Statistically Robust Neural Network Classification (Supplementary material)

Benjie Wang[1] **Stefan Webb**[2] **Tom Rainforth**[3]

[1]Department of Computer Science, University of Oxford
[2]Twitter Cortex, San Francisco
[3]Department of Statistics, University of Oxford

A PROOFS

Recall the risk definitions:

$$r_{\mathcal{D}}^{\text{stat}}(f_\theta) \triangleq \mathbb{E}_{(X,Y)\sim p_{\mathcal{D}}}\left[\mathbb{E}_{X'\sim p(\cdot|X)}\left[\phi(f_\theta(X'),Y)\right]\right] \quad (6)$$

$$R_{N,C}^{\text{statMC}}(f_\theta) = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{C}\sum_{m=1}^{C}\phi(f_\theta(x'_{n,m}),y_n) \quad (9)$$

Theorem 0. *Suppose ϕ is bounded in $[0,c]$, and γ-Lipschitz in the first argument. For $m = 1,...,C$, define $S'_m = \{(x'_{1,m},y_1),...,(x'_{N,m},y_N)\}$. In other words, S'_m contains the mth perturbed point from each of the N original input points. For any $\delta \in (0,1)$, with probability at least $1-\delta$, the following holds for all $f \in \mathcal{F}$:*

$$\begin{aligned} &r_{\mathcal{D}}^{stat}(f) - R_{N,C}^{statMC}(f) \\ &\quad \leq 2c\gamma\overline{\text{Rad}_{S'}(\mathcal{F})} + 3c\sqrt{\log(2/\delta)/(2N)} \end{aligned}$$

where

$$\overline{\text{Rad}_{S'}(\mathcal{F})} \triangleq \frac{1}{C}\sum_{m=1}^{C}\text{Rad}_{S'_m}(\mathcal{F})$$

Proof. We can rewrite the SRR as a single expectation over (X',Y) using the law of total expectation:

$$\begin{aligned} r_{\mathcal{D}}^{\text{stat}}(f) &= \mathbb{E}_{(X,Y)\sim p_{\mathcal{D}},X'\sim p(\cdot|X)}\left[\phi(f_\theta(X'),Y)\right] \\ &= \mathbb{E}_{(X',Y)\sim q_{\mathcal{D}}}\left[\phi(f_\theta(X'),Y)\right] \end{aligned}$$

where $q_{\mathcal{D}}((X',Y)) \propto \int_{\mathcal{X}} p_{\mathcal{D}}(X,Y)p(X'|X)dX$.

For $C = 1$, we can apply the ERC generalization bound from Theorem 1 (on the distribution $q_{\mathcal{D}}$).

For $C > 1$ this is not directly possible because the $x'_{n,m}$ are not i.i.d. with respect to this distribution: for a fixed n, the $\{x'_{n,m} : m = 1,...,C\}$ are dependent as they come from the same point x_n.

In the general case, we will use the idea of ***independent blocks*** [Mohri and Rostamizadeh, 2008]. That is, the fact that while the variables within each block $\{x'_{n,m} : m = 1,...,C\}$ dependent, the blocks themselves are independent. To work at the block level, we need to rework our loss and risk definitions.

Recall the definition of the loss function class $L_{\mathcal{F}} = \{(X,Y) \to \phi(f(X),Y) : f \in \mathcal{F}\}$. Now we will define the "aggregate" loss function class $L_{\mathcal{F}}^C : \mathcal{X}^C \times \mathcal{Y} \to \mathbb{R}$ to include, for each function $l \in L_{\mathcal{F}}$, the following function:

$$l'((x_{n,1},...,x_{n,C}),y_n) = \frac{1}{C}\sum_{m=1}^{C} l(x_{n,m},y_n)$$

That is, functions in this class compute the average loss for a neural network in $\mathcal{F}$ on the C different points forming a block.

Note that if $L_{\mathcal{F}}$ is bounded in $[0,c]$, then so is $L_{\mathcal{F}}^C$. Further, notice that there is a 1-to-1 correspondence between NN functions $f \in \mathcal{F}$ and loss functions $l' \in L_{\mathcal{F}}^C$. Thus we can define the SRR and MC estimate of the SRR in terms of l': $R_{N,C}^{\text{statMC}}(l') = \frac{1}{N}\sum_{n=1}^{N} l'((x_{n,1},...,x_{n,C}),y_n)$, and $r_{\mathcal{D}}^{\text{stat}}(l') = \mathbb{E}\left[R_{N,C}^{\text{statMC}}(l')\right]$, and these are the same as for the corresponding f.

