Probabilistic Robustness Estimates for Feed-forward Neural Networks

Nicolas Couellan

Probabilistic Robustness Estimates for Feed-forward Neural Networks

Nicolas Couellan*

ENAC, Université de Toulouse, 7 Avenue Edouard Belin, 31400 Toulouse, France

Institut de Mathématiques de Toulouse, Université de Toulouse, UPS IMT, F-31062 Toulouse Cedex 9, France

April, 2021

Abstract

Robustness of deep neural networks is a critical issue in practical applications. In the general case of feed-forward neural networks (including convolutional deep neural network architectures), under random noise attacks, we propose to study the probability that the output of the network deviates from its nominal value by a given threshold. We derive a simple concentration inequality for the propagation of the input uncertainty through the network using the Cramer-Chernoff method and estimates of the local variation of the neural network mapping computed at the training points. We further discuss and exploit the resulting condition on the network to regularize the loss function during training. Finally, we assess the proposed tail probability estimates empirically on various public datasets and show that the observed robustness is very well estimated by the proposed method.

1 Introduction

Deep neural networks have proven to be very effective in practice to perform highly complex learning tasks [11]. Due to this success, they have gained a great deal of attention these past few years and they have been applied widely. However, they also have been found to be very sensitive to data uncertainties [9, 23] to the point that a whole research community is now addressing the so-called network attacks in order to study and design input noise that can fool the network decision. Attacks can be random when data are corrupted by some

*nicolas.couellan@recherche.enac.fr

random noise or adversarial when the noise is specifically designed to alter the network output [23]. Even though both types of attacks are related since they are both addressing the robustness of the network, we will only focus in this article on the random case. Most data are usually uncertain, either because the data are related to naturally noisy phenomenon and we only have access to some of its statistics or because assessing devices to do not have sufficient accuracy to record precisely the data. In this study, we will therefore assume that the network input data are corrupted by some additive bounded random noise.

Robustness to bounded input perturbations has been analyzed in the past few years. Most people have addressed the problem through the use of regularization techniques [12, 19, 26, 10]. The main idea is to consider the neural network as a Lipschitz map between the input and output data. The Lipschitz constant of the network is then estimated or upper bounded by the norm of the layer-by-layer weights product. This estimates the expansion or contraction capability of the network and is then used to regularize the loss during training. Often, there is a price to pay: the expressiveness of the network may be reduced, especially if the weights are too constrained or constrained layer by layer instead of constrained accross layer [7]. Such strategies are enforcing robustness but do not provide guarantees or estimates on the level of robustness that has been achieved. In the case of adversarial perturbation, some authors have proposed methods for certifying robustness [15, 4]. Recently, a probabilistic approach has also been proposed in the case of random noise for convolutional neural networks [27]. Pointing out that the threat of random noise may have been overlooked by the research community in favor of adversarial attacks, the authors have proposed probabilistic bounds based on the idea that the output of the network can be lower and upper bounded by two linear functions. The work proposed here is along the same line but distinct in several aspects. It combines upper bounds on tail probabilities calculated by deriving a specific Cramer-Chernoff concentration inequality for the propagation of uncertainty through the network with a network sensitivity estimate based on a network gradient calculation with respect to the inputs. The network gradient is computed by automatic differentiation and estimates the local variation of the output with respect to the input of the network. The estimation is carried out and averaged over the complete training set. A maximum component-wise gradient variation is also calculated in order to give probabilistic certificates rather than estimates. The certificates can be used in place of estimates whenever guaranteed upper bounds are needed, however they are often not as accurate since they are based on variation bounds rather than averages. For the specific case of piece-wise linear activation functions, we also propose an alternative bound based on the calculation of an average activation operator matrix computed at each layer using also the training samples. We then discuss the use of the derived bounds and estimate to regularize the neural network during training in order to reach regions of the weight space that ensure greater robustness properties. We then design experiments in order to assess the robustness probabilistic estimates for various regularization strategies.

The article is organized as follows: Section 2 provides the specific neural network concentration inequality using the Cramer-Chernoff method and the calculation of the network gradient estimate and the average activation operator for the case of piece-wise linear activations. Section 4 deals with training of the neural network and its regularization issues to increase its robustness. Section 5 provides the results of an empirical evaluation of the neural network robustness for various public datasets. Finally, Section 6 concludes the article.

2 Probabilistic certificates of robustness

Consider feed-forward neural networks that we represent as a successive composition of linear weighted combination of functions such that $x^l = f^l((W^l)^\top x^{l-1} + b^l)$ for $l = 1, \dots, L$, where $x^{l-1} \in \mathbb{R}^{n_{l-1}}$ is the input of the l-th layer, the function f^l is the L_f^l-Lipschitz continuous activation function at layer l, and $W^l \in \mathbb{R}^{n_{l-1} \times n_l}$ and $b^l \in \mathbb{R}^{n_l}$ are the weight matrix and bias vector between layer $l-1$ and l that define our model parameter $\theta = \{W^l, b^l\}_{l=1}^L$ that we want to estimate during training. The network can be seen as the mapping $g_\theta : x^0 \to g_\theta(x^0) = x^L$. The training phase of the network can be written as the minimization of the empirical loss $\mathcal{L}(x, y, \theta) = \frac{1}{n} \sum_{i=1}^n l_\theta(g_\theta(x_i), y_i)$ where l_θ is a measure of discrepancy between the network output and the desired output.

Assume now that we only have access to noisy observations x_i of the input sample. However, we know that these observations are drawn from a distribution D with finite support. We first consider the special case where the functions f^l are linear or piece-wise linear (this includes the case of ReLu activation functions) and then extend the analysis to more general functions. We start by considering the one layer neural network setting to further extend the idea to several layers.

2.1 The single layer architecture

In this section, we consider the simple case where the outputs of the network $y = x^L$ (with $L = 1$ in this case) depends linearly of the inputs $x = x^0 \in \mathbb{R}^n$ as follows: $w^\top x + b$ where $w = W^1$ is the single layer weights vector and $b = b^1$ is the layer bias.
We assume that our input observations are corrupted by some additive noise ϵ such that $\epsilon \sim D$ and $\forall i = 1, \dots, n$, we have $\epsilon_i \in [-\gamma, \gamma]$ with $\gamma < +\infty$. Our objective is to ensure the following property:

$$\mathbb{P}_{\epsilon \sim D} \left(\|y - y_\epsilon\| \leq \Gamma \right) \geq 1 - \alpha \tag{1}$$

where $y_\epsilon = w^\top (x + \epsilon) + b$, Γ is the allowed output uncertainty and $1 - \alpha$ is some predefined level of confidence.

In the following proposition, we give a condition on w that will ensure, with probability greater than $1 - \alpha$, that the output uncertainty remains below Γ.

Proposition 2.1 *If the network inputs are subject to an additive uncertainty* ϵ *where* $\forall i = 1, \ldots, n$, $\epsilon_i \sim D$ *and* $\text{supp}(D) \subset [-\gamma, \gamma]$ $(\gamma < +\infty)$*, then for a given* $\alpha \in (0, 1]$ *and a given output uncertainty level* Γ*, the following condition holds:*

If the layer weights vector w *satisfies* $\|ww^\top\|_F \leq \frac{\Gamma^2}{\gamma^2\sqrt{2\log(1/\alpha)}}$*, then* $\mathbb{P}_{\epsilon\sim D}\left(\|y - y_\epsilon\| \leq \Gamma\right) \geq 1 - \alpha$.

