fit the pre-activation vectors. As briefly mentioned before, this is true when the channel size goes to infinity by the central limit theorem. In practice, when the channel size is large enough, e.g., a ResNet-based architecture [20], this assumption may be acceptable with a small error (evaluated later in experiments). Therefore, we will only consider the first two moments, which is reasonable for Gaussian perturbation [49].

**Second moments:** Our second assumption is that in each layer of a convolution network, the second moments are identical for a fixed Gaussian perturbation \( \varepsilon \). Due to weight sharing and the linearity of the convolution operators, the second moments will only depend on the kernel matrix without the position information. A more detailed discussion is in Obs. 2 and the appendix.

**Notations and setup:** Let \( N \) be the number of channels. We use \( \Sigma \) as the covariance matrices of the perturbation across the channels unless otherwise noted. The input perturbation comes from Gaussian perturbation \( \varepsilon \) where \( \Sigma = \sigma^2 I \). As the image passes through the network, the input perturbation directly influences the output at each pixel as a function of the network parameters. We use \( \Sigma_i \in \mathbb{R}^{N \times N} \), shorthand for \( \Sigma_{x_{i}} \), to denote the covariance of the perturbation distribution associated with pixel \( i \) of image \( x \) denoted as \( x_i \). We call \( \Sigma[i, j] \) as the \((i, j)\)-entry of \( \Sigma \). Notice that the \( N \) changes from one layer to the other as the number of channels are different. So, the size of \( \Sigma \) will change. Let \( M_{q} \) be the number of pixels in the \( q^{th} \) layer input, i.e., for \( q = 1, M_{1} \) is the number of pixels in the 1st hidden layer of the network.

Similarly, \( \mu_{x_i} \) or \( \mu_{i} \in \mathbb{R}^N \) is the mean of the distribution of the pixel \( x_i \) intensity after the perturbation. In the input layer, since the perturbation \( \varepsilon \sim \mathcal{N}(0, \sigma^2 I) \), \( \mu_{i} = x_i \). At the \( u_0 \) layer, the number of channels is the number of classes, with the number of pixels being 1. We use \( \mu_{\{c_{x}\}} \) and \( \Sigma\{c_{x_i}, c_{x}\} \) to denote the \( c_{x_i} \) component of \( \mu \) and \( \Sigma \{c_{x_i}, c_{x}\} \)-entry of \( \Sigma \) respectively. To denote the cross-correlation between two pixels \( x_i, x_j \), we use \( E_{x_i, x_j} \) or \( \Sigma_{ij} \in \mathbb{R}^{N \times N} \). Note that this cross-correlation is across channels. For the special case with channel size \( N = 1 \), we will use \( \sigma(1) \) to represent the variance in the \( i^{th} \) layer. Let us define

\[
e_{\xi} = \arg\max_{c \in \mathcal{Y}} \mu_{c}, \quad \overline{c} = \arg\max_{c \in \mathcal{Y}, c \neq c_{\xi}} \mu_{c}
\]

Let the number of classes \( C = |\mathcal{Y}| \). Then, we can state the following.

**Observation 1.** Using \( u_0 \), the prediction of the model can be written as \( f_0(x) = \arg\max_{c \in \mathcal{Y}} \text{softmax}(u_0(x)) \). Assume \( u_0(x) \sim \mathcal{N}(\mu, \Sigma) \), where \( x \sim \mathcal{N}(\mu_{x}, \Sigma_{x}) \), \( \mu \in \mathbb{R}^C \) and \( \Sigma \in \mathbb{R}^{C \times C} \). Then the estimation of \( p_{c_{x_0}} \) is

\[
p_{c_{x_0}} = \Phi \left( \frac{\mu_{\{c_{x}\}} - \mu_{\overline{c}}}{\sqrt{\Sigma\{c_{x_i}, c_{x}\} + \Sigma\{c_{\xi}, \overline{c}\} - 2\Sigma\{c_{x_i}, \overline{c}\}}} \right)
\]
Notice that propagating \( \mu \) through the network is simple, since tracking the mean is the same as directly passing it through the network when there is no nonlinear activation, and requires no cross-correlation between pixels. But tracking \( \Sigma \) at each step of the network can be challenging and some approximating techniques have been used in literature for simple networks [41]. To see this, let us consider a simple 1-D example.