As noted by Mohri and Rostamizadeh [2008], by viewing each block as an i.i.d. point and applying McDiarmid's inequality, the excess risk has the standard probabilistic bound. That is, with probability at least $1-\delta$, for all $l' \in L_{\mathcal{F}}^C$:

$$\begin{aligned} &r_{\mathcal{D}}^{\text{stat}}(l') - R_{N,C}^{\text{statMC}}(l') \\ &\quad \leq 2c\text{Rad}_N(L_{\mathcal{F}}^C) + c\sqrt{\log(1/\delta)/(2N)} \end{aligned}$$

where $\text{Rad}_N(L_{\mathcal{F}}^C)$ is the (non-empirical) Rademacher complexity of $L_{\mathcal{F}}^C$ over N inputs.

Once again applying McDiarmid's inequality in the standard way, we can convert to a bound involving the empirical Rademacher complexity over the sample $S' =$

$\{((x_{1,1},...,x_{1,C}),y_1),...,((x_{N,1},...,x_{N,C}),y_N)\}$:

$$r_{\mathcal{D}}^{\text{stat}}(l') - R_{N,C}^{\text{statMC}}(l') \leq 2c\text{Rad}_{S'}(L_{\mathcal{F}}^C) + 3c\sqrt{\log(2/\delta)/(2N)}$$

It remains to determine $\text{Rad}_{S'}(L_{\mathcal{F}}^C)$.

Lemma 1. $\text{Rad}_{S'}(L_{\mathcal{F}}^C) \leq \frac{1}{C}\sum_{m=1}^C \text{Rad}_{S'_m}(L_{\mathcal{F}})$

Proof. Consider the following function class $U_{\mathcal{F}}^C : \mathcal{X}^C \times \mathcal{Y} \to \mathbb{R}$:

$$U_{\mathcal{F}}^C((x_{n,1},...,x_{n,C}),y_n) = \{\frac{1}{C}\sum_{m=1}^C l_m(x_{n,m},y_n) : l_1,...,l_C \in L_{\mathcal{F}}\}$$

Since $U_{\mathcal{F}}^C$ is a linear combination of C classes $L_{\mathcal{F}}$ divided by C, its ERC is given by $\text{Rad}_{S'}(U_{\mathcal{F}}^C) = \frac{1}{C}\sum_{m=1}^C \text{Rad}_{S'_m}(L_{\mathcal{F}})$. This can be seen as follows:

$$\begin{aligned}\text{Rad}_{S'}(U_{\mathcal{F}}^C) &= \frac{1}{N}\mathbb{E}_\sigma\left[\sup_{u\in U_{\mathcal{F}}}\sum_{n=1}^N \sigma_n u((x_{n,1},...,x_{n,C}),y_n)\right] \\ &= \frac{1}{N}\mathbb{E}_\sigma\left[\sup_{l_1,...,l_C\in L_{\mathcal{F}}}\sum_{n=1}^N \sigma_n \frac{1}{C}\sum_{m=1}^C l_m(x_{n,m},y_n)\right] \\ &= \frac{1}{N}\mathbb{E}_\sigma\left[\sup_{l_m\in L_{\mathcal{F}}}\frac{1}{C}\sum_{m=1}^C\left(\sum_{n=1}^N \sigma_n l_m(x_{n,m},y_n)\right)\right] \\ &= \frac{1}{C}\sum_{m=1}^C\frac{1}{N}\mathbb{E}_\sigma\left[\sup_{l\in L_{\mathcal{F}}}\sum_{n=1}^N \sigma_n l(x_{n,m},y_n)\right] \\ &= \frac{1}{C}\sum_{m=1}^C \text{Rad}_{S'_m}(L_{\mathcal{F}})\end{aligned}$$

Further, since $L_{\mathcal{F}}^C$ is a subset of $U_{\mathcal{F}}^C$ (the former is $U_{\mathcal{F}}^C$ with the restriction that all the constituent loss functions are the same), it has ERC at most $\text{Rad}_{S'}(U_{\mathcal{F}}^C)$. □

The result follows using the Ledoux-Talagrand contraction lemma to bound the ERC of the loss function class in terms of the ERC of the neural network function class.

□

B EFFECT OF STATISTICALLY ROBUST TRAINING

In Section 3.1, we discussed one of the major shortfalls of adversarial risk: namely, that it loses information about how robust points are. We now demonstrate that this can be seen in practice with trained networks.