Proof: We start by observing that in general we have

$$\mathbb{P}_{\epsilon\sim D}\left(\|y - y^\epsilon\| \leq \Gamma\right) \geq \mathbb{P}_{\epsilon\sim D}\left(tr\left(w^\top\epsilon(w^\top\epsilon)^\top\right) \leq \Gamma^2\right), \tag{2}$$

where $tr(A)$ defines the trace of a matrix A. Note that in the particular case where all activation functions are linear, equality is achieved.
Therefore inequality (1) may be satisfied by ensuring that

$$\mathbb{P}_{\epsilon\sim D}\left(tr(w^\top\epsilon(w^\top\epsilon)^\top) \geq \Gamma^2\right) = \mathbb{P}_{\epsilon\sim D}\left(tr(\epsilon\epsilon^\top ww^\top) \geq \Gamma^2\right) \leq \alpha.$$

We now state and prove the following Lemma which provides a simple concentration inequality for the above probability.

Lemma 2.2 *For any random matrix* $M \in \mathbb{R}^{n\times n}$ *of the form* $M = vv^\top$ *where* v *is a random vector such that for all* i *in* $\{1, \ldots, n\}$*,* v_i *are all independent, have finite support included in* $[-\delta, \delta]$ *and* $\mathbb{E}(v_i) = 0$*, we have*

$$\forall Q \in \mathbb{R}^{n\times n} \setminus \{0_n\}, \forall t > 0, \quad \mathbb{P}\left(tr(MQ) \geq t\right) \leq e^{-\frac{t^2}{2\beta^2\delta^4\|Q\|_F^2}}. \tag{3}$$

where 0_n *is the* $n \times n$ *zero matrix and* $\beta = \min\{\rho > 0 \ \ s.t. \ \ \forall z \in \mathbb{R}, \frac{(\rho z)^2}{2} - \log(cosh(z)) \geq 0\}$.

Proof: The proof is based on the Cramer-Chernoff method [5] to bound the tail probability of the random variable $tr(MQ)$. Applying Markov inequality to the left hand side of (3), we have:

$$\mathbb{P}\left(\text{tr}(MQ) \geq t\right) \leq \frac{\mathbb{E}(\phi(\text{tr}(MQ)))}{\phi(t)}$$

for any positive and non decreasing function ϕ. Therefore, since $m_{ij} = v_iv_j = m_{ji}$, for any $p \in \mathbb{R}^+$, we have

$$\begin{aligned}
\mathbb{P}\left(tr(MQ) \geq t\right) &\leq e^{-pt}\mathbb{E}\left(e^{p\,tr(MQ)}\right) \\
&\leq e^{-pt}\mathbb{E}\left(e^{p\sum_{i,j=1}^n m_{ij}q_{ji}}\right) \\
&\leq e^{-pt}\mathbb{E}\left(\left(\prod_{i=1}^n e^{pm_{ii}q_{ii}}\right)\left(\prod_{i,j=1:i<j}^n e^{2pm_{ij}q_{ji}}\right)\right). \qquad (4)
\end{aligned}$$

Observe now that since for all $i \neq j$, we have that $\text{cov}(m_{ii}, m_{jj})$=0 and since $\mathbb{E}(v_i) = \mathbb{E}(v_j) = 0$ that

$$\text{cov}(m_{ii}, m_{ij}) = \text{cov}(v_iv_i, v_iv_j) = 0.$$

Therefore, from (4), we have

$$\mathbb{P}\left(tr(MQ) \geq t\right) \leq e^{-pt} \prod_{i,j=1}^{n} \mathbb{E}\left(e^{pm_{ij}q_{ji}}\right).$$

Let now $\psi_{\mathrm{tr}(MQ)}$ be the moment generating function of $\mathrm{tr}(MQ)$, its Cramer transform obtained by Fenchel-Legendre duality is

$$\psi^*_{\mathrm{tr}(MQ)}(t) = \sup_{p \geq 0}(pt - \psi_{\mathrm{tr}(MQ)}(p)).$$

Note that $m_{ij} \in [-\delta^2, \delta^2]$ and that by convexity of the exponential function, we can bound the moment generating function of m_{ij} as follows:

$$\forall z \in \mathbb{R}, \quad \mathbb{E}(e^{zm_{ij}}) \leq \frac{1}{2}e^{-z\delta^2} + \frac{1}{2}e^{z\delta^2} = cosh(z\delta^2).$$

Therefore, around zero, we can write

$$\log(\mathbb{E}(zm_{ij})) \leq \log(cosh(z\delta^2)) \leq \frac{(\beta z\delta^2)^2}{2}$$

where β is a coefficient in $(0, 1]$ that tightens the bound as much as possible and can be defined as $\beta = \min\{\rho > 0 \,\text{s.t.}\, \frac{(\rho z)^2}{2} - \log(cosh(z)) \geq 0\}$. Replacing z by pq_{ji} with $p > 0$, we can derive the following upper bound for the Cramer transform of $\psi^*_{\mathrm{tr}(MQ)}$:

$$\forall t \in \mathbb{R}, \quad \psi^*_{\mathrm{tr}(MQ)}(t) \leq \min_{p>0} \left\{ -pt + \sum_{i,j=1}^{n} \frac{(pq_{ji}\beta\delta^2)^2}{2} \right\}.$$

The minimum in the right hand side of the expression above is reached at $p^* = \frac{t}{\beta^2\delta^4\|Q\|_F^2}$ and therefore, applying Cramer inequality, we finally get

$$\log(\mathbb{P}\left(\mathrm{tr}(MQ) \geq t\right)) \leq -\frac{t^2}{2\beta^2\delta^4\|Q\|_F^2},$$

which completes the proof of the lemma. □

Note that $\forall (i, j) \in \mathbb{R}^{n\times n}$, we have $(\epsilon\epsilon^\top)_{ij} \in [-\gamma^2, \gamma^2]$. Hence, applying Lemma 2.2 to bound $\mathbb{P}_{\epsilon\sim D}\left(\mathrm{tr}(\epsilon\epsilon^\top ww^\top) \geq \Gamma^2\right)$ will lead to

$$\mathbb{P}_{\epsilon\sim D}\left(\mathrm{tr}(\epsilon\epsilon^\top ww^\top) \geq \Gamma^2\right) \leq e^{-\frac{\Gamma^4}{2\beta^2\gamma^4\|ww^\top\|_F^2}} \leq e^{-\frac{\Gamma^4}{2\gamma^4\|ww^\top\|_F^2}},$$

since $\beta \in (0, 1]$. In order to remain below the level α, we then need that

$$-\frac{\Gamma^4}{2\gamma^4\|ww^\top\|_F^2} \leq \log(\alpha). \tag{5}$$

This can also be written as $\|ww^\top\| \leq \frac{\Gamma^2}{\gamma^2\sqrt{2\log(1/\alpha)}}$, proving Proposition 2.1. □

Note that in (5), one can keep the tightening coefficient β to get a sharper bound whenever needed and write $\|ww^\top\| \leq \frac{\Gamma^2}{\beta\gamma^2\sqrt{2\log(1/\alpha)}}$. The value of β can be estimated numerically.

2.2 Extension to deeper architectures

In this section, we address the case where the network is composed of several layers $l = 1, \ldots, L$ with L_f^l-Lipschitz activation functions. Property (1) should now relate to the output of the last layer as follows:

$$\mathbb{P}_{\epsilon\sim D}\left(\|x^L - x_\epsilon^L\| \leq \Gamma\right) \geq 1 - \alpha \tag{6}$$

where x_ϵ^L is the output of the layer L when a noisy input $x_\epsilon^0 = x^0 + \epsilon$ is propagated through the network and ϵ is the additive input noise such that $\epsilon \sim D$ and $\forall i = 1, \ldots, n$, we have $\epsilon_i \in [-\gamma, \gamma]$ ($\gamma < +\infty$). With this setting, we can now state and prove the following:

Proposition 2.3 *If the network inputs are subject to an additive uncertainty ϵ where $\forall i = 1, \ldots, n$, $\epsilon_i \sim D$ and $\mathrm{supp}(D) \subset [-\gamma, \gamma]$ ($\gamma < +\infty$), then for a given $\alpha \in [0, 1]$ and a given output uncertainty level Γ, the following condition holds:*

If the layer weights vector w satisfies $\left\|\left(\prod_{l=1}^L W^l\right)\left(\prod_{l=1}^L W^l\right)^\top\right\|_F \leq \frac{\Gamma^2}{\gamma^2 L_f^2\sqrt{2\log(1/\alpha)}}$ then $\mathbb{P}_{\epsilon\sim D}\left(\|x^L - x_\epsilon^L\| \leq \Gamma\right) \geq 1 - \alpha$ with $L_f = \left(\prod_{l=1}^L L_f^l\right)$.