**Bookkeeping problem:** Consider a simple 1-D convolution with a kernel size \( k \). By Obs. 1, we will need the distribution of \( u_0(x) \) of the \( i^{th} \) layer (i.e., the network without the softmax layer). Directly, this will involve taking into account \( k^1 \) pixels in the \((i - 1)^{th}\) layer, and \( k^2 \) pixels in the \((i - 2)^{th}\) layer. We must calculate the covariance \( \Sigma \) and also calculate the cross-correlation \( E \) between all \( k^3 \) pixels in \((i - q)^{th}\) layer. This trend stops when we hit \( q^q > M_{i-q} \), where \( M_{i-q} \) is the number of pixels at \((i - q)\) layer, but it is impractical anyway.

If we temporarily assume that the network involves no activation functions, and if the input perturbation is identical for all pixels, then the variance of all pixels after perturbation is also identical. Thus, the variance of each pixel only relies on the variance of the perturbation and not on the pixel intensity. This may allow us to track one covariance matrix instead of \( M \) for all \( M \) pixels.

**Observation 2.** With the input perturbation \( \varepsilon \) set to be identical along the spatial dimension and without nonlinear activation function, for \( q^{th} \) hidden convolution layer with \( \{h_i\}_{i=1}^{M_q} \) output pixels, we have \( \Sigma_{h_i}^{(q)} = \Sigma_{h_j}^{(q)}, \forall i, j \in \{1, \cdots, M_q\} \).

The Obs. 2 only reduces the cost marginally: instead of computing all the covariances of the perturbation for all pixels, \( \Sigma_{i}^{(i-q)}, \Sigma_{j}^{(i-q)}, \cdots, \Sigma_{k}^{(i-q)} \), we only need to compute a single \( \Sigma^{(i-q)} \). Unfortunately, we still need to compute all different \( E_{ij}^{(i-2)} \) that will contribute to \( u_0(x) \) (also see worked out example in the appendix). Thus, due to these cross-correlation terms \( E_{ij} \), the overall computation is still not feasible. In any case, the assumption itself is unrealistic: we do need to take nonlinear activations into account which will break the identity assumption of the second moments. For this reason, we explore a useful approximation which we discuss next.

**3.1. How To Make Distribution Tracking Feasible**

From the previous discussion, we observe that a key bottleneck of tracking distribution across layers is to track the interaction between pairs of pixels, i.e., cross-correlations. Thus, we need an estimate of the cross-correlations between pixels. In [19], the authors provide an upper-bound on the joint distribution of two multivariate Gaussian random variables such that the upper bounding distribution contains no cross-correlations. This result will be crucial for us.

Formally, let \( x_1, x_2 \in \mathbb{R}^N \) be two random vectors representing two pixels with \( N \) channels. Without any loss of generalization, assume that \( x_1 \sim \mathcal{N}(0, \Sigma_1) \), and \( x_2 \sim \mathcal{N}(0, \Sigma_2) \) (if the mean is not \( 0 \), we can subtract the mean without affecting the covariance matrix). Also, assume that we do not know the cross-correlation between \( x_1 \) and \( x_2 \), i.e., \( E_{12} \). Instead, the correlation coefficient \( r \) is bounded by \( r_{\text{max}} \), i.e., \(|r| \leq r_{\text{max}}\).

With the above assumptions, we can bound the covariance matrix of the joint distribution of two \( N \)-dimensional random vectors \( x_1, x_2 \) by two independent random vectors \( \tilde{x}_1, \tilde{x}_2 \). We will use the notation \( "\tilde{\cdot}" \) to denote the upper bound estimation of \( \cdot \). The upper bound here means that \( \tilde{\Sigma} - \Sigma \) is a positive semi-definite matrix, where \( \tilde{\Sigma} \) is the joint distribution of the two independent random vectors \( \tilde{x}_1 \) and \( \tilde{x}_2 \). Here, \( \tilde{\Sigma} \) is the joint distribution of \( x_1, x_2 \) with correlation.