We analyse networks trained on CIFAR-10 either using corruption training (uniform over $\epsilon = 0.157$ L_∞ ball) or PGD/adversarial training (over ϵ L_∞ ball). As we saw in Section 6.2 and Table 1, the former attains 92.4% on the TSRM metric, while the latter attains 88.1%. We thus wish to examine to what the additional robust performance of the corruption trained network is attributable to.

Recall that the pointwise statistical robustness of a point (x,y) can be defined as $PSR(x,y) = \mathbb{E}_{X'\sim p(\cdot|x)}\left[\mathbb{1}_{f_\theta(x')\neq y}\right]$ (note that, as throughout this paper, we have swtiched to using the CI rather than PC definition of adversarial examples). This expresses how robust the point is; if this is 0, the network almost always predicts incorrectly, while if it is 1, it almost always predicts correctly. In Figure 1, we plot the empirical CDF of $PSR(x,y)$ over the 10000 points in the CIFAR-10 dataset, for both trained networks.

We see that both networks have $PSR(x,y) = 1$ for the majority of points (more than 80%). The main difference, then, is in that fact that the corruption trained network has a large number of points with $PSR(x,y)$ between 0 and 1. In contrast, the adversarially trained network is very polarized, with the vast majority of points having $PSR(x,y) = 0$ or 1. That is to say, the former achieves better TSRM not by being fully robust on more points, but rather by ensuring as many points as possible have *some* robustness. Returning to the medical imaging analogy in Section 3.1, this means correct diagnoses for more patients, under random noise corruptions.

C ESTIMATION AND TRAINING

In Section 3.4, we discussed a simple Monte Carlo estimation/training scheme, which involves sampling C perturbed points around each input:

$$R_{N,C}^{\text{statMC}}(f_\theta) = \frac{1}{N}\sum_{n=1}^N\frac{1}{C}\sum_{m=1}^C \phi(f_\theta(x'_{n,m}),y_n) \quad (9)$$

C.1 EFFECT OF C

As we justified in Section 3.4, taking $C = 1$ is, somewhat surprisingly, sufficient to estimate the SRR accurately, meaning that we can evaluate test loss/accuracy for a network by simply perturbing each sample once. We found that this also holds true for training: rather than taking multiple samples around each datapoint, as is often done in e.g. Gaussian data augmentation, we find that perturbing each sample once doing training is sufficient, achieving similar performance to taking more samples (e.g. $C = 5$) while reducing training time to the same as standard training.

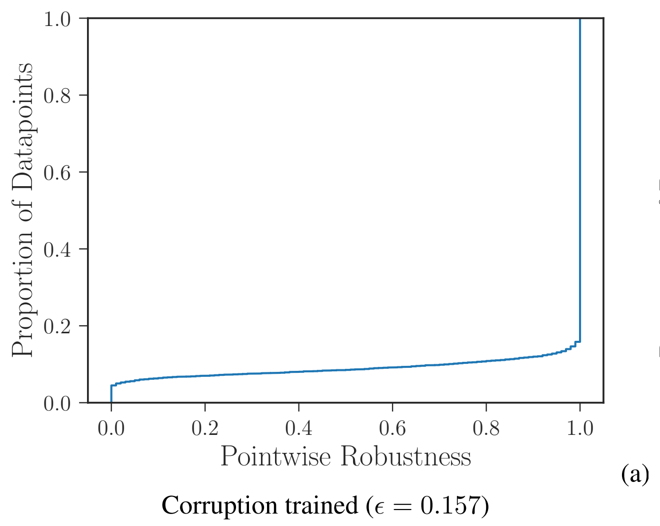

Figure 1: Empirical cumulative distribution function (ECDF) for the pointwise statistical robustness (CI definition), over all 10000 images in the CIFAR-10 test dataset.

C.2 ALTERNATIVE METHODS

Nonetheless, other schemes may sometimes be more efficient in different situations, such as when our dataset is small, or when the number of "non-robust" points is small.