Proof: Propagating forward the input uncertainties through the network, we can write:

$$\|x^L - x_\epsilon^L\| \leq L_f^L\|(W^L)^\top\epsilon^{L-1}\| \leq L_f\left\|\left(\prod_{l=1}^L W^l\right)^\top\epsilon\right\|$$

where ϵ^l is the propagated noise from layer 1 to layer l and $L_f = \left(\prod_{l=1}^L L_f^l\right)$. Furthermore, we have

$$\begin{aligned}
\|x^L - x_\epsilon^L\|^2 &\leq L_f^2\ \mathrm{tr}\left(\left(\left(\prod_{l=1}^L W^l\right)^\top\epsilon\right)\left(\left(\prod_{l=1}^L W^l\right)^\top\epsilon\right)^\top\right)\\
&\leq L_f^2\ \mathrm{tr}\left(\left(\prod_{l=1}^L W^l\right)^\top\epsilon\epsilon^\top\left(\prod_{l=1}^L W^l\right)\right)\\
&\leq L_f^2\ \mathrm{tr}\left(\epsilon\epsilon^\top\left(\prod_{l=1}^L W^l\right)\left(\prod_{l=1}^L W^l\right)^\top\right).
\end{aligned}$$

Therefore, applying again Lemma 2.2 to upper bound the probability in (6), the following condition on the layer weights matrices to ensure property (6) is directly obtained:

$$\left\| \left(\prod_{l=1}^{L} W^l \right) \left(\prod_{l=1}^{L} W^l \right)^\top \right\|_F \leq \frac{\Gamma^2}{\gamma^2 L_f^2 \sqrt{2\log(1/\alpha)}}. \tag{7}$$

□

2.3 The case of convolutional layers

Deep neural networks often involved other types of layer structures such as convolutional layers where connections within the layer are sparse [11]. Sparsity arise from the fact that weights of convolutional filters are shared across neurons within the layer when applied to the layer input. However, this structure is not very different from the case of fully connected networks. Indeed, a convolutional layer performs discrete convolutions with several filters. This can be seen as a weight matrix multiplication operation where the weight matrix is very sparse due to parameter sharing and sparse connectivity induced by the small size of filters usually used in practical applications. Therefore, the output of a l-th convolutional layer can also be written as $x^l = W^l x^{l-1} + b^l$ where x^{l-1} is a flattening transformation of a tensor X^{l-1} of feature maps (transformation of tensor into vector) and b^l is also a flattened version of the bias of the l-th layer (see [11, 12] for more details about linearity of convolutional layers). In this specific case, the weight matrix W^l has a Toeplitz structure. This, in general, helps when Toeplitz matrices are to be multiplied, since this can be done in $\mathcal{O}(n^2)$ if the matrices belong to $\mathbb{R}^{n\times n}$. The Toeplitz property is not preserved by matrix multiplication which renders this result of little help if the network involve more than two convolutional layers or if layers are combined with other types of layers as discussed next. However, in [29, 30], the specific role of the Toeplitz structure of W^l has been emphasized and exploited to prove that CNN have universal approximation properties. More specifically, following a similar analysis of the sparsity and redundancy in W^l may actually help in deriving lower bounds for $\left\| \left(\prod_{l=1}^{L} W^l \right) \left(\prod_{l=1}^{L} W^l \right)^\top \right\|_F$ and eventually tighter bounds for (3).

In convolutional architecture, usually convolution is combined with other types of structures. Activation layers (ex: ReLu activation) provide non linearity to the network and, from the point of view of our study, these can be treated by integrating their Lipschitz constant L_f^l (which in many practical cases is 1) in the calculation of the bounds. Pooling layers are also used to reduce the dimension of feature maps by summarizing extracted information and can also be viewed as an operator on the output of the convolutional layer that has a Lipschitz constant smaller than 1 [12].

Therefore, from the point of view of noise contraction or expansion developed above, combining the various types of layers usually found in CNN architecture,

is another particular case of the general framework we have considered above. Hence, the techniques developed in the following fully apply to deep network architectures and the conclusions that are drawn in the sequel of the article are also valid.

3 Improving the bound by taking the network non-linearity into account

3.1 General neural network activations

Remember that the bound derived above relies on the fact that we have considered the linear upper bound of the neural network response. Therefore, the inequality (2) applied to the multi-layers case gives

$$\mathbb{P}_{\epsilon\sim D}\left(\|x^L - x^L_\epsilon\| \leq \Gamma\right) \geq \mathbb{P}_{\epsilon\sim D}\left[tr\left(\epsilon\epsilon^\top \left(\prod_{l=1}^{L} W^l\right)\left(\prod_{l=1}^{L} W^l\right)^\top\right) \leq \frac{\Gamma^2}{L_f^2}\right] \quad (8)$$

Even if L_f^l is known for all levels (ex: $L_f^l = 1$ for all l if all network activations are Relu), their product L_f may be a loose bound for the network Lipschitz constant. This means that the Chernoff bound proposed above may be tight with respect to the right hand side of the above inequality, however, with respect to the left hand side, in general, (8) is not tight since the deep complex layer structure of the neural network generates a highly non linear behavior where some neurons are activated and some are not in a complex manner. Actually, it has been proven that computed the actual neural network Lipschitz constant is an NP-hard problem [26]. Therefore, the above bound as such may often be of poor quality. To address this issue, we propose an alternative to estimate the variation of the neural network response. In the following, we will consider that the network output is 1-dimensional. The following developments also generalize to the multi-dimensional output case by using differential calculus for multi-dimensional valued mappings but at the unnecessary expense of clarity. Since $\|\epsilon\|$ may be considered small with respect to the magnitude of the network inputs, we are interested in the local behavior of the network output that we will approximate as follows:

$$x^L_\epsilon - x^L = F(x+\epsilon) - F(x) \simeq \nabla_x F(x)^\top \epsilon \quad (9)$$

where $F(x) = f^L((W^L)^\top f^{L-1}((W^{L-1})^\top f^{L-2}(\dots f^0((W^0)^\top x + b^0)) + b^{L-2}) + b^{L-1})$ and $\nabla_x F$ is the gradient of F with respect to the network input x.
The linear approximation (9) is only local but its advantage is that it can easily be evaluated at many x values. Indeed, while visiting all input vector x during training, this information is usually available at a very low extra computational cost through automatic differentiation in most training computer algorithms and packages for neural networks (such as `TensorFlow` [1] and `Pytorch` [21]). Therefore, from local estimates at various training points x_i^0 for $i = 1, \dots, n$, we

calculate two n_0-dimensional vectors of network variation estimates $\widehat{\nabla_x F}$ and $\overline{\nabla_x F}$ defined as follows:

$$\left(\widehat{\nabla_x F}\right)_k = \text{sign}\left(\left(\nabla_x F(x^0_{i_k})\right)_k\right) \times v_k \quad \overline{\nabla_x F} = \frac{1}{n}\sum_{i=1}^{n} \nabla_x F(x^0_i) \tag{10}$$

where

$$(v_k, i_k) = \left(\max_{i\in\{1\dots,n\}} \left|\left(\nabla_x F(x^0_i)\right)_k\right|, \operatorname*{argmax}_{i\in\{1\dots,n\}} \left|\left(\nabla_x F(x^0_i)\right)_k\right| \right).$$