**Theorem 2.** [19] When \( \tilde{x}_1 \sim \mathcal{N}(0, \tilde{\Sigma}_1) \), and \( \tilde{x}_2 \sim \mathcal{N}(0, \tilde{\Sigma}_2) \), the covariance matrix \( B = \tilde{\Sigma} = \begin{bmatrix} \tau_1 \Sigma_1 & 0 \\ 0 & \tau_2 \Sigma_2 \end{bmatrix} \) bounds the joint distribution of \( x_1 \) and \( x_2 \), i.e., \( B \succeq \Sigma = \begin{bmatrix} \tau_1 \Sigma_1 & E_{12} \\ E_{21} & \Sigma_2 \end{bmatrix} \), where \( \tau_1 = \frac{1}{\sqrt{1 - \kappa^2}}, \tau_2 = \frac{1}{\sqrt{1 - \kappa^2}}, \kappa^2 \leq \frac{1 - 2r_{\text{max}}}{1 + 2r_{\text{max}}} + \eta^2, \) and \( 0.5 \leq \eta \leq \frac{1}{1 + r_{\text{max}}} \).

With this result in hand, we now discuss how to use it to makes the tracking of moments across layers feasible.

**How to use Theorem 2?** By Obs. 2, we can store one covariance matrix over the convolved output pixels at each layer. Notice that due to the presence of the cross-correlation between output pixels, we also need to store cross-correlation matrices, which was our bottleneck! But with the help of Theorem 2, we can essentially construct independent convolved outputs, called \( \{h_i\}_i \), that bound the covariance of the original convolved outputs, \( \{h_i\}_i \). To apply this theorem, we need to estimate the bounding covariance matrix \( B \), which can be achieved with the following simple steps (the notations are consistent with Theorem 2)

(a) We estimate the bound on correlation coefficient \( r_{\text{max}} \)
(b) Assign \( \eta = \frac{1}{1 + r_{\text{max}}} \)
(c) Assign \( \kappa = 0 \) which essentially implies \( \tau_1 = \tau_2 \)

**Remark:** When computing the variance \( \Sigma[e_x, e_x] \) in the \( i^{th} \) layer, we need only \( k \) upper bound of covariances \( \Sigma^{(i-1)}, \Sigma^{(i-1)}, \cdots, \Sigma^{(i-1)} \) from the \((i - 1)^{th}\) layer. Moreover, using the assumption that the covariance matrices of the \((i - 1)^{th}\) layers to be identical across pixels when the input perturbation is identical, we only compute \( \Sigma^{(i-1)} \), which in turn requires computing only one upper bound of covariance. Hence, the computational cost reduces to linear in terms of the depth of the network.

**Ansatz:** The assumption of identical pixels (when removing the mean) is sensible when the network is linear. But the
assumption is undesirable. So, we will need a mechanism to deal with the nonlinear activation function setting. Further, we will need to design the mechanics of how to track the mean and covariance for different type of layers. We will describe the details next.

3.2. Robust Training By Propagating Covariances

Overview: We described simplifying the computation cost by tracking the upper bounds on the perturbation of the independent pixels. We introduce details of an efficient technique to track the covariance of the distribution across different types of layers in a CNN. We will also describe how to deal with nonlinear activation functions.

We treat the $i^{th}$ pixel, after perturbation, as drawn from a Gaussian distribution $x_i \sim \mathcal{N}(\mu_i, \Sigma)$, where $\mu_i \in \mathbb{R}^{N}$ and $\Sigma$ is the covariance matrix across the channels (note that $\Sigma$ is the same across pixels for the same layer). We may remove the indices to simplify the formulation and avoid clutter. A schematic showing propagating the distribution across LeNet [6] model, for simplicity, is shown in Fig. 3(a) denoted by the colored ovals.

To propagate the distribution through the whole network, we need a way to propagate the moments through the layers, including commonly used network modules, such as convolution and fully connected layers. Since the batch normalization layer normally has a large Lipschitz constant, we do not include the batch normalization layer in the network. We will introduce the high-level idea, while the low-level details are in the appendix.

Convolution layer: Since the convolution layer is a linear operator, the covariance of an output pixel $\Sigma_h \in \mathbb{R}^{N_{out} \times k^2} \times N_{out}$ is defined as $\Sigma_h = W^T \Sigma W$. Here, $x \in \mathbb{R}^{N_{in} \times k^2}$ be the vector consisting of all the independent variables inside a $k \times k$ kernel $\{x_k\}, \Sigma \in \mathbb{R}^{N_{in} \times k^2 \times N_{in} \times k^2}$ is the covariance of the concatenated $x$, $W$ is the reshaped weight matrix of the shape $N_{in} \times k^2 \times N_{out}$.