For example, suppose that we wish to evaluate the TSRM ($\phi(f_\theta(X'), Y) = \mathbb{1}_{f_\theta(X') \neq Y}$) given a relatively small number of datapoints, say $N = 100$. This can be achieved using our previous Monte Carlo method:

$$R^{\text{statMC}}_{100,C}(f_\theta) = \frac{1}{100}\sum_{n=1}^{N}\frac{1}{C}\sum_{m=1}^{C}\phi(f_\theta(x'_{n,m}), y_n) \qquad (9)$$

However, since N is much smaller, the law of large numbers over N (datapoints) no longer applies and we must estimate the robustness of each point precisely:

$$L^{\text{stat}}(X, Y, f) \triangleq \mathbb{E}_{p(X'|X)}\left[\phi(f_\theta(X'), Y)\right] \qquad (7)$$

For TSRM, $0 \leq L^{\text{stat}}(x_n, y_n, f) \leq 1$ for all n. However, many such points will have $L^{\text{stat}}(x_n, y_n, f)$ close to 0 or 1 (i.e. almost all of the perturbation region is classified correctly, or almost all is classified incorrectly), in which case sampling from $p(\cdot|x_n)$ to estimate this quantity will have very low variance. Thus it can be more efficient not to uniformly increase the number of samples for each point x_n by increasing C in the Monte Carlo estimator, but to adaptively select which points to sample from based on the variance observed thus far.

This is a stratified sampling problem which has connections to multi-armed bandits, where the ultimate goal is to produce a low-variance estimate of $r^{\text{stat}}(p, f_\theta)$. Carpentier et al. [2015] suggested an algorithm (***adaptive stratified sampling***) for this purpose. In Fig 2 we investigated the variance of adaptive stratified sampling for estimating TSRM, and found that it was somewhat lower given the same number of epochs (i.e. same number of forward passes using the network) compared to Monte Carlo. For larger datasets, however, adaptive sampling provides no noticeable advantage, which is why we employ Monte Carlo in our other experiments.

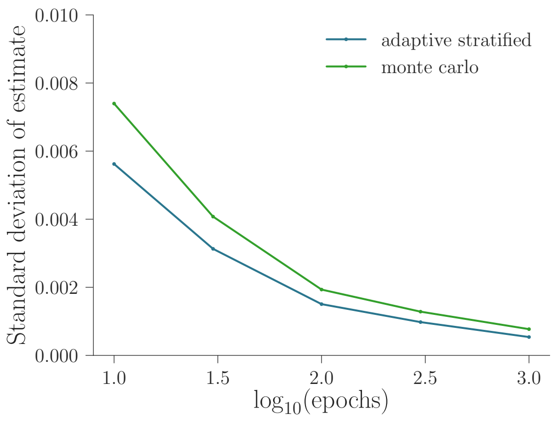

Figure 2: Standard deviation over 100 runs of the adaptive stratified and Monte Carlo algorithms for estimating TSRM. The horizontal axis represents the number of forward passes allowed.

Alternative methods for training networks with respect to the SRR are also possible, and also could potentially be beneficial particularly when we have limited data. To this end we applied an importance sampling scheme to the setting in Experiment 6.3, but found that this did not provide noticeable improvements. Investigation of adaptive training methods, perhaps borrowing ideas from active learning, could comprise interesting avenues for future work.

D COMPARISON TO RANDOMIZED SMOOTHING

Randomized smoothing is a recently proposed technique that applies post-hoc to a classifier in order to improve its adversarial robustness. Given a base classifier f, a new "smoothed classifier" g is defined as follows:

$$g(x) \triangleq \arg\max_c \mathbb{P}_{X' \sim p(\cdot|x)}(f(X') = c) \qquad (1)$$

Here, $p(\cdot|x)$ is some perturbation distribution, usually taken to be additive Gaussian noise, i.e. $X' \sim N(x, \sigma^2 I)$. For a given input point x, g picks the class c which is predicted most often by f in the perturbation region. In practice, since it is not possible to evaluate predictions over the whole distribution, we sample from the distribution, and choose the class which is predicted most frequently (majority vote). This naturally leads to more smooth/invariant predictions, and in fact it is possible to directly obtain certified adversarial robustness (accuracy) guarantees in l_2 norm for each point: that is, a radius R_x such that:

$$g(x_{\text{pert}}) = g(x) \qquad \forall x_{\text{pert}} \text{ s.t. } ||x_{\text{pert}} - x||_2 < R_x$$

The size of this radius R_x depends on both the probabilities of the most likely classes predicted by f in $N(x, \sigma^2 I)$, as well as the standard deviation σ of the Gaussian distribution. Intuitively, if x_{pert} is sufficiently close to x, then the distributions $N(x_{\text{pert}}, \sigma^2 I)$ and $N(x, \sigma^2 I)$ overlap sufficiently that the most likely predictions of f on both are the same. There exists a tradeoff between smoothness and precision here: increasing σ increases the certified adversarial radius, but can also make the classifier "too smooth" and lose standard accuracy. It can be seen that in the limit $\sigma \to \infty$, g will simply predict the same class for all input points x.