The quantity $\widehat{\nabla_x F}$ accounts for the maximum variation of the network response in every component direction of the network input encountered during the training and gives therefore a tighter linear upper bound than $\left(\prod_{l=1}^{L} W^l\right)$ when input data are part of the training set. The quantity $\overline{\nabla_x F}$ does not provide a linear upper bound but estimates an average linear behavior of the network response that be used in practice to estimate the required Γ value to reduce $\mathbb{P}_{\epsilon\sim D}\left(\|x^L - x^L_\epsilon\| \geq \Gamma\right)$ below α. The larger is the training dataset, the higher is the quality of these estimates. Replacing $L_f\left(\prod_{l=1}^{L} W^l\right)$ by these quantities in (7), we derive the following robustness bound for the network:

$$\left\| \widehat{\nabla_x F}\, \widehat{\nabla_x F}^\top \right\|_F \lesssim \frac{\Gamma^2}{\gamma^2\sqrt{2\log(1/\alpha)}}. \tag{11}$$

and the following estimate of Γ to achieve a robustness confidence level of $1-\alpha$:

$$\Gamma \gtrsim \gamma\left(2\log(1/\alpha)\|\overline{\nabla_x F}\,\overline{\nabla_x F}^\top\|_F^2\right)^{\frac{1}{4}} \tag{12}$$

where the symbols $\lesssim$ and $\gtrsim$ stand respectively for upper bound and lower bound approximations, which does not strictly guarantee the inequalities.

3.2 The specific case of piece-wise linear activations

When activation functions are piece-wise linear such as in the case of ReLu functions or when activations can be easily approximated by piece-wise linear functions such as in the case of sigmoids, it is possible to compute explicitly the above gradient $\nabla_x F$ and its average value over the entire training set $\overline{\nabla_x F}$. To this end, we introduce the *activation operator* and its average counterpart *average activation operator*.

Definition 3.1 *Consider a neural network layer l given by its weights matrix W^l and a piece-wise activation function f^l such that $f^l(t) = a_p t + b_p$ for $t \in I_p$ and where $I_p \subset \mathbb{R}$ for $p \in \{1, \cdots, P\}$ are P disjoint intervals such that $\text{dom}\, f = \cup_{p=1}^{P} I_p$ and $(a_p, b_p) \in \mathbb{R}^2$. Consider also a given input sample x that is propagated through the first l layers of the network to give the l-th layer input*

x^l. *We define as the* l*-th* activation operator *of* x*, the matrix* $M^{(l)} \in \mathbb{R}^{n_{l-1} \times n_l}$ *given by its rows as follows:*

$$(M^{(l)}(x))_i = (\underbrace{a_p \quad \cdots \quad a_p}_{n_{l-1}}) \quad \text{for} \quad \sum_{j=1}^{n_{l-1}} W_{ji}^l x^{(l-1)} \in I_p$$

and for $i \in \{1, \cdots, n_l\}$.

In other words, each row of the l-th activation operator $M^{(l)}(x)$ relates to one neuron of the l-th layer and it records the slope of the active linear piece of the neuron activation when an input x is propagated from the first layer to the l-th layer. Therefore the complete activation operator matrix records the various active slopes for all the neuron structure of the l-layer. The following definition considers the computation of such operators for all the dataset samples and averages row-wise (meaning neuron-wise) their slope values:

Definition 3.2 *Consider a training set* $S = \{x_1, \cdots, x_n\} \in \mathbb{R}^{n_0 \times n}$*, a neural network layer* l *given by its weights matrix* W^l *and a piece-wise activation function* f^l *such that* $f^l(t) = a_p t + b_p$ *for* $t \in I_p$ *and where* $I_p \subset \mathbb{R}$ *for* $p \in \{1, \cdots, P\}$ *are* P *disjoint intervals such that* $\operatorname{dom} f = \cup_{p=1}^{P} I_p$ *and* $(a_p, b_p) \in \mathbb{R}^2$*. We define as the* l*-th layer* average activation operator*, the matrix* $\overline{M}^{(l)} \in \mathbb{R}^{n_{l-1} \times n_l}$ *given by its rows as follows:*

$$(\overline{M}^{(l)})_i = \frac{1}{n} \sum_{m=1}^{n} (M^{(l)}(x_m))_i$$

where $M^{(l)}(x_m)$ *is the* l*-th activation operator of* x_m *and* $i \in \{1, \cdots, n_l\}$.

It is important to observe that the computation of the average activation operator is at little extra costs during training since a forward pass through the network is anyway necessary during training and these slope values can be accumulated during the process. This cost is mainly a storage cost. The associated computational cost is the cost for averaging which is even smaller than the cost for computing the network gradient we have previously seen since it requires automatic differentiation.
In the case of piece-wise linear function, the inequality (8) can therefore be written as

$$\mathbb{P}_{\epsilon \sim D}\left(\|x^L - x_\epsilon^L\| \leq \Gamma\right) \gtrsim \mathbb{P}_{\epsilon \sim D}\left[tr\left(\epsilon\epsilon^\top \left(\prod_{l=1}^{L} \overline{M}^{(l)} W^l\right)\left(\prod_{l=1}^{L} \overline{M}^{(l)} W^l\right)^\top\right) \leq \Gamma^2\right]. \tag{13}$$

Additionally, for a given confidence $1 - \alpha$, we have

$$\Gamma \gtrsim \gamma \left(2\log(1/\alpha) \left\| \left(\prod_{l=1}^{L} \overline{M}^{(l)} W^l\right)\left(\prod_{l=1}^{L} \overline{M}^{(l)} W^l\right)^\top \right\|_F^2\right)^{\frac{1}{4}} \tag{14}$$

Similar bound estimates could be derived also by defining a *maximum activation operator* and by taking the row-wise maximum activation operator over the entire dataset.

3.3 Discussion on the various bounds

The bounds given in ((11),(13)) and ((12),(14)) are interesting in practice in several ways. First, they can be used as a network probabilistic robustness estimates. Indeed, assume that the magnitude of the input uncertainty γ is known and a level of uncertainty Γ is required for a practical application. Then set a level of confidence $1-\alpha$. After training, if the network weights satisfy the bounds ((11),(13)), it implies that the output uncertainty will be below the required level Γ with a propability around $1-\alpha$. Of course, obtaining strict guarantees of robustness would be even more interesting. However, getting a strict bound inequalities would require a strict upper bound on the network local variation. An available upper bound is the global upper bound that we have calculated in Section 2.2 using layer Lipschitz constant upper bounds but at the expense of bound tightness. For this reason, instead of computing upper bounds that are loose, we propose estimates of bounds that do not provide strict certificate of robustness but are tighter to what is observed in practice as we will see in Section 5.

Alternatively, assume now that you know γ and after training you observe the quantity $\|\overline{\nabla_x F}\,\overline{\nabla_x F^\top}\|_F$, or $\left\|\left(\prod_{l=1}^{L}\overline{M}^{(l)}W^l\right)\left(\prod_{l=1}^{L}\overline{M}^{(l)}W^l\right)^\top\right\|_F$. For a level of confidence $1-\alpha$, ((12),(14)) give estimates of the variation Γ of the output that you might expect. Therefore, for a specific application, the calculated Γ provides an additional performance criterion for the trained network. In addition to validation accuracy and generalization performance, the Γ value gives an estimate of the sensitivity of the network to input uncertainty. The larger is Γ, the more sensitive to uncertainty is the network since, for the same level of α and γ, one has to expect that the output should deviate (in norm) from a nominal value by Γ.
Finally, these bounds emphasize also the relationship between the robustness of the network and the weights values achieved after training. One idea is to use this knowledge to drive the training process to regions where these bound estimates are small in order to design networks that are intrinsically robust. This is the objective of the next section.