We need to apply Theorem 2 to compute the upper bound of $\Sigma_h$ as $\Sigma_h = (1 + r_{max})W^T \Sigma W$ to avoid the computational costs of the dependency from cross-correlations. A pictorial description of propagating moments through the convolution layer is shown in Fig. 3(b).

First (and other) linear layers: The first linear layer can be viewed as a special case of convolution with kernel size equal to the input spatial dimension. Since there will only be one output neuron $h$ (with channels), there is no need to break the cross-correlation between neurons. Thus, $\Sigma_h = W^T \Sigma W$ and takes a form similar to the convolution layer.

Special case: From Obs. 1, we only need the largest two intensities to estimate the $p$ in the $u_o(x)$ layer. Thus, if there is only one linear layer as the last layer in the $u_o(x)$, as in most of ResNet like models, this can be further simplified. We only need to consider the covariance matrix between $c_{x}$ and $\tilde{c}$ index of $u_o(x)$. Thus, this will need calculating a $2 \times 2$ covariance matrix instead of a $C \times C$ matrix.

On the other hand, if the network consists of multiple linear layers, calculating the moments of the subsequent linear layers must be handled differently. Let $x \sim \mathcal{N}(\mu_x, \Sigma_x)$ be the input of the $i^{th}$ linear layer given by $h = W^T_i x + b_i$, then

$$h \sim \mathcal{N}(W^T_i \mu_x + b_i, W^T_i \Sigma_x W_i).$$

Here, $W_i \in \mathbb{R}^{N_{i} \times N_{i+1}}, b_i \in \mathbb{R}^{N_{i+1}}$, and $h \in \mathbb{R}^{N_{i+1}}$.

Pooling layer: Recall that the activation of a max pooling layer is $\{x_i\}$ where each $x_i \in \mathbb{R}^{N_i}$ and the index $i$ varies over the spatial dimension. Observe that as we identify each $x_i$ by the respective distribution $\mathcal{N}(\mu_i, \Sigma)$, applying max pooling over $x_i$ essentially requires computing the maximum over $\{\mathcal{N}(\mu_i, \Sigma)\}$, which is not a well-defined operation. Thus, we restrict ourselves to average pooling. This can be viewed as a special case of the convolution layer with no overlapping and the fixed kernel: $h \sim \mathcal{N}\left(\frac{1}{W} \sum_{x_i \in W} \mu_i, \sum_{x_i \in W} \Sigma \right)$. $W$ is the kernel window.

Normalizations layer: For the normalization layer, given by $h = (x - \mu')/\sigma'$, where $\mu', \sigma'$ can be computed in different ways [24, 4, 47], we have $h \sim \mathcal{N}\left((\mu_x - \mu')/\sigma', \Sigma_x/\sigma'^2\right)$. However, as the normalization layers often have large Lipschitz constant [3], we omit these layers in this work.

Activation layer: This is the final missing piece in efficiently tracking the moments. The overall goal is to find an identical upper bound of the second moments after the activation layer when the input vectors share identical second moments. Also, the first moments should be easier to compute, and ideally, will have a closed form. In [7, 29], the authors introduced a scheme to compute the mean and variance after a ReLU operation. Since ReLU is an element-wise operation, for each element (a scalar), assume $x \sim \mathcal{N}(\mu, \sigma^2)$. After ReLU activation, the first and second moments of the output are given by:

$$\text{E} \left(\text{ReLU}(x)\right) = \frac{1}{2} \mu - \frac{1}{2} \mu \text{erf}\left(\frac{-\mu}{\sqrt{2} \sigma}\right) + \frac{1}{\sqrt{2 \pi}} \sigma \exp\left(-\frac{\mu^2}{2 \sigma^2}\right),$$

$$\text{var} \left(\text{ReLU}(x)\right) = \text{var}(x).$$

Here, $\text{erf}$ is the Error function. Since we want an identical upper bound of the covariance matrix after ReLU, as well as the closed form of the mean, we use ReLU(x) $\sim \mathcal{N}(\mu_a, \Sigma_a)$ where

$$\mu_a = \frac{1}{2} \mu - \frac{1}{2} \mu \text{erf}\left(\frac{-\mu}{\sqrt{2} \sigma}\right) + \frac{1}{\sqrt{2 \pi}} \sigma \exp\left(-\frac{\mu^2}{2 \sigma^2}\right),$$

$$\Sigma_a \leq \Sigma$$

$\sigma$ is the square root of the diagram of $\Sigma$. $\mu$ is the mean of the input vector. All the operators in the first equation are element-wise operators.