At first glance, the smoothing operation (1) appears to resemble the pointwise statistical robustness metric from Section 2.3:

$$\mathcal{I}[p] \triangleq \mathbb{E}_{X' \sim p(\cdot|x)}\left[\mathbb{1}_{f(X') \neq f(x)}\right] \qquad (3)$$

as both involve probabilistic perturbations around point x. However, notice that the former is an operation, which produces a new classifier, while the latter is a metric for the original classifier f. In particular, the smoothing $f \to g$ does not directly minimize or target the statistical robustness metric in any meaningful way. That said, we could of course apply the metric to g, which would involve a "double averaging". The inner component of (3) would be $\mathbb{1}_{g(X') \neq g(x)}$, which tests whether the smoothed classifier classifies X', x the same, or, in other words, whether f agrees sufficiently on $N(X', \sigma^2 I)$ and $N(x, \sigma^2 I)$. For similar reasons to the adversarial robustness guarantees above, we would expect the pointwise statistical robustness metric to be small on g (in fact, 0 if $p(\cdot|x)$ is chosen such that its support lies entirely within the l_2 ball of radius R_x around x).

We now consider our total statistical robustness metric defined in Section 3.2, which extends to the whole data distribution $p_\mathcal{D}$. Recall that, crucially, when moving to the TSRM from the pointwise metric, we changed to the "corrupted instance" (CI) definition of adversarial examples; that is, we compare the perturbed prediction to the true class label, rather than the original prediction:

$$\mathcal{I}_{\text{total}}[p] = \mathbb{E}_{(X,Y)\sim p_\mathcal{D}}\left[\mathbb{E}_{X' \sim p(\cdot|X)}\left[\mathbb{1}_{f(X') \neq Y}\right]\right], \qquad (5)$$

The CI definition requires that the classifier is not just "smooth" in the sense of producing the same prediction at nearby points, but also that this prediction matches the true label Y. Now consider applying randomised smoothing to the base classifier f, in addition to or in lieu of SRR training. As before, the randomized smoothed classifier g does not directly target TSRM. However, unlike the pointwise metric, we would not necessarily even expect g to obtain a better TSRM than f. This is because g makes f smoother, but this can come at the cost of making the correct prediction.

We empirically tested applying randomized smoothing to our classifiers, and as we expected, this provided no benefit in terms of SRR/TSRM, usually performing slightly worse on these metrics. The smoothed classifiers did exhibit greater adversarial robustness, as randomized smoothing is designed to achieve. Thus, while randomized smoothing makes use of probabilistic perturbations, it does so in a very different way to SRR training.

E IMPLEMENTATION DETAILS

All experiments were performed using Python 3 with the PyTorch framework, using a single NVIDIA Tesla T4 GPU.

In Experiment 6.1, we used the MNIST dataset with the standard train/test split (60000/10000), with pixels scaled to $[0, 1]$. We used a dense ReLU network architecture with an input layer of size 784, a hidden layer of size 256, and an output layer of size 10. Training was performed for 50 epochs, each time using default initialization and with the Adam optimizer with learning rate $1\text{e}{-3}$.

In Experiment 6.2, we used the CIFAR-10 dataset with the standard train/test split (50000/10000), with pixels normalized by per-channel mean and standard deviation. We use a wide residual network architecture [Zagoruyko and Komodakis, 2016] with depth 28, widening factor 10, and dropout rate 0.3. Training was performed using SGD with a staggered learning rate, starting at 0.01 and ending at 0.0004. We report the train/test scores after 30 epochs (averaged over 5 runs) in Table 1. Adversarial training was applied using

7-step PGD to find the most adversarial perturbation for each training point; this took significantly longer (approx. 6 times) compared to natural or corruption training.

In Experiment 6.3, we again used the MNIST dataset with the same train/test split and the same network architecture. However, we instead trained for 1000 epochs in each run, and used a learning rate of $5\mathrm{e}{-5}$ for the Adam optimizer.

References

Alexandra Carpentier, Remi Munos, and András Antos. Adaptive strategy for stratified monte carlo sampling. *Journal of Machine Learning Research*, 16(68):2231–2271, 2015.

Mehryar Mohri and Afshin Rostamizadeh. Rademacher complexity bounds for non-i.i.d. processes. In *Proceedings of the 21st International Conference on Neural Information Processing Systems*, page 1097–1104, 2008.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2016.