4 Controlling the bound during training

In this section, we are interesting in exploiting the bounds derived above during the process of training the neural network. The main idea would be to ensure that optimal weights after training are satisfying the bound constraint (11) or (13). Naturally, this could be formulated as a constrained optimization training

problem. Stochastic projected gradient techniques [18, 16] could be used to solve such a problem. However, in the general case, the projection operator for such constraint is not simple and would require important computational effort. Therefore, instead of ensuring the constraint, we propose to regularize the loss function during training by adding a penalization term as follows:

$$\min_{\theta=(W,b)} \frac{1}{\lambda}\mathcal{L}(x,y,\theta) + \|Q(W)\|_F^2$$

where λ is a positive parameter, $\mathcal{L}$ is a loss function (mean squared error for example) and $\|Q(W)\|_F$ is the Frobenius norm of a matrix $Q(W)$ that could be chosen among $Q(W) = \widehat{\nabla_x F}\,\widehat{\nabla_x F}^\top$, $Q(W) = \overline{\nabla_x F}\,\overline{\nabla_x F}^\top$, $Q(W) = \left(\prod_{l=1}^L W^l\right)\left(\prod_{l=1}^L W^l\right)^\top$ or $Q(W) = \left(\prod_{l=1}^L \overline{M}^{(l)} W^l\right)\left(\prod_{l=1}^L \overline{M}^{(l)} W^l\right)^\top$, depending on which bound from above we want to exploit.
Regularization is a common practice in machine learning [3, 11] and is usually proposed to avoid overfitting and increase model generalization. The connection between generalization and robustness to input uncertainty in machine learning models has been established in several studies [28, 22].
The neural network update rule during training can be expressed as

$$\frac{1}{\lambda}\nabla_\theta \mathcal{L}(x,y,\theta) + \nabla_\theta \|Q(W)\|_F^2$$

The second term corresponding to the regularization acts as a correction term of the gradient of the loss to select direction of move that will ensure that $\|Q(W)\|_F$ will be decreased at each iteration. Of course, the overall optimization clearly depends on the nature of $Q(W)$. For choices of $Q(W)$ listed above and especially for the network gradient or activation operator methods, depending on the training samples, the regularization may not be convex and gradient descent convergence may be slow to reach a local minimum. However, in practice, during experiments (see Section 5), no convergence issue was encountered.
Intuitively, the $\|Q(W)\|_F^2$ regularization term acts as a special weight contraction and it is natural to consider alternative possibilities to reduce the magnitude of the network weights. One alternative is the squared spectral norm (largest eigenvalue) of $Q(W)$ that would also account for the maximum absolute contraction of a vector when multiplied by $Q(W)$. Finally, in [7], the product $\prod_{l=1}^L \|W^l\|$ which is an upper bound of the Lipschitz constant of the network has also been proposed as regularization that promotes robustness. It accounts for the overall Lipschitz regularity of the network and acts also as an overall control on the contraction power of the network by coupling layers and allowing some weights to grow for some layers as long as in other layers others weights are getting smaller to compensate.
When $Q_W = \left(\prod_{l=1}^L W^l\right)\left(\prod_{l=1}^L W^l\right)^\top$, its Frobenius norm and the Lipschitz constant gradient can be explicitely derived and integrated into the backpropagation scheme and chain rule of gradients in order to optimize the augmented loss during the training phase. However, for the spectral norm, approximation

methods are necessary and gradient will have to be computed using numerical differentiation techniques. Among available approximation methods, the power iteration algorithm [2], or preferably the Lanczos algorithm [17] since $Q(W)$ is symmetric, is well suited for the purpose. Note that there is no real need to approximate $\lambda_{max}(Q(W))$ (the largest eigenvalue of $Q(W)$) with great accuracy as it is only used as a regularization function to guide the optimization process towards optimal regions where the spectral norm is reduced. Therefore, an alternative approximation technique is to use an upper bound of $\lambda_{max}(Q(W))$. As $Q(W)$ is positive definite, we propose to use the Dembo's upper bound [8] defined as follows:
Let $A_n \in \mathbb{R}^{n\times n}$ be an Hermitian positive definite matrix and let $\lambda_1^{(n)} \leq \ldots \leq \lambda_n^{(n)}$ be the eigenvalues of A_n. The matrix A_n can be written as

$$A_n = \begin{pmatrix} A_{n-1} & b \\ b^H & c \end{pmatrix}$$

where b^H denotes the Hermitian transpose and the largest eigenvalue of A_n satisfies

$$\lambda_n^{(n)} \leq \frac{c + \lambda_n^{(n-1)}}{2} + \sqrt{\frac{(c - \lambda_n^{(n-1)})^2}{4} + b^H b}$$

In the next section, we propose to carry out experiments with these various regularization strategies and evaluate their respective impact on the robustness properties of the network.

5 Experiments

In order to assess the quality of the estimated probability bounds, experiments are conducted on two types of datasets (regression and classification data). The neural network, its training and testing are implemented in the `python` [24] environment using the `keras` [6] library and `Tensorflow` [1] backend. Results for the general network gradient strategy and the activation operator strategy presented in section 3 are tested and presented next.

5.1 Experiments with network gradient bounds (11) and (12)

In this experiment, we consider the `BOSTON` [13], `DIABETES` [25] and `CALIFORNIA` [20] datasets. Several neural network architectures are considered and tested with the constraint of having sufficient network depth in order for the regularization schemes to achieve a balance between weight values and network expressiveness [7]. We retain the following architecture: 4 dense hidden layers with 50 ReLu activations neuronal units and one dense linear output layer. All results that are presented below are average results from 10 independent runs that are carried out after random shuffling and random splitting of datasets. All

Table 1: Dataset information and experimental setup

Dataset	# inputs	# samples	test/train ratio	batch size	# epochs	learning rate
`BOSTON`	13	606	0.2	50	100	0.001
`DIABETES`	10	442	0.2	200	30	0.001
`CALIFORNIA`	8	20640	0.4	600	30	0.001

dataset samples are scaled so that they lie in $[-1, 1]$. All neural network training procedures are executed with the `ADAM` stochastic optimization algorithm with default parameters as given in [14]. Additional details about the datasets dimensions and training parameters are given in Table 1.

All comparison results provided below are referring to the following methods as described in section 4:

- `no reg`: training procedure with mean squared error loss without regularization
- `Lipschitz reg`: training procedure with mean squared error loss with the $\prod_{l=1}^{L} \|W^l\|$ regularizer as described in [7].
- `Gradient reg`: training procedure with mean squared error loss with the $\|Q(W)\|^2$ regularizer as described above and where $Q(W) = \overline{\nabla_x F}\,\overline{\nabla_x F}^{\top}$.
- `MaxEig reg`: training procedure with mean squared error loss with the $\lambda_{max}(Q(W))^2$ regularizer.

In order to estimate the probability $\mathbb{P}_{\epsilon\sim D}\left(\|y - y_\epsilon\| \leq \Gamma\right)$, the following procedure is applied. For each test sample from the validation set, we generate random vectors ϵ_j with $j \in \{1, \ldots, 10\}$ and calculate the following probability estimate:

$$\frac{1}{10 \times T} \sum_{i=1}^{T} \sum_{j=1}^{10} \mathbf{1}_{\{\|y^{(i)} - y^{(i)}_{\epsilon_j}\| \leq \Gamma\}}(\epsilon_j) \tag{15}$$

where T is the number of samples in the testing set, $y^{(i)}$ is the desired output for the i-th testing sample and $y^{(i)}_{\epsilon_j}$ is the output of the network calculated via a forward pass through the network for the input vector $x^{(i)} + \epsilon_j$. In all experiments, we have $supp(D) = [-\gamma, \gamma]$ with various levels of γ. In the figures described below, γ is referred as `gamma` and Γ as `Gamma`.