Last layer/prediction: The last layer is the layer before softmax layer, which represents the “strength” of the model.
for a specific class. By Obs. 1, we have the estimate

\[ p_{c_x} = p_{\bar{c}_x} = \Phi \left( \frac{\mu[c_x] - \mu[c]}{\sqrt{\Sigma[c_x, c_x] + \Sigma[c, c] - 2\Sigma[c_x, c]}} \right) \]

and \( p_{\bar{c}_x} = 1 - p_{c_x} \) as an upper bound estimation. By Theorem 1, the certified radius is

\[ C_R = \frac{1}{\sigma} \left( \Phi^{-1}(p_{c_x}) - \Phi^{-1}(p_{\bar{c}_x}) \right) \]

\[ = \frac{\mu[c_x] - \mu[c]}{\sqrt{\Sigma[c_x, c_x] + \Sigma[c, c] - 2\Sigma[c_x, c]}}. \]

**Network structures used:** In the experiment, we applied two types of network on different dataset, LeNet [6] and PreActResnet-18 [21].

LeNet requires convolution layer, average pooling layer, activation layer, and linear layer. We build the network with three convolution layers with activation and pooling after each layer, and two linear layers.

The structure of PreActResnet-18 is similar with two major differences – the residual connection and it involves only one linear layer. For the residual connection, it can be viewed as a special type of linear layer. Due to the assumption of independence, the final covariance is the addition of two inputs. Also, there is only one linear layer as the final layer. Thus, we can reduce the cost of computing the whole covariance matrix to only computing the covariance matrix of the largest two intensities.

As discussed above, we removed all batch normalization layers within the network as well as replaced all max pooling operations to the average pooling layer in the network structure. Our experiments suggest that there is minimal impact on performance.

### 3.3. Training Loss

In the spirit of [52], the training loss consists of two parts: the classification loss and the robustness loss, i.e., the total loss \( l(g_0; x, y) = l_C(g_0; x, y) + \lambda l_R(g_0; x, y) \).

Similar to the literature, we use the softmax layer on the expectation to compute the cross-entropy of the prediction and the true label, given by

\[ l_C(g_0; x, y) = y \log(\text{softmax}(E[u_0(x)])) \]

Here, \( l_C(g_0; x, y = c_x) \) is

\[ \max(0, \Gamma - \sigma \frac{\mu[c_x] - \mu[c]}{\sqrt{\Sigma[c_x, c_x] + \Sigma[c, c] - 2\Sigma[c_x, c]}}) \]

Thus, minimizing the loss of \( l_{C_R} \) is equivalent to maximizing \( C_{R} \). \( \Gamma \) is the offset to control the certified radius.

### 4. Experiments

In this section, we discuss the applicability and usefulness of our proposed model in two applications namely (a) image classification tasks to show the performance of our proposed model both in terms of performance and speed (b) trainability of our model on data with noisy labels.

#### 4.1. Robust Training

Similar to [12], we use the approximate certified test set accuracy as our metric, which is defined as the percentage of test set whose \( C_R \leq r \). For a fair comparison, we use the Monte Carlo method introduced in [12] Section 3.2 to compute \( C_R \) here just as our baseline model does. Recall that \( C_R = 0 \) if the classification is wrong. Otherwise, \( C_R = \sigma \Phi^{-1}(p_A) \) (please refer to the pseudocode in [12]).

In order to run certification, we used the code provided by
We report the average certified radius (ACR), which is defined as \( \frac{1}{N} \sum_{i=1}^{N} C_R(x^i) \) over the test set.

Datasets and baselines: We evaluate our proposed model on five vision datasets: MNIST [27], SVHN [35], Cifar-10 [26], ImageNet [13], and Places365 [57]. We modify LeNet for MNIST dataset and PreActResnet-18 [21] for SVHN, Cifar-10, ImageNet, and Places365 datasets similarly as in [12]. Our baseline model is based on Monte Carlo samples, which requires a large number of samples to make an accurate estimation. In the rest of the section, we will observe that our model can be at best 5 times faster than the baseline model. For the larger dataset, since MACER [52] uses a reduced number of MC samples, our model is 1.5 times faster.

Model hyperparameters: During training, we use a similar strategy as our baseline model. We train the base classifier first and then fine-tune our model considering the aforementioned robust error. We train a total of 200 epochs with the initial learning rate to be 0.01 for MNIST, SVHN, and Cifar-10. The learning rate decays at 100, 150 epochs respectively. The \( \lambda \) is set to be 0 in the initial training step, and changes to 4.0 at epoch 100 for MNIST, SVHN, and Cifar-10. For ImageNet and Places365 dataset, we train 120 epochs with \( \lambda \) being 0.5 after 30 epochs. The initial learning rate is 0.01 and decays linearly at 30, 60, 90 epochs.

Results: We report the numerical results in Table 3, where, the number reported in each column represents the ratio of the test set with the certified radius larger than the header of that column. Thus, the larger the number is, the better the performance of different models. The ACR is the average of all the certified radius on the test set. Note that the certified radius is 0 when the classification result is wrong. It is noticeable that our model strikes a balance of robustness and the training speed. We achieve 5 times speed-up over [52], which uses the Monte Carlo method during the training phase, as shown in Fig. 5. On the other hand, compared with [12], our model achieves competitive accuracy and certified radius. We like to point out that although SmoothAdv [40] is a powerful model, we did not compare with SmoothAdv because MACER [52] performs better than SmoothAdv [40] in terms of ACR and training speed.

<table>
<thead>
<tr>
<th>Layer number</th>
<th>1</th>
<th>5</th>
<th>9</th>
<th>13</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC (1000 samples)</td>
<td>0.243</td>
<td>0.913</td>
<td>2.740</td>
<td>2.999</td>
<td>0.712</td>
</tr>
<tr>
<td>Upper bound</td>
<td>0.256</td>
<td>1.126</td>
<td>4.069</td>
<td>5.367</td>
<td>1.208</td>
</tr>
</tbody>
</table>

Separate from the quantitative performance measures, we also evaluate the validity of Gaussian assumption on the pre-activation vectors within the network. Here, we choose PreActResNet-18 on ImageNet to visualize the first two channels across different layers. The detailed results are shown in Fig. 6 and Table 2. These results not only show that the assumption is reasonable along the neural network, but also demonstrates that our method can estimate the covariance matrix well when the depth of network is moderate. Deeper networks, where the bounds get looser, are described in the appendix.

Ablation study: We perform an ablation study on the choice of the hyperparameters for Places365 dataset. We fix \( \sigma \) of the perturbation to be 0.5. We first test the influence of \( \lambda \) which is the balance between the accuracy (first moments) and the robustness (second moments). Also, to verify the estimation of \( r_{\text{max}} \), we tried different \( r_{\text{max}} \) estimates while fixing \( \lambda = 0.5 \). The detailed results are shown in Table 4.

Discussion: A key benefit of our method is the training time. As shown in Fig. 5, our method can be 5 times faster on Cifar-10, dataset, with a comparable ACR as MACER. For larger datasets, since MACER reduces the number of MC samples in their algorithm, our method is only 1.5 times faster with a slightly better ACR than MACER. Hence, our method is a cheaper substitute of the SOTA with a marginal performance compromise.

Limitations: There are some limitations due to the simplifications incorporated in our model. When the network is extremely deep, e.g., Resnet-101, the estimation of the second moments tends to be looser as the network grows deeper. Another minor issue is when the input perturbation is large. As observed from Table 3, the ACR drops for \( \sigma = 1.0 \) from \( \sigma = 0.5 \). The main reason is the assumption that samples are Gaussian distributed. Hence, as the perturbation grows larger, the number of channels, by the

![Figure 5: The training speed for three models on Cifar-10 and ImageNet dataset, including [12], MACER [52], and ours.](image)