Figure 1(right) reports these observed probabilities together with the estimated tail probabilities given by

$$e^{-\frac{\Gamma^4}{2\gamma^4 \|\overline{\nabla_x F}\,\overline{\nabla_x F}^{\top}\|_F^2}}$$

for various values of Γ, while Figure 1(left) reports the corresponding mean validation error achieved during the training process. The probability level

$1-\alpha$ (with $\alpha = 0.05$) is also marked with a blue dashed line on each plot on the right. Figure 2 provides further details about the magnitude of the norm of the network gradient $\|\widehat{\nabla_x F}\|$ and $\lambda_{max}(Q_W)^2$ (re-scaled by a factor 10 to ease the reading of the plot) for each dataset and the four regularization strategies. Finally, Figure 3 provides, for each dataset and each regularization strategies a comparison of the Γ values achieved to reach a $1-\alpha$ probability level. Three values are reported each time, $\Gamma(max) = \left(2\log(1/\alpha)\|\widehat{\nabla_x F}\,\widehat{\nabla_x F}^\top\|_F^2\right)^{\frac{1}{4}}$, $\Gamma(mean) = \left(2\log(1/\alpha)\|\overline{\nabla_x F}\,\overline{\nabla_x F}^\top\|_F^2\right)^{\frac{1}{4}}$ and the Γ value observed so that the probability given in (15) reaches a level $1-\alpha$.

We see in Figure 1 that, for the three datasets and for a probability of 0.95, the calculated Γ value (x axis) obtained by the expression of the exponential tail probability, provides a very good estimate of the Γ value given by the observed probability (probability that the output deviates from its nominal value by more than Γ). This validates experimentally, at least for these datasets, the relevance of the estimate given in (12). For the `BOSTON` dataset, all regularizing strategies give similar Γ values whereas for the `DIABETES` and `CALIFORNIA`, the Γ values are more sensitive to the type of regularization employed. However, surprisingly, no general rule can be given from these results. It is difficult to say which regularizer performs best. It is dataset dependent. The left hand side plots of Figure 1, representing the validation mean absolute error, show that similar training performance were achieved by the four regularizing methods and do not provide further explanation of this phenomenon. We believe, without providing any evidence of it, that the high non linearity of the neural network error surface may explain it. Indeed, after training, the optimization algorithm has reached a local minimum where the loss value may not have decreased sufficiently to really express the regularization power of the regularizer. This depends on the geometry of the error surface that is greatly dependent on input data.

On Figure 2, the norm of the network gradient tends to be slightly smaller for the `Gradient reg` strategy. This would confirm that regularizing by the network gradient would help in achieving better robustness. Additionally, the figure also shows that the maximum eigenvalue regularization is not correlated to the network gradient norm and may not be a suitable alternative for robustness purposes. The Γ value comparison in Figure 3 confirms that the Γ estimates calculated by the proposed method are very closed to the observed values. This is true for all datasets and regularizing strategies. Furthermore, the Γ upper bound values ($\Gamma(max)$), are loose as expected in Section 3 but provide certificates for robustness. These certificates follow the same pattern as the norm of the network gradient in Figure 2, which was also expected since their expression in (11) are directly dependent. Therefore, as for the network

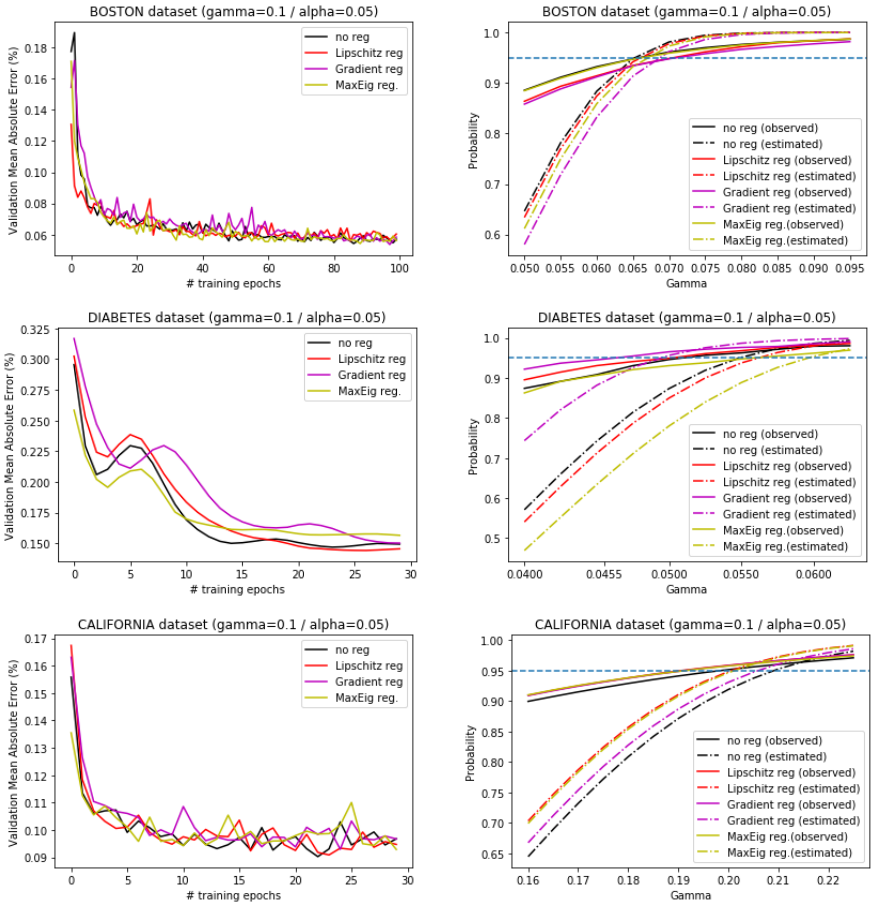

Figure 1: Mean absolute validation error profiles during training with $Q(W) = \overline{\nabla_x F}\,\overline{\nabla_x F}^{\top}$ (left) & Comparison of $\mathbb{P}_{\epsilon\sim D}\left(\|y - y_\epsilon\| \leq \Gamma\right)$ and exponential tail bound for various levels of Γ (right)

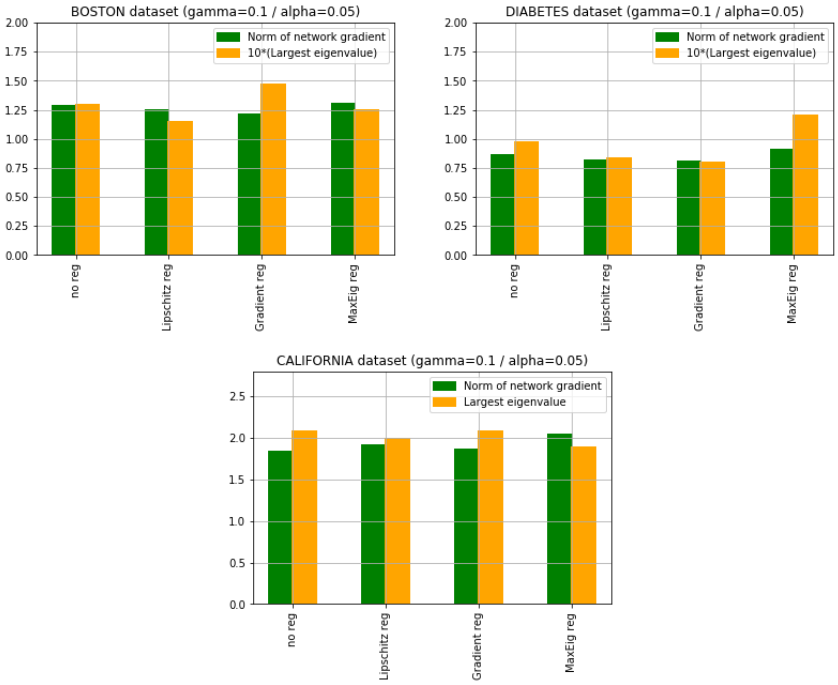

Figure 2: Norm of network gradient $\|\widehat{\nabla_x F}\|$ and Estimate of $\lambda_{max}(Q(W))$ after training