![Figure 6: A visualization of the first two channels within the neural network across different layers. The dots are the actual MC samples and the color represents the density at that point. The blue oval is generated from the covariance matrices we are tracking.](image)
Table 3: The results on MNIST, SVHN, Cifar-10, ImageNet, and Places365 with the certified robustness. The number reported in each column represents the ratio of the test set with the certified radius larger than the header of that column under the perturbation $\sigma$. ACR is the average certified radius of all the test samples. A larger value is better for all the numbers reported.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Method</th>
<th>$\sigma$</th>
<th>0.00</th>
<th>0.25</th>
<th>0.50</th>
<th>0.75</th>
<th>1.00</th>
<th>1.25</th>
<th>1.50</th>
<th>1.75</th>
<th>ACR</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>Cohen [12]</td>
<td>0.25</td>
<td>0.99</td>
<td>0.97</td>
<td>0.94</td>
<td>0.89</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.537</td>
</tr>
<tr>
<td></td>
<td>MACER [52]</td>
<td>0.99</td>
<td>0.99</td>
<td>0.97</td>
<td>0.95</td>
<td>0.92</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.518</td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>0.99</td>
<td>0.98</td>
<td>0.96</td>
<td>0.92</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.504</td>
</tr>
<tr>
<td>SVHN</td>
<td>Cohen [12]</td>
<td>0.25</td>
<td>0.99</td>
<td>0.97</td>
<td>0.94</td>
<td>0.91</td>
<td>0.84</td>
<td>0.75</td>
<td>0.59</td>
<td>0.33</td>
<td>1.455</td>
</tr>
<tr>
<td></td>
<td>MACER [52]</td>
<td>0.99</td>
<td>0.98</td>
<td>0.96</td>
<td>0.94</td>
<td>0.90</td>
<td>0.83</td>
<td>0.75</td>
<td>0.50</td>
<td>0.35</td>
<td>1.283</td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>0.98</td>
<td>0.98</td>
<td>0.95</td>
<td>0.91</td>
<td>0.87</td>
<td>0.77</td>
<td>0.62</td>
<td>0.37</td>
<td>0.25</td>
<td>1.085</td>
</tr>
<tr>
<td>Cifar-10</td>
<td>Cohen [12]</td>
<td>0.25</td>
<td>0.90</td>
<td>0.70</td>
<td>0.44</td>
<td>0.26</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.469</td>
</tr>
<tr>
<td></td>
<td>MACER [52]</td>
<td>0.86</td>
<td>0.72</td>
<td>0.56</td>
<td>0.39</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.540</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>0.89</td>
<td>0.68</td>
<td>0.48</td>
<td>0.36</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.509</td>
<td></td>
</tr>
<tr>
<td>ImageNet</td>
<td>Cohen [12]</td>
<td>0.25</td>
<td>0.67</td>
<td>0.48</td>
<td>0.37</td>
<td>0.24</td>
<td>0.14</td>
<td>0.08</td>
<td>0.06</td>
<td>0.03</td>
<td>0.434</td>
</tr>
<tr>
<td></td>
<td>MACER [52]</td>
<td>0.61</td>
<td>0.53</td>
<td>0.44</td>
<td>0.35</td>
<td>0.24</td>
<td>0.15</td>
<td>0.19</td>
<td>0.09</td>
<td>0.04</td>
<td>0.586</td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>0.67</td>
<td>0.53</td>
<td>0.36</td>
<td>0.29</td>
<td>0.19</td>
<td>0.12</td>
<td>0.07</td>
<td>0.03</td>
<td>0.475</td>
<td></td>
</tr>
<tr>
<td>Places365</td>
<td>Cohen [12]</td>
<td>0.25</td>
<td>0.75</td>
<td>0.60</td>
<td>0.43</td>
<td>0.26</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.416</td>
</tr>
<tr>
<td></td>
<td>MACER [52]</td>
<td>0.81</td>
<td>0.71</td>
<td>0.59</td>
<td>0.43</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.556</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>0.80</td>
<td>0.72</td>
<td>0.55</td>
<td>0.37</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.518</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>0.65</td>
<td>0.54</td>
<td>0.41</td>
<td>0.32</td>
<td>0.25</td>
<td>0.15</td>
<td>0.09</td>
<td>0.04</td>
<td>0.491</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>0.66</td>
<td>0.60</td>
<td>0.53</td>
<td>0.46</td>
<td>0.38</td>
<td>0.29</td>
<td>0.19</td>
<td>0.12</td>
<td>0.726</td>
</tr>
</tbody>
</table>