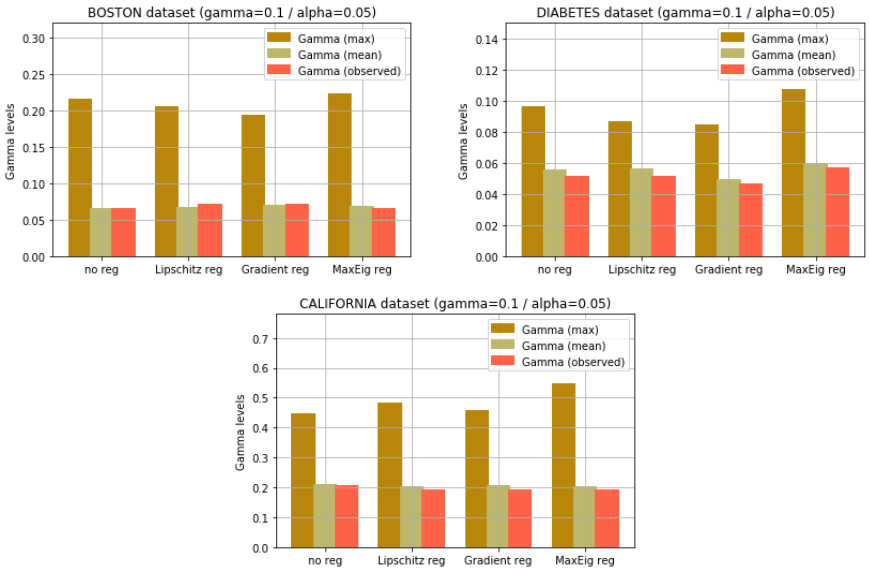

Figure 3: Comparison of calculated $\Gamma(max)$ and $\Gamma(mean)$ values for 0.95 confidence level using $\widehat{\nabla_x F}$ and $\overline{\nabla_x F}$ respectively with the $\Gamma(observed)$ value observed so that the probability given in (15) reaches a level 0.95

gradient, we observed that these certificates are better (tighter but still quite loose) for the network gradient regularizing strategy.

5.2 Experiments with activation operator bound (13)

The setting of this experiment is identical to the previous one. For the same datasets, the same network architecture, the same data preparation process and the same noise generation for estimating the probability (15), we now consider an additional regularization scheme corresponding to the average activation operator method and compute an estimated tail probability with respect to this method rather than the network gradient method. More specifically, the additional regularization scheme is as follows:

- `WMRO reg`: training procedure with mean squared error loss with the $\|Q_W\|^2$ regularizer as described above and where

$$Q(W) = \left(\prod_{l=1}^{L} \overline{M}^{(l)} W^l\right)\left(\prod_{l=1}^{L} \overline{M}^{(l)} W^l\right)^\top$$

and for all regularization strategies we also report the following estimated tail probabilities:

$$e^{-\frac{\Gamma^4}{2\gamma^4\|Q(W)\|_F^2}}$$

Figure 4(right) reports the above tail probability estimates for various levels of Γ as well as the observed probability (15) after training.

In Figure 1, we see that, for the three datasets and for a probability of 0.95, the calculated Γ value (x axis) obtained by the expression of the exponential tail probability, provides also a good estimates of the Γ value given by the observed probability, although not as sharp as in the case of the general network gradient method.

Experiments with real life data

Next, we use the MNIST dataset to carry out similar experiments on classification real life data. The dataset contains 60000 training samples and 10000 validation samples. Each sample is a 28×28 image of handwritten characters. As before, the network architecture is composed of a 5 fully connected layers of 50 neurons each . Hidden activations are ReLu activations and output activation is the softmax activation. The exact same perturbation process is used to calculate after training the observed probability that the output deviates from its nominal value by more than Γ. To illustrate how these noise perturbations act on the image samples, Figure 5 shows 5 input samples where various noise level η were applied. Figure 6(center) reports the observed probability and the calculated tail probability estimates for various levels of Γ. Figure 6(right) shows the difference between observed and estimated probability the various levels of Γ . As before, Figure 6(left) provides the average validation accuracies achieved by the various regularized loss minimization during training.
We see in Figure 6(right) that among the various strategies, the `Gradient reg` and `WMRO reg` achieve the smallest values, meaning that there are better estimates of the observed probability. Again, as in the previous experiment, the network gradient method tends to give slightly better results than the mean activation operator method.

We see in Figure 6(right) that among the various strategies, the `Gradient reg` and `WMRO reg` achieve the smallest values, meaning that there are better estimates of the observed probability. Again, as in the previous experiment, the network gradient method tends to give slightly better results than the mean activation operator method.

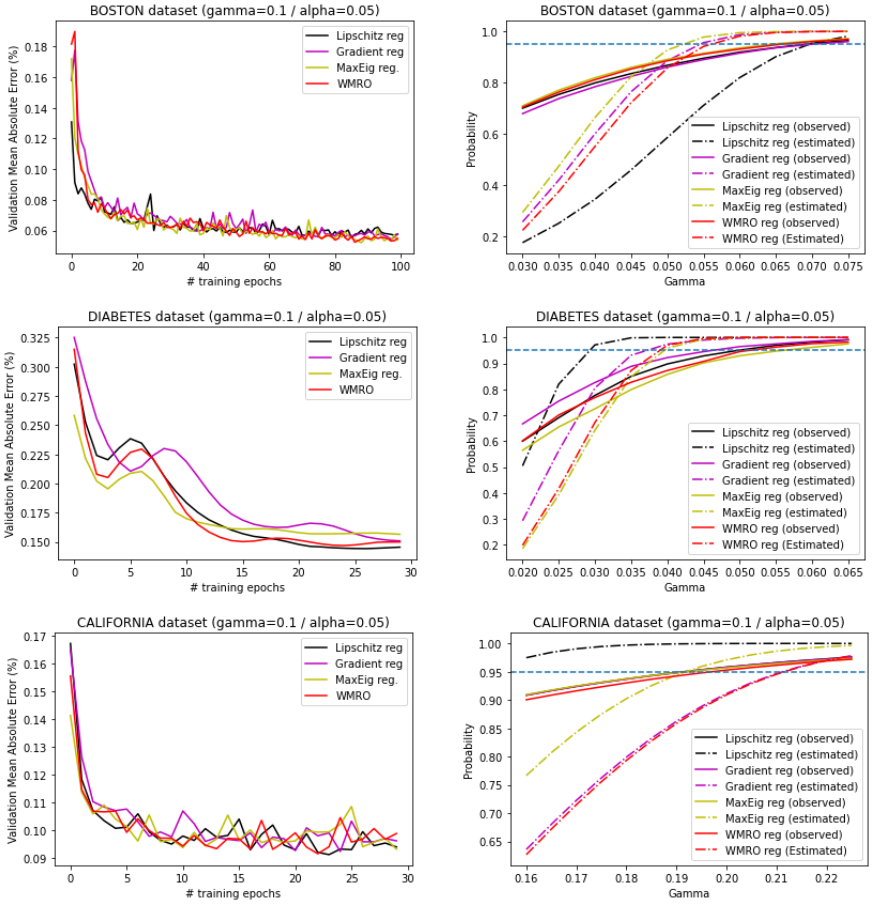

Figure 4: Mean absolute validation error profiles during training with $Q(W) = \left(\prod_{l=1}^{L} \overline{M}^{(l)} W^l\right) \left(\prod_{l=1}^{L} \overline{M}^{(l)} W^l\right)^\top$ (left) and comparison of $\mathbb{P}_{\epsilon \sim D}\left(\|y - y_\epsilon\| \leq \Gamma\right)$ vs exponential tail bound for various levels of Γ (right)