Table 4: Ablation experiment on Places365 with $\sigma = 0.5$. We perform the choice of $\lambda$ and $r_{max}$ as the hyper-parameters.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
<th>0.00</th>
<th>0.25</th>
<th>0.50</th>
<th>0.75</th>
<th>1.00</th>
<th>1.25</th>
<th>1.50</th>
<th>1.75</th>
<th>ACR</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda$</td>
<td>0.0</td>
<td>0.43</td>
<td>0.38</td>
<td>0.35</td>
<td>0.28</td>
<td>0.23</td>
<td>0.19</td>
<td>0.17</td>
<td>0.12</td>
<td>0.484</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>0.47</td>
<td>0.44</td>
<td>0.39</td>
<td>0.34</td>
<td>0.29</td>
<td>0.23</td>
<td>0.19</td>
<td>0.14</td>
<td>0.565</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>0.44</td>
<td>0.41</td>
<td>0.34</td>
<td>0.30</td>
<td>0.26</td>
<td>0.23</td>
<td>0.20</td>
<td>0.14</td>
<td>0.530</td>
</tr>
<tr>
<td>$r_{max}$</td>
<td>0.0</td>
<td>0.43</td>
<td>0.38</td>
<td>0.36</td>
<td>0.31</td>
<td>0.27</td>
<td>0.21</td>
<td>0.16</td>
<td>0.13</td>
<td>0.509</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>0.47</td>
<td>0.44</td>
<td>0.39</td>
<td>0.34</td>
<td>0.29</td>
<td>0.23</td>
<td>0.19</td>
<td>0.14</td>
<td>0.565</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>0.46</td>
<td>0.44</td>
<td>0.39</td>
<td>0.35</td>
<td>0.31</td>
<td>0.28</td>
<td>0.23</td>
<td>0.16</td>
<td>0.573</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>0.46</td>
<td>0.44</td>
<td>0.41</td>
<td>0.35</td>
<td>0.29</td>
<td>0.23</td>
<td>0.19</td>
<td>0.15</td>
<td>0.573</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>0.44</td>
<td>0.40</td>
<td>0.36</td>
<td>0.31</td>
<td>0.27</td>
<td>0.23</td>
<td>0.17</td>
<td>0.12</td>
<td>0.520</td>
</tr>
</tbody>
</table>

4.2. Training With Noisy Labels
As we discussed in Section 1, training a robust network has a side effect on smoothing the margin of the decision boundary, which enables training with noisy labels. **Problem statement:** Here, we consider a challenging noise setup called “pair flipping”, which can be described as follows. When noise rate is $p$ fraction, it means $p$ fraction of the $y_{ih}$ labels are flipped to the $(i+1)_{th}$. In this work, we test our method on the high noise rate 0.45. **Dataset:** The dataset we considered for this analysis is Cifar-10. To generate noisy labels from the clean labels of the dataset, we stochastically changed $p$ fraction of the labels using the source code provided by [18]. We perform a comparative analysis of our method with Bootstrap [38], S-model [15], Decoupling [32], MentorNet [25], Co-teaching [18], and Trunc $\mathbb{L}_q$ [56].

**Model hyperparameters:** Similar to training robust network with the clean labels, we first treat the noisy labels as “clean” to train our model. After 60 epochs, we remove the classification loss for the data with top 10% $C_R$ to fine-tune the network. The initial learning rate is set to 0.01 and decays at 30, 60, 90 epochs, respectively.

**Results:** The results are shown in Table 5, where, it is noticeable that even under this strong label corruption, our model outperforms most baseline results as well as stays stable over different epochs.

5. Conclusions
Developing mechanisms that enable training certifiably robust neural networks nicely complements the rapidly evolving body of literature on adversarial training. While certification schemes, in general, have typically been limited to small sized networks, recent proposals related to randomized smoothing have led to a significant expansion of the type of models where these ideas can be used. Our proposal here takes this line of work forward and shows that bound propagation ideas together with some meaningful approximations can provide an efficient method to maximize the certified radius – a measure of robustness of the model. We show that the strategy achieves competitive results to other baselines with faster training speed. We also investigate a potential use case for training with noisy labels where the behavior of such ideas has not been investigated, but appears to be promising.

Acknowledgments
Research supported in part by grant NIH RF1 AG059312, NIH RF1AG062336, NSF CCF #1918211 and NSF CAREER RI #1252725. Singh thanks Loris D’Antoni and Aws Albarghouthi for several discussions related to the work in [14].