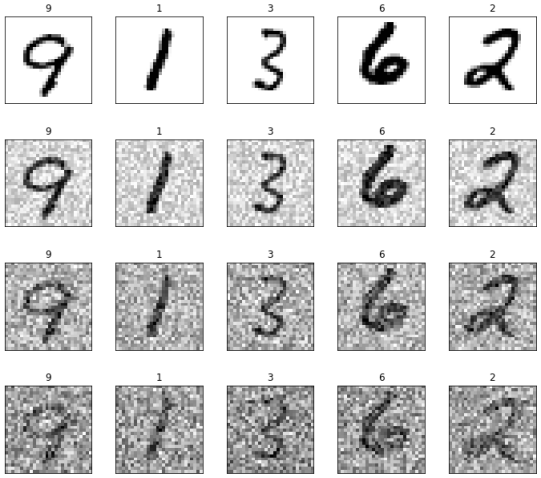

Figure 5: MNIST dataset samples with various levels γ of random noise ($\eta \in \{0.0, 0.2, 0.4, 0.6\}$ from top to bottom)

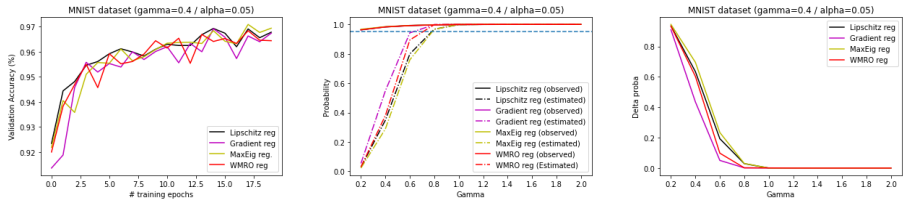

Figure 6: Mean absolute validation error profiles during training with $Q(W) = \left(\prod_{l=1}^{L} \overline{M}^{(l)} W^l\right) \left(\prod_{l=1}^{L} \overline{M}^{(l)} W^l\right)^\top$ (left), Comparison of $\mathbb{P}_{\epsilon \sim D}\left(\|y - y_\epsilon\| \leq \Gamma\right)$ vs exponential tail bound for various levels of Γ (center) and difference between observed probability $\mathbb{P}_{\epsilon \sim D}\left(\|y - y_\epsilon\| \leq \Gamma\right)$ and exponential tail bound (right)

6 Conclusions

In this study, we have proposed analytical probabilistic estimates (and certificates) for feed-forward neural networks. The idea combines tail probability bound calculation using the Cramer-Chernoff scheme and the estimation of the network local variation. The network gradient computation is using the automatic differentiation procedure available in many neural network training packages and carried out only at the training samples which does not require much extra computational cost. In the case of piece-wise linear activations, an alternative bound calculation was proposed. It is also based on the local behavior of the network at the training samples but provides an explicit average activation operator calculation that does not require automatic differentiation. Experiments with these methods have been conducted on public datasets and have shown that the calculated robustness performance estimates well the observed empirical performance. Further analysis on these datasets show that the quality of the estimates is not really impacted by the regularization strategy, however, the network gradient regularization tends to generate slightly more robust network architectures.

Acknowledgements

Our work has benefited from the AI Interdisciplinary Institute ANITI. ANITI is funded by the French "Investing for the Future - PIA3" program under the Grant agreement # ANR-19-PI3A-0004.

References

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

[2] G. Allaire. *Numerical Analysis and Optimization.* Oxford Science Publications, 2007.

[3] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag, Berlin, Heidelberg, 2006.

[4] A. Boopathy, T.-W. Weng, P.-Y. Chen, S. Liu, and L. Daniel. Cnn-cert: An efficient framework for certifying robustness of convolutional neural networks, 2018.

[5] S. Boucheron, G. Lugosi, and P. Massart. *Concentration Inequalities, a nonasymptotic theory of independence*. Oxford, 2013.

[6] F. Chollet et al. Keras. `https://keras.io`, 2015.

[7] N. Couellan. The coupling effect of Lipschitz regularization in deep neural networks. *SN Computer Science*, 2021. to appear.

[8] A. Dembo. Bounds on the extreme eigenvalues of positive-definite toeplitz matrices. *IEEE Transactions on Information Theory*, 34(2):352–355, 1988.

[9] A. Fawzi, S. Moosavi-Dezfooli, and P. Frossard. The robustness of deep networks: A geometrical perspective. *IEEE Signal Processing Magazine*, 34(6):50–62, 2017.

[10] C. Finlay, A. Oberman, and B. Abbasi. Improved robustness to adversarial examples using lipschitz regularization of the loss. *arXiv:1810.00953v3*, 2018.

[11] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[12] H. Gouk, E. Frank, B. Pfahringer, and M. Cree. Regularisation of neural networks by enforcing lipschitz continuity. *arXiv:1804.04368v2*, 2018.

[13] D. Harrison and D. Rubinfeld. Hedonic prices and the demand for clean air. *J. Environ. Economics and Management*, 5:81–102, 1978.

[14] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[15] J. Z. Kolter and E. Wong. Provable defenses against adversarial examples via the convex outer adversarial polytope. *CoRR*, abs/1711.00851, 2017.

[16] S. Lacoste-Julien, M. Schmidt, and F. Bach. A simpler approach to obtaining an o(1/t) convergence rate for the projected stochastic subgradient method, 2012.

[17] C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of Research of the National Bureau of Standards*, 45(4):255 – 282, 1950.

[18] A. Nedic and S. Lee. On stochastic subgradient mirror-descent algorithm with weighted averaging. *SIAM Journal on Optimization*, 24(1):84–107, 2014.

[19] A. Oberman and J. Calder. Lipschitz regularized deep neural networks converge and generalize. *arXiv:1808.09540v3*, 2018.

[20] R. K. Pace and R. Barry. Sparse spatial autoregressions. *Statistics and Probability Letters*, 33(3):291 – 297, 1997.

[21] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[22] M. Staib and S. Jegelka. Distributionally robust optimization and generalization in kernel methods, 2019.

[23] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, and I. G. et al. Intriguing properties of neural networks. *International Conference on learning representations (ICLR)*, 2014.

[24] P. C. Team. Python: A dynamic, open source programming language, python software foundation. URL https://www.python.org/, 2015.

[25] R. Tibshirani, I. Johnstone, T. Hastie, and B. Efron. Least angle regression. *The Annals of Statistics*, 32(2):407–499, 2004.

[26] A. Virmaux and K. Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. *Advances in Neural Information Processing Systems*, 2018.

[27] L. Weng, P.-Y. Chen, L. Nguyen, M. Squillante, A. Boopathy, I. Oseledets, and L. Daniel. PROVEN: Verifying robustness of neural networks with a probabilistic approach. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6727–6736, 2019.

[28] H. Xu, C. Caramanis, and S. Mannor. Robustness and regularization of support vector machines. *J. Mach. Learn. Res.*, 10:1485–1510, 2009.

[29] D.-X. Zhou. Deep distributed convolutional neural networks: Universality. *Analysis and Applications*, 16(06):895–919, nov 2018.

[30] D.-X. Zhou. Universality of deep convolutional neural networks. *Applied and Computational Harmonic Analysis*, 48(2):787–794, 2020